universität wien

DISSERTATION

Titel der Dissertation

„Wissensrepräsentation und automatisierte Entscheidungsfindung am Beispiel des Kriegsopferversorgungsgesetzes“

Verfasser

Mag. iur. Johannes Sebastian Scharf

angestrebter akademischer Grad

Doktor der Rechtswissenschaften (Dr. iur.)

Wien, 2015

Studienkennzahl lt. Studienblatt:	A 783 101
Dissertationsgebiet lt. Studienblatt:	Rechtswissenschaften
Betreuerin / Betreuer:	ao. Univ.-Prof. MMag. DDr. Erich Schweighofer

Für meine Großeltern Leopoldine und Johann Mayer

Danksagung

Diese herausfordernde und vor allem für das Gebiet der Rechtswissenschaften außergewöhnliche Dissertation wurde größtenteils von Oktober 2013 bis Februar 2015 verfasst. Es war eine spannende, aber auch ungemein fordernde Zeit, in der mir der Zuspruch meiner Familie und meiner Freunde die notwendige Kraft gegeben hat, die Arbeit letztendlich abschließen zu können.

Insbesondere möchte ich meinem Betreuer Prof. Erich Schweighofer danken, der mich immer wieder motiviert und angespornt hat, weitere Themen zu untersuchen und meine Erkenntnisse auf wissenschaftlichen Konferenzen zu präsentieren. Die Behandlung dieses komplexen Themas wäre ohne seine Hilfe undenkbar gewesen. Weiters sei Prof. Raschauer und Prof. Lachmayer herzlich für ihre Bereitschaft zur Übernahme der Begutachtung gedankt.

Ferner danke ich meinen Freunden Andreas Zinkl, Karl Senft, Mathias Burger und Robert Scheibelhofer für die Diskussionen, die dazu beigetragen haben, neue Sichtweisen zu entwickeln. Auch meinen Mitstreitern auf Konferenzen, Anton Geist und Thomas Preiß, möchte ich für den regen Gedankenaustausch und ihre kritischen Kommentare zu meiner Arbeit danken.

Großer Dank gebührt auch Eva und Roman Leodolter sowie Franziska Stohl, da sie stets ein offenes Ohr für meine Anliegen hatten und mir bei geselligen Zusammenkünften geholfen haben, Termine und Anspannung eine Zeit lang zu vergessen, um wieder neue Kraft zu schöpfen.

Die größte Bürde, neben der ungeheuren Komplexität des Themas, war der Umstand, dass mich die Verwirklichung der Dissertation zu einem gestressten und anstrengenden Menschen gemacht hat. Obwohl diese Last auf ihren Rücken fiel, hat meine liebe Lebensgefährtin Christina, der Eckpfeiler meines Lebens, nie die Geduld mit mir verloren und mich in Phasen der Erschöpfung immer wieder emutigt und aufgerichtet, oft ohne ein Wort zu sagen. Ihre großartige Unterstützung kann ich ihr niemals vergelten.

Diese Dissertation widme ich meinen Großeltern Leopoldine und Johann Mayer, die mich in jungen Jahren beherbergt haben. Ohne sie wäre mein Weg unmöglich gewesen.

Wien, im März 2015 Johannes Scharf

Inhaltsverzeichnis

1. EINLEITUNG 1

1.1. MOTIVATION 1

1.2. HYPOTHESEN UND FORSCHUNGSFRAGEN 5

1.2.1. Hypothesen 5

1.2.2. Forschungsfragen 5

1.3. ONTOLOGIEN 8

1.4. FORSCHUNGSANSATZ 13

1.5. BEITRAG UND ÜBERBLICK 14

2. E-GOVERNMENT 19

2.1. WAS IST E-GOVERNMENT? 19

2.2. AKTUELLER STAND 20

2.3. RECHTSGRUNDLAGEN 22

2.4. ADRESSATEN 25

2.5. ZIELE 26

2.5.1. Steigerung der Effizienz der Verwaltung 26

2.5.2. Zugang für alle Bürger zu E-Government-Diensten 28

2.5.3. Datenschutz und Sicherheit 30

2.6. KRITIK 31

2.7. ZUKUNFTSAUSSICHTEN 33

3. STATUS QUO DER VERWALTUNGSAUTOMATISIERUNG 37

3.1. EINLEITUNG 37

3.2. ANFORDERUNGEN AN JURISTISCHE EXPERTENSYSTEME 38

3.3. GÄNGIGE LÖSUNGSANSÄTZE 42

3.3.1. Objektorientierte Programmierung 43

3.3.2. Modellgetriebene Softwareentwicklung 53

3.4. DISKUSSION 56

4. DIE FORMALISIERUNG DES RECHTS 59

4.1. EINLEITUNG 59

4.2. VORÜBERLEGUNGEN 60

4.3. BEGRIFFSBESTIMMUNG 61

4.3.1. Die Formalisierung im engeren Sinn 62

4.3.2. Die Formalisierung im weiteren Sinn 63

4.3.3. Abschließende Bemerkungen 65

4.4. DIE AXIOMATISIERUNG 65

4.4.1. Die Axiomatisierung als theoretischer Ansatz 65

4.4.2. Kritik an der Axiomatisierung *68*
4.4.3. Abschließende Bemerkungen *77*
4.5. Die Formalisierung im Kontext der Dissertation 78
4.6. Vorgehensweise 80
4.7. Die Relativität der Formalisierung 82
4.8. Grundlagen der Rechtsanwendung 83
4.8.1. Einleitung *83*
4.8.2. Die Subsumtion *84*
4.8.3. Die Interpretation von Rechtsvorschriften *86*
4.9. Grenzen 89
4.9.1. Einleitung *89*
4.9.2. Unbestimmte Gesetzesbegriffe *90*
4.9.3. Ermessen *91*
4.9.4. Rechtsfortbildung *97*
4.9.5. Lösungsansätze *99*
4.10. Diskussion 104

5. WISSENSREPRÄSENTATION 109

5.1. Einleitung 109
5.1.1. Künstliche Intelligenz *109*
5.1.2. Wissensrepräsentation *111*
5.2. Über Modelle 114
5.3. Wissen und Wissensarten 116
5.3.1. Was ist Wissen? *116*
5.3.2. Wissensarten *118*
5.3.3. Qualitäten des Wissens *119*
5.3.4. Abschließende Bemerkungen *121*
5.4. Ansätze zur juristischen Wissensrepräsentation 122
5.4.1. Überblick *122*
5.4.2. Der regelbasierte Ansatz *123*
5.4.3. Der fallbasierte Ansatz *127*
5.4.4. Der logische Ansatz *129*
5.4.5. Argumentation *135*
5.4.6. Sonstige Ansätze *139*
5.4.7. Abschließende Bemerkungen *140*
5.5. Aktuelle Probleme 141
5.5.1. Die Ungenauigkeit der natürlichen Sprache und von rechtlichen Begriffen *143*
5.5.2. Ähnlichkeit und Analogie *144*
5.5.3. Die Repräsentation des Kontexts *145*

5.5.4. Das Problem der Symbolverankerung ... *147*
5.5.5. Die Repräsentation der Kausalität, von Bedingungen und Alternativen ... *149*
5.5.6. Die Gewichtung von Argumenten ... *150*
5.5.7. Das grundlegende Problem der Kontinuität ... *151*
5.5.8. Menschliche Intelligenz ... *152*
5.5.9. Abschließende Bemerkungen ... *154*

6. SEMANTIC WEB ... 155

6.1. Einleitung ... 155
6.2. Bausteine des Semantic Web ... 155
6.3. Schichten des Semantic Web ... 158
6.4. Was ist eine Ontologie? ... 161
6.5. Die Relevanz von Ontologien für die Formalisierung ... 163
6.6. Sonstige Einsatzmöglichkeiten von Ontologien ... 166
6.6.1. Integration heterogener Daten ... *166*
6.6.2. Automatische Verwendung von (neuen) Services ... *168*

7. DIE ANFORDERUNGEN AN SPRACHEN ZUR FORMALISIERUNG VON VERWALTUNGSRECHT ... 169

7.1. Einleitung ... 169
7.2. Die Besonderheiten des Rechts ... 170
7.3. Der Begriff „Regel“ ... 174
7.4. Charakteristika von Rechtsnormen ... 176
7.4.1. Semantische Eigenschaften ... *176*
7.4.2. Logische Eigenschaften ... *177*
7.4.3. Rechtsprozessuale Eigenschaften ... *180*
7.5. Zusätzliche Anforderungen ... 181
7.6. Die Gewichtung der Anforderungen ... 183
7.7. Diskussion ... 184

8. SYSTEMDESIGN UND ARCHITEKTUR ... 189

8.1. Einleitung ... 189
8.2. Software zur semi-automatisierten Vollziehung des KOVG ... 189
8.3. Das Potential semi-automatisierter Vollziehung ... 193

9. DIE FORMALISIERUNG DES KRIEGSOPFERVERSORGUNGSGESETZES IN JAVA ... 195

9.1. Einleitung ... 195
9.2. Das Kriegsopferversorgungsgesetz ... 195
9.3. Die Beschädigtenrente ... 197
9.3.1. Die gesetzliche Grundlage ... *197*
9.3.2. Das Berechnungsschema ... *204*

9.4. Systembeschreibung 205
9.4.1. Architektur 205
9.4.2. Das Subsumtionsmodul 208
9.4.3. Die Funktionsweise des Systems 210
9.4.4. Abschließende Bemerkungen 212
9.5. Die Formalisierung in Java 213
9.5.1. Das Programm im Überblick 213
9.5.2. Anforderungen 216
9.5.3. Designprinzipien 216
9.5.4. Das Programm in Aktion 221
9.5.5. Entwicklungsmethode 223
9.5.6. Entwicklungsumgebung 224
9.5.7. Architektur 224
9.5.8. Das Domänenmodell 228
9.5.9. Die Implementierung der automatisierten Vollziehung 231
9.5.10. Datenzugriff 246
9.6. Diskussion 247
9.6.1. Die Behandlung von Konflikten 247
9.6.2. Gruppierung von Regeln 249
9.6.3. Schlussbetrachtung 250

10. DIE ONTOLOGIE DES KRIEGSOPFERVERSORGUNGSGESETZES 253

10.1. Einleitung 253
10.1.1. Die LKIF-Core Ontology 255
10.1.2. Das KOVG 256
10.2. Die Verwendung von OWL zur automatisierten Ermittlung von Ansprüchen 257
10.3. Methodologie 258
10.4. Die Ontologie des Kriegsopferversorgungsgesetzes im Überblick 262
10.5. Die Domänenontologie 265
10.5.1. Taxonomie 266
10.6. Eine simple Ontologie zur Repräsentation von Normen 269
10.6.1. Die Klassifizierung des Antragstellers 269
10.6.2. Die automatisierte Feststellung der Ansprüche 270
10.6.3. Beispiele 274
10.7. Die Verwendung der LKIF-Core Ontologie 278
10.7.1. Die Formalisierung von Legaldefinitionen 279
10.7.2. Die Formalisierung von präskriptiven Normen und Sachverhalten 280
10.7.3. LKIF-Core Light 285
10.7.4. Abschließende Bemerkungen 287

10.8. Eine hybride Architektur zur Unterstützung der rechtlichen Entscheidungsfindung 289
10.9. Verwandte Arbeiten 293
10.9.1. Die Formalisierung von Verboten, Geboten und Erlaubnissen mit OWL 2 293
10.9.2. Die Formalisierung in Java 295
10.10. Diskussion 297
10.10.1. Vor- und Nachteile des ontologischen Ansatzes 298
10.10.2. Konklusion und Empfehlungen 304
10.10.3. Zukünftige Forschungen 306

11. ERKENNTNISSE DER FORMALISIERUNG 309

11.1. Einleitung 309
11.2. Zeitlicher Geltungsbereich und substantielle rechtliche Beurteilung 310
11.3. Rechtsquellen des Verwaltungsrechts und Perspektiven der Formalisierung 313
11.3.1. Mehrschichtige Repräsentation 314
11.4. Formale Sprache, Modell und Inferenzmaschine 315
11.5. Zeitliche Aspekte von Rechtsnormen 321
11.5.1. Die primären zeitlichen Dimensionen 321
11.5.2. Analyse der vorhandenen Literatur 323
11.5.3. Interne und externe Zeit 330
11.5.4. Dauer von normativen Effekten 331
11.5.5. Gültigkeit von Normen 333
11.6. Änderungen der Rechtsordnung und von Normen 336
11.7. Die anfechtbare Schlussfolgerung 339
11.7.1. Kollisionen 341
11.7.2. Wiederinkraftsetzung 343
11.7.3. Die Überwindung der Anfechtbarkeit juristischer Schlussfolgerungen 343
11.7.4. Technische Perspektive 345
11.8. Modularität 347
11.9. Reihenfolge der Normanwendung 348
11.10. Hypothetische Schlussfolgerung 351

12. STATE-OF-THE-ART-TECHNOLOGIEN 353

12.1. JBoss Drools 353
12.1.1. Überblick 353
12.1.2. Die Verwendung von Drools im juristischen Bereich 354
12.1.3. Das temporale Modell 355
12.1.4. Die anfechtbare Schlussfolgerung 356
12.1.5. Constraint-Programmierung 356
12.1.6. Experimentelle Ansätze 358

12.1.7. Abschließende Bemerkungen ... 358
12.2. SPINDLE ... 360
12.2.1. Überblick ... 360
12.2.2. Grundlagen der anfechtbaren Logik ... 360
12.2.3. Das temporale Modell ... 361
12.2.4. Abschließende Bemerkungen ... 362
12.3. LEGALRULEML ... 363
12.3.1. Überblick ... 363
12.3.2. Features ... 364
12.3.3. Abschließende Bemerkungen ... 364

13. ROWLER – EINE HYBRIDE RULE ENGINE ZUR JURISTISCHEN ENTSCHEIDUNGSFINDUNG ... 365

13.1. EINLEITUNG ... 365
13.1.1. Motivation ... 365
13.1.2. Anforderungen ... 366
13.1.3. Designprinzipien ... 368
13.2. ARCHITEKTUR ... 369
13.2.1. Die Prozessschicht ... 369
13.2.2. Die Regelschicht ... 371
13.2.3. Die ontologische Schicht ... 372
13.2.4. Elektronische Dokumentensammlung ... 375
13.3. SCHLUSSFOLGERUNGSMODUL UND ALGORITHMUS ... 376
13.3.1. Die Ermittlung der anwendbaren Rechtslage ... 377
13.3.2. Die rechtliche Beurteilung ... 379
13.3.3. Abschließende Bemerkungen ... 381
13.4. DAS TEMPORALE MODELL ... 382
13.4.1. Überblick ... 382
13.4.2. Versionierung legislativer Ressourcen ... 382
13.4.3. Methoden der computergestützten Versionierung von Normen ... 383
13.4.4. Versionierungsmodell ... 385
13.4.5. Die Ermittlung der anwendbaren Rechtsnormen ... 398
13.4.6. Abschließende Bemerkungen ... 409
13.5. GRUNDPFEILER DES FORMALEN MODELLS ... 410
13.5.1. Isomorphismus ... 411
13.5.2. Temporale Dimensionen ... 412
13.5.3. Metadaten ... 413
13.6. DIE MODELLIERUNG VON NORMEN ... 414
13.6.1. Unterstützung verschiedener Normtypen ... 415
13.6.2. Normative Ausnahmen und Prioritäten ... 417

13.6.3. Unterstützung mehrerer Interpretationen ... *418*
13.6.4. Sanktion und Wiedergutmachung ... *419*
13.7. Verwandte Arbeiten ... 421
13.7.1. JBoss Drools ... *421*
13.7.2. SPINdle ... *422*
13.7.3. Carneades ... *423*
13.7.4. Temporal Defeasible Logic ... *424*
13.7.5. Norma-System ... *425*
13.7.6. Akoma Ntoso ... *426*
13.7.7. LegalRuleML ... *427*
13.8. Diskussion ... 427
14. KONKLUSION ... 431
14.1. Zusammenfassung ... 431
14.2. Wichtigste Ergebnisse ... 434
14.3. Zukünftige Forschungen ... 436
14.4. Abschließende Bemerkungen ... 439
LITERATURVERZEICHNIS ... 443
ABBILDUNGSVERZEICHNIS ... 477
ABSTRACT ... 479
ABSTRACT (ENGLISCH) ... 481
CURRICULUM VITAE ... 483
ANHANG ... I

1. Einleitung

1.1. Motivation

Bedingt durch die stetig steigende Anzahl an Gesetzen auf nationaler und europäischer Ebene, bei gleichzeitiger Forderung nach höherer Effizienz und Qualität von Entscheidungen der öffentlichen Verwaltung[1], besteht erhöhter Bedarf an einer entsprechenden Repräsentation juristischen Wissens und dessen Implementierung in wissensbasierten Systemen (WBS).

Die Forderung nach höherer Effizienz der Verwaltung ist nicht neu, wird aber in wirtschaftlich schwierigen Zeiten besonders laut. Eine solche Effizienzsteigerung versucht man oftmals durch organisatorische Änderungen zu erreichen, wofür die Reform der Verwaltungsgerichtsbarkeit in Österreich ein Beispiel ist.

Es ist jedoch zu erwarten, dass jede Steigerung der Effizienz der Verwaltung sofort durch eine steigende Nachfrage zunichtegemacht wird. Für diese Hypothese kann ein recht einfaches Argument vorgebracht werden: Die Beschreitung des Rechtswegs ist das letzte teure Mittel, um die normative Ordnung in der Gesellschaft zu bewahren. Daher führt jede Reduktion der Kosten unmittelbar zu einer erhöhten Inanspruchnahme der Gerichte und Verwaltungsbehörden durch die Bürger (Boer, 2009, S. 14).

Ein noch größerer Mangel an Rechtsexperten in naher Zukunft scheint unvermeidbar zu sein. Aus diesem Grund rückt die Entwicklung von *wissensbasierten Systemen* in den Mittelpunkt des Interesses der öffentlichen Verwaltung. Solche Systeme sollen den menschlichen Benutzer bei der Lösung wissensintensiver Aufgaben unterstützen und, wenn möglich, Entscheidungen treffen. Damit das System diese Aufgabe bewältigen kann, benötigt es eine adäquate interne Repräsentation des jeweiligen Ausschnitts der Wirklichkeit, damit es sich „intelligent“ verhalten kann.

Modelle spiegeln die Wirklichkeit nicht direkt wider, sondern repräsentieren das zur Bewältigung der zugewiesenen Aufgaben notwendige Wissen eines Bereichs. Es bedarf daher für die Entwicklung von wissensbasierten Systemen einer Übersetzung des Wissens eines Experten in das formale Modell, das vom Computer verwendet wird. Die Übersetzung

[1] Der Begriff „Verwaltung“ wird in dieser Dissertation in einem organisatorischen Sinn verstanden und umfasst vor allem die Organe des Bundes.

und Repräsentation von menschlichem Wissen ist eines der grundlegenden Probleme der künstlichen Intelligenz (Newell, 1982).

Wissensbasierte Systeme sind in der Lage einfache Aufgaben selbst zu erledigen oder zumindest weniger qualifiziertes Personal zu unterstützen, damit dieses im Stande ist Aufgaben zu lösen, für die sonst Expertenwissen notwendig ist. Dies führt zu einer Entlastung der Experten, die sich dann den schwierigen Fällen widmen können. Der Einsatz von wissensbasierten Systemen trägt so maßgeblich zur Effizienzsteigerung und Erhöhung der Qualität von Entscheidungen bei.

Hinsichtlich der Rolle von Experten im Entscheidungsprozess ist ein Wandel vom Entscheidungsträger zum Ersteller von Richtlinien für wissensbasierte Systeme zu beobachten, wobei zu erwarten ist, dass sich dieser Trend noch verstärkt. Vermehrt werden Experten in Zukunft Entscheidungen nicht mehr selbst fällen, sondern vielmehr ihr Wissen zur Verfügung stellen, das schließlich in einem wissensbasierten System implementiert wird und gegebenenfalls korrigierend in den Entscheidungsprozess eingreifen. Da nicht zu erwarten ist, dass bald signifikant mehr Juristen von der Verwaltung beschäftigt werden, lässt sich nur durch den Einsatz von wissensbasierten Systemen die stetig steigende Menge an Fällen, bei zukünftig noch höherer Regelungsdichte, bewältigen. Dabei darf ebenso nicht der Aspekt vergessen werden, dass das nationalstaatliche Recht immer stärker mit internationalen Bestimmungen, insbesondere dem Recht der Europäischen Union, verknüpft ist. Selbst für einen Juristen kann es damit außerordentlich schwierig sein, alle relevanten Normen im Einzelfall zu berücksichtigen. Meines Erachtens wird sich nur durch den verstärkten Einsatz von wissensbasierten Systemen die stetig steigende Menge an Fällen, bei in Zukunft noch höherer Regelungsdichte, bewältigen lassen.

Dieser Paradigmenwechsel ist sehr bedeutend, da sich mit ihm die Auffassung der Rolle juristischer Experten maßgeblich ändert. Damit in Zukunft Experten ihr Wissen in wissensbasierte Systeme einpflegen können, ist erforderlich, dass eine geeignete Sprache zur Modellierung juristischen Wissens, gegebenenfalls unterstützt durch spezielle Editoren, zur Verfügung steht.

Wissensbasierte Systeme operieren auf einer symbolischen Repräsentation des Expertenwissens, um den Benutzer bei seinen Aufgaben zu unterstützen und anzuleiten. Sie tragen so maßgeblich zur Vereinheitlichung der Entscheidungspraxis einer Behörde bei,

wobei gleichzeitig die Qualität und Nachvollziehbarkeit von Entscheidungen verbessert wird. Insgesamt führt der Einsatz von wissensbasierten Systemen zur Steigerung der Transparenz behördlicher Entscheidungen, einem zentralen Anliegen von E-Government.

In Anbetracht des technologischen Entwicklungsstandes und der Erkenntnisse, die auf dem Forschungsgebiet der Künstlichen Intelligenz und Recht gewonnen wurden, scheint es angebracht, die gängigen Ansätze zur Implementierung von juristischen Expertensystemen für die Verwaltung zu überdenken. Die Technologien des Semantic Web und insbesondere Ontologien, einer Art von computergestütztem Modell, erscheinen vielversprechend, den State of the Art der Verwaltungsautomatisierung zu verbessern.

Denn obwohl Softwaresysteme bereits seit den 1970er-Jahren in der Verwaltung zum Einsatz kommen[2], um die Durchführung von Verfahren zu unterstützen und damit eine nicht bestreitbare praktische Eignung besitzen, ist die Entwicklung von juristischen Expertensystemen noch immer teuer und fehleranfällig. Viele Softwareprojekte im E-Government, die nach einer Modernisierung streben, verzögern sich aus diesem Grund oder scheitern sogar gänzlich. Es besteht demnach durchaus Verbesserungsspotential, um in Zukunft die Entwicklung von Expertensystemen effizienter zu machen und das Scheitern solcher Projekte zu verhindern.

Es kann beobachtet werden, dass sich die Forschungen auf dem Gebiet Künstliche Intelligenz und Recht und die Praxis der Verwaltungsautomatisierung in Österreich über viele Jahrzehnte parallel entwickelt haben, ohne voneinander Notiz zu nehmen. Die Formalisierung von Normen erfolgt in der Praxis gewissermaßen ad hoc, ohne auf rechtstheoretische Überlegungen Bedacht zu nehmen.

Die wissenschaftliche Untersuchung und Analyse von Möglichkeiten zur Verbesserung des State of the Art der Verwaltungsautomatisierung, im Kontext neuer Technologien, ist der erste Schritt in Richtung einer soliden theoretischen Grundlage für die Entwicklung von juristischen Expertensystemen zur Unterstützung der Verwaltung[3]. Diese Dissertation versucht über den offenbar bestehenden Graben zwischen Theorie und Praxis eine Brücke zu

[2] Exemplarisch sei das sogenannte „Wiener System“ erwähnt (Svoboda, 1975).

[3] Mit „Expertensysteme zur Unterstützung der Verwaltung“ sind Systeme gemeint, die Verwaltungsorgane bei der Vollziehung von Gesetzen unterstützen.

schlagen, indem vorhandene Ansätze evaluiert und Verbesserungsmöglichkeiten durch den Einsatz neuer Technologien aufgezeigt werden. Damit soll auch ein Überblick über verfügbare theoretische Ansätze und experimentelle Technologien gegeben werden, die Forschungen auf dem Gebiet der Künstlichen Intelligenz und Recht hervorgebracht haben. Dieser Diskurs hilft den Blick auf potentiell nützliche Ansätze zu schärfen, um so deren praktische Verwendung zu steigern. Umgekehrt wird damit auch die praktische Eignung theoretischer Ansätze als Grundlage für juristische Expertensysteme aufgezeigt.

Auf lange Sicht wird ein flexibles Framework für die Entwicklung von juristischen Expertensystemen benötigt, das auf offenen Standards aufbaut und bewährte Ansätze implementiert, um die Wiederverwendbarkeit zu steigern. Weiters muss solch ein Framework von einer einheitlichen Methode zur Formalisierung von Rechtsnormen komplementiert werden.

Die wichtigsten Beiträge der Dissertation, um diese Ziele zu erreichen, sind die Evaluierung von Ontologien zur Unterstützung der automatisierten Entscheidungsfindung im Verwaltungsrecht und die Konzeption einer modularen Rule Engine, deren temporales Modell in der Lage ist, mit der Dynamik des Rechts auf effiziente Art und Weise umzugehen.

Es liegt auf der Hand, dass der Diskurs keinesfalls danach trachtet sämtliche Materien des Verwaltungsrechts zu behandeln, da dieses in unglaublicher Vielfalt in Erscheinung tritt (Raschauer, 2013, S. 7 ff.). Die Untersuchung beschränkt sich daher exemplarisch auf das Kriegsopferversorgungsgesetz BGBl. Nr. 152/1957 (KOVG), einer Sondermaterie, die dem sogenannten „Versorgungsrecht“ zuzuordnen ist und dem besonderen Verwaltungsrecht angehört. Das Versorgungsrecht umfasst neben dem KOVG noch weitere Gesetze, wie insbesondere das Heeresversorgungsgesetz BGBl. Nr. 27/1964 (HVG). Ein Großteil der Normen des Kriegsopferversorgungsgesetzes Gesetzes befasst sich mit der Berechnung von monetären Renten. Es wird daher der Anspruch gestellt, dass die gewonnenen Erkenntnisse auf ähnliche Gesetzesmaterien, wie das Pensionsrecht und das Steuerrecht, übertragen werden können.

Aufgrund des interdisziplinären Forschungsansatzes dieser Dissertation ist diese für Juristen, Softwareentwickler, Entscheidungsträger und allgemein für all jene von Interesse, die mit der Formalisierung und Automatisierung juristischer Entscheidungsfindung befasst sind.

1.2. Hypothesen und Forschungsfragen

Ausgehend vom dargestellten Stand der Verwaltungsautomatisierung und der neuen technologischen Möglichkeiten, werden nun Hypothesen und Forschungsfragen zu deren Überprüfung formuliert.

1.2.1. Hypothesen

Folgende Hypothesen sollen in dieser Dissertation untersucht werden:

1. Die in der Praxis gängigen Ansätze zur Formalisierung von Rechtsnormen weisen erhebliche Unzulänglichkeiten auf.

2. Die Berücksichtigung der auf dem Gebiet der Künstlichen Intelligenz und Recht entwickelten Theorien und Ansätze kann die formale Repräsentation von Rechtsnormen verbessern.

1.2.2. Forschungsfragen

Zur Überprüfung der aufgeworfenen Hypothesen und um den wissenschaftlichen Diskurs zu fokussieren, werden folgende Forschungsfragen formuliert:

1. Welchen Beitrag leisten formale Modelle zum E-Government?

Die Ziele von E-Government sind unter anderem Transparenz und Effizienz. Wie tragen formale Modelle zur Erreichung dieser Ziele bei? Welche Möglichkeiten eröffnet der Einsatz semantischer Technologien für E-Government-Anwendungen?

2. Welche Nachteile weisen bestehende juristische Expertensyteme und Ansätze der Formalisierung auf?

Schon seit Langem existieren in Österreich im öffentlichen Sektor Softwarelösungen zur Unterstützung von Verwaltungsverfahren. Auf welchen Technologien basieren diese und mit welchen Nachteilen sind diese behaftet?

3. Welche Hindernisse stehen der Formalisierung von Rechtsnormen und der Automatisierung der juristischen Entscheidungsfindung entgegen?

Schon früh gab es Versuche das Recht zu formalisieren. Welche Probleme stehen derartigen Unternehmungen entgegen und welche Hindernisse müssen überwunden werden?

4. Welchen Einfluss hat die Theorie auf die Entwicklung computergestützter Modelle des Rechts?

Theorien tragen maßgeblich zum fundierten Verständnis eines Bereiches bei – aber sind diese auch als Grundlage für formale Modelle geeignet? Hinter jeder formalen Sprache stehen tiefgehende theoretische Überlegungen. Durch die Wahl einer Sprache nimmt man auch die vorhandenen Einschränkungen in Kauf. Unterstützen die in der Theorie entwickelten Ansätze die Modellierung von Rechtsnormen?

5. Welche Ansätze zur juristischen Wissensrepräsentation existieren?

Seit mehreren Jahrzehnten wird auf dem Gebiet der Künstlichen Intelligenz und Recht intensiv nach geeigneten Sprachen und Modellen zur Repräsentation von Normen geforscht. Welche theoretischen Ansätze zur juristischen Wissensrepräsentation existieren und wozu eignen sie sich?

6. Welche Anforderungen werden an formale Sprachen zur Modellierung von Verwaltungsrecht gestellt?

Die Anforderungen an Modelle und Sprachen zur Formalisierung ergeben sich vor allem aus dem materiellen Verwaltungsrecht. Überdies können jedoch weitere Anforderungen aus dem formellen Recht und den Unzulänglichkeiten bestehender Ansätze abgeleitet werden. Welche Anforderungen müssen Sprachen zur Modellierung von Verwaltungsrecht erfüllen?

7. Eignen sich Ontologien für die Unterstützung der automatisierten Entscheidungsfindung?

Die einzelnen Sprachen unterscheiden sich durch Ausdrucksstärke voneinander, die für die Erstellung von Modellen genutzt werden kann. Inwieweit erfüllen die neuen Sprachen und vorhandenen Ontologien die Anforderungen? Die Erstellung von formalen Modellen ist

schwierig und sehr aufwändig, wohingegen deren Nutzen nicht immer sofort erkennbar ist. Führen die neuen Sprachen zu einer Verbesserung gegenüber den gängigen praktischen Ansätzen, insbesondere zu einer Steigerung der Effizienz?

8. Eignen sich State-of-the-Art-Technologien für die Automatisierung der juristischen Entscheidungsfindung?

Es existieren viele Technologien, von denen manche auch für den rechtlichen Bereich konzipiert wurden. Welche Technologien existieren und wie tragen diese zur Formalisierung von Normen und der Automatisierung der juristischen Entscheidungsfindung bei?

Der Gegenstand der Dissertation ist die Suche des Autors nach Antworten auf die aufgeworfenen Fragen. Die Fragen sollen jedoch nicht nacheinander beantworten werden, sondern sind vielmehr als Hintergrund zu verstehen, vor dem sich die folgenden Kapitel entfalten.

Die Kapitel 4, 5, 6 und 7 formieren die theoretische Grundlage für den Diskurs, die in Kapitel 11 noch weiter vertieft wird, um die Entwicklung eines eigenen Ansatzes zu ermöglichen. Die Kapitel 2 und 3 behandeln die Rolle semantischer Technologien im E-Government und beantworten Frage 1. Kapitel 3 fokussiert stark auf Frage 2 und zeigt die Schwächen gängiger Ansätze zur Formalisierung von Normen auf. In den Kapiteln 4 und 5 wird auf Frage 3 eingegangen und es werden die Hindernisse diskutiert, die der Formalisierung von Rechtsnormen entgegenstehen. Frage 4 ist eine der wichtigsten Fragen, der sich daher ein Großteil der Dissertation in den Kapiteln 5, 6, 7, 9, 10, 11 und 13 widmet. Kapitel 5 baut auf dem in Kapitel 4 erarbeiteten Grundverständnis der Formalisierung auf und befasst sich insbesondere mit Frage 5. Frage 6 wird im Rahmen der Abhandlung in Kapitel 7 erörtert. Kapitel 10 wendet die Theorie an um ein ontologisches Modell des Gesetzes zu kreieren und schafft damit die Grundlage für die Beantwortung von Frage 7, auf die zum Teil ebenfalls Kapitel 11 eingeht. Kapitel 12 präsentiert State-of-the-Art-Technologien und verschreibt sich Frage 8. In Kapitel 13 wird ein neuartiger Ansatz zur automatisierten juristischen Entscheidungsfindung entwickelt und dient damit insbesondere der Untersuchung von Hypothese 2.

1.3. Ontologien

Die oben formulierten Fragen sollen im Kontext eines speziellen computergestützten Modells – der Ontologie – beantwortet werden. Dieses Unterkapitel soll zeigen, warum Ontologien als Grundlage für diese Untersuchung besonders geeignet erscheinen.

All jene, die bereits mit dem „Semantic Web“ in Berührung gekommen sind, dürften mit dem Begriff „Ontologie“ bereits einigermaßen vertraut sein. Die Einsatzgebiete für Ontologien sind vielfältig, ihre Nützlichkeit ist weitgehend anerkannt und sie sind zentrales Thema vieler wissenschaftlicher Konferenzen.

Die grundlegende Frage, was eine Ontologie ist, ist jedoch relativ schwierig zu beantworten. Bei der Beantwortung tendieren Experten unbewusst dazu, diese nicht wertfrei, sondern beeinflusst von ihrem eigenen Hintergrund zu beantworten. Wie gezeigt wird, sind durch die Pluralität der Einflüsse mehrere Antworten zulässig. Die Bedeutung des Begriffs „Ontologie“ ist aus historischen Gründen überladen, da in ihm Erkenntnisse aus der Philosophie, der Künstlichen Intelligenz, dem Systems Engineering, dem Informationsmanagement, der Computerlinguistik und der Kognitionspsychologie zusammentreffen.

Diese interdisziplinäre Grundlage und die daraus resultierende Vielzahl an Einflüssen haben zu ertragreichen Diskussionen beigetragen und die Entwicklung von Ontologien positiv beeinflusst. Andererseits lassen sich dadurch die verfügbaren Methoden und Theorien für die Erstellung von Ontologien sowie die bereits vorhandenen Ontologien kaum mehr überblicken (Casellas, 2011; Delgado, Gallego, Llorente, & García, 2003; R. García & Delgado, 2005; Hoekstra, Breuker, Di Bello, & Boer, 2007; Liebwald, 2009; Rodríguez-Doncel, Santos, & Casanovas, 2014; Rubino, Rotolo, & Sartor, 2006; Wyner, 2008). Die Verwendung von Ontologien zur Modellierung von Normen wird dadurch ungemein erschwert.

In dieser Dissertation wird jedenfalls die Interpretation des Begriffs „Ontologie“ aus dem Bereich der Wissensrepräsentation[4] übernommen. Die Wissensrepräsentation ist ein Teilgebiet der Künstlichen Intelligenz, welches die Probleme zu lösen versucht, die bei der Übersetzung von Wissen in computergestützte Modelle auftreten. Die formale Repräsentation von Wissen erlaubt es Computersystemen – den sogenannten

[4] Engl. „Knowledge representation“ (KR).

wissensbasierten Systemen[5] – „intelligente“ Entscheidungen zu treffen. Wissensbasierte Systeme erledigen Aufgaben, die sonst von Experten gelöst werden. Oftmals ist mit wissensintensiven Aufgaben die Verarbeitung großer Datenmengen verbunden, deren Verarbeitung durch den Menschen aufwändig und fehleranfällig ist. Wissensbasierte Systeme können den Experten beim Umgang mit großen Datenmengen erheblich unterstützen. Das Knowledge-Engineering befasst sich, analog zum Software-Engineering, mit dem Design und der Entwicklung von wissensbasierten Systemen (WBS). Ein wichtiger Teilbereich des Knowledge-Engineerings ist die Gewinnung von Expertenwissen und dessen Organisation in wiederverwendbarer Form.

Im Kontext der Wissensrepräsentation können Ontologien im Wesentlichen als standardisierte Vokabulare verstanden werden, mit deren Hilfe das in einem Bereich vorhandene Wissen, in Form von Begriffen und deren Beziehungen, repräsentiert wird. Es handelt sich demnach im Grunde um eine stark strukturierte Sammlung von Begriffen, ähnlich einem Thesaurus, jedoch mit erweiterter Funktionalität und formaler Semantik.

Der Begriff „Ontologie“ (oder „Ontologia“) stammt ursprünglich aus der Philosophie, wo er offenbar das erste Mal im Jahr 1606 im Buch „Ogdoas Scholastica“ des deutschen Scholastikers und Protestanten Jacob Lorhard verwendet wurde. Für Lorhard und viele ihm nachfolgende Philosophen war der Begriff „Ontologie“ ein Synonym für die „Metaphysik“. Der Begriff „Metaphysik“ bedeutet wörtlich übersetzt so viel wie „das was nach der Physik kommt“ und wurde als so etwas wie die „Wissenschaft des Seins“ begriffen (Smith, 2011).

Betrachtet man den Reifegrad des Ontology-Engineerings kann heute gesagt werden, dass Ontologien „erwachsen geworden sind“ (McGuinness, 2003). Eine beachtliche Anzahl an Forschungsprojekten hatte die Anwendung von Technologien des Semantic Web auf das Recht zum Gegenstand. Dies hat zu einem erheblichen Anstieg an Wissen im Bereich des Ontology-Engineerings geführt. Juristische Ontologien wurden als Modell für verschiedenste Anwendungszwecke in Betracht gezogen, wie unter anderem für die Informationsgewinnung, Interoperabilitäts-Frameworks und die Entscheidungsfindung. Dies erklärt auch die vielen Arten von Ontologien, die nach ihrer Granularität (domain-specific vs. core), dem Grad der Formalität (hoch axiomatisch vs. lexikalisch oder sprachorientiert) und

[5] Engl. „Knowledge-based systems“ (KBS).

nach der Quelle für die Extraktion von Begriffen (offizielle Rechtsquellen, Expertenwissen und ethnografische Studien) unterschieden werden können. Die Diversität der methodologischen Ansätze und der verfügbaren Theorien zeigen, dass es sich um ein florierendes Forschungsgebiet handelt, dies erschwert jedoch zugleich die Verwendung von Ontologien in der Praxis, wie weiter oben bereits erwähnt wurde.

Die Entwicklung des juristischen Ontology-Engineerings wird nicht bloß von der Neugierde der Wissenschaftler vorangetrieben, neue Technologien und Werkzeuge einzusetzen, sondern vor allem von der Notwendigkeit Anwendungen zu entwickeln, die – vor dem Hintergrund des rasanten technologischen Fortschritts – besser die Anforderungen der Praxis und Bürger erfüllen.

Derzeit existieren mehr als sechzig Ontologien mit juristischem Bezug (Casanovas, Sartor, Biasiotti, & Fernández-Barrera, 2011, S. 5 ff.). Darüber hinaus beschäftigten sich mehrere Dissertationen mit Wissensrepräsentation und der Entwicklung von Ontologien (Casellas, 2011; Hoekstra, 2009).

Die große Anzahl an Ontologien zeigt jedenfalls eindrucksvoll die Stärke und Dynamik dieses Bereichs. Nur wenige Ontologien sind jedoch über das Stadium eines erweiterten und verbesserten Prototyps hinausgewachsen. Dieser Umstand deckt sich auch mit Umfragen, die zeigen, dass juristische Expertensysteme kaum verbreitet sind (Cardoso, 2007). Auch in der Praxis der Verwaltungsautomatisierung in Österreich existieren nur vereinzelt Softwaresysteme zur Unterstützung der Verwaltung und semantischen Technologien wird kaum Beachtung geschenkt.

Es scheint sehr wahrscheinlich, dass sich dies in naher Zukunft ändern wird. Ähnlich wie das Web 2.0 treibende Kraft für das E-Government 2.0 ist[6], bereitet das Web 3.0 durch den Einsatz von semantischen Technologien den Weg zur Erreichung des E-Government 3.0[7]. Unter E-Government 2.0 versteht man die Verwendung von „Social Media“ – Blogs, Wikis, Soziale Netzwerke etc. – durch Behörden, um insbesondere demokratische Prozesse zu unterstützen (Chun, Shulman, Sandoval, & Hovy, 2010). Das Web 3.0 wird als eine

[6] In der Tat existieren im E-Government-Umfeld mehrere Anwendungen, die auf dem Web 2.0 aufbauen (Büschenfeldt & Scholl, 2014; Röchert-Voigt & Gronau, 2010; Ventzke, Weiß, Wimmer, & Makolm, 2010).

[7] Der Begriff „E-Government 3.0“ ist nicht gebräuchlich und wird hier gewissermaßen als Synthese aus Web 3.0 und E-Government 2.0 verstanden.

Kombination der semantischen und sozialen Dimension (Web 2.0) gesehen (Davies et al., 2009)[8].

Der Einsatz von hoch axiomatisierten Ontologien und zentralisierten Datensammlungen sollte in Anbetracht der neuesten Entwicklungen hinterfragt werden. Es scheint zielführender die Intelligenz von Anwendungen nach der Fähigkeit zu beurteilen, mit heterogenen und verteilten Daten umgehen zu können (Casanovas et al., 2011, S. 4).

Das Design von Ontologien spielt jedenfalls eine wichtige Rolle für die Wissensrepräsentation und das Knowledge-Engineering. Die Forschung hat zahlreiche Methoden und Designprinzipien für die Erstellung von Ontologien hervorgebracht. Beispielsweise sei hier die „Web Ontology Language" (OWL) erwähnt, die speziell für die Modellierung von Ontologien entwickelt wurde. OWL basiert, wie jede formale Sprache zur Wissensrepräsentation, auf einem Kompromiss zwischen Ausdrucksstärke und Entscheidbarkeit (Levesque & Brachman, 1987). Ohne eine Einschränkung der Ausdrucksstärke könnten bestimmte Eigenschaften der Sprache im Hinblick auf deren formale Semantik nicht gewährleistet werden.

Die Erstellung von Ontologien in OWL ist eine herausfordernde Aufgabe. Dies hat zur Entwicklung diverser Tools zur Unterstützung und zur Spezifikation von Entwurfsmustern für das Design von Ontologien geführt. Überdies wurde eine beachtliche Zahl von Ontologien mit dem Ziel entwickelt, ein generisches und einheitliches Framework für die Entwicklung von Ontologien bereitzustellen. Es wird erwartet, dass solche Core-Ontologien die Entwicklung von speziellen Domain-Ontologien anleiten und fördern können.

Zusammengefasst ist die Entwicklung von Ontologien ein etabliertes Forschungsthema, das reichlich Inspiration für die Untersuchung der formulierten Forschungsfragen bietet:

- Ontologien können als strukturierte Sammlung von Begriffen und deren Beziehungen verstanden werden. Das ontologische Modell repräsentiert die Normen des Gesetzes und zeigt, ob diese Darstellung dessen Verständlichkeit erhöht (Frage eins).

[8] Bereits heute leistet das „Web" und insbesondere das „Web 3.0" wertvolle Hilfe bei der Unterstützung von Entscheidungsprozessen in Communitys (C. Miller & Smith, 2011). Dieser Trend wird sich meines Erachtens maßgeblich durch den Einsatz von semantischen Technologien in Zukunft verstärken.

- Mehrere (juristische) Ontologien existieren bereits, die als Grundlage für die Entwicklung weiterer Ontologien dienen sollen. Diese Forschungen offenbaren nicht nur Perspektiven für das Design von Ontologien, sondern liefern überdies Erkenntnisse für die Spezifikation von Ontologien in einer speziellen Sprache und zeigen, welche theoretischen Überlegungen praktisch umsetzbar sind und tragen so zur Beantwortung der Fragen vier und fünf bei.
- Ontologien können mithilfe speziell dafür entwickelter Sprachen zur Wissensrepräsentation modelliert werden, die stets bedeutsamen Einschränkungen unterliegen. Der Versuch der Repräsentation von Normen in einer Ontologie zeigt, welche Anforderungen an Sprachen zur Modellierung gestellt werden und insbesondere, ob diese zur Formalisierung von Verwaltungsrecht geeignet sind. Der Diskurs dient insbesondere der Untersuchung der Fragen sechs, sieben und acht.
- Der Begriff „Ontologie“ stammt ursprünglich aus der philosophischen Theorie und wurde in den mehr anwendungsorientierten Bereich der Künstlichen Intelligenz und des Semantic Web übernommen. Die bereits auf dem Gebiet der Philosophie und Rechtstheorie entwickelten Einsichten werden für die Modellierung von Normen, wenn zweckdienlich, herangezogen. Das Zusammenspiel zwischen diesen beiden Bereichen bietet eine spannende Grundlage für die Untersuchung der Fragen vier und sieben.
- In der Rechtstheorie existieren bereits Lösungsansätze für Probleme der juristischen Entscheidungsfindung. Die Forschungen haben sich unter anderem mit der Strukturierung von juristischem Wissen befasst. Die Verwendung rechtstheoretischer Überlegungen als Inspiration und Wegweiser für das Design von Ontologien soll zeigen, wie die Theorie zur Entwicklung computergestützter Modelle des Rechts beitragen kann und dient so der Beantwortung von Frage vier.
- Ontologien spielen eine bedeutende Rolle im Semantic Web und werden für äußerst unterschiedliche Zwecke verwendet. Sie sind damit nicht bloß abstrakte, theoretische Gebilde, sondern finden auch praktische Verwendung. Ein eingehender wissenschaftlicher Diskurs zur Verwendung von Ontologien zur Formalisierung von Normen und Unterstützung der automatisierten Entscheidungsfindung fehlt jedoch im Kontext der Verwaltung. Aus diesem Grund soll die Untersuchung aller Fragen vor dem Hintergrund des Semantic Web und Ontologien stattfinden und so zu einem

besseren Verständnis der Rolle von Ontologien bei der Formalisierung von Normen beitragen.

1.4. Forschungsansatz

Für die Erforschung und Implementierung juristischer Expertensysteme existiert keine anerkannte Forschungsmethode. Aus diesem Grund und zum besseren Verständnis der Dissertation, soll an dieser Stelle der gewählte Forschungsansatz umrissen werden.

Der Einsatz von Technologien des Semantic Web verlangt eine hohe Struktur der Wissensbasis und umfangreiches Wissen über die Modellierung. Für die adäquate Repräsentation von Rechtsnormen ist überdies Kenntnis der Rechtstheorie erforderlich.

Um ein solides Fundament für die Untersuchung von Ontologien zur Modellierung von Normen zu schaffen, wird mit einer Darstellung der Formalisierung und Wissensrepräsentation begonnen. Auf dieser Grundlage erfolgt sodann die Evaluierung von formalen Modellen des Kriegsopferversorgungsgestzes[9], geschrieben in Java und OWL 2. Aus den Modellen werden neue Erkenntnisse gewissermaßen induktiv abgeleitet, mit theoretischen Ausführungen vertieft und diskutiert.

Die Kombination vorhandener Theorien und der Erkenntnisse der praktischen Modellierungsversuche schafft eine ausgezeichnete Grundlage für die Entwicklung eines neuartigen Ansatzes zur Repräsentation von Normen.

Da die Theorie deduktiv auf die Modellierung von Rechtsnormen angewendet wird und neue Erkenntnisse aus den entwickelten Modellen induktiv abgeleitet werden, handelt es sich in einer Gesamtbetrachtung um einen hybriden Forschungsansatz[10]. Der wissenschaftliche Diskurs wird durch die formulierten Hypothesen und Forschungsfragen zu deren Überprüfung angeleitet.

---

[9] Das Kriegsopferversorgungsgesetz wird dabei stellvertretend für Gesetzesmaterien untersucht, die ebenso die Berechnung von monetären Beträgen vorsehen, wie das Pensionsrecht und das Steuerrecht.

[10] Da das Wissen sozusagen top-down aus generellen Theorien abgeleitet, neue Erkenntnisse aber botton-up aus praktischen Erfahrungen gewonnen werden, könnte auch von einem middle-out Ansatz gesprochen werden. Der Begriff „middle-out“ wird jedoch bereits im Bereich des Ontology Engineerings verwendet und soll nicht mit zusätzlicher, möglicherweise irreführender, Bedeutung überladen werden.

1.5. Beitrag und Überblick

Die Kapitel der Dissertation untersuchen die Themen, die in diesem Kapitel vorgestellt wurden, wie folgt:

Kapitel 2 - E-Government gibt eine Einführung in das Thema E-Government und legt damit den Kontext fest, in den sich Expertensysteme im öffentlichen Bereich einfügen müssen. Zu Beginn wird der aktuelle Stand des E-Governments in Österreich und dessen Rechtsgrundlagen dargestellt. Die folgenden Abschnitte widmen sich den Adressaten des E-Governments und dessen Zielen, die – zumindest indirekt – auch die Entwicklung von juristischen Expertensystemen beeinflussen. Das Konzept des bereichsspezifischen Personenkennzeichens wird einer kritischen Betrachtung unterzogen. Die Diskussion der zukünftigen Entwicklung des E-Governments im Lichte neuer Technologien des Semantic Web schließt das Kapitel ab.

Kapitel 3 - Status quo der Verwaltungsautomatisierung thematisiert den aktuellen Stand der Verwaltungsautomatisierung und stellt die Nachteile der in der Praxis gängigen Ansätze zur Formalisierung von Normen dar. Es wird verdeutlicht, warum in diesem Bereich Verbesserungspotential vorhanden ist. Die Diskussion schließt mit einem Ausblick auf die Untersuchung der Verwendung von Ontologien zur Repräsentation von Normen und auf den erhofften Nutzen dieses neuartigen Ansatzes.

Kapitel 4 - Die Formalisierung des Rechts dient der Begriffsbestimmung und Präsentation der theoretischen Grundlage der Formalisierung von Rechtsnormen. Dabei wird erörtert, welches Verständnis der Formalisierung dieser Dissertation zugrunde liegt und gezeigt, dass sich dieses von der ursprünglich stark logikorientierten Konnotation unterscheidet. Im Anschluss an diese Ausführungen wird auf die Grundlagen der Rechtsanwendung eingegangen, um ein Verständnis für die Problematik der Automatisierung der juristischen Entscheidungsfindung zu schaffen. Ein wesentlicher Teil des Kapitels befasst sich mit der Diskussion der Grenzen der Formalisierung und Automatisierung und zeigt mögliche Lösungsansätze auf.

Kapitel 5 - Wissensrepräsentation befasst sich mit den Themen Künstliche Intelligenz und Wissensrepräsentation. Die Wissensrepräsentation ist eines der Kernthemen der Künstlichen Intelligenz, da die Strukturierung und Aufbereitung von Wissen notwendige Vorausetzungen für die Implementierung von „intelligenten“ Systemen sind. Jedes wissensbasierte System benötigt eine geeignete Repräsentation des Wissens zur Problemlösung, das in einem Modell implementiert wird. Das Kapitel stellt die Funktionen von Modellen dar und gibt einen Überblick über die verschiedenen Wissensarten. Danach werden die Ansätze zur juristischen Wissensrepräsentation präsentiert und kritisch gewürdigt. Diese Abhandlung schafft die Grundlage für die Diskussion der Probleme, die es bei der juristischen Wissensrepräsentation zu lösen gilt. Es wird gezeigt, dass nicht für alle Herausforderungen bislang eine brauchbare Lösung gefunden werden konnte.

Kapitel 6 - Semantic Web stellt die dem Semantic Web zugrunde liegenden Konzepte, insbesondere Ontologien und die Web Ontology Language (und deren Nachfolgerin OWL 2), vor. Die Technologien des Semantic Web sind in mehreren aufeinander aufbauenden Schichten, im sogenannten Semantic Web Stack, angeordnet. Der Semantic Web Stack wurde für den juristischen Bereich adaptiert und zum Legal Semantic Web Stack erweitert. Ein wesentlicher Teil dieser Dissertation ist die Untersuchung von Ontologien zur Modellierung von Normen. Dementsprechend befasst sich das Kapitel mit einer Beschreibung von Ontologien als computergestützte Modelle. Weiters wird der erhoffte Nutzen des Einsatzes von Ontologien zur Formalisierung von Normen im Kontext der automatisierten Entscheidungsfindung thematisiert. Abschließend wird auf weitere Einsatzmöglichkeiten von Ontologien im E-Government-Umfeld und deren Potential zur Verbesserung von Expertensystemen eingegangen.

Kapitel 7 - Die Anforderungen an Sprachen zur Formalisierung von Verwaltungsrecht beschäftigt sich mit der Erhebung der Anforderungen an Sprachen zur Formalisierung von Verwaltungsrecht. Diese Darstellung dient als Grundlage für die Modellierung von Normen und entwickelt Kriterien, an denen sich vorhandene Ansätze und Sprachen messen lassen müssen. Dabei werden zunächst die Besonderheiten des Rechts dargestellt und das in Kapitel 5 gewonnene Grundverständnis der juristischen Wissensrepräsentation vertieft. Der

Dissertation liegt die Annahme zugrunde, dass Normen durch formale Regeln repräsentiert werden können. Dementsprechend widmet sich ein Abschnitt der Repräsentation von Normen durch formale Regeln, bestehend aus Antezedens und Konsequenz[11]. Dabei werden die Charakteristika von Rechtsnormen diskutiert, die es bei der formalen Repräsentation durch Regeln zu berücksichtigen gilt. Erkenntnisse aus der Implementierung von Verwaltungsrecht in Expertensystemen fließen in den Diskurs mit ein, um eine möglichst vollständige Abhandlung zu gewährleisten. Die erhobenen Anforderungen werden schließlich im gegenständlichen Kontext gewichtet.

Kapitel 8 - Systemdesign und Architektur widmet sich der Darstellung der Funktionsweise des Expertensystems aus fachlicher Sicht. Die Funktionen des Systems werden dabei anhand eines fiktiven Verfahrensablaufs skizziert. Anschließend wird das Potential semi-automatischer Expertensysteme diskutiert und aufgezeigt, dass es sich um einen praktisch höchst relevanten Ansatz handelt.

Kapitel 9 - Die Formalisierung des Kriegsopferversorgungsgesetzes in Java wendet die in den vorhergehenden Kapiteln dargestellten theoretischen Kenntnisse an und beschreibt die Formalisierung des Kriegsopferversorgungsgesetzes in Java. Das entwickelte Modell kann als Referenz verstanden werden, um den in Kapitel 10 entwickelten ontologischen Ansatz zu bewerten. Dabei wird besonderes Augenmerk darauf gelegt, die Struktur des Gesetzes und der Normen im Sinne des Isomorphismus möglichst exakt im Programmcode zu repräsentieren. Ein weiterer Schwerpunkt befasst sich mit den Möglichkeiten, die im Recht immanenten Ausnahmebestimmungen in Java abzubilden und mit Konflikten zwischen Normen umzugehen. Das Kapitel schließt mit einer eingehenden Schlussbetrachtung und Diskussion der Erkenntnisse.

Kapitel 10 - Die Ontologie des Kriegsopferversorgungsgesetzes stellt den experimentellen Versuch dar, das Kriegsopferversorgungsgesetz in einer Ontologie mit OWL 2 zu modellieren.

[11] Auch genannt „Sukzedens“ oder „Konklusion“.

Für die Repräsentation der Normen kommt ausschließlich Beschreibungslogik zum Einsatz. Das ontologische Modell besteht aus mehreren Modulen zur Formalisierung der Begrifflichkeit des Gesetzes, der Rechtsnormen und der Fakten des Einzelfalls. Das Kapitel behandelt zwei Ontologien: Eine simple Ontologie und eine erweiterte, die auf der LKIF-Core Ontologie aufbaut. Die Leistungsfähigkeit beider Ansätze wird sodann diskutiert und auf Schwächen hingewiesen. Das entwickelte ontologische Modell wird mit verwandten Arbeiten, insbesondere der Formalisierung von Normen in Java, verglichen. Das Ergebnis der Untersuchung wird am Ende des Kapitels dargestellt und ausführlich diskutiert sowie Empfehlungen für die Verwendung von Ontologien abgegeben.

Kapitel 11 - Erkenntnisse der Formalisierung stellt die wichtigsten Erkenntnisse der Formalisierung von Verwaltungsrecht in Java und OWL 2 dar. Die Darstellung beschränkt sich jedoch nicht nur auf die Zusammenfassung der Ergebnisse, sondern vertieft die wichtigsten Aspekte der automatisierten Rechtsanwendung. Zu diesen zählen insbesondere die mehrschichtige Repräsentation des Rechts, die Formalisierung von zeitlichen Aspekten von Rechtsnormen und die anfechtbare Schlussfolgerung. Diese ausführliche theoretische Abhandlung bietet eine vorzügliche Grundlage für die Entwicklung eines neuartigen Ansatzes zur Formalisierung von Normen, der in Kapitel 13 präsentiert wird.

Kapitel 12 - State-of-the-Art-Technologien bewertet verschiedenste Technologien im Sinne der im vorhergehenden Kapitel erhobenen Aspekte. Dadurch wird die Eignung dieser Technologien für die Formalisierung von Normen und die automatisierte Entscheidungsfindung ermittelt. Gemeinsam mit Kapitel 11 schafft dieses Kapitel einen Startpunkt für einen neuen Wettstreit der „AI and Law Community“, hin zur automatisierten juristischen Entscheidungsfindung durch den Einsatz von Informationstechnologie.

Kapitel 13 - rOWLer – Eine hybride Rule Engine zur juristischen Entscheidungsfindung präsentiert den wichtigsten wissenschaftlichen Beitrag dieser Dissertation. Es wird das Konzept einer Rule Engine dargestellt, die auf einer hybriden Wissensbasis, bestehend aus Regeln und einer Ontologie, operiert. Das formale Modell kombiniert die Ausdrucksstärke

von Regeln und OWL 2, um eine adäquate Repräsentation der Normen zu gewährleisten und die juristische Entscheidungsfindung durchzuführen. Die Engine ist konzipiert für den Einsatz in der Verwaltung (Steuerrecht, Sozialrecht etc.) und stellt eine flexible modulare Architektur zur Verfügung, die eine Anpassung an unterschiedliche Anforderungen erlaubt. Das Design der Engine baut damit auf den Erkenntnissen auf, die in vorherigen Kapiteln gewonnen wurden. Das System beruht auf einem ausgefeilten temporalen Modell, das mit den zeitlichen Aspekten von Rechtsnormen umgehen kann und basiert auf einem nicht-monotonen Algorithmus zur Schlussfolgerung.

2. E-Government

2.1. Was ist E-Government?

„E-Government“ (auch eGovernment, electronic government) wird von der Europäischen Kommission definiert als *„Einsatz der Informations- und Kommunikationstechnologien (IKT) in öffentlichen Verwaltungen in Verbindung mit organisatorischen Änderungen und neuen Fähigkeiten, um öffentliche Dienste und demokratische Prozesse zu verbessern und die Gestaltung und Durchführung staatlicher Politik zu erleichtern.“*[12]

Für die Bürger und Unternehmen soll eine einzige virtuelle Anlaufstelle („one stop shop“) geschaffen werden. Über ein solches Portal wird der Zugriff auf diverse E-Government Anwendungen an einer Stelle gebündelt. Die Möglichkeit die Dienste einer Behörde über das Internet in Anspruch zu nehmen, kann unter dem Slogan „Die Akten sollen laufen, nicht die Bürger“ verstanden werden.

Die elektronische Verfahrensabwicklung erfordert jedoch nicht nur die Schaffung von neuen Anwendungen für das Web, vielmehr müssen auch interne Behördenabläufe restrukturiert und verbessert werden, damit neue Anwendungen, die diese internen Abläufe unterstützen, die bestmögliche Effektivität und Effizienz erreichen können.

Erhöhte Transparenz soll zu gesteigertem Vertrauen in die Verwaltung führen, die mit einem relativen Machtübergewicht den Bürgern gegenübersteht. Die gesamte Verwaltung soll mehr auf den Bürger bezogen werden, der nicht mehr als bloßer Bittsteller angesehen wird. Mit dem E-Government erfolgt daher eine Änderung des Selbstbildes der Verwaltung.

Durch die elektronische Zustellung mit Hilfe so genannter Zustelldienste[13] können die Kosten für die Zustellung von Erledigungen, wie beispielsweise Bescheide in Papierform, drastisch reduziert werden. Bisher findet die elektronische Zustellung jedoch bei Privatpersonen wenig Akzeptanz. Das Zustellgesetz sieht auch die Möglichkeit der nachweislichen Zustellung im Auftrag von Privaten vor[14].

---

[12] KOM (2003) 567 endg.

[13] §§ 28 ff. ZustellG BGBl. Nr. 200/1982 i.d.F. BGBl. I Nr. 33/2013.

[14] § 29 Abs. 3 ZustellG BGBl. Nr. 200/1982 i.d.F. BGBl. I Nr. 33/2013.

Obwohl E-Government in Österreich schon eine relativ lange Tradition hat und es im europäischen Vergleich zu den führenden Nationen gehört, sind noch bei Weitem nicht alle Dienste online zugänglich. Der Großteil der Verwaltungsverfahren wird nach wie vor auf traditionelle Art in Form von Papierakten geführt.

Für die Nutzung vieler E-Government-Dienste ist Voraussetzung, dass die Benutzer – Bürger aber auch Unternehmen – über eine sogenannte „Bürgerkarte" verfügen. Das E-Government-Gesetz (§ 2 Zf. 10 E-GovG BGBl. I Nr. 10/2004 i.d.F. BGBl. I Nr. 7/2008) definiert diese als eine *„logische Einheit, die unabhängig von ihrer technischen Umsetzung eine qualifizierte elektronische Signatur mit einer Personenbindung und den zugehörigen Sicherheitsdaten und –funktionen sowie allenfalls mit Vollmachtsdaten verbindet".*

Es wurde bewusst eine technologieneutrale Formulierung gewählt, um bereits im Vorhinein für künftige technische Entwicklungen eine gesetzliche Grundlage zu schaffen.

Ein wichtiges Ziel des E-Government-Gesetzes ist, dass Dienste barrierefrei zur Verfügung gestellt werden und auch von Menschen mit besonderen Bedürfnissen genutzt werden können, ohne dass diese der Hilfe bei der Bedienung durch eine andere Person bedürfen.

Die Einführung und Forcierung von E-Government-Diensten darf unter keinen Umständen dazu führen, dass Menschen mit besonderen Bedürfnissen oder Bürger, die keinen Zugang zu Breitbanddiensten haben, von der Verwaltung und damit von der Ausübung ihrer Rechte „abgeschnitten" werden. Die zuletzt genannte Problematik wird unter dem Begriff „Digital Divide" verstanden und näher im Kapitel 2.5.2. thematisiert.

2.2. Aktueller Stand

Im Rahmen der Initiative „Digital Agenda for Europe" wird von der Europäischen Kommission regelmäßig der „eGovernment Benchmark" beauftragt, der die Benutzerzentriertheit, die Transparenz, die grenzüberschreitende Mobilität und die Grundvoraussetzungen für E-Government bewertet. Der „EU eGovernment Report 2014" (European Commission, 2014) ist der erste Benchmark, der auf dem neuen „eGovernment Benchmark Framework 2012-2015" basiert. Die folgenden Ausführungen beziehen sich auf den eben genannten Benchmark.

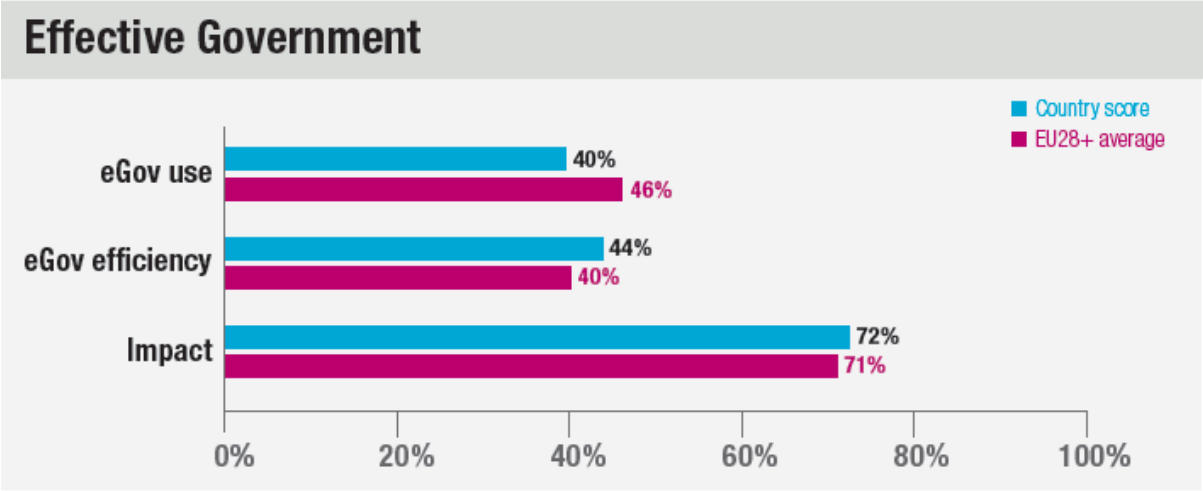

Abbildung 1: Benchmark Effective Government[15]

Abbildung 1 zeigt einen Ausschnitt aus dem Datenblatt für Österreich des E-Government-Benchmarks aus 2014. Wie ersichtlich, liegt Österreich hinsichtlich der Benutzung von E-Government knapp hinter dem EU-Durchschnitt zurück. Das zeigt, dass in dieser Hinsicht durchaus noch Verbesserungsbedarf besteht, sodass die Bürger E-Government-Services vermehrt nutzen.

Die ermittelten Werte decken sich im Wesentlichen mit innerstaatlichen Messungen, die von der Statistik Austria durchgeführt wurden. Nach dieser Statistik aus dem Jahr 2011 nutzt ein Großteil der Unternehmen in Österreich E-Government-Dienste. Bei privaten Haushalten ergab sich 2012 insofern ein anderes Bild, als dass 53 % der befragten Personen angaben, E-Government-Angebote in den letzten zwölf Monaten vor der Befragung genutzt zu haben[16].

Die Werte des europäischen und österreichischen Benchmarks können nicht direkt miteinander verglichen werden, da unterschiedliche Messkriterien und Befragungsmethoden herangezogen wurden. Dennoch zeigt sich der Trend, dass unter Privatpersonen die Akzeptanz von E-Government nach wie vor gesteigert werden muss.

Ein möglicher Grund dafür ist wohl, dass die Bürger im Durchschnitt wenig Kontakt mit Behörden haben und die traditionelle „Bürgerkarte“ in Form der Chipkarte einen relativ

---

[15] *Europäische Kommission*, EU eGovernment Report 2014 - Country Factsheets E-Government. http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=5543 aufgerufen: 18.11.2014 (2014).

[16] *Digitales Österreich*, IKT-Einsatz in Haushalten und Unternehmen 2012. http://www.digitales.oesterreich.gv.at/site/cob__40198/5428/default.aspx aufgerufen: 16.11.2014 (2012).

hohen administrativen Aufwand für die Aktivierung mit sich bringt. Für Privatpersonen verursacht die notwendige Registrierung damit einen verhältnismäßig hohen Aufwand im Vergleich zum erzielten Nutzen.

Das Problem wurde erkannt und durch die Einführung der Handysignatur adressiert, die seit Ende des Jahres 2009 verfügbar ist und eine einfachere Aktivierung der „Bürgerkarte" ermöglichen soll. Betrachtet man die aktuellen Werte, so scheint diese bisher nicht den großen Durchbruch hinsichtlich der Nutzung von E-Government-Diensten bewirkt zu haben[17].

Im Internet findet sich auf der offiziellen Webseite der „Bürgerkarte"[18] eine recht beachtliche Anzahl von E-Government-Anwendungen. Das stetig größer werdende Angebot an E-Government-Diensten könnte die Verbreitung der „Bürgerkarte" unter Privatpersonen weiter steigern. Vor allem, wenn der Bürger durch die Verwendung von E-Government-Diensten Kosten und Zeit spart oder längere Fristen für die Einbringung von Anträgen und die Ergreifung von Rechtsmitteln im Verfahren hat, wird er den Nutzen solcher Dienste erkennen.

2.3. Rechtsgrundlagen

Die Behörden und deren interne Organisation sind zwar vom Verfassungsrecht in Grundzügen vorgegeben, im Detail jedoch durch einfache Gesetze und Verordnungen ausgestaltet (Raschauer, 2009, S. 89 ff., 181 f.). In diesem Zusammenhang sei noch darauf hingewiesen, dass die Behördenorganisation nicht der Kompetenz der Europäischen Union, sondern den Nationalstaaten unterliegt.

Sobald jedoch die Verwaltung, in welcher Form auch immer, gegenüber den Rechtsunterworfenen in Erscheinung tritt, also „nach außen" tätig wird, erfordert das verfassungsrechtliche Legalitätsprinzip (Art. 18 B-VG) eine strenge rechtliche Determinierung des Verwaltungshandelns.

---

[17] Möglicherweise muss in Zukunft das Konzept der Bürgerkarte flexibler ausgestaltet und mit Aspekten der „Identity Federation" kombiniert werden, um den Anforderungen einer immer stärker vernetzten Gesellschaft zu genügen (Schweighofer & Hötzendorfer, 2012).

[18] *A-SIT Zentrum für sichere Informationstechnologie - Austria*, Das kann die Bürgerkarte. https://www.buergerkarte.at/anwendungen-karte.html aufgerufen 16.11.2014.

Bei der Vielfalt moderner Kommunikationsmöglichkeiten muss darauf geachtet werden, dass verfassungs- (bspw. die Gewaltenteilung) und datenschutzrechtliche Bestimmungen eingehalten werden. Im Folgenden werden die wichtigsten Rechtsgrundlagen von E-Government überblicksmäßig dargestellt.

Das E-Government-Gesetz BGBl. I Nr. 10/2004 (E-GovG), das am 1. März 2004 in Kraft trat, ist die wichtigste Rechtsgrundlage für die Bereitstellung elektronischer Dienste und ein sicheres E-Government. Es handelt sich dabei innerhalb Europas um ein beispielhaftes Gesetz, das aus diesem Grund für Interessierte auf internationaler Ebene auch ins Englische übersetzt wurde. Um mit den rasch voranschreitenden technischen Entwicklungen Schritt halten zu können, wurde im E-Government-Gesetz bewusst eine technologieneutrale Formulierung gewählt. Der Begriff „Bürgerkarte" bezeichnet beispielsweise eine logische Einheit und nicht zwingend eine Plastikkarte im Scheckkartenformat.

Flankiert wird das E-Government-Gesetz von einer Reihe weiterer Gesetze und Verordnungen. Das Signaturgesetz BGBl. I Nr. 190/1999 (SigG) regelt die rechtlichen Rahmenbedingungen für die Erstellung von elektronischen Signaturen sowie die Erbringung von Signatur- und Zertifizierungsdiensten, ohne die ein sicheres E-Government nicht gewährleistet werden könnte. Das Zustellgesetz BGBl. Nr. 200/1982 (ZustG) regelt unter anderem die für das E-Government unerlässliche elektronische Zustellung und die Anforderungen an Zustelldienste[19].

Auf Grundlage dieser Gesetze wurde die Stammzahlenregisterbehördenverordnung 2009 BGBl. II Nr. 330/2009 (StZRegBehV 2009), die Ergänzungsregisterverordnung 2009 BGBl. II Nr. 331/2009 (ERegV 2009) und die E-Government-Bereichsabgrenzungsverordnung BGBl. II Nr. 289/2004 (E-Gov-BerAbgrV) erlassen.

Für den Schutz personenbezogener Daten im automationsunterstützten Datenverkehr ist das Datenschutzgesetz 2000 BGBl. I Nr. 165/1999 (DSG 2000) zu berücksichtigen. Hervorzuheben ist, dass weder das E-Government-Gesetz noch die E-Government-Bereichsabgrenzungsverordnung ein neues Datenschutzregime konstituieren, sodass stets die Bestimmungen des DSG für die Beantwortung datenschutzrechtlicher Fragen

[19] Es besteht jedoch die Möglichkeit für den Gesetzgeber, die elektronische Zustellung abweichend zu regeln, wie beispielsweise in § 89a GOG RGBl. Nr. 217/1896 i.d.F. BGBl. I Nr. 26/2012.

heranzuziehen sind. Neben dem DSG existiert eine Reihe von Spezialgesetzen zum Schutz bestimmter Arten von Daten. Beispielsweise regelt das Gesundheitstelematikgesetz 2012 BGBl. I Nr. 111/2012 (GTelG 2012) die Verwendung personenbezogener elektronischer Gesundheitsdaten durch Gesundheitsdiensteanbieter.

Am 25.1.2012 veröffentlichte die EU-Kommission den Vorschlag für eine Datenschutz-Grundverordnung[20]. Diese soll die aus dem Jahr 1995 stammende Datenschutzrichtlinie[21] ersetzen und zu einer Vereinheitlichung des Datenschutzes in der EU führen. Bisher existiert die Verordnung nur als Entwurf und wurde noch nicht im Amtsblatt der Europäischen Union kundgemacht. Im Fall des Inkraftretens ist mit gewichtigen Auswirkungen auf den Datenschutz und E-Government-Anwendungen zu rechnen (Knyrim, 2013). Mit einer Verabschiedung ist aus derzeitiger Sicht jedenfalls noch im Lauf des Jahres 2015 zu rechnen.

Die Durchführung des Verfahrens selbst wird durch das formelle Recht bestimmt, wobei unter Umständen Sonderbestimmungen auf die elektronische Verfahrensführung Bedacht nehmen[22]. Das E-Government-Gesetz, BGBl. I Nr. 10/2004 (E-GovG), wurde dafür geschaffen, den elektronischen Verkehr mit öffentlichen Stellen zu erleichtern. Als prominenter Vertreter der Verfahrensvorschriften sei das Allgemeine Verwaltungsverfahrensgesetz 1991, BGBl. Nr. 51/1991 (AVG), genannt.

Weiters stecken die E-Government-Strategie der EU und deren nationale „Umsetzung“[23] den Rahmen für die Realisierung von Projekten ab, in den sich Neuentwicklungen einfügen sollten. Freilich handelt es sich bei den Strategievorgaben der EU nicht um für die Mitgliedsstaaten verbindliche Normen, da die EU keine Kompetenz hat den Staaten den Aufbau ihrer inneren Organisation vorzuschreiben. Die Einhaltung eines gemeinsamen Rahmens kann jedoch die Interoperabilität zwischen Systemen verschiedener Mitgliedstaaten erleichtern. Unter Interoperabilität versteht man in diesem Zusammenhang

[20] KOM (2012) endg.

[21] Datenschutzrichtlinie 95/46/EG.

[22] Beispielsweise nehmen die §§ 89a ff. GOG, RGBl. Nr. 217/1896 i.d.F. BGBl. I Nr. 34/2015, sowie die §§ 16 Abs. 2, 18 Abs. 3 und 4 AVG auf die elektronische Verfahrensführung Bedacht.

[23] *Digitales Österreich*, E-Government Strategie des Bundes. http://www.digitales.oesterreich.gv.at/site/5237/default.aspx aufgerufen: 16.11.2014.

die Fähigkeit von verschiedenen Systemen über eine gemeinsame Schnittstelle miteinander zu kommunizieren.

2.4. Adressaten

E-Government richtet sich nicht nur an die Bürger, sondern auch an Unternehmen und Behörden selbst. Um einen möglichst effektiven Verfahrensablauf zu gewährleisten ist es außerdem erforderlich, dass auch die Kommunikation zwischen den Behörden elektronisch abläuft. Damit dies überhaupt möglich ist, muss auch das Backoffice, d.h. der behördeninterne Bereich, in dem die Aufgaben der Behörde ablaufen, elektronisch unterstützt werden. Kernstück eines solchen Systems ist die elektronische Aktenführung, die den ortsunabhängigen Zugriff auf die Verfahrensdaten jederzeit ermöglicht und damit auch eine Parallelbearbeitung erlaubt.

Die Adressaten des E-Governments lassen sich in folgende Kategorien einteilen:

Government to Citizen (G2C): Darunter versteht man Beziehungen zwischen staatlichen Einrichtungen, die in der Regel mit hoheitlicher Gewalt auftreten (Behörden) und einzelnen Bürgern. Die Rechtsunterworfenen sollen dabei aber nicht mehr als bloße Bittsteller angesehen werden, die einem „übermächtigen“ Verwaltungsapparat gegenüber stehen, sondern verstärkt in den Entscheidungsprozess miteinbezogen werden. Der Bürger kann, falls notwendig, sein Wissen in den Prozess einbringen. Dies macht die Entscheidung der Behörde nachvollziehbar und transparenter, was wiederum zu einem gesteigerten Vertrauen in den Entscheidungsprozess führen kann.

Government to Business (G2B): In diese Beziehungsebene fallen geschäftliche Kontakte zwischen öffentlichen Behörden und Unternehmen, z.B. e-Procurement. E-Government erweist sich mehr und mehr als wichtiger Standortfaktor für in- und ausländische Unternehmen. Vor allem ausländische Investoren werden ihr Unternehmen eher in einem Land betreiben, in dem ein transparentes und effizientes Verfahren gewährleistet ist. Dies betrifft nicht nur die Gründung des Unternehmens, sondern auch den gesamten späteren Kontakt mit Behörden, der aufgrund des täglichen Betriebs des Unternehmens notwendig ist (z.B. die Abgabe der Steuererklärung).

Government to Government (G2G): Um den gesamten Verwaltungsapparat eines Staates effizienter zu machen ist es unerlässlich, dass auch die Kommunikation zwischen einzelnen

Behörden durch IKT unterstützt wird. Daher zählen letztlich die Behörden selbst zu den Adressaten des E-Governments. Kommunikationsflüsse können dabei aufgrund der hierarchischen Organisation von Bund und Ländern sowohl auf horizontaler Ebene (Legislative, Exekutive, Judikative) als auch vertikal zwischen verschiedenen Ebenen auftreten. Als anschauliches Beispiel für vertikale Kommunikation kann die Unterstützung eines Berufungsverfahrens gegen einen Bescheid mittels E-Government dienen. In solch einem Fall muss der durchgängige Informationsfluss von der erstinstanzlichen Behörde bis zur ersten (und eventuell zweiten) Berufungsinstanz mittels IKT gewährleistet werden.

Sämtliche Kommunikation – auch innerhalb einer Behörde selbst – sollte jedenfalls ohne Medienbrüche erfolgen. Unter einem Medienbruch versteht man den Wechsel des informationstragenden Mediums innerhalb eines Informationsbeschaffungs- oder Verarbeitungsprozesses. Um dieses Ziel zu erreichen, ist es erforderlich, dass sämtliche Bearbeitungsprozesse auf elektronischem Wege erfolgen. Der gesamte Aktenlauf, vom Eingang bis zum Ausgang z.B. in Form eines Bescheides, sollte elektronisch und nicht in Form eines traditionellen Aktes in Papierform geführt werden.

2.5. Ziele

2.5.1. Steigerung der Effizienz der Verwaltung

Ziel von E-Government ist es möglichst effiziente und transparente Verfahrensvorgänge zu schaffen und mittels IKT[24] zu unterstützen. Um Verfahren kostengünstiger und effizienter zu machen, ist es nicht mit der bloßen Abbildung bestehender Verfahrensabläufe und Arbeitsweisen mittels moderner Technologien getan. Vielmehr sollten bestehende Abläufe im Vorfeld analysiert und gegebenenfalls neu modelliert werden, um sie von unnötigem Ballast zu befreien. Nur selten kann mit der Einführung neuer Technologien alleine die gewünschte Effizienzsteigerung erzielt werden.

Freilich bietet die gesetzliche Grundlage selbst auch Raum für Verbesserungen. Der technischen Umsetzung sollte daher gegebenenfalls eine gesetzliche Reform vorausgehen. Auf einer „schlankeren“ gesetzlichen Grundlage lassen sich schließlich auch einfachere Verwaltungsvorgänge modellieren, die dann von der Software implementiert werden. Die

[24] „IKT“ steht für Informations- und Kommunikationstechnik.

Novellierung von Gesetzen ist ein eher langsamer Prozess, der nicht immer im Rahmen von E-Government-Projekten durchgeführt werden kann. Je klarer die gesetzliche Grundlage ist, desto nachvollziehbarer wird die Entscheidung, auch für juristische Laien, sein. Erhält der Betroffene noch zusätzlich online Auskunft über den aktuellen Stand des laufenden Verfahrens, wird dies zu einem gesteigerten Vertrauen in E-Government und die gesamte Verwaltung führen.

Ziel eines E-Government-Projektes sollte also nicht nur eine bloße Technologieablöse sein, sondern auch eine Optimierung der gesetzlichen Grundlage und der Verwaltungsorganisation, die dem Softwareprojekt vorgeschaltet ist oder zumindest mit diesem Hand in Hand geht. Änderungen der Behördenorganisation und Novellen von Gesetzen sind jedoch oftmals schwerfällig und ziehen sich über lange Zeit hin, sodass die Implementierung der Systeme zwangsläufig früher beginnt.

E-Government kann demnach wesentlich die Qualität eines Verwaltungsverfahrens erhöhen („Effektivität“ bzw. Zielerreichungsgrad). Daten können vom System automatisiert einer Plausibilitätsprüfung, eventuell unter Zuhilfenahme anderer Systeme, unterzogen werden. Nur „Ausreißer“, d.h. Fälle, die bestimmten Kriterien der Plausibilitätsprüfung widersprechen, werden einem Sachbearbeiter zur weiteren Prüfung vorgelegt. Der Benutzer kann durch automatisches Ausfüllen von Formularen aufgrund bereits vorhandener Daten beim Einbringen seiner Anträge wesentlich unterstützt werden. Dies kann außerdem die Eingabe falscher Daten vermeiden und so zusätzlich die Qualität der Anträge verbessern.

Für den Bürger soll eine einheitliche Anlaufstelle („one stop shop“) geschaffen werden, bei der er sämtliche Anträge online einbringen kann. Dieser One-Stop-Shop präsentiert sich im Web in der Form eines Portals, das z.B. nach Lebenssituationen gegliedert ist und dem Bürger Informationen anbietet[25]. Behörden können auch Anträge online entgegennehmen und das System leitet diese an die zuständigen Behörden weiter. Dem Betroffenen bleibt damit eine lange Suche nach der richtigen Anlaufstelle für sein Anliegen erspart.

[25] Siehe beispielsweise das Portal http://www.help.gv.at/ aufgerufen: 18.3.2014.

2.5.2. Zugang für alle Bürger zu E-Government-Diensten

Die Problematik der „digitalen Kluft“ (engl. „digital divide“) wurde zuvor bereits kurz angesprochen. Man versteht darunter die ungleiche Verteilung des Zugangs zu digitalen Informations- und Kommunikationstechnologien (IKT) in der Gesellschaft, also die Unterscheidung zwischen „Onlinern“ und „Offlinern“ (Promberger, Bernhart, & Früh, 2010, S. 14 ff.). Dabei berücksichtigt die OECD[26] in ihrer Definition nicht nur den Zugangsaspekt, sondern bezieht auch Unterschiede bezüglich der tatsächlichen Anwendung von IKT im Alltag mit ein. Selbst wenn die entsprechenden technischen Strukturen vorhanden sind, bedeutet das nicht, dass sich daraus der gleichmäßige tatsächliche Zugang aller Bevölkerungsgruppen ergibt. Man kann daher zwischen einer „ersten digitalen Kluft“ (engl. „first digital divide“) und einer „zweiten digitalen Kluft“ (engl. „second digital divide“) unterscheiden. Erstere bezieht sich auf Unterschiede im Hinblick auf den technologischen Zugang, letztere hingegen auf soziale Differenzen betreffend der Anwendung neuer Technologien.

Wie schon erwähnt, führen die ungleiche Nutzung und der ungleiche Zugang zu IKT letztendlich zu einem ungleichen Zugang zu Verwaltungsleistungen. Der Staat steht dabei vor der Herausforderung – anders als Unternehmen – sämtlichen Bürgern dienen zu müssen, unabhängig von deren persönlicher Situation und Wohnort. Die Frage nach dem konkreten Inhalt und Umfang dieser Pflicht soll an dieser Stelle nicht weiter behandelt werden.

Voraussetzungen, um E-Government-Dienste überhaupt in Anspruch nehmen zu können, sind demnach ein ausreichend schneller Internetzugang und die notwendigen Kenntnisse für die Benutzung.

Der Zugang zu elektronischen Kommunikationstechnologien soll durch das Telekommunikationsgesetz (TKG) gewährleistet werden. Das TKG definiert so genannte Universaldienste (§ 26 TKG BGBl. I Nr. 70/2003 i.d.F. BGBl. I Nr. 102/2011). Gemäß der Definition versteht man darunter *„ein Mindestangebot an öffentlichen Diensten, zu denen alle Endnutzer unabhängig von ihrem Wohn- oder Geschäftsort zu einem erschwinglichen Preis Zugang haben müssen.“*.

[26] Organisation for Economic Co-operation and Development.

Das Gesetz bleibt jedoch eine genaue Definition der Mindestgeschwindigkeit für Internetzugänge schuldig. Zur Interpretation kann Erwägungsgrund 8 der Universaldiensterichtlinie[27], die im TKG umgesetzt wurde, herangezogen werden. Demnach handelt es sich bei einem Universaldienst nicht um einen Breitbanddienst, sondern um Dienste mit einer Höchstgeschwindigkeit von 56 Kilobit pro Sekunde (Schmalband), was einem traditionellen Internetzugang mittels Modem über das Telefonnetz entspricht. Moderne Webseiten besitzen eine ausgereifte Benutzeroberfläche, die traditionellen Fat-Client-Anwendungen um nichts nachsteht, und stellen höhere Ansprüche an die erforderliche Übertragungsgeschwindigkeit des Internetzugangs. Moderne Webangebote sind daher ohne eine leistungsfähige Breitbandverbindung nur sehr eingeschränkt nutzbar.

Eine der größten politischen Herausforderungen ist es daher der digitalen Kluft entgegenzuwirken und unter anderem Breitbanddienste auch für ländliche Gegenden zugänglich zu machen. Das Bestreben des Staats sollte sein, dass auch in abgelegenen Gebieten ein ausreichend schneller Internetzugang bereitgestellt wird, damit die Bürger E-Government Dienste vernünftig nutzen können. Da die Erschließung von dünn besiedelten Gegenden mit Breitbanddiensten für private Unternehmen in der Regel unrentabel ist, liegt es hier am Staat durch Subventionen den Ausbau von Breitbanddiensten (weiter) voranzutreiben. Derzeit muss allerdings, wie oben bereits ausgeführt, nur der Zugang zu Schmalbanddiensten gewährleistet werden[28].

Wie bereits eingangs erwähnt, ist der Staat verpflichtet allen Bürgern gleichmäßig zu dienen. Im Gesetz findet sich diese Anforderung im verfassungsrechtlichen Gleichheitssatz (Art. 7 B-

---

[27] UDRL 2002/22/EG.

[28] Exkurs zur Zulässigkeit staatlicher Subventionen: Bei der Subvention einzelner Unternehmen müssen die Grenzen des europäischen und nationalen Wettbewerbsrechts eingehalten werden. Der Telekommunikationssektor unterliegt (noch) nicht ausschließlich europäischen Wettbewerbsregeln, sondern einer sektorspezifischen Regulierung, durch die sukzessive ein wirksamer Wettbewerb auf diesen Märkten geschaffen werden soll. Der Telekommunikationsmarkt ist auf europäischer Ebene durch eine Reihe von Richtlinien geregelt: Rahmen-RL 2002/21/EG, Genehmigungs-RL 2002/20/EG, Zugangs-RL 2002/19/EG, Universaldienste-RL 2002/22/EG und die Datenschutz-RL 2002/58/EG. In Österreich wurde das Telekompaket mit dem Telekommunikationsgesetz 2003 BGBl. I Nr. 70/2003 umgesetzt.

Zentrales Anliegen der Universaldienste-RL ist unter anderem die Schaffung eines Mindestangebots an sogenannten „Universaldiensten“. Es obliegt den Mitgliedsstaaten zu garantieren, dass dieses Angebot auch erbracht wird. Damit Unternehmen solche Dienste erbringen können, ist es zulässig, dass die Kosten aus öffentlicher Hand gedeckt werden, falls der Dienst unrentabel ist. Im Unterschied dazu verbieten auf anderen Märkten, bis auf wenige Ausnahmen, die EU-Wettbewerbsregeln staatliche Subventionen, um Verzerrungen des Wettbewerbs zu verhindern (Birnstiel, Bungenberg, & Heinrich, 2013).

VG) und im E-Government-Gesetz (§ 1 Abs. 3 E-GovG) wieder. Zentrales Ziel von E-Government ist demnach, dass sämtliche Dienste auch für Benutzer mit Einschränkungen zugänglich sind. Um dies zu gewährleisten, muss bei der Entwicklung von Webseiten auf die Einhaltung von Standards – wie z.B. der „Web Content Accessibility Guidlines“[29] – geachtet werden.

Die barrierefreie Gestaltung von Webauftritten steht in einem gewissen Spannungsfeld zu Anforderungen an die Bedienbarkeit und Benutzerfreundlichkeit, da bestimmte Technologien und Elemente durch die Standards nicht verwendet werden dürfen. Die Lösung dieses Problems erhöht in den meisten Fällen den Aufwand für die Entwicklung erheblich.

2.5.3. Datenschutz und Sicherheit

In vielen Fällen wird es sich bei den in ein Verwaltungsverfahren eingebrachten Daten um (höchst) persönliche Daten des Einbringers handeln. Gemeint sind Daten, bei denen es sich in der Terminologie des Datenschutzgesetzes um „personenbezogene Daten“ (§ 4 Zf. 1 DSG) oder „sensible Daten“ (§ 4 Zf. 2 DSG) handelt.

Aus diesem Grund muss bei E-Government Anwendungen besonderes Augenmerk auf den Datenschutz und Sicherheitsmaßnahmen gelegt werden. Während der gesamten Verarbeitung der Daten, von der Eingabe über die Speicherung bis zur Übertragung von Daten an andere Systeme, muss gewährleistet sein, dass diese vor dem unberechtigten Zugriff Dritter geschützt werden. Versäumnisse beim Datenschutz können einen schwer wieder gut zu machenden Imageschaden für die Verwaltung und E-Government bedeuten.

Eine weitere Notwendigkeit für sicheres E-Government ist die eindeutige Identifizierung des Antragstellers. Diese Authentifizierung kann auf herkömmliche Weise mittels Benutzername und Passwort erfolgen. Die weitaus sicherere Variante der Authentifizierung mittels Bürgerkarte war in Österreich von Anfang an vorgesehen. Aufgrund der noch spärlichen Verbreitung derselben bei Privatpersonen bieten viele E-Government-Services nach wie vor einen Zugang auf herkömmliche Weise an. Vornehmliches Ziel muss sein, die Verbreitung der Bürgerkarte – vor allem bei Privatpersonen – weiter zu fördern. Die Einführung der

[29] *W3C*, Web Content Accessibility Guidelines (WCAG) 2.0. http://www.w3.org/TR/WCAG20/ aufgerufen: 4.12.2010.

Handysignatur ist ein neuerlicher Versuch die Bürgerkarte einem breiteren Teil der Bevölkerung zugänglich zu machen.

Die Bürgerkarte ist außerdem Voraussetzung für die Erzeugung qualifizierter elektronischer Signaturen. Mit Hilfe von kryptographischen Verfahren werden elektronische Signaturen erzeugt, die sicherstellen sollen, dass Daten nicht verfälscht werden können (Integrität) und tatsächlich vom jeweiligen Antragsteller stammen (Authentizität).

2.6. Kritik

Durch E-Government steigt stetig die Zahl der IT-basierten Systeme, die Daten von natürlichen und juristischen Personen verarbeiten. Neue Technologien erlauben es verschiedene Systeme (von Behörden, Unternehmen etc.) über die Grenzen einer Behörde hinweg mithilfe gemeinsamer Schnittstellen zu verbinden. Die so entstehenden integrierten Systeme ermöglichen den Austausch beliebiger Daten. So sind unter anderem automatisierte Abfragen und Auswertungen der gespeicherten Daten möglich.

Mit der Sammlung von Daten und der Möglichkeit der automatisierten Abfrage und Verknüpfung steigt die Gefahr des potentiellen Missbrauchs an. Dieses Problem wird im E-Government-Umfeld besonders augenscheinlich, da oftmals auch sensible Daten natürlicher Personen von IT-basierten Systemen verarbeitet werden. Anders als bisher bei Akten, die in diversen Behörden in Papierform geführt werden, ermöglicht die Software Verknüpfungen beliebiger Daten aus verschiedenen Systemen, solange diese einer bestimmten Person mithilfe identifizierender Daten zugeordnet werden können. An dieser Stelle sei erwähnt, dass auch Anbieter (Unternehmen) aus dem privaten Bereich die Authentifizierung ihrer Kunden mittels Bürgerkarte verwenden können. Eine Verknüpfung auch mit Daten von Unternehmen ist somit denkbar.

Um dieses, nicht nur aus datenschutzrechtlicher Sicht, bedenkliche Szenario des „gläsernen Menschen“ zu verhindern, wurden in Österreich mehrere Sicherheitsmaßnahmen ergriffen.

Jede natürliche Person wird durch die Stammzahl, die von der ZMR-Zahl aus dem Zentralenmelderegister durch eine symmetrische Verschlüsselung abgeleitet wird, identifiziert. Die Stammzahl für natürliche Personen kann damit nicht auf die ZMR-Zahl zurückgeführt werden. Für juristische Personen wird als Stammzahl die Firmenbuchnummer

bzw. die ZVR-Zahl aus dem Zentralen Vereinsregister verwendet. Für sonstige Betroffene ist dafür die im Ergänzungsregister vergebene Ordnungsnummer zu verwenden.

Um zu verhindern, dass verschiedene staatliche Stellen in andere Verfahren einer natürlichen Person Einblick erhalten, wird die Stammzahl nicht direkt verwendet, sondern ein weiteres Mal verschlüsselt, um das so genannte „bereichsspezifische Personenkennzeichen“ (bPK)[30] zu erhalten (Karning, 2004, S. 50 ff.). Dieses ist für verschiedene Verwaltungsbereiche unterschiedlich, siehe E-Government-Bereichsabgrenzungsverordnung. Eine direkte Rückführung des bPK zur Stammzahl ist nicht möglich. Benötigt eine Behörde Zugriff auf die Daten eines anderen Verwaltungsbereichs, beispielsweise im Zuge der Amtshilfe, oder liefert diese Daten zu Statistikzwecken an die Statistik Austria, so kann mithilfe des Stammzahlenregisters das bPK des anderen Verwaltungsbereichs errechnet werden. Dieses sogenannte Fremd-bPK darf nur in verschlüsselter Form gespeichert werden.

Dieser Ansatz ist im Grunde durchaus geeignet, unzulässige Verknüpfungen von Daten zu verhindern. Die Verwendung von bPKs zur Identifikation von natürlichen Personen ist jedoch nicht zwingend und bleibt der jeweiligen Behörde überlassen. In der Praxis findet sich bis heute eine große Anzahl von IT-Anwendungen, die wie bisher beispielsweise den Namen oder die Versicherungsnummer verwenden um Personen zu identifizieren. Aufgrund des Anpassungsaufwandes und der technischen Schwierigkeiten verzichten viele öffentliche Stellen auf die durchgängige Verwendung von bPKs.

Die bloß fakultative Verwendung von bPKs und der daraus folgende Gebrauch von identifizierenden Kriterien wie Name, Geburtsdatum und Versicherungsnummer etc. ermöglicht eine systemübergreifende Verknüpfung von Daten einer Person, wenn diese in Schnittstellen zum Einsatz kommen. Diese Problematik sollte bei der Konzeption von E-Government-Anwendungen berücksichtigt und gegebenenfalls gelöst werden.

Wie jedes Gesetz unterliegen auch die Regelungen des E-Governments ständigem Wandel und sind nicht vor politischen Einflüssen gefeit. Die derzeitige Rechtslage ist damit keinesfalls

[30] Für eine Einführung zum „bereichsspezifischen Personenkennzeichen“ siehe *Bundeskanzleramt*, Bürgerkarte. http://oesterreich.gv.at/site/cob__19457/5244/Default.aspx#a2 aufgerufen: 20.10.2010.

für alle Zeit „eingefroren“ und kann in Zukunft durchaus Änderungen unterworfen sein, die dem Datenschutz keinen so hohen Stellenwert einräumen.

Zudem haben die jüngsten Enthüllungen von Snowden offenbart, dass Geheimdienste im großen Stil die weltweite Internetkommuniktion überwachen. Diese Entwicklungen legen es nahe, die vorhandenen Mechanismen zum Schutz von (personenbezogenen) Daten zu überprüfen und gegebenenfalls zu verbessern (Hötzendorfer & Schweighofer, 2014).

Weiteren Anlass zur Kritik bietet die ungleiche Verteilung des Nutzens von E-Government-Anwendungen, die sich notwendigerweise aus der digitalen Kluft ergibt. Es ist zu befürchten, dass Menschen, die keinen Zugang zum Internet haben, im Hinblick auf die Wahrung ihrer Rechte, zu „Bürgern zweiter Klasse“ werden.

2.7. Zukunftsaussichten

Der derzeitige Stand der Technik verlangt es, dass Daten in einer vordefinierten Struktur eingegeben und übermittelt werden müssen. Eingabeformulare auf Webseiten dienen dem Zweck, die Daten in strukturierter Form an das Computersystem zu übermitteln. Die Software versteht den Inhalt der Daten nicht, sondern operiert auf einer rein syntaktischen Ebene. Die Verarbeitung der Daten erfolgt ausschließlich aufgrund deren Struktur. Die Analyse und Verarbeitung von frei eingegebenem Text durch ein Computersystem – ähnlich der intellektuellen Tätigkeit eines Juristen – ist derzeit noch eine Zukunftsvision.

Nicht nur bei der Erfassung müssen die Daten in eine bestimmte Form gebracht werden, sondern auch beim Austausch von Daten mit anderen Systemen. Dabei muss das aufrufende System seine eigenen Daten in ein Format transformieren, das durch die Schnittstellenbeschreibung des aufgerufenen Systems vorgegeben ist. Nur so kann das Zielsystem die übermittelten Daten verarbeiten. Die Transformation betrifft aber nicht nur das Format der Daten, z.B. Übertragung im XML-Format, sondern auch den Inhalt (Semantik). Ein bestimmter Begriff (Wert) kann in verschiedenen Systemen eine unterschiedliche Bedeutung haben und so zu inkorrekten Ergebnissen führen.

Aufgrund dieser Umstände ist die Integration von heterogenen Systemen schwierig. Zumindest das aufrufende System muss Kenntnis über die Schnittstelle des aufgerufenen Systems haben, um die Daten entsprechend transformieren und übermitteln zu können.

Wegen abweichender Semantik ist die Integration von Systemen nicht immer nahtlos möglich.

Abhilfe für dieses Dilemma könnte das „Semantic Web“ – auch als Web 3.0 bezeichnet – schaffen. Stark verkürzt geht es darum, dass es Computersystemen möglich sein soll Inhalte zu „verstehen“. Daten werden dabei mit Metadaten angereichert. Mit Hilfe von Ontologien und Inferenzregeln können Maschinen schließlich Schlüsse aus den modellierten Daten ableiten.

Ähnlich wie das Web 2.0 mit seinen Blogs, Wikis und Sozialen Netzwerken zur Entwicklung des E-Government 2.0 (Chun et al., 2010) beigetragen hat, ist zu erwarten, dass das Web 3.0 (Davies et al., 2009), wie das Semantic Web auch genannt wird, durch den Einsatz von semantischen Technologien den Weg hin zum E-Government 3.0 bereitet.

Auf dem Semantic Web basieren Technologien wie die sogenannten „Semantic Web Services“ (Mc Ilraith, Son, & Zeng, 2001), deren Schnittstellen mit semantischen Daten angereichert werden, was deren automatisierte Auffindung und Verwendung erleichtert (J. M. García, Ruiz, & Ruiz-Cortés, 2012). Mithilfe semantischer Technologien ist der einfachere Austausch von Daten zwischen heterogenen Systemen, auch über Staatsgrenzen hinweg, möglich.

Ansatzweise werden semantische Technologien schon heute von der Verwaltung verwendet. Der „Semantische Dienstleistungsassistent“[31] verwendet eine Ontologie zur Generierung der Formulare, die zur Datenerfassung dienen. Er basiert auf einem Framework namens SeGoF (Semantic eGovernment Forms) (Salhofer, Stadlhofer, & Tretter, 2009; Stadlhofer & Salhofer, 2008).

Vermehrt geht die Verwaltung dazu über vorhandene Daten im Sinne von „Open Data“ öffentlich zur Verfügung zu stellen. Im Rahmen von „Linked (Open) Data“ (Bizer, Heath, & Berners-Lee, 2009) wird dieser Gedanken weitergeführt und mithilfe von semantischen Technologien (URIs und RDF) können die vorhandenen Daten verknüpft werden. Damit entsteht ein weltweites Netz, das auch „Linked Open Data Cloud“ genannt wird. Zuvor

[31] *Bundesministerium für Wissenschaft*, Forschung und Wirtschaft, Dienstleistungsassistent. https://www.usp.gv.at/Portal.Node/usp/public/content/gruendung/dienstleistungsassistent/41113.html aufgerufen: 12.2.2015 (2015)

isolierte Datenbestände entfalten durch die Kombination mit anderen Informationen ihr volles Potential.

3. Status quo der Verwaltungsautomatisierung

3.1. Einleitung

Der Einsatz von juristischen Expertensystemen[32] zur Untersützung der Vollziehung durch Verwaltungsbehörden hat in Österreich eine lange Tradition, die bis in die 1970er-Jahre zurück reicht (Svoboda, 1975). Als wohl populärster Vertreter solcher Systeme sei FinanzOnline[33] genannt, das seit vielen Jahren ermöglicht, dass der Steuerpflichtige Erklärungen an die Finanzbehörden online übermitteln kann (Kutschera, 2004). Darüber hinaus werden in vielen Behörden im verwaltungsinternen Backoffice Programme zur Unterstützung diverser Aufgaben eingesetzt. Fielder (2004) spricht in diesem Zusammenhang von „stiller Formalisierung", da sie von der Rechtsinformatik zu wenig beachtet wurde.

Zu Beginn der Dissertation wurde bereits erwähnt, dass die in der Praxis gängigen Ansätze zur Implementierung von juristischen Expertensystemen in der Verwaltung mit einigen Problemen behaftet sind. Dieses Kapitel widmet sich diesen Problemen und unterzieht die gängigen Ansätze einer kritischen Würdigung[34].

Vorausgeschickt sei, dass die Implementierung von juristischen Expertensystemen in der Praxis ohne Berücksichtigung der existierenden Theorien erfolgt[35]. Die Forschungen auf dem Gebiet der Künstlichen Intelligenz und Recht und die Praxis der Verwaltungsautomatisierung haben sich parallel entwickelt. Die Formalisierung von Normen geschieht gewissermaßen ad hoc durch den Softwareentwickler[36], ohne Berücksichtigung rechtstheoretischer

---

[32] Der Begriff „Expertensystem" wird in diesem Kapitel recht weit zur Bezeichnung von Software zur Unterstützung der Vollziehung durch Verwaltungsorgane verstanden. Nicht gemeint sind damit juristische wissensbasierte Systeme, da für diese charakteristisch ist, dass die formale Wissensrepräsentation vom Lösungsalgorithmus getrennt ist. Diese Eigenschaft weisen jedoch gängige Systeme in der Praxis nicht auf.

[33] *BMF*, FinanzOnline. https://finanzonline.bmf.gv.at/fon/ aufgerufen: 4.1.2015.

[34] Da in der Literatur eine entsprechende Referenz fehlt, basieren die Ausführungen in diesem Kapitel hauptsächlich auf den Erfahrungen des Autors, die dieser während seiner langjährigen Tätigkeit als Entwickler und Softwarearchitekt im E-Government-Umfeld gemacht hat.

[35] Eine ähnliche Beobachtung hat Moles (1992) gemacht, der unter anderem den regelbasierten Ansatz zur Repräsentation von Normen kritisiert und für eine stärkere Berücksichtigung der Rechtstheorie eintritt.

[36] „Ad hoc" deshalb, weil kein abstraktes Konzept, gestützt von einer soliden Theorie, zur Verfügung steht, wie Normen repräsentiert werden sollen. Ebenso fehlt es an einer einheitlichen Methode, wie bei der Formalisierung von Normen vorzugehen ist. Es steht dem Softwareentwickler weitgehend frei, wie dieser Normen im Code implementiert.

Überlegungen. Meines Erachtens ist dieser Umstand die Hauptursache für die im Folgenden diskutierten Unzulänglichkeiten gängiger Ansätze der Formalisierung.

Im Fokus der Untersuchung stehen jene Ansätze, die am häufigsten in der Praxis der Verwaltungsautomatisierung zur Implementierung von Expertensystemen zum Einsatz kommen. Die aufgezeigten Probleme sind im Kontext von Expertensystemen zur Unterstützung der Vollziehung zu verstehen.

3.2. Anforderungen an juristische Expertensysteme

Die Problemstellung, die es bei der Implementierung von juristischen Expertensystemen zu lösen gilt, ist die Übersetzung von einem abstrakten Modell – den Normen – in ein anderes – die Software. Die normative Grundlage für die zu implementierenden Funktionen stellen dabei Gesetze, Verordnungen und (verwaltungsinterne) Weisungen[37] dar. Für eine eingehende Darstellung der Rechtsquellen, die es bei der Implementierung von juristischen wissensbasierten Systemen im öffentlichen Bereich zu berücksichtigen gilt, sei auf Abschnitt 11.3. verwiesen.

Das Recht weist einige Charakteristika auf, die sich auf die Implementierung von juristischen wissensbasierten Systemen auswirken. Aus diesem Grund soll hier auf diese Besonderheiten und die Anforderungen, die aus diesen erwachsen, näher eingegangen werden. Diese Feststellungen ermöglichen überhaupt erst die Identifizierung von potentiell brauchbaren Lösungsansätzen.

Insbesondere im Verwaltungsrecht herrscht eine recht hohe Regelungsdichte. Der Gesetzgeber trachtet oftmals danach möglichst alle denkbaren Sachverhalte abschließend zu regeln, was zu einer Vielzahl von Varianten und Sonderfällen im Gesetz führt. Aus Sicht der Softwareentwicklung ist dies einerseits positiv hervorzuheben, da relativ genau ausgedrückt wird wie in einem bestimmten Fall zu verfahren ist. Das Gesetz gibt so eine exakte Anleitung, was das Programm im Einzelfall tun soll.

Andererseits stellt sich die normative Grundlage durch die zahlreichen Sonderfälle relativ komplex und unübersichtlich dar. Man denke hier vor allem an Rentenberechnungen mit

[37] Bei „Weisungen“ handelt es sich um behördeninterne Anweisungen, die unter anderem anordnen, wie in bestimmten Fällen zu verfahren ist. Sie konkretisieren also unter Umständen Gesetze und Verordnungen näher.

zahlreichen Staffelungen und „Einschleifregelungen“[38], so wie sie auch dem KOVG immanent sind. Die hohe Komplexität überträgt sich schließlich auch auf die Software. Die zahlreichen Varianten des Gesetzes führen in der Regel zu einer fast unüberschaubaren Anzahl von Verzweigungen innerhalb des Programmcodes. Letzterer wird dadurch komplex und unübersichtlich, was letzten Endes Anpassungen schwierig macht und so schließlich zu einer Verteuerung der Wartung führt. Ein Expertensystem muss also mit dieser Komplexität umgehen können und die Regeln korrekt abbilden. Der Code muss zudem möglichst einfach, d.h. insbesondere gut strukturiert und lesbar, gehalten werden.

Das in Art. 18 B-VG[39] verankerte Legalitätsprinzip stellt die wohl wichtigste Handlungsanweisung für Verwaltungsbehörden dar. Es postuliert die unbedingte Bindung des behördlichen Handelns an die Gesetze. Die Einrichtung und der innere Aufbau von Behörden sind jedoch in der Verfassung nur in Grundzügen vorgegeben. Durch die Verfassung und insbesondere die Verfahrensgesetze sind damit zwar Inhalt und Ablauf von Verwaltungsverfahren bestimmt, die organisatorische Ausgestaltung von Behörden ist von diesen aber nur in Grundzügen betroffen. Damit bleibt es der vollziehenden Behörde überlassen, welcher Hilfsmittel sich diese bedient um ein Verfahren durchzuführen. Im gegenständlichen Zusammenhang bedeutet das, dass es der Behörde freisteht sich eines Expertensystems zur Unterstützung zu bedienen. Die Behörde ist jedoch selbstverständlich weiterhin den Rechtsunterworfenen für Verfahrensfehler etc. verantwortlich.

Die Qualität von Expertensystemen muss sich demnach am strengen Maßstab des Legalitätsprinzips messen lassen. Weitere Anforderungen an das behördliche Handeln ergeben sich insbesondere aus den Verwaltungsverfahrensgesetzen. Damit sind an die Software hinsichtlich der Richtigkeit der Ergebnisse besonders hohe Ansprüche gestellt. Dies erscheint auch gerechtfertigt, da unter Umständen derartige Expertensysteme sogar Entscheidungen, insbesondere Bescheide, vorbereiten oder sogar vollautomatisiert erlassen, die Rechtswirkungen für die Parteien erzeugen. Angemerkt sei, dass eine automatisierte

[38] Damit sind hier Normen gemeint, die eine Leistung aliquot in Abhängigkeit von einer bestimmten Grenze (oft das Einkommen) gewähren. Der Anspruch auf eine Leistung besteht nur insoweit, als dass üblicherweise das Einkommen die jeweilige Grenze nicht erreicht. Wird die Grenze erreicht oder überschritten, entfällt der Anspruch vollständig. Vgl. § 12 Abs. 2 KOVG BGBl. Nr. 152/1957 geändert durch BGBl. II Nr. 468/2012.

[39] BGBl. Nr. 1/1930 i.d.F. BGBl. I Nr. 51/2012.

Entscheidung dann „richtig“ ist, wenn das zugrundeliegende Gesetz fehlerfrei angewendet wurde und der erstellte Bescheid den gesetzlichen Inhaltserfordernissen entspricht.

Eng verbunden mit dem Legalitätsprinzip und ebenso ein Ausfluss des in der Verfassung verankerten Rechtsstaatsprinzips ist das Erfordernis der Begründung von Entscheidungen. Diese in § 58 Abs. 2 AVG[40] normierte Pflicht wurde darüber hinaus durch die Judikatur der Gerichtshöfe des öffentlichen Rechts zu einem rechtsstaatlichen Grundsatz erhoben (Wielinger, 2008, S. 21 Rz. 32 ff.). Die Pflicht Verwaltungsentscheidungen zu begründen ermöglicht es den Parteien überhaupt erst, sich wirksam gegen diese im Instanzenzug zu verteidigen. Ohne diese obligatorischen Ausführungen, die eine Entscheidung untermauern, wäre diese letztlich einem rechtlichen Diskurs entzogen und der Willkür Tür und Tor geöffnet.

Daraus folgt, dass das Ergebnis eines juristischen Expertensystems stets den Konnex zur Rechtsgrundlage bewahren muss, damit die Entscheidung begründet und damit rechtlich gerechtfertigt werden kann. Zusätzlich müssen die Entscheidungen, die von der Software bei der Subsumtion getroffen werden, auf nachvollziehbare Art und Weise dokumentiert werden, sodass erkennbar ist, warum das System eine Norm für anwendbar hält oder nicht.

An juristische Expertensysteme bestehen demnach aufgrund rechtlicher Erfordernisse erhöhte Anforderungen, was die Dokumentation und Nachvollziehbarkeit des Ergebnisses betrifft.

Gesetze unterliegen als Verhaltensanordnung, deren Kern auch auf moralischen Wertungen fußt, dem ständigen Wandel der Zeit. So werden diese, wie beispielsweise das KOVG, jährlich durch eine Verordnung novelliert, um eine Indexanpassung der normierten Beträge vorzunehmen. Im Gegensatz zu anderen Bereichen unterliegt das Recht ständigen Änderungen, die notwendigerweise auch zu einer Anpassung der im System implementierten Normen führen. Formale Modelle des Gesetzes müssen daher möglichst flexibel auf Änderungen der Normen reagieren können, sodass die neue Gesetzeslage dem System zur Verfügung steht.

Unter Umständen muss ein juristisches Expertensystem mehrere Fassungen eines Gesetzes gleichzeitig unterstützen. Dies ist insbesondere dann der Fall, wenn bereits verwirklichte

[40] BGBl. Nr. 51/1991.

Sachverhalte nach der alten Rechtslage beurteilt werden müssen oder diese für bereits anhängige Verfahren weiterhin anwendbar bleibt[41]. Das System muss damit mit dem zeitlichen Anwendungsbereich von Normen umgehen können, dies bedingt, dass die formale Repräsentation der Normen mit einem Geltungszeitraum versehen werden kann. Angemerkt sei, dass in der Praxis dieses Problem oft dadurch umgangen wird, dass laufende Verfahren nach Möglichkeit vor dem Inkrafttreten einer Novelle abgeschlossen werden. Dadurch kann die Notwendigkeit, dass ein Expertensystem mehrere Fassungen eines Gesetzes unterstützen muss, zurückgedrängt werden. Freilich ist dies nicht immer durchgängig möglich.

Es liegt auf der Hand, dass juristische Expertensysteme gesetzliche Änderungen mit dem Tag beherrschen müssen, an dem diese in Kraft treten. Bevor die Novelle jedoch kundgemacht wird, bestehen oftmals nur vage Informationen hinsichtlich der bevorstehenden Anpassungen, sodass der Zeitraum für Änderungen der Software denkbar knapp ist. Jede Änderung verursacht nämlich nicht nur einen Implementierungsaufwand, sondern muss auch einen entsprechenden Qualitätssicherungszyklus, also diverse Tests, durchlaufen und bestehen. Juristische Expertensysteme müssen daher in der Lage sein möglichst schnell auf Änderungen der Normen zu reagieren, wobei die Modifikation der Wissensbasis mit möglichst geringem Aufwand verbunden sein soll.

Damit Änderungen einen möglichst geringen Aufwand verursachen, muss ein juristisches wissensbasiertes System bestimmte Voraussetzungen erfüllen. Erstens muss die Stelle, an der ein Expertensystem geändert werden muss, einfach lokalisiert werden können, um auf eine Änderung der Rechtsquellen zu reagieren. Dieses Problem betrifft den Isomorphismus zwischen der Rechtsquelle und der Repräsentation der Normen. Zweitens sollten Änderungen nur an einer einzigen Stelle durchgeführt werden müssen. Noch besser wäre, wenn an einer Stelle alle Systeme modifiziert werden können, die auf derselben Rechtsquelle basieren. Soweit wie möglich sollte daher die Wissensbasis aus wiederverwendbaren Wissenskomponenten (knowledge components) zusammengestellt werden, die an einer einzigen Stelle gewartet werden können (Boer, 2009, S. 29).

[41] Novellen sehen oftmals entsprechende Übergangsbestimmungen vor.

Neben dem formalen Modell der Normen muss ein juristisches Expertensystem auch den Sachverhalt im Einzelfall angemessen repräsentieren. Die formale Darstellung des Sachverhalts umfasst Akteure, Handlungen, Objekte und komplexe Beziehungen zwischen diesen sowie deren zeitliche Abfolge. Aus diesem Grund besteht die Wissensbasis eines juristischen Expertensystems oftmals aus mehreren Komponenten, die Normen und Sachverhalte repräsentieren[42].

Zusammengefasst muss ein Expertensystem zur Unterstützung der Vollziehung folgende Anforderungen erfüllen: (1) Repräsentation von Normen und Fällen. (2) Die Qualität des Ergebnisses muss sehr hohen Ansprüchen genügen. (3) Das Ergebnis muss nachvollziehbar und begründet sein. (4) Die Software muss in der Lage sein auf häufige Änderungen der Gesetzeslage zu reagieren. (5) Der zeitliche Anwendungsbereich von Normen muss berücksichtigt werden, sodass mehrere Fassungen eines Gesetzes unterstützt werden können. (6) Ergänzend zu den beiden vorherigen Anforderungen müssen Anpassungen der Wissensbasis möglichst rasch und effizient durchgeführt werden können.

Im Anschluss werden die in der Praxis gängigen Lösungsansätze zur Modellierung von Rechtsnormen dargestellt und einer kritischen Würdigung unterzogen.

3.3. Gängige Lösungsansätze

Die objektorientierte Programmierung (OOP) ist das heutzutage gebräuchlichste Programmierparadigma. Sprachen wie Java wurden in einer Vielzahl von Projekten – auch im öffentlichen Bereich – eingesetzt, um Expertensysteme zu implementieren. Aus diesem Grund wird im Folgenden mit der Darstellung der Probleme des objektorientierten Ansatzes bei der Implementierung von juristischen Expertensystemen begonnen. Auf die strukturierte Programmierung soll nicht näher eingegangen werden, da diese heute als veraltet gilt. Es erfolgt jedoch zuerst eine Abgrenzung zur objektorientierten Programmierung.

Zu den populärsten Programmiersprachen, die dem Paradigma der strukturierten Programmierung zugezählt werden, gehören wohl C und COBOL. Heute kommen diese Programmiersprachen in der Regel nur mehr auf Großrechnern zum Einsatz. Die strukturierte Programmierung kann als Vorgänger der objektorientierten Programmierung

[42] Für die Modellierung von Normen und Sachverhalten können auch unterschiedliche Sprachen zum Einsatz kommen.

verstanden werden, aus der sich letztere entwickelt hat. Angemerkt sei, dass der Begriff „prozedurale Programmierung“ in der Literatur unterschiedliche Bedeutungen hat. Einerseits wird dieser als Synonym für den Begriff „strukturierte Programmierung“ verstanden, für andere Autoren jedoch ist die prozedurale Programmierung, mit dem Ansatz, Algorithmen in überschaubare Teile zu zerlegen, ein Teilgebiet der strukturierten Programmierung. Dieser Meinungsstreit ist aber für die gegenständliche Betrachtung unerheblich. Sowohl die strukturierte Programmierung als auch die prozedurale Programmierung, ungeachtet ob letztere als Synonym oder Teil ersterer verstanden wird, sollen hier als Gegenstück zur objektorientierten Programmierung begriffen werden. Es soll nicht verschwiegen werden, dass moderne objektorientierte Sprachen wie Java ebenso Elemente der strukturierten Programmierung aufweisen. In Kontrollstrukturen, wie Schleifen und Bedingungen, ist die strukturierte Programmierung als Ursprung objektorientierter Sprachen erkennbar.

Bevor auf die einzelnen Ansätze im Anschluss näher eingegangen wird, sei angemerkt, dass jede Softwarelösung zur automatisierten Vollziehung (noch immer) weitgehend speziell angefertigt werden muss. Änderungen und Erweiterungen der Software, die durch gesetzliche Novellen bedingt sein können, führen daher zwangsweise zur Notwendigkeit, den Programmcode anzupassen. Dieser Umstand führt wiederum zu gesteigerten Kosten für die Erstellung und Änderung von Software. Die Effizienz derartiger Systeme könnte maßgeblich erhöht werden, würde es gelingen die Notwendigkeit Programmcode zu ändern zu beseitigen.

3.3.1. Objektorientierte Programmierung

Objektorientierte Sprachen[43] haben bereits eine langjährige Entwicklung hinter sich und wurden in einer Vielzahl von Projekten eingesetzt. Das objektorientierte Paradigma bietet mit den grundlegenden Konzepten der Datenkapselung, Polymorphie und Vererbung einen mittlerweile erprobten und bewährten Ansatz, um die Komplexität von Softwaresystemen in den Griff zu bekommen.

Dabei werden erprobte Entwurfsmuster („Patterns“) zur Lösung von Problemen angewendet, da die Objektorientierung alleine nicht auf alle Probleme eine Antwort kennt. Durch die immer stärker werdende Verbreitung des Internets auch im öffentlichen Bereich

[43] Z.B. Java von Oracle oder C# von Microsoft.

werden E-Government-Systeme heutzutage üblicherweise als Webanwendungen konzipiert. Ohne dass hier auf die technischen Details eingegangen werden soll, sind darunter Programme zu verstehen, die über ein Netzwerk („das Internet“) zugänglich sind und mit einem herkömmlichen Browser bedient werden können.

Webanwendungen ermöglichen, dass die Bürger die Anträge online, in elektronischer Form, stellen können. Aus architektonischer Sicht besteht eine Webanwendung aus mehreren „Schichten“, auch genannt „Module“, die unterschiedliche Aufgaben erfüllen und bestimmte Funktionen anbieten. Module werden durch eindeutig definierte Schnittstellen begrenzt, womit ein modularer Aufbau von Programmen erreicht wird. Die konkrete Implementierung der Schnittstelle bleibt dabei jedoch, gemäß dem Prinzip der Datenkapselung, verborgen. Änderungen der Implementierung wirken sich damit nicht auf Programmteile außerhalb des Moduls aus. Neben einer besseren Übersichtlichkeit des Programms trägt die Modularisierung so maßgeblich zur Verringerung des Wartungsaufwands bei.

Exkurs zur Architektur von Webanwendungen: Webanwendungen – aber auch Fat-Client-Anwendungen – bestehen üblicherweise aus drei aufeinander aufbauenden Schichten (Modulen), die jeweils bestimmte Aufgaben erfüllen: Die unterste Schicht, der „Data Layer“, ist für den Datenzugriff, die mittlere Schicht, der „Service Layer“, für die Implementierung der Geschäftsregeln, und schließlich die oberste Schicht, der „Web Layer“[44], für die Darstellung der Benutzeroberfläche zuständig. In der Regel werden diese drei Schichten durch übergreifende, sozusagen vertikal aus Sicht der Architektur angeordnete Module, die Aufgaben wie Authentifizierung und Autorisierung übernehmen, ergänzt.

Der „Data Layer“ übernimmt den Zugriff auf die persistierten Daten, die oftmals in einer relationalen Datenbank gespeichert werden. Immer stärkere Verbreitung finden auch die sogenannten NoSQL-Datenbanken, wie Graphendatenbanken oder Key-Value-Datenbanken, die allenfalls herkömmliche relationale Datenbanken ersetzen oder ergänzend für bestimmte Daten eingesetzt werden. Diese Schicht abstrahiert die zum Einsatz kommende

[44] Die oberste Schicht wird auch „Presentation Layer“ genannt, da sie für die Darstellung des GUI zuständig ist. Im Fall einer Webanwendung besteht das GUI aus den generierten HTML-Daten, die im Browser angezeigt werden.

Technologie zur Datenspeicherung und bietet entsprechende Zugriffsmethoden, die Objekte zur Kapselung der Daten liefern.

Auf dem Data Layer setzt der sogenannte „Service Layer“ auf. In diesem Layer wird üblicherweise die Geschäftslogik implementiert. Darunter sind sämtliche Regeln zu verstehen, die die Tätigkeit eines Unternehmens (der Behörde) steuern. Im gegenständlichen Kontext werden im „Service Layer“ insbesondere Subsumtionsregeln implementiert. Die im Anschluss aufgezeigten Probleme betreffen demnach vor allem den Service Layer von Expertensystemen.

Die oberste Schicht, der „Web Layer“, erzeugt die Benutzeroberfläche des Programms, die dann im Browser des Benutzers angezeigt wird. Technisch wird HTML-Code erzeugt, der die eigentliche Webseite beschreibt und im Browser angezeigt wird.

Kommen wir nun nach diesem Exkurs zur Architektur von Webanwendungen zurück zum eigentlichen Thema dieses Abschnitts, zur Diksussion der objektorientierten Programmierung zur Implementierung von Normen. Stellvertretend für objektorientierte Programmiersprachen werden hier die Probleme anhand von Java diskutiert. Die Ausführungen lassen sich aber weitgehend auch auf andere Vertreter des objektorientierten Ansatzes, wie C#, übertragen.

Java ist eine sogenannte „General Purpose Language“[45] und kann daher grundsätzlich zur Programmierung beliebiger Anwendungen eingesetzt werden. Der vorhandene Funktionsumfang ist groß und die Syntax ausdrucksstark. Die Syntax kennt eine Vielzahl von Kontrollstrukturen, die miteinander kombiniert werden können, um auch komplexe Abläufe zu implementieren. Die Sprache ist damit zwar nicht auf die Lösung bestimmter Probleme wie z.B. die Implementierung mathematischer Algorithmen zugeschnitten, die Syntax ist jedoch ausdrucksstark und flexibel genug, um grundsätzlich Lösungen für beliebige Problemstellungen zu implementieren.

Dies soll nun am Beispiel der automatisierten Subsumtion näher erläutert werden. Um die Anwendbarkeit einer Norm auf einen bestimmten Sachverhalt feststellen zu können, muss

[45] Das Gegenteil dazu ist eine „Domain Specific Language“, die auf einen bestimmten Problembereich („Problem Domain“) zugeschnitten ist.

der Sachverhalt zuerst in eine strukturierte Form, also in ein Objekt, übersetzt werden. Diese Strukturierung erfolgt heutzutage üblicherweise durch ein Zusammenspiel zwischen Mensch und Maschine. Der Sachbearbeiter analysiert den fraglichen Sachverhalt und gibt die relevanten Tatbestandsmerkmale in ein elektronisches Formular ein. Das Formular enthält dabei sämtliche Fakten, die zur Beurteilung notwendig sind, und unterstützt so auch den Sachbearbeiter bei der Erhebung der relevanten Umstände. Damit wird der Fall also durch den Menschen strukturiert erfasst und so einer elektronischen Subsumtion zugänglich gemacht.

Danach beginnt die eigentliche automatisierte Subsumtion. Der Sachverhalt, repräsentiert durch Objekte bzw. deren Attribute, wird mithilfe von Bedingungen analysiert. Je nachdem ob eine Bedingung erfüllt ist oder nicht, werden unterschiedliche Zweige des Programms abgearbeitet. Welche Bedingungen und Verzweigungen im Programm implementiert werden müssen, ergibt sich aus dem Tatbestand der Normen.

Der Gesetzestext muss zuerst interpretiert und anschließend in die Syntax und Semantik der jeweiligen Programmiersprache übersetzt werden. Dabei stellt sich das Problem, dass Juristen, die befähigt sind Normen zu interpretieren und zu verstehen, in der Regel nur wenig mit der technisch orientierten Syntax und Semantik einer Programmiersprache anfangen können. Umgekehrt erschließt sich die wahre Bedeutung einer Norm nicht immer aus dem Wortlaut. Techniker beherrschen üblicherweise nicht die juristische Terminologie und die Interpretationsmethoden der Rechtswissenschaft. Es bedarf also einer gründlichen Aufbereitung der rechtlichen Anforderungen, bevor mit der Implementierung eines Expertensystems begonnen werden kann. Dieser Übersetzungsprozess ist, nicht zuletzt aufgrund der Komplexität mancher Gesetze, fehleranfällig.

Das Gesetz weist naturbedingt oftmals einen hohen Grad an Unbestimmtheit[46] auf und wird in ein anderes informales Modell, den Anforderungskatalog[47], übersetzt. Basierend auf den gesammelten Anforderungen wird sodann das Expertensystem in der jeweiligen Programmiersprache implementiert. Die Aufgabe, die es demnach bei der Implementierung

[46] Hart bezeichnet diese Eigenschaft von Normen als „offene Textur“ (Hart, 2009).

[47] Abhängig von der eingesetzen Methode zur Projektentwicklung wird dieser „Lastenheft“ genannt. Bei agilen Vorgehensmodellen, wie Scrum, entfällt die Erstellung eines förmlichen Lastenhefts, das sämtliche Anforderungen möglichst detailliert enthält.

von juristischen Expertensystemen zu bewältigen gilt, ist die Übersetzung des informalen Gesetzes in ein formales Modell, das computergestützt verarbeitet werden kann.

3.3.1.1. Die Repräsentation zeitlicher Dimensionen

In diesem Abschnitt erfolgt eine überblicksmäßige Darstellung, wie mit zeitlichen Änderungen computergestützter Modelle, die vor allem durch Novellen bewirkt werden, in der Praxis der Verwaltungsautomatisierung umgegangen wird. Es wird gezeigt, wie mehrere Fassungen eines Gesetzes im Programmcode implementiert werden.

Die Ausführungen beziehen sich auf in Java implementierte Modelle von Gesetzen, wie sie in der Praxis vorherrschend sind. Nachdem sich die Art und Weise, wie Normen in der Praxis formalisiert werden, seit den Anfängen der Automatisierung in den 1970er-Jahren kaum verändert hat, sollten sich die Anmerkungen weitgehend auch auf frühere Ansätze mit COBOL oder PL/1 übertragen lassen. Java ist im Gegensatz zu den genannten Sprachen jedoch eine objektorientierte Programmiersprache.

Sämtlichen Ansätzen im Bereich der Verwaltungsautomatisierung ist gemein, dass zeitliche Aspekte direkt im Programmcode mithilfe von Bedingungen geprüft werden. Die Prüfung zeitlicher Aspekte findet damit nicht auf einer Metaebene statt, sondern ist Teil der automatisierten Anwendung des juristischen Syllogismus und damit mit der rechtlichen Beurteilung im engeren Sinn vermengt.

Es fehlt an einem geeigneten Ansatz, der auf eine solide theoretische Grundlage zur Formalisierung zeitlicher Aspekte des Rechts gestützt werden kann. Die Formalisierung zeitlicher Charakteristika von Normen findet in der Praxis gewissermaßen ad hoc, in Form von Bedingungen im Programm, statt. Besser wäre jedoch, wenn ein temporales Modell wiederverwendet werden könnte, damit Fehler bei der Formalisierung vermieden werden. Dazu ist erforderlich, dass das Modell ein adäquates Abstraktionsniveau aufweist, das nicht relevante Details der Normen ignoriert.

Bevor auf die Nachteile der in der Praxis geläufigen Modelle eingegangen wird, sei vorausgeschickt, dass es diesen meiner Ansicht nach an einer geeigneten Repräsentation der Norm fehlt. Eine solche abstrakte Manifestation einer Norm, repräsentiert durch eine formale Regel, bestehend aus Antezedens und Konsequenz, ist jedoch Voraussetzung, um zeitliche Aspekte des Modells auf einer Metaebene behandeln zu können. Von diesem

Standpunkt aus ist eine Norm als Objekt zu betrachten, an das unter anderem zeitliche Eigenschaften, wie beispielsweise das Inkrafttreten, geknüpft werden können. Diese Ausführungen lassen erkennen, dass auch im Hinblick auf die zeitlichen Dimensionen des Rechts das Prinzip des Isomorphismus nur unzureichend berücksichtigt wird.

Zeitliche Charakteristika des Rechts werden in der Praxis vom Programmcode nicht adäquat repräsentiert, und die gängigen Ansätze zur Formalisierung der temporalen Dimensionen von Normen sind mit einigen gewichtigen Nachteilen behaftet:

(1) Unzureichender Isomorphismus: Es ist nicht erkennbar, welche Regeln im Programm zu einer bestimmten Rechtslage gehören und zu welchem Zeitpunkt diese anwendbar sind. Die vom Programm produzierten Ergebnisse können daher nur schwer validiert werden.

(2) Schwierige Änderbarkeit: Aus dem ersten Problem folgt, dass die Auswirkungen von Änderungen des formalen Modells nur schwer abgeschätzt werden können. Im Fall von Novellen lässt sich kaum feststellen, welche Regeln von Änderungen betroffen sind.

(3) Fehlende Reifikation: Es fehlt an einer geeigneten Repräsentation der Normen, die es erlaubt, diese als Objekt mit bestimmten Eigenschaften zu betrachten. Ohne diese ist es nicht möglich, die temporalen Aspekte einer Norm getrennt von der eigentlichen Subsumtion zu behandeln. Die Trennung der materiellrechtlichen Beurteilung von der Behandlung von zeitlichem Bedingungs- und Rechtsfolgenbereich einer Norm ist jedoch meines Erachtens Voraussetzung, um die Wiederverwendbarkeit von temporalen Modellen steigern zu können.

Der Rest dieses Abschnitts befasst sich mit der Kategorisierung der Änderungen von Rechtsvorschriften, aus Sicht der in der Praxis vorherrschenden programmatischen Modelle von Gesetzen. Der Schwerpunkt der Darstellung wird dabei auf die Möglichkeiten und Grenzen objektorientierter Modelle gelegt, mit Novellen des Gesetzes umzugehen.

Aus der Tatsache, dass einmal kompilierter Programmcode zur Laufzeit nicht mehr geändert werden kann, ergeben sich im Hinblick auf die Möglichkeiten, das Programm an Novellen anzupassen, einige Einschränkungen.

Gänzlich ausgeschlossen ist eine Änderung der im Programm codierten Anweisungen zur Laufzeit. Im Umkehrschluss bedeutet dies, dass Normen, die direkt durch Code repräsentiert

werden, nur durch eine Änderung des Programmcodes selbst angepasst werden können. In der Praxis muss daher im Fall einer Novelle der Programmcode vom Programmierer angepasst, kompiliert, getestet und das Programm neu ausgerollt werden. Dabei werden in den Code neue Bedingungen aufgenommen, die dafür sorgen, dass vor dem Inkrafttreten der Novelle die alte und danach die neue Rechtslage vom Programm angewendet wird. Je nachdem, ob eine Novelle rückwirkend erlassen wurde, muss auch auf diesen Aspekt im Code durch entsprechende Bedingungen Bedacht genommen werden.

Um die Nachteile der statischen Natur von Programmcode zu mindern, wurden Methoden entwickelt, die es erlauben, das Programm bis zu einem gewissen Maß zu parametrisieren. Diese Externalisierung von „Programmlogik" reicht von einfachen Wertevorräten und Bedingungen bis zu ausgeklügelten Strukturen wie Entscheidungstabellen.

In vielen Gesetzen ist es Brauch, dass Geldbeträge jährlich an den Verbraucherpreisindex angepasst werden[48]. Solche Anpassungen des Gesetzes führen nicht zu einer Änderung der Struktur des formalen Modells, da weder neue Normen hinzugefügt noch bestehende entfernt werden, und seien daher als „nicht-strukturelle Modifikationen" bezeichnet.

Um Änderungen des Programmcodes möglichst zu vermeiden, werden daher Geldbeträge und numerische Werte allgemein in eine Datenbank oder in Dateien (XML etc.) ausgelagert. Die externen Wertevorräte werden üblicherweise neben einer Identifikationsnummer (ID), damit diese aufgefunden werden können, mit einem Datum versehen, ab dem diese gültig sind.

Diese Methode stößt jedoch schnell an ihre Grenzen, sobald bestehende Bedingungen geändert oder neue hinzugefügt werden müssen. In solchen Fällen ist eine Änderung der Struktur des formalen Modells erforderlich, indem in bestehende Regeln eingegriffen und neue hinzugefügt werden. Dementsprechend soll von „strukturellen Modifikationen" gesprochen werden.

Änderungen des Rechts, die zu strukturellen Modifikationen des computergestützten Modells führen, können nicht bloß durch die Auslagerung von Wertevorräten bewältigt

[48] Vgl. § 63 KOVG BGBl. Nr. 152/1957 i.d.F. BGBl. I Nr. 48/2005 und § 108f ASVG BGBl. Nr. 189/1955 i.d.F. BGBl. I Nr. 35/2012.

werden. Um möglichst flexibel auf Änderungen der Rechtslage reagieren zu können, werden analog zu numerischen Werten auch Bedingungen aus dem Programmcode ausgelagert.

Für diesen Zweck sich sogenannte „Entscheidungstabellen“ (decision tables) in der Praxis etabliert, um komplexe Entscheidungssituationen möglichst übersichtlich abzubilden. Dabei werden Regeln mit ihren Bedingungen und Aktionen in tabellarischen Strukturen abgebildet. Bedingungen und Aktionen werden dabei in die Zellen der Tabelle eingetragen. Bedingungen in diesem Sinne sind einfache Vergleiche von Zahlen, Zeichen und Datumswerten etc. Ein Programm kann die zu einer Regel gehörenden Bedingungen zur Laufzeit laden und prüfen, ob diese erfüllt sind, und gegebenenfalls die zugehörigen Aktionen ausführen. Je nach Ausgestaltung wird für jede Regel eine eigene Tabelle verwendet oder werden mehrere Regeln in einer gemeinsamen Tabelle abgebildet. Im zweiten Fall wird eine Regel üblicherweise anhand des Spaltenamens bzw. einer Identifikationsnummer (ID) identifiziert.

Mithilfe von Entscheidungstabellen kann ein Programm bis zu einem gewissen Grad feststellen, welche Regeln zu einem bestimmten Zeitpunkt anwendbar sind. Dazu werden in die Bedingungen Datumswerte aufgenommen, die den Beginn der Gültigkeit der Regeln markieren. Die Ausdrucksstärke und Flexibilität dieses Ansatzes, mit temporalen Dimensionen von Normen umzugehen, ist jedoch begrenzt. Grundsätzlich ist die Menge der Regeln nicht dynamisch veränderbar. Im Programmcode ist nämlich an entsprechender Stelle der Aufruf der Regeln vorgesehen, womit deren Existenz und Semantik dem Programm bekannt sein muss. Neue Regeln können damit nicht einfach hinzugefügt und verwendet werden.

Diese Limitierung kann durch den Einsatz mehrerer Entscheidungstabellen und ausgeklügelter Programmlogik teilweise überwunden werden. Eine Entscheidungstabelle enthält dabei das Datum, ab dem eine Regel angewendet werden soll, mit deren eindeutiger Identifikationsnummer. Mithilfe der ermittelten Identifikationsnummer wird die eigentliche Regel aus einer anderen Entscheidungstabelle selektiert, und so werden die zum jeweiligen Zeitpunkt anwendbaren Regeln ermittelt.

Angemerkt sei, dass diese Methode selbstverständlich nur dann funktioniert, wenn sämtliche Regeln dieselbe Struktur und semantisch idente Bedingungen aufweisen. Sind diese Voraussetzungen erfüllt, kann an geeigneter Stelle im Programm der soeben beschriebene, aus zwei Schritten bestehende Algorithmus zur Ermittlung der anwendbaren

Regeln verwendet werden. Noch komplizierter und mit diesem Ansatz nicht mehr sinnvoll abbildbar sind Normen, deren Bedingungs- und Rechtsfolgenbereich voneinander abweicht oder die rückwirkende Rechtswirkungen erzeugen.

Entscheidungstabellen zur Repräsentation temporaler Charakteristika sind jedoch noch einer weiteren Einschränkung unterworfen: Die Anzahl der Bedingungen und Konklusionen einer Regel und deren Semantik sind statisch. Das Programm setzt eine bestimmte Anzahl von Bedingungen und Semantik voraus, damit es die formulierten Regeln verarbeiten kann. Dies folgt aus dem Umstand, dass das Programm die jeweiligen internen Variablen mit den Werten der Bedingungen vergleichen muss. Es muss damit bereits zum Zeitpunkt der Implementierung feststehen, wie viele und welche Bedingungen zur Laufzeit in die Entscheidungstabelle ausgelagert und geprüft werden. Selbiges lässt sich im übertragenen Sinne von den Konklusionen einer Regel sagen[49].

Im Licht dieser Ausführungen wird deutlich, dass Entscheidungstabellen im Wesentlichen komplexere Anordnungen von Wertevorräten sind. Damit sind diese jedoch von ähnlichen Limitierungen wie diese betroffen. So sind die in einer Entscheidungstabelle enthaltenen Regeln mit ihren Bedingungen und Konklusionen statisch. Neue Regeln und Bedingungen müssen grundsätzlich dem Programm bekannt gemacht werden, damit diese verwendet werden können.

Zusammenfassend lässt sich feststellen, dass Entscheidungstabellen kein probates Mittel darstellen, um mit temporalen Charakteristika des Rechts umzugehen. Ein vom Bedingungsbereich abweichender Rechtsfolgenbereich sowie rückwirkend erlassene Normen können nicht angemessen repräsentiert werden, ohne die Komplexität um ein nicht hinnehmbares Maß zu steigern.

Zu der Auslagerung von Werten in externe Datenstrukturen im Allgemeinen ist zu sagen, dass dadurch das Programm schnell unübersichtlich wird. Durch eine Sichtung des Quellcodes lässt sich nicht immer feststellen, welche Operationen vom Programm zur Laufzeit ausgeführt werden. Die Verteilung von Logik auf das Programm und externe

[49] Damit unterscheiden sich „herkömmliche“ Programme von Rule Engines, da bei letzteren die Anzahl der Regeln dynamisch ist, sodass zur Laufzeit bei Bedarf neue geladen werden können. Überdies können Regeln beliebige Bedingungen enthalten und werden üblicherweise in einer speziellen Sprache modelliert, die abhängig von der jeweiligen Rule Engine variiert.

Strukturen, wie Entscheidungstabellen, führt in der Regel dazu, dass Entwickler mit unterschiedlichen Technologien umgehen müssen. Üblicherweise werden Entscheidungstabellen in Datenbanken abgebildet.

Die Verteilung der Programmlogik auf unterschiedliche Systemteile, die unter Umständen mit anderen Technologien erstellt werden, führt dazu, dass das Programm unübersichtlich wird. Es lässt sich nur schwer nachvollziehen, auf welcher Rechtsgrundlage bestimmte Anweisungen im Programm und den Entscheidungstabellen beruhen. Das Prinzip des Isomorphismus ist damit unzureichend erfüllt.

3.3.1.2. Kritische Würdigung

Im Anschluss erfolgt eine kritische Würdigung des objektorientierten Ansatzes zur Formalisierung von Rechtsnormen. Hinsichtlich der Repräsentation der zeitlichen Dimensionen des Rechts sei auf die Diskussion im vorhergehenden Abschnitt verwiesen.

Objektorientierte Programmiersprachen sind nicht für die Modellierung von Normen und die Lösung juristischer Aufgabenstellungen zugeschnitten. Juristen können die technisch orientierte Syntax und Semantik von Programmiersprachen nur schwer verstehen. Umgekehrt ist das Gesetz zwar auch für juristische Laien lesbar, die exakte Bedeutung (Semantik) von Begriffen stimmt jedoch nicht immer mit dem allgemeinen Sprachgebrauch überein.

Dieser syntaktische und semantische Graben zwischen Juristen und Softwareexperten führt dazu, dass sich Fehler bei der Implementierung von Expertensystemen einschleichen können, die auf Missverständnissen über die Bedeutung von Formulierungen beruhen. Außerdem ist es nahezu unmöglich, dass Juristen durch Sichtung des Programmcodes dessen Richtigkeit überprüfen können. Die Validierbarkeit des Modells ist daher nur unzureichend gewährleistet. Damit gehen schwierige Fragen hinsichtlich der Authentizität des Modells und der vom Programm gestützten juristischen Entscheidung einher. Das Problem der fehlenden Rechtsverbindlichkeit und des vom Programm erstellten Ergebnisses wird in der Praxis geschickt umschifft, indem das rechtlich zuständige Organ die Entscheidung legitimiert.

Objektorientierten Programmiersprachen fehlt es gemäß dieser Ausführungen weniger an Ausdrucksstärke als einem geeigneten Abstraktionsniveau zur Modellierung von Normen und Sachverhalten. Es wäre äußerst hilfreich, wenn Juristen selbst Normen und Fälle

modellieren könnten und das erstellte Modell von der Software direkt ausgeführt wird. Das niedrige Abstraktionsniveau von Programmcode verhindert, dass das vorhandene Wiederverwendungspotential ausgeschöpft werden kann. Eine weitere wichtige Voraussetzung für die Wiederverwendbarkeit der formalen Repräsentation von Normen, nämlich die Trennung der Wissensrepräsentation vom Algorithmus, wird von heutigen Systemen nicht erfüllt. Damit erhöht sich insgesamt der Aufwand für die Implementierung von Expertensystemen, da bereits gelöste Aufgabenstellungen immer wieder neu implementiert werden.

Im Programm ist oftmals nicht ersichtlich, auf welcher Rechtsgrundlage der Code beruht. Der Isomorphismus zwischen der formalen Repräsentation der Normen im Code und der Rechtsquelle ist nicht gewährleistet. Die unzureichende Berücksichtigung des Prinzips des Isomorphismus hat weitreichende Folgen. Es lässt sich nur schwer feststellen, wo ein Programm geändert werden muss, wenn die zugrunde liegenden Normen novelliert werden. Dies führt zu erheblichem Wartungsaufwand von Expertensystemen in der Praxis. Der mangelnde Isomorphismus ist, neben der unzureichenden Repräsentation temporaler Eigenschaften des Rechts, eines der Hauptprobleme von objektorientierten juristischen Expertensystemen.

Zusammengefasst sind objektorientierte Programmiersprachen sehr ausdrucksstark, weisen jedoch keine geeignete Semantik zur Modellierung von Normen auf. Es mangelt an einem geeigneten Abstraktionsniveau, sodass die Normen nicht vom Juristen selbst modelliert werden können. Der implementierte Code ist schwer zu warten und nicht wiederverwendbar. Der syntaktische und semantische Graben zwischen Softwareexperten und Juristen muss überwunden werden, um eine effiziente Modellierung von Normen zu gewährleisten (Gosh, 2011, S. 3 ff.).

3.3.2. Modellgetriebene Softwareentwicklung

Wie oben bereits ausgeführt, bieten Programmiersprachen nicht immer die passende Abstraktionsstufe um Probleme eines Fachbereichs zu beschreiben. Um die Entwicklung von objektorientierten Programmen zu unterstützen, werden eigene Sprachen und Werkzeuge zur Modellierung eingesetzt. Ziel dieses Ansatzes ist die Unterstützung der Entwicklung, indem Teile einer Software mithilfe einer grafischen Notation modelliert werden. Im

Mittelpunkt des Interesses modellgetriebener Ansätze steht daher die Trennung der technischen Abbildung und des fachlichen Modells.

Dadurch wird die Entwicklung der Software maßgeblich erleichtert, da die Experten der Domäne die Sachverhalte unabhängig von der technischen Implementierung modellieren können. Dies erleichtert die Weiterentwicklung und Wartung des Expertensystems erheblich.

Die erstellten Modelle können entweder direkt interpretiert und ausgeführt werden, oder aus diesen muss erst Code generiert werden. Als prominenter Vertreter des **generativen Ansatzes** sei die Sprache UML (Unified Modeling Language) genannt. Die Entwicklung von UML begann bereits 1995, die derzeit letzte Version ist 2.4.1[50]. Die Notation von UML bietet verschiedenste Diagrammtypen an. Exemplarisch seien hier das Klassendiagramm, mit dessen Hilfe sich die wesentlichen Teile eines Programms, nämlich Klassen und Interfaces, entwerfen lassen sowie das Sequenzdiagramm, mit dem Abläufe modelliert werden können, genannt. Für eine Einführung in die diversen Diagrammtypen sei auf Balzert (2001, S. 19 ff.) verwiesen. Aus den UML-Modellen kann mithilfe entsprechender Werkzeuge der Programmcode generiert werden. Dieser muss in der Regel an manchen Stellen noch überarbeitet und ergänzt werden, um die geforderte Funktionalität vollständig zu erfüllen. Der Vorteil dieses Ansatzes ist, dass der Programmcode einer beliebigen Sprache generiert werden kann und damit das Modell nicht an eine bestimmte Technologie gebunden ist.

Obwohl die Ausdrucksstärke von UML – insbesondere mit Version 2.0 – stetig gestiegen ist und der Grad der Formalisierung zunahm, wurde bisher keine vollständige formale Sprachbasis etabliert (Jeckle, Rupp, Zengler, Queins, & Hahn, 2004, S. 327 f.). Es bleibt jedoch fraglich, ob die Syntax von UML bereits mächtig genug ist, um Normen und komplexe juristische Sachverhalte zu modellieren. Aufgrund der in der Praxis gesammelten Erfahrungen scheint UML nur bedingt dafür geeignet zu sein[51]. Insbesondere zeitliche Charakteristika des Rechts und von Handlungen lassen sich nicht angemessen modellieren. Mit dem Anstieg der Ausdrucksstärke hat auch der Umfang der Syntax von UML

[50] Stand vom 2.1.2015. Für einen Überblick über die einzelnen Versionen siehe *Object Management Group*, Unified Modeling Language. http://www.omg.org/spec/UML/ aufgerufen: 2.1.2015 (2014).

[51] Beispielsweise haben van Engers et al. (2001) UML erweitert und zusätzlich OCL (Object Constraint Language) verwendet, um Normen für ihre Zwecke zu modellieren.

zugenommen. Es ist umfangreiches Wissen notwendig, um sämtliche Features von UML nutzen zu können. UML ist wie Java eine „General Purpose Language“, die nicht speziell für die Modellierung juristischer Aufgabenstellungen entwickelt wurde.

Ebenso gibt es Modelle, die einen **interpretativen Ansatz** verfolgen. Im Unterschied zu generativen Modellen werden interpretative zur Laufzeit von der Software eingelesen und direkt ausgeführt. Das Modell liegt somit in einer „eindeutigen“ Form vor, sodass die Generierung des Programmcodes, im Gegensatz zum generativen Ansatz, nicht erforderlich ist. Bei beiden Ansätzen wird letzten Endes das Modell vom Computer ausgeführt.

Spezielle formale Notationen, sogenannte „Domain Specific Languages“, finden sich üblicherweise nur vereinzelt für die Modellierung bestimmter Probleme. An dieser Stelle sei beispielsweise die Sprache BPMN[52] erwähnt, deren Notation die Formalisierung und grafische Darstellung von Geschäftsprozessen („Workflows“) ermöglicht. Zur Implementierung gibt es mehrere Bibliotheken, die in BPMN modellierte Workflows verarbeiten können. An dieser Stelle sei als Vertreter solcher Bibliotheken die Open-Source-Bibliothek „Activiti“ erwähnt (Rademakers, 2012).

3.3.2.1. Kritische Würdigung

UML ist eine universelle Modellierungssprache und ist als solche, ähnlich wie Java, nicht für die Modellierung von Normen und juristischen Problemen optimiert. Es mangelt an einer adäquaten Abstraktionsstufe, die es erlaubt, Normen und juristische Aufgabenstellungen präzise und zielgerichtet zu repräsentieren. Obwohl UML-Diagramme abstrakter sind als in Java geschriebener Code, sind diese dennoch nicht an den Juristen adressiert und für diesen unter Umständen nicht verständlich. Damit können diese nicht von juristischen Experten auf ihre Richtigkeit überprüft werden.

Ferner besitzt UML nur teilweise eine formale Semantik, sodass Unschärfen bei der Generierung von Programmcode aus den Modellen auftreten können (Broy et al., 2007; Liu et al., 2013). Verschiedene Interpretationen der in UML modellierten Repräsentationen der Normen sind damit zulässig. Damit können sich Abweichungen bei der Übersetzung der

[52] Business Process Model and Notation.

UML-Modelle in den Code der jeweiligen Programmiersprache einschleichen, die verhindern, dass die Semantik der Normen korrekt wiedergegeben wird.

Die Syntax von UML ist derzeit, trotz stetiger Verbesserung mit jeder Version, noch zu ausdrucksschwach, um alle notwendigen Details zu modellieren, die für die Repräsentation von Normen erforderlich sind. Mit UML lassen sich lediglich Klassen und Abläufe in Grundzügen beschreiben. In dieser Hinsicht bleibt UML jedenfalls hinter den Möglichkeiten von Java zurück. Der Vorteil von UML, als eine universelle Sprache zur Modellierung, gegenüber Java ist, dass die in UML erstellten Modelle grundsätzlich unabhängig von einer bestimmten Programmiersprache sind. Dieser Vorteil von UML vermag im gegenständlichen Kontext angesichts der evidenten Schwächen, jedoch nicht zu überzeugen.

Fehlende formale Semantik und mangelnde Ausdrucksstärke verbieten es, dass Normen mit ausreichender Präzision modelliert werden. Weiters scheint die Beschreibung komplexer Strukturen, wie von juristischen Argumenten und Wertungen, mit UML nicht möglich. Überdies ist die Modellierung der zeitlichen Dimensionen des Rechts mit UML kaum vorstellbar, ohne eine nicht hinnehmbare Komplexität des Modells in Kauf nehmen zu müssen.

Sprachen zur Erstellung interpretativer Modelle von Normen, wie LegalRuleML (Athan et al., 2013b), sind derzeit noch experimentell und nicht weit verbreitet. Mithilfe solch spezieller Sprachen wäre es möglich, Modelle des Gesetzes zu erstellen, die eine formale Semantik aufweisen und direkt ausgeführt werden können.

Zusammengefasst weist die modellgetriebene Softwareentwicklung folgende Schwächen im Hinblick auf die Modellierung von Normen auf: (1) Zu niedrige Abstraktionsstufe und keine geeignete Syntax[53], (2) nur teilweise formale Semantik, (3) fehlende Ausdrucksstärke und (4) zeitliche Dimensionen können nicht adäquat repräsentiert werden.

3.4. Diskussion

Dieses Kapitel befasste sich mit der Darstellung der in der Praxis der Verwaltungsautomatisierung gängigen Ansätze, nämlich der objektorientierten Programmierung und der modellgetriebenen Softwareentwicklung, die einer kritischen

[53] Dieses Problem haftet auch dem objektorientierten Ansatz an.

Würdigung im Hinblick auf die Formalisierung von Normen unterzogen wurden. Beide Ansätze weisen schwerwiegende Nachteile auf, die zu hohen Kosten für die Entwicklung und Wartung von juristischen Expertensystemen führen.

In den nachfolgenden Kapiteln wird auf die Herausforderungen der Formalisierung und Wissensrepräsentation im rechtlichen Bereich eingegangen. Diese Ausführungen bilden die Grundlage für einen experimentellen Ansatz zur Formalisierung, der ein ontologisches Modell zur Repräsentation der Normen einsetzt.

Die Untersuchung der Formalisierung von Normen in einer Ontologie soll die Grenzen dieses Ansatzes ausloten. Die Ontologie dient dabei als formales und direkt ausführbares Modell des Gesetzes. Exemplarisch wird das KOVG zur Formalisierung herangezogen. Die Ontologie des KOVG soll dabei auf der im Projekt Estrella[54] entwickelten LKIF-Core Ontology aufbauen (Breuker et al., 2007).

Eine Ontologie besteht im Wesentlichen aus Klassen (Begriffen), Instanzen dieser Klassen und Beziehungen zwischen diesen. Vorteilhaft ist, dass die Semantik dieser ontologischen Komponenten durch einen logischen Formalismus[55] eindeutig definiert ist. Der Aufbau von Axiomen der Ontologie ist an die natürliche Sprache angelehnt, wodurch es möglich sein sollte, die Terminologie und Struktur des Gesetzes möglichst direkt in die Ontologie zu übernehmen.

Durch Abstraktion sollen aus der normativen Grundlage Kategorien von Begriffen gebildet und in der Ontologie modelliert werden. Für die Modellierung der Ontologie ist die Interpretation der Normen erforderlich, um deren Bedeutung zu ermitteln. Zur besseren Strukturierung des ontologischen Modells wird dieses zum einen aus einer Ontologie bestehen, die die Begrifflichkeit abbildet, und zum anderen aus einer zweiten, auf dieser aufsetzenden Ontologie, die Beschreibungen von Fällen und Normen enthält.

[54] *Estrella*, LKIF-Core Ontology. http://www.estrellaproject.org/?page_id=3 aufgerufen: 10.1.2013.

[55] In diesem Fall OWL 2.

Die Untersuchung wird zeigen, ob Ontologien geeignet sind, einfache Fälle[56] zu lösen und inwiefern diese juristische Expertensysteme verbessern können.

[56] Damit sind Fälle gemeint, in denen die Fakten eindeutig unter den Tatbestand einer Norm subsumiert werden können. Solche Fälle können weitgehend „mechanisch“ beurteilt werden, indem die Regeln auf die Fakten angewendet und die Rechtsfolgen deduktiv abgeleitet werden.

4. Die Formalisierung des Rechts

4.1. Einleitung

Erste Überlegungen und Aufsätze zur „Formalisierung des Rechts“ gab es bereits in den 1970er-Jahren (Morscher, 1988; Reisinger, 1975). Trotz dieser frühen Forschungen und der beachtlichen Entwicklungen auf dem Gebiet „Artificial Intelligence and Law“ (Bench-Capon et al., 2012) sind Maschinen noch immer nicht in der Lage, das Recht wie Menschen zu verstehen (Franklin, 2012; Prakken & Sartor, 1998, 2007, 2013; Prakken, 2008; Wyner, Bench-Capon, & Atkinson, 2011).

Bis heute fehlt ein einheitlicher Standard für die Formalisierung von Normen. Aufgrund der Verschiedenheiten der Rechtsordnungen und der strukturellen Unterschiede der einzelnen Rechtsgebiete scheint ein solcher zunächst in weiter Ferne. Dennoch haben sich in jüngster Zeit Anstrengungen verdichtet, einen gemeinsamen anerkannten Standard zur Modellierung von Normen zu etablieren, was unter der Schirmherrschaft des OASIS Konsortiums zur Entwicklung von LegalRuleML geführt hat (Athan et al., 2013b; Palmirani, Governatori, Rotolo, et al., 2011). Die praktische Nützlichkeit von LegalRuleML bei der Entwicklung von Anwendungen im öffentlichen Sektor wird sich erst zeigen. Es fehlt jedenfalls an einer einheitlichen Methode und an Werkzeugen zur Entwicklung formaler Modelle des Rechts.

Dem stetigen technologischen Fortschritt und der sprunghaft ansteigenden Leistungsfähigkeit von Computersystemen zum Trotz gibt es auch heute noch bei der Formalisierung von Normen und der darauf aufbauenden Automatisierung Hindernisse, die noch später genauer dargestellt werden. Vorweg sei erwähnt, dass diese Probleme aus den Besonderheiten des Rechts, insbesondere der Struktur der Rechtsnormen und der Komplexität sprachlicher Formulierungen, herrühren (Franklin, 2012; Morscher, 1988, S. 116 ff.; Reisinger, 1975, S. 42 ff.).

Die Besonderheit des Rechts spiegelt sich deutlich in der juristischen Arbeitsweise wider, wie insbesondere die Notwendigkeit der Interpretation von Normen und die Auflösung von Normkonflikten durch Anwendung bekannter Grundsätze wie „lex spexialis derogat legi generali“ zeigen. Es scheint daher sinnvoll, sich den Grenzen der Formalisierung über eine theoretische Darstellung der normativen Grundlage und der juristischen Arbeitsweise zu nähern.

Die Formalisierung von Normen ist zentrales Thema der Dissertation, wobei der Fokus der Untersuchung auf das Verwaltungsrecht gerichtet ist. Selbstverständlich würde eine Untersuchung aller Rechtsvorschriften, die dem Verwaltungsrecht zuzuordnen sind, den Rahmen der Dissertation bei weitem sprengen. Exemplarisch wird daher das Kriegsopferversorgungsgesetz, eine Sondermaterie des besonderen Verwaltungsrechts, für die Formalisierung herangezogen. Dieses weist einen eher mathematischen Inhalt auf, die Ergebnisse der Untersuchung sollten sich daher auf das Steuer- und Pensionsrecht etc. übertragen lassen, da diese Rechtsmaterien ebenso zahlreiche Berechnungen enthalten.

Der erste Teil dieses Kapitel beschäftigt sich daher mit der Frage, was unter dem Begriff der „Formalisierung“ in diesem Kontext zu verstehen ist?

Um ein grundlegendes Verständnis der Problematik zu schaffen und weil die in der Literatur verwendete Nomenklatur nicht einheitlich ist, soll zuerst der Begriff der „Formalisierung“ näher bestimmt werden. Es wird dabei gezeigt, dass der Dissertation ein differenziertes Verständnis vom ursprünglich stark logikorientierten Begriff der „Formalisierung“ zugrunde liegt. Überdies gilt es, die Formalisierung insbesondere von Axiomatisierung und Automatisierung abzugrenzen.

Diese eingängliche Begriffsbestimmung der Formalisierung ist die Grundlage für die weiteren Ausführungen hinsichtlich der Axiomatisierung und die anschließenden Erläuterungen über die Formalisierung im Kontext der Dissertation.

Der zweite Teil des Kapitels beschäftigt sich mit den Grundlagen der Rechtsanwendung und versucht die Frage zu beantworten, welche prinzipiellen Grenzen der Formalisierung entgegenstehen? Dabei wird insbesondere auf die Probleme eingegangen, die es bei der juristischen Subsumtion zu lösen gilt. Dieses Problemverständnis dient als Grundlage für die anschließenden Ausführungen über die Hindernisse, die der Formalisierung von Rechtsnormen entgegenstehen. Abschließend werden Lösungsansätze für die identifizierten Hindernisse präsentiert.

4.2. Vorüberlegungen

Man könnte die Frage stellen, welchen Zweck und Nutzen rechtstheoretische Überlegungen im Kontext der Formalisierung und Automatisierung überhaupt haben?

Wie noch gezeigt wird, werden die Normen durch die Formalisierung auf eine rein syntaktische Dimension reduziert, die versucht, die wesentlichen Eigenschaften zu repräsentieren (Fiedler, 2004; Morscher, 1988, S. 96, 106 f.). Es ist daher notwendig, zuerst das theoretische Fundament der Formalisierung und Automatisierung darzustellen, um die damit verbundenen Probleme verstehen zu können. Zu diesen grundlegenden Ausführungen erscheint es ebenso zweckmäßig, die juristische Arbeitsweise für ein weitergehendes Problemverständnis zu behandeln. Die Maschine bzw. die Software muss nämlich die juristische Entscheidungsfindung in gewisser Weise nachahmen, um zu einer korrekten Entscheidung zu gelangen. Ohne diese Ausführungen könnte nicht sinnvoll auf die Probleme der Formalisierung, insbesondere hinsichtlich vager Gesetzesbegriffe, eingegangen werden.

Die derzeitigen Expertensysteme zur Unterstützung von Verwaltungsverfahren weisen einige Unzulänglichkeiten auf. Der ständig wachsenden Komplexität solcher Systeme begegnet man in der Praxis ausschließlich mit technischen Mitteln, wie serviceorientierten Architekturen.

Eine eingehende theoretische Aufarbeitung des Themas ist meines Erachtens unerlässlich, wenn in Zukunft die Schwächen derzeitiger Systeme ausgemerzt werden sollen. Ein „optimaler“ Ansatz für die Entwicklung von Systemen zur automatisierten Vollziehung muss bereits im Vorfeld, bei der Konzeption, ansetzen. So müssen unter anderem die Struktur der Rechtsnormen sowie die „Metaregeln“, wie zum Beispiel der Geltungszeitraum von Rechtsvorschriften, berücksichtigt werden. Der Programmcode muss erkennen lassen, auf welcher Rechtsgrundlage dieser basiert (Bench-Capon & Coenen, 1992). Normen, Sachverhalte und Verfahren müssen auf einer höheren Abstraktionsebene modelliert werden als dies mit bisherigen Ansätzen möglich ist.

4.3. Begriffsbestimmung

Die folgenden Abschnitte dienen dem Versuch einer Begriffsbestimmung der „Formalisierung“. Diese Ausführungen erscheinen zwecks Schaffung einer Grundlage für die Dissertation unerlässlich. Die in der Literatur vorhandenen Interpretationen des Begriffs „Formalisierung“ sind nämlich keineswegs einheitlich (Fiedler, 2004; Morscher, 1988, S. 86 ff.).

4.3.1. Die Formalisierung im engeren Sinn

Der Begriff „Formalisierung“ ist selbst mehrdeutig und wird in der Literatur unterschiedlich verwendet. Nach Morscher kann unter „Formalisierung“ eine bestimmte Methode, die (erfolgreiche) Anwendung einer Methode oder deren Ergebnis verstanden werden (Morscher, 1988, S. 107). Mit dieser Feststellung ist noch nicht viel gewonnen. Es bedarf weiterer Ausführungen zur Klarstellung. Reisinger analysiert verschiedene Auffassungen der Formalisierung in der Literatur und kommt zu dem Ergebnis, dass unter „Formalisierung“ regelmäßig „die Darstellung eines rechtlich relevanten Bedeutungsinhalts in ‚Formeln‘, dh in formallogischen und/oder mathematischen Ausdrücken“ verstanden wird (Reisinger, 1975, S. 24).

Diese noch recht vage Definition versucht Reisinger durch einen Umweg über den *Kalkülbegriff* näher zu definieren. Unter einem Kalkül versteht man allgemein ein System zur Herstellung von Figuren aus gewissen Ausgangspositionen, den Grundfiguren, nach bestimmten Herstellungsvorschriften, den Grundregeln (Ritter, 1976, S. 672). Reisinger geht davon aus, dass ein formalwissenschaftlicher Kalkül durch die Mengen M_0, M_1, M_2 und M_3 beschrieben wird (Reisinger, 1975, S. 24).

Die Menge M_0 repräsentiert die Grundsymbole oder Mengenzeichen des Kalküls. Die Menge M_1 besteht aus Kalkül- oder Zeichenreihen, die durch eine endliche Reihung von Symbolen aus M_0 entstehen. Aus der Menge M_1 wird die Teilmenge M_2 durch bestimmte strukturelle Bedingungen ausgesondert. Damit wird der Begriff des Kalkülausdrucks definiert. Aus der Menge der Kalkülausdrücke M_2 wird schließlich die Teilmenge M_3, die sogenannten Kalkülaussagen oder Kalkülsätze, ausgesondert.

Für die Aussonderung der Kalkülaussagen (M_3) aus den Kalkülausdrücken (M_2) gibt es nach Reisinger zwei Methoden (Reisinger, 1975, S. 24 ff.):

(1) Die semantische Methode geht von der Allgemeingültigkeit logischer Ausdrücke aus. Ein logischer Ausdruck ist allgemeingültig, wenn er bei jeder beliebigen Interpretation seiner Prädikate und für beliebige Individuenbereiche eine wahre Aussage ergibt. Eine solche Aussageform heißt Tautologie (Zoglauer, 2008, S. 47).

(2) Andererseits kann man auch von Prinzipien der logischen Syntax ausgehen. Dazu ist es notwendig, spezielle Ausdrücke, die Axiome, zu definieren. Aus diesen Axiomen werden

dann mithilfe der Ableitungsregeln die weiteren Sätze (Theoreme) gewonnen. Diese Ableitungsrelation kann rein syntaktischer Natur sein, d.h. es erfolgt keine Interpretation der Zeichen. Überdies sind jedoch auch semantische Ableitungsrelationen möglich, die im Rahmen einer Interpretation des Kalküls definiert werden (Reisinger, 1975, S. 25). Für weitere Erläuterungen zur *semantischen* und *syntaktischen Wahrheit*, sei auf die Darstellung von Zoglauer (2008, S. 107 f.) verwiesen.

Nach Auffassung Reisingers lässt sich die Formalisierung als die „Beschreibung eines substanzwissenschaftlichen Objektbereichs mittels eines Kalküls“ definieren (Reisinger, 1975, S. 26):

„Ein bestimmter Objektbereich werde durch eine nichtformalisierte Sprache beschrieben. Formalisierung bedeutet dann die Ersetzung der Aussagen der nichtformalisierten Sprache durch Aussagen des Kalküls, wobei als Interpretation der formalisierten Sprache der genannte Objektbereich zugrunde gelegt wird.“

Reisinger stellt also auf die Verwendung des Begriffs „Formalisierung“ als eine Methode ab, lässt aber weitgehend offen, welche Objekte Gegenstand der Formalisierung sein können und wie die angesprochene Ersetzung zur erfolgen hat.

Die soeben getroffene Definition bezeichnet die „Formalisierung im engeren Sinn“. Diese ist von der „Formalisierung im weiteren Sinn“ zu unterscheiden, die sogleich näher dargestellt wird.

4.3.2. Die Formalisierung im weiteren Sinn

Die folgenden Abschnitte dienen der Abgrenzung der „Formalisierung im engeren Sinn“ von der „Formalisierung im weiteren Sinn“.

4.3.2.1. Symbolisierung

Unter „Symbolisierung“ wird die bloße Übersetzung eines Zeichensystems, z.B. des Textes einer natürlichen Sprache, in entsprechende Symbole einer anderen Sprache verstanden. Diese Symbole sind in der Regel einfacher, die vorgegebene Reihenfolge der Elemente im Text wird jedoch nicht verändert. Das Resultat der Symbolisierung ist im Wesentlichen also eine Art „Kurzschrift“.

Durch die Formalisierung i.e.S. kommt es häufig auch zu einer Übersetzung in Symbole der jeweiligen Sprache, womit durch diese auch eine Symbolisierung stattfindet. Die Symbolisierung darf aber keinesfalls die logische Struktur des ursprünglichen Textes verdecken (Reisinger, 1975, S. 26 f.).

In der Literatur werden allerdings „Symbolisierung“ und „Formalisierung“ gelegentlich synonym verwendet.

4.3.2.2. Axiomatisierung

Ein „Axiom“ ist ein Grundsatz einer Theorie, der mit den anderen Axiomen an den Anfang der Theorie gestellt wird. Axiome werden nicht aus anderen Sätzen der Theorie abgeleitet oder bewiesen. Sie stellen also gleichsam die Grundlage der Theorie dar. Aus den Axiomen können die weiteren Aussagen der Theorie, die Theoreme, deduktiv abgeleitet werden (Opalek & Wolenski, 1975, S. 51 f.; Reisinger, 1975, S. 26; Zoglauer, 2008, S. 102 ff.).

Eine Axiomatisierung soll demnach zwischen Aussagen oder Normen einen Ableitungszusammenhang herstellen. Für Reisinger stellt eine Axiomatisierung lediglich eine spezielle Form der Definition gültiger Kalkülsätze dar. Weiter wird betont, dass einer Axiomatisierung nicht auch unbedingt eine Formalisierung vorhergehen muss. Letztere ist jedoch ein vorzügliches Mittel, um die Axiomatisierung voll zur Geltung kommen zu lassen. Reisinger merkt an, dass daher die im Schrifttum häufig geäußerten Bedenken gegenüber einer Axiomatisierung nicht zwingend auch für eine Formalisierung gelten müssen (Reisinger, 1975, S. 26)[57].

4.3.2.3. Algorithmisierung

Unter einem Algorithmus versteht man ein schematisches Verfahren zur Lösung bestimmter Klassen von Aufgaben, wobei jeder einzelne Schritt dieses Verfahrens genau definiert ist (Reisinger, 1975, S. 27).

Die Formalisierung i.e.S ist eine notwendige Voraussetzung, damit rechtliche Probleme durch ein Computerprogramm mithilfe eines entsprechenden Algorithmus gelöst werden

[57] Zu den Bedenken hinsichtlich einer Axiomatisierung siehe Opalek und Wolenski (1975, S. 58 ff.) und insbes. die Ausführungen in Kapitel 4.4.2.

können. Der Algorithmus kann in diesem Sinne als eine Problemlösungsstrategie verstanden werden.

Unter „Algorithmisierung“ kann man in diesem Zusammenhang daher die Entwicklung eines entsprechenden Algorithmus zur Lösung rechtlicher Probleme verstehen.

4.3.2.4. Automatisierung

Die letzte Phase, die Automatisierung, ist das eigentliche Ziel der Bemühungen um eine Formalisierung rechtlicher Probleme. Die vorhergehenden Phasen dienen daher als Vorbereitung für die Automatisierung. Eine Software soll dabei, basierend auf dem erstellten formalen Modell, durch Anwendung der entwickelten Algorithmen rechtliche Problemstellungen behandeln und eine Lösung finden (Reisinger, 1975, S. 27 f.).

4.3.3. Abschließende Bemerkungen

Durch die Formalisierung i.e.S. erfolgt eine Reduktion der Rechtsnormen auf eine rein syntaktische Ebene (Morscher, 1988, S. 96). Für die automatisierte Rechtsanwendung ist jedoch von größter Bedeutung, dass ein logischer Ausdruck nur dann „wahr“ ist, wenn er den Tatsachen entspricht. Aus diesem Grund ist jedenfalls erforderlich, dass dem logischen System eine Interpretation zugrunde gelegt wird, siehe insbesondere Morscher (1988, S. 100 f.) und zur *semantischen Wahrheitstheorie* Zoglauer (2008, S. 28 ff.).

Abschließend sei noch auf den ebenfalls logischen Ansatz von Morscher verwiesen, der seine Darstellung danach gliedert, welches „Ausgangsobjekt“ formalisiert werden soll, und das gewünschte Resultat der jeweiligen Formalisierung beschreibt (Morscher, 1988).

4.4. Die Axiomatisierung

4.4.1. Die Axiomatisierung als theoretischer Ansatz

Bevor die Axiomatisierung als Methode einer kritischen Betrachtung unterzogen werden kann, bedarf es einiger grundlegender Ausführungen zur Axiomatisierung selbst, mit denen sich dieser Abschnitt beschäftigt.

Vom theoretischen Standpunkt her erscheint es angebracht darauf hinzuweisen, dass die Anwendung einer axiomatischen Methode auf rechtliche Probleme eine zweifache sein kann. Einerseits kann eine Axiomatisierung der Rechtswissenschaft oder eines ihrer Teile und andererseits die Axiomatisierung des Rechts selbst angestrebt werden.

Eine Axiomatisierung ist immer, unabhängig ob man das Recht oder die Rechtswissenschaft selbst axiomatisieren will, ein theoretisches Unterfangen. So formuliert Gordon bezüglich der Axiomatisierung eins Rechtsgebiets treffend (Gordon, 1994a, S. 25):

„Briefly, the idea of axiomatization is to represent a theory of the law of some domain, such as contracts or tort law, as a finite set of propositions."

Gordon rekurriert im zitierten Artikel auf das Werk von Rödig und kritisiert den von Klug in den 50ern entwickelten Ansatz zur Axiomatisierung, der auf Prädikatenlogik erster Stufe basiert (Rödig, 1980)[58].

Ferrajoli sei an dieser Stelle als Vertreter der Axiomatisierung der Rechtswissenschaft selbst genannt. Im ersten Teil seines Werks *Principia iuris*, dessen Fertigstellung vierzig Jahre gedauert hat, entwickelt er eine „axiomatische Theorie des Rechts" (Ferrajoli, 2007). Die Grundzüge dieser Theorie stellt Sandro in seinem Aufsatz dar (Sandro, 2011).

An dieser Stelle sei nur so viel angemerkt, dass Ferrajoli die Axiomatisierung eines Rechtssystems für undurchführbar hält. Paradoxerweise sieht Ferrajoli gerade darin den Grund warum, die Rechtstheorie axiomatisiert werden sollte. Seiner Ansicht nach ist eine allgemeine Theorie des Rechts ein Diskurs, durch den normative Aussagen mittels Deduktion gewonnen werden können. Positives Recht hingegen ist immer ein Produkt eines Rechtssetzungsaktes einer dazu berufenen Autorität und kann daher niemals abgeleitet, sondern nur produziert werden. Diese Trennung ist wesentlich für die Frage der Axiomatisierbarkeit der Rechtstheorie, und Ferrajolis Werk beschäftigt sich im Kern exakt mit diesem erkenntnistheoretischen Unterschied (Sandro, 2011, S. 344 f.).

Die folgenden Ausführungen beschäftigen sich jedenfalls mit der Axiomatisierung des Rechts[59] selbst und nicht der Rechtswissenschaft.

Was steckt nun hinter der Idee der Axiomatisierung? Anknüpfend an das oben stehende Zitat von Gordon wird zuerst eine Theorie eines Gesetzes durch eine begrenzte Anzahl von *Axiomen*, den Grundsätzen, repräsentiert. Von diesen Axiomen werden schließlich weitere Aussagen, die *Theoreme*, deduktiv abgeleitet. Die Axiomatisierung einer Theorie ist nur dann

[58] Zu der von Gordon an Rödigs Werk geäußerten Kritik siehe Kapitel 4.4.2.5.

[59] Mit „Recht" ist in diesem Abschnitt ein Rechtssystem gemeint und nicht die Rechtswissenschaft selbst.

korrekt, wenn alle auf diese Art gewonnenen Theoreme Bestandteile der repräsentierten Theorie sind. Sie ist nur dann *vollständig*, wenn jede Aussage der Theorie auch ein Theorem ist (Gordon, 1994a, S. 25).

Allgemein verlangt man von einem Axiomensystem, dass es vollständig, widerspruchsfrei und unabhängig ist. Es ist *vollständig*, wenn alle allgemeingültigen Sätze der Theorie aus ihm ableitbar sind. *Widerspruchsfrei*, wenn aus den Axiomen keine Widersprüche abgeleitet werden können, und schließlich *unabhängig*, wenn es nicht möglich ist, ein Axiom (oder dessen Negation) aus den anderen Axiomen abzuleiten (Zoglauer, 2008, S. 103).

Die Axiomatisierung kann einerseits ein Mittel zur Systematisierung einer bereits existierenden Theorie sein oder andererseits kann eine neue Theorie von Anfang an auf axiomatische Weise aufgebaut werden. Diese Unterscheidung scheint praktisch nicht von großer Relevanz zu sein, da die Axiome immer auf gewisse Kenntnisse, die den Anfang einer Theorie repräsentieren, gestützt sind (Opalek & Wolenski, 1975, S. 53).

Das Postulat der *Axiomatisierung des Rechts* steht eng in Verbindung mit dem Streben nach Modernisierung. Die Rechtssprache soll dabei in eine symbolische Sprache übersetzt und das Rechtssystem als ein axiomatisches System dargestellt werden. Damit soll die Kohärenz des Rechtssystems gewährleistet und es Maschinen ermöglicht werden, aus allgemeinen Vorschriften konkrete und umgekehrt abzuleiten. Das Streben nach Axiomatisierung des Rechts ist jedoch keinesfalls eine neue Erscheinung, sondern war bereits deutlich in den naturrechtlichen Doktrinen erkennbar. Den naturrechtlichen Ansätzen ist gemein, dass sie das Recht auf allgemeine, unveränderliche Grundsätze zurückführen (Opalek & Wolenski, 1975, S. 54 f.).

Aber auch in anderen Disziplinen wurde schon früh versucht eine Axiomatisierung durchzuführen. So hat beispielsweise Spinoza seine Ethik im 17. Jahrhundert *more geometrico* nach Art der Mathematik verfasst (Spinoza, 1976). Das eben zitierte Werk ist ein gutes Beispiel dafür, dass eine Axiomatisierung auch unabhängig von einer Formalisierung durchgeführt werden kann (Reisinger, 1975, S. 26). Spinoza bediente sich der lateinischen Sprache zur Definition der Axiome, auf die er seine Theorie stützte.

4.4.2. Kritik an der Axiomatisierung

Obwohl es mehrere Analogien[60] zwischen dem Rechtssystem und dem axiomatischen System zu geben scheint (Opalek & Wolenski, 1975, S. 55 ff.), gab es schon früh beachtliche Kritik an diesem Ansatz, die im Folgenden dargestellt werden soll.

4.4.2.1. Normen als Objekte der Axiomatisierung

Die axiomatische Methode wurde ursprünglich in der Mathematik angewendet. Im Gegensatz zum Recht sind dort bereits logische Sätze das Objekt der Axiomatisierung. Um zu einer Lösung dieses Problems zu gelangen, gibt es grundsätzlich drei Ansätze (Opalek & Wolenski, 1975, S. 58 f.):

Erstens kann man annehmen, dass es sich bei Normen bereits um Sätze im logischen Sinn handelt. Dieser Auffassung muss entgegengehalten werden, dass Normen häufig einen solchen Inhalt besitzen, der *nicht* zulässt, dass die Frage, ob ein konkreter Sachverhalt unter den Tatbestand zu subsumieren ist, eindeutig mit „ja“ oder „nein“ beantwortet werden kann[61]. Normen unterscheiden sich damit maßgeblich von logischen Aussagen, die immer entweder „wahr“ oder „falsch“ sind.

Zweitens kann man an der Möglichkeit der Konstruktion einer „echten“ Normenlogik festhalten, die dieselbe Funktion wie die Logik der Sätze erfüllen müsste[62]. Obwohl es immer schon Bemühungen auf diesem Gebiet gab, blieb der Durchbruch einer Normenlogik bisher aus. Seit jeher gibt es Zweifel, ob es eine „Logik der Normen“ überhaupt gibt (Cornides, 1975).

Obwohl die praktische Relevanz einer Normenlogik fraglich ist, gibt es auch noch in jüngster Zeit Stimmen, die für die Notwendigkeit einer Normenlogik einstehen. Weinberger zeichnet

---

[60] Man denke hier beispielsweise an den Grundsatz „pacta sunt servanda“, der nicht expressis verbis im Gesetz zum Ausdruck kommt und doch unbestritten Teil des Rechtssystems ist.

[61] Siehe dazu insbes. die Ausführungen im Kapitel 4.8. .

[62] Opalek und Wolenski unterscheiden eine solche offenkundig von der „deontischen Logik“.

beispielsweise ein Konzept für eine „echte“ Normenlogik[63], sozusagen als eine Weiterentwicklung der deontischen Logik[64] (Weinberger, 2006).

Drittens kann man davon ausgehen, dass der Inhalt der Normen sich mithilfe von deontischen Sätzen abbilden lässt. Man nimmt also an, dass die deontischen Sätze den Rechtsinhalt adäquat wiedergeben (Opalek & Wolenski, 1975, S. 59). Dieser Ansatz erscheint vielversprechend, da – im Gegensatz zur echten Normenlogik – eine deontische Logik bereits entwickelt wurde (Zoglauer, 2008, S. 135 ff.).

An dieser Stelle scheint es angebracht darauf hinzuweisen, dass in der Literatur der Begriff „Normenlogik“ oft synonym mit „deontischer Logik“ (Zoglauer, 2008, S. 135 ff.) bzw. die Normenlogik als Überbegriff (Weinberger, 2006, S. 160), der die deontische Logik einschließt, verwendet wird. Daher wurde hier auch explizit darauf hingewiesen, dass die oben genannten Autoren ein differenziertes Verständnis von einer „echten“ Normenlogik im Gegensatz zur deontischen Logik haben.

4.4.2.2. Der logische Status juristischer Folgerungen

Juristen entwickeln das Rechtssystem durch Anwendung verschiedener Methoden, die oftmals als deduktiv erachtet werden. Es wird jedoch auch die Meinung vertreten, dass die Regeln zur Schlussfolgerung, die bei der Entscheidungsfindung angewendet werden, einen für das juristische Denken spezifischen – nicht deduktiven – Charakter haben. Diese Meinung kann auf zwei Gründe zurückgeführt werden (Opalek & Wolenski, 1975, S. 58 ff.).

Der erste Grund ist, dass, wie bereits oben ausgeführt wurde, man die Ansicht vertreten kann, Normen unterscheiden sich von logischen Sätzen. Damit verleiht man der juristischen Schlussfolgerung einen spezifischen Charakter. Dieser Ansicht kann nicht gefolgt werden, schließt man sich der im letzten Abschnitt zuletzt dargestellten Meinung an, dass Normen durch deontische Sätze adäquat wiedergegeben werden können.

Der zweite, meines Erachtens schwerwiegendere, Grund ist die Tatsache, dass die Struktur juristischer Folgerungen, wie z.B. des „argumentum a fortiori“[65], nicht immer durch logische

---

[63] Zu den philosophischen Grundfragen, die von der deontischen Logik und der echten Normenlogik unterschiedlich beantwortet werden, siehe Weinberger (2006, S. 166).

[64] Es geht bei der deontischen Logik um die Darstellung von Geboten, Erlaubnissen und Verboten (D. Gabbay, Horty, Parent, van der Meyden, & van der Torre, 2013).

Schlüsse adäquat wiedergegeben werden kann. Die Annahme, die juristische Schlussfolgerung folge in jedem Fall den Regeln der formalen Logik und sei damit deduktiv, erscheint verfehlt. Juristen bedienen sich vielmehr zusätzlich auch anderer Schemata zur Lösung von Normkonflikten, die an dieser Stelle nicht weiter verbreitert werden sollen. Ein Festhalten an der ausschließlichen Anwendung formaler Logik zur Durchführung juristischer Schlussfolgerung führt in vielen Fällen unweigerlich zu intuitionswidrigen Resultaten (Opalek & Wolenski, 1975, S. 60, 64 f.).

Abschließend sei darauf hingewiesen, dass einen wesentlichen Teil der Rechtsanwendung die Auflösung von Normkonflikten ausmacht (Koller, 1997, S. 193 ff.). Als Beispiel sei hier die – zumindest unter Juristen – allgemein bekannte Regel „lex spexialis derogat legi generali" angeführt, wonach bei Normkonkurrenzen die speziellere die allgemeinere Norm verdrängt. Um zu einer Entscheidung zu gelangen reicht es nicht aus, die Fakten unter eine Norm zu subsumieren, sondern es müssen auch rechtliche Metaregeln berücksichtigt werden. Gemeint sind damit Regeln, die nicht direkt im Rechtstext zum Ausdruck kommen, jedoch implizit bei der Subsumtion angewendet werden müssen (Opalek & Wolenski, 1975, S. 60).

Die Unterscheidung zwischen Metaregeln des Rechts und dem Inhalt von Rechtsnormen selbst kann durch einen logischen Formalismus nur unzureichend ausgedrückt werden. Zwar kann mithilfe der Logik bis zu einem gewissen Grad beschrieben werden, welche Normen Ausnahmebedingungen für andere Normen sind, jedoch kommt es unweigerlich zu einer Vermischung von Metaregeln und Objektregeln (dem normativen Inhalt). Eine solche Vermengung ist tunlichst zu vermeiden, da sie die Komplexität der logischen Sätze unweigerlich erhöht und den Aufbau des Rechts nicht adäquat wiedergibt (Breuker et al., 2007, S. 120).

4.4.2.3. Die Unabhängigkeit des Rechtssystems

Für die weiteren Überlegungen ist zunächst die eindeutige Definition des „Axiomatisierens" erforderlich. Streng mathematisch definiert bedeutet das Axiomatisieren der Klasse D, eine solche Klasse D_0, in D enthalten zu finden, dass $D = Cn(D_0)$ (Opalek & Wolenski, 1975, S. 61).

---

[65] Man unterscheidet hier die beiden Größenschlüsse „argumentum a maiore ad minus" und „argumentum a minore ad maius" (Bydlinski, 2005, S. 63 ff.).

Gemäß den Autoren soll die Axiomatisierung der Klasse D sozusagen aus ihrer Reduktion zur Klasse, die ihren fundamentalen Teil bildet, bestehen.

Die Klasse D_0 enthält dabei die spezifischen Axiome, die für die Deduktion der übrigen Sätze der Klasse D notwendig sind. Diese Deduktion erfolgt mithilfe der Regeln der Inferenz, die abhängig sind vom jeweiligen gewählten logischen Formalismus, z.B. der deontischen Logik. Die Klasse $Cn(D_0)$ enthält dabei alles, was durch Anwendung der logischen Inferenz von der Klasse D_0 abgeleitet werden kann. Ein axiomatisiertes Rechtssystem besitzt also die Eigenschaft, dass sämtliche Normen aus der Klasse der Axiome, den „Grundsätzen", abgeleitet werden können (Reisinger, 1975, S. 61).

Nach Opalek und Wolenski ist das Rechtsystem jedoch als eine Klasse der unabhängigen Normen zu betrachten. Dies folgt aus dem Postulat der Rechtsauslegung, welches es verbietet, bei der Auslegung von Rechtstexten irgendeine Formulierung des Textes des Rechts außer Acht zu lassen. Diese Eigenschaft muss nach Ansicht der Autoren auch den deontischen Sätzen zugeschrieben werden, die das Rechtssystem repräsentieren. Die Klasse D muss damit definitionsgemäß als unabhängige Klasse betrachtet werden. Ein beliebiger Satz von D ist somit keine logische Konsequenz der übrigen Sätze (Opalek & Wolenski, 1975, S. 61).

Es erschließt sich nicht, wie die Autoren vom oben zitierten „Postulat der Rechtsauslegung" zur „Unabhängigkeit des Rechtssystems" kommen. Noch dunkler bleibt die Begründung für die darauf aufsetzende Schlussfolgerung, dass demnach kein Satz der Klasse D eine logische Konsequenz der übrigen Sätze ist.

Ohne noch weiter auf das Postulat der Rechtauslegung als Ursprung der Ausführungen näher eingehen zu wollen sei angemerkt, dass mit der letzten Schlussfolgerung möglicherweise auf den Stufenbau der Rechtsordnung hingewiesen wird, der stark verkürzt besagt, dass die Geltung aller Normen eines Rechtssystems letztlich auf eine oberste Erzeugungsregel zurückführbar sein muss (Koller, 1997, S. 118 ff.). Die in einem Rechtstext enthaltenen Normen sind jedoch in der Regel keine „Erzeugungsregeln" für Normen derselben Stufe. Damit könnte die Schlussfolgerung, dass ein Satz der Klasse D keine Konsequenz der übrigen Sätze ist, erklärt werden. Dieser Erklärungsversuch setzt jedoch voraus, dass Opalek und Wolenski ebenfalls unter der Klasse D die Gesamtheit der deontischen Sätze eines Gesetzestextes oder zumindest von Gesetzestexten derselben Rechtsstufe verstehen.

Nachdem gegen die Axiomatisierung ohnehin weitaus gewichtigere Argumente ins Treffen geführt werden können, sei der Diskurs hinsichtlich dieser Kritik an dieser Stelle beendet. Ohne weitere Informationen ließe sich einer längeren Diskussion kaum ein Mehrwert abgewinnen.

4.4.2.4. Der rechtliche Status der Axiome

Aufgrund der im letzten Abschnitt vorgebrachten Kritik hinsichtlich der Bildung der Axiome aus den deontischen Sätzen der Klasse D selbst schreitet man oft dazu, aus einem konkreten Rechtsystem einige wenige „Grundsätze", auch genannt „Hauptnormen", zu extrahieren.

Nun stellt sich die Frage nach dem rechtlich Status der Axiome, also der Klasse D_0. Gemäß der oben getroffenen Definition, ist die Klasse D_0 stets ein Teil der Klasse D. Die ausgewählten Axiome gehören also per Definition der Axiomatisierbarkeit zur Klasse D, die das Rechtssystem repräsentiert. Mit der Annahme der Systemzugehörigkeit der Axiome sind jedoch keineswegs alle theoretischen Probleme beseitigt (Reisinger, 1975, S. 62 f.).

Die „Grundsätze" werden nämlich aus dem Rechtstext auf verschiedene Arten, quasi induktiv, abgeleitet. Die Axiome ergeben sich somit nicht aus einer deduktiven Ableitung von den explizit im Text vorhandenen Normen, sie sind also keine logische Konsequenz derselben und von größerer Allgemeinheit. Dies widerspricht der Definition der Axiome als Sätze, die ohne Beweis als wahr angenommen werden.

Es wird klar, dass die Axiome ursprünglich nicht in der Klasse der Normen vorhanden sind, sondern durch einen induktiven Prozess, sozusagen außerhalb des Rechtssystems, entstehen. Das Rechtssystem wird durch die Axiomatisierung also faktisch erweitert, was in letzter Konsequenz dazu führt, dass das axiomatisierte System sich vom ursprünglichen Rechtssystem unterscheidet.

Nach Ansicht von Opalek und Wolenski widerspricht eine solche Bereicherung des Rechtsinhalts wohl verfassungsrechtlichen Prinzipien[66], da nach deren Ansicht eine solche „nur durch einen Akt der Rechtssetzung geschehen kann."[67] Es entsteht dadurch eine über

[66] Op. cit. sind die Autoren der Ansicht, dass ein solches Verfahren mangelhaft vom Standpunkt der Intuitionen über das Funktionieren des Rechts zu sein scheint. Damit kann m.E. nur gemeint sein, dass Rechtsnormen ausschließlich von verfassungsmäßig ermächtigten Organen erzeugt werden dürfen.

[67] Dazu sei angemerkt, dass auch der Richter, bis zu einem gewissen Grad, im Rahmen der Rechtsfortbildung rechtsschöpfend tätig sein kann, wodurch es ebenso zu einer Bereicherung des Rechtsinhalts kommt.

das positive Recht emporgehobene naturrechtliche Hyperstruktur (Opalek & Wolenski, 1975, S. 62 f.).

4.4.2.5. Der Wert der Axiomatisierung des Rechts und deren Sinnhaftigkeit

Um den relativen breiten Diskurs über die Axiomatisierung nun zu einem Ende zu führen, soll der Wert der Idee der Axiomatisierung des Rechts selbst beleuchtet werden. Für diesen Zweck ist der Aufsatz von Gordon hilfreich, in dem dieser den Versuch von Rödig (1980), das Werk von Klug (1982) zu verteidigen, kritisiert.

Klugs Vorschlag bestand darin, die axiomatische Methode anzuwenden, um Theorien des Rechts zu repräsentieren. Seiner Ansicht nach sollten Richter ihre Entscheidungen öffentlich mithilfe deduktiver Beweise rechtfertigen müssen, indem sie solche Axiome verwenden. Gordon behandelt in seinem Aufsatz die Probleme dieses Vorschlags, mit denen sich der Rest dieses Abschnitts beschäftigt.

(1) Welchen Wert hat die Idee das Recht zu axiomatisieren?

Der Zweck der Axiomatisierung ist, die versteckten Bekenntnisse einer Theorie sichtbar zu machen, vorausgesetzte Prämissen offen zu legen und logische Abhängigkeiten explizit darzustellen. Demnach wird die Axiomatisierung als ein Werkzeug zur Bewertung von Theorien des Rechts gesehen (Gordon, 1994a, S. 25).

Gordon kritisiert die fragwürdige Unterscheidung zwischen einer Theorie und deren Repräsentation als eine Menge von Axiomen. Es existiert nämlich keine allgemeine, unabhängige Theorie eines Rechtssystems, daher ist es nicht einmal möglich zu überprüfen, ob eine vorgeschlagene Axiomatisierung korrekt und vollständig ist. Sollten inakzeptable Aussagen aus einer Menge von Axiomen folgen, so ist es unmöglich festzustellen, ob die Axiomatisierung inkorrekt ist oder das Problem vielmehr aus der Theorie herrührt, die axiomatisiert wurde. Wenn nun die Unterscheidung zwischen einer Axiomatisierung und der zugrundeliegenden Theorie nicht mehr aufrechterhalten werden kann, hat es nach der Auffassung von Gordon keinen Sinn, noch länger von einer Axiomatisierung zu sprechen (Gordon, 1994a, S. 25).

(2) Ist ein deduktiver Beweis überhaupt notwendig und hinreichend, um richterliches Ermessen zu beschränken?

Im Wesentlichen kann die Forderung von Klug und Rödig darauf reduziert werden, dass die Entscheidungsgewalt des Richters beschränkt werden sollte, indem gefordert wurde, dass richterliche Entscheidungen durch deduktive Beweise gerechtfertigt werden müssen. Die Entscheidung sollte eine logisch notwendige Konsequenz der veröffentlichten Meinung[68] des Richters sein (Gordon, 1994a, S. 25).

Für Rödig sind die Vorteile eines solchen Verfahrens augenscheinlich, sodass dieser keine Bemühungen zur Rechtfertigung seiner Behauptungen macht. Gordon stellt Rödigs Ansatz in Frage und nähert sich dessen These durch einen Exkurs über das Prinzip der Gewaltenteilung[69] (Gordon, 1994a, S. 26 ff.).

Das Prinzip der Gewaltenteilung soll für Gleichheit und Rechtssicherheit sorgen. Urteile sollen vorhersehbar und unwillkürlich erlassen werden, und die persönlichen Vorurteile oder privaten Interessen des Richters dürfen keinen Einfluss auf die Entscheidung nehmen.

Besonders stark war die Meinung nach der französischen Revolution vertreten, richterliche Entscheidung müsse auf die korrekte Anwendung der Deduktion beschränkt werden, ohne dass ein Spielraum für Interpretation bestünde. Die Erlaubnis des Richters zur Interpretation bringe die Gefahr, dass der Wille des Gesetzgebers untergraben wird. Bei Unklarheiten sei der Richter verpflichtet, den Gesetzgeber zu konsultieren um zu klären, ob und wie ein Gesetz anzuwenden ist (Gordon, 1994a, S. 26).

Es stellte sich jedoch bald heraus, dass dieser Ansatz impraktikabel ist, da die Gerichte dadurch massiv bei ihrer Arbeit behindert wurden. Wichtiger ist jedoch, dass durch diese Forderung, im Gegensatz zur eigentlichen Intention dieses Ansatzes, das Prinzip der Gewaltentrennung unterlaufen wurde. Genauso wie die Gerichtsbarkeit nicht zur Gesetzgebung berufen ist, überschreitet es die Kompetenzen des Gesetzgebers, wenn dieser zur Entscheidung im Einzelfall angerufen wird. Es besteht die Gefahr, dass der Grundsatz der „Gleichheit“ durch ein Gesetz verletzt wird, das bloß aus dem Anlass, einen bestimmten Fall zu entscheiden, erlassen wird. Aus diesem Grund sah der spätere „Code Napoleon“ vor, dass

[68] Darunter sind wohl die festgestellten Tatsachen, also der Sachverhalt, zu verstehen, von dem der Richter überzeugt ist und der in Folge für die rechtliche Beurteilung als Grundlage dient.

[69] Das Prinzip der Gewaltenteilung ist in Österreich in Art. 94 B-VG verankert.

Richter verpflichtet sind, jeden Fall selbst zu entscheiden, selbst wenn die Rechtslage unklar oder unvollständig ist (Gordon, 1994a, S. 26).

Zunächst hat es den Anschein, die Forderung nach einem deduktiven Beweis der richterlichen Entscheidung stehe in Einklang mit der Forderung nach Trennung der Gerichtsbarkeit von der Gesetzgebung. Es wurde für möglich gehalten, dass Fälle beurteilt werden können, indem das Recht auf den Sachverhalt deduktiv angewendet wird.

Wie oben jedoch gezeigt wurde, gibt es triftige Gründe, die bloße Deduktion als unzureichend für die Beurteilung von Fällen zu erachten. Für Gordon ist es unverständlich, warum trotz dieser Bedenken nicht schon früher von der Forderung nach einem deduktiven Beweis der Entscheidung abgewichen wurde. Seiner Ansicht nach ist ein deduktiver Beweis weder *notwendig* noch *hinreichend*, um die Grundsätze der Bestimmtheit und Gleichheit zu schützen (Gordon, 1994a, S. 27):

„Deductive proof is neither necessary nor sufficient to protect the interests of certainty and equality. It is clearly not sufficient, as any proposition entails itself. Merely assuming the decision would not justify it."

Ein „deduktiver Beweis" ist nach Gordon deshalb nicht *hinreichend*, da jede logische Aussage sich selbst impliziert[70]. Um diese Behauptung zu verstehen, scheinen ein paar erläuternde Ausführungen angebracht.

Es handelt sich um eine Eigenschaft der klassischen Logik, dass $\{P\} \vDash P$. Diese formale Definition geht auf die Arbeiten zur Logik von Tarski in der ersten Hälfte des zwanzigsten Jahrhunderts zurück. Die Folgerungsrelation[71] der klassischen Logik erfüllt Tarskis Bedingungen für Folgerungsrelationen. Die erste, in diesem Kontext wichtigste Bedingung ist, dass eine Menge von Aussagen X eine Teilmenge der logischen Folgerungen von X ist. Daraus folgt, dass P aus {P} folgen muss – für eine detailliertere Erklärung siehe (Jansana, 2011).

In diesem Kontext bedeutet das, dass ein deduktiver Beweis nicht hinreichend ist, um eine (juristische) Entscheidung zu rechtfertigen, da die Prämissen selbst ausreichend begründet

[70] Dieser Grundsatz wird in der englischsprachigen Literatur „immediate entailment" bezeichnet.

[71] Engl. „consequence relation".

sein müssen und nicht nur ihre logischen Konsequenzen. Dürften die Prämissen frei gewählt werden, so könnte man einfach annehmen, was man zu beweisen versucht. Es ist also die Qualität der Prämissen selbst entscheidend und nicht bloß die aus diesen folgende, deduktive Ableitung der Konklusion (Gordon, 1994a, S. 27).

Anhand eines Beispiels zeigt Gordon weiter, dass ein deduktiver Beweis ebenso nicht *notwendig* ist. Die wesentlichsten Aussagen sollen nun dargestellt werden.

Angenommen der Kläger ist der Ansicht, dass eine Norm A → B bedeutet, und der Beklagte meint hingegen, dieselbe Vorschrift bedeutet C → B. Weiter sei angenommen, dass beide zwar über die Interpretation der Vorschrift unterschiedliche Meinungen haben, jedoch übereinstimmen, dass sowohl A als auch C stattgefunden hat. Demnach ist unwichtig, welche Interpretation letztendlich richtig ist, es ist klar, dass B geboten ist (Gordon, 1994a, S. 27).

Um B deduktiv zu begründen, ohne zwischen A → B und C → B wählen zu müssen, könnte man annehmen, es würde ausreichen, die Aussage A → B ∨ C → B hinzuzufügen. Jedoch müsste diese zusätzliche Aussage aus der Rechtsordnung abgeleitet werden können, damit diese rechtlich gültig ist. Die einfache Annahme dieser Aussage ohne konkrete Rechtfertigung durch die zugrundeliegenden Normen würde die Absichten, die hinter der Forderung nach einer deduktiven Begründung stehen, ad absurdum führen (Gordon, 1994a, S. 27).

Eine Lösung für dieses Dilemma könnte sein, B einfach als gegeben anzunehmen. Zum Beispiel könnte {A,C} alleine als ein Argument für B angenommen werden. Obwohl nun B keine logische Konsequenz aus diesem Argument ist, kann letzteres kritisch auf seine rechtliche Richtigkeit geprüft werden. Die Prozedur, um ein solches „elliptisches Argument“ zu prüfen, unterscheidet sich von der Überprüfung einer als deduktiver Beweis formulierten Entscheidung. Es handelt sich um einen abduktiven Prozess, der eine akzeptable Erklärung für die Entscheidung zu finden versucht. Letztere ist dann akzeptabel, wenn zusätzlich zu den im Urteil zum Ausdruck gebrachten Aussagen weitere gefunden werden können, mit deren Hilfe es letztendlich möglich ist, die Entscheidung abzuleiten (Gordon, 1994a, S. 27).

Dieser Ansatz ermöglicht es, dass eine Entscheidung durch eine höhere Instanz, z.B. ein Berufungsgericht, gekippt wird, indem dieses (bessere) Argumente findet, die gegen das Urteil sprechen. Den Grundsätzen der Gleichheit und Bestimmbarkeit wird so besser

entsprochen, da die höheren Instanzen stets als Kontrollinstanz fungieren und so möglichst ressourcenschonend arbeiten können. Es wäre nicht sinnvoll, würden die Ressourcen höherer Instanzen bloß für den Zweck in Anspruch genommen werden, einen deduktiven Beweis für die Entscheidung zu konstruieren (Gordon, 1994a, S. 27).

Warum hielten Klug und Rödig trotz dieser naheliegenden und doch gewichtigen Kritik an ihrem Ansatz fest? Nach Gordon wurden sie von der damals vorherrschenden Meinung fehlgeleitet, dass Prädikatenlogik erster Stufe ein universell anwendbares Modell korrekten Denkens sei. Wie hier gezeigt wurde, ist dieser Ansatz jedoch mit einigen schwerwiegenden Problemen behaftet (Gordon, 1994a, S. 27).

4.4.3. Abschließende Bemerkungen

Kommen wir nun nach diesem etwas längeren Exkurs über die Axiomatisierung zu den abschließenden Bemerkungen. Nach den in diesem Kapitel vorgebrachten Bedenken gegen eine Axiomatisierung des Rechts erscheint es wenig verwunderlich, dass bis heute keine dem Autor bekannten Versuche unternommen wurden, ein Rechtssystem oder zumindest Teile davon, beispielsweise ein bestimmtes Gesetz, zu axiomatisieren.

Bereits Opalek und Wolenski hielten es für bedenklich, ein Rechtssystem als ein axiomatisches System zu betrachten, das aus nicht näher bestimmten „Grundsätzen“ besteht. Die von ihnen vorgebrachte Kritik hinsichtlich der Frage, ob Normen als logische Sätze abgebildet werden können, ist berechtigt, scheint jedoch hinsichtlich gewichtigerer Bedenken in den Hintergrund zu treten. Die genannten Autoren stellten bereits den rechtlichen Status eines axiomatisch konstruierten Systems in Frage. Sie bemerkten zu Recht, dass eine Erweiterung des Rechtssystems durch Axiome aufgrund rechtlicher Prinzipien bedenklich ist.

Gordon geht sogar noch einen Schritt weiter und hält die Unterscheidung einer nicht näher bestimmbaren Theorie und deren Repräsentation durch Axiome für nicht haltbar. Damit stellt sich die Frage, ob eine Axiomatisierung überhaupt als probates Mittel zur Prüfung der Vollständigkeit und Widerspruchslosigkeit einer Theorie geeignet ist?

Das im aktuellen Kontext jedoch meines Erachtens weit wichtigere Vorbringen ist Gordons zweiter Kritikpunkt. Gordon hält die Forderung nach einer deduktiven Begründung juristischer Entscheidungen als unhaltbar und bestätigt damit die Kritik von Opalek und

Wolenski. Er untermauert seine Kritik, indem er seine Formulierungen weiter präzisiert und einen deduktiven Beweis weder für notwendig noch für hinreichend erachtet. Angemerkt sei, dass die Begriffe „notwendig“ und „hinreichend“[72] aus der Logik stammen und im Sinne von „Notwendiger und hinreichender Bedingung“ zu verstehen sind (Brennan, 2012). Es handelt sich bei diesen Begriffen im Wesentlichen um logische Eigenschaften der Beziehung zwischen zwei Aussagen.

Gordons treffende Ausführungen und insbesondere die Feststellung, dass eine ausschließlich deduktiv konstruierte Begründung von Entscheidungen qualitativ unzureichend ist, reicht wohl aus, um die Idee der Axiomatisierung des Rechts zu verwerfen. Die Kritik von Gordon entwickelt jedoch auch eine über die Axiomatisierung hinausgehende Bedeutung, worauf noch näher in Kapitel 4.5. eingegangen wird.

4.5. Die Formalisierung im Kontext der Dissertation

Kommen wir nun zu einer ersten Zusammenfassung der Erwägungen. Wie den Ausführungen der vorhergehenden Kapitels zu entnehmen ist, scheint es kaum von Nutzen zu sein, an der Idee der Axiomatisierung eines Gesetzes festzuhalten. Der Vollständigkeit halber sei überdies noch angemerkt, dass keinesfalls die Formalisierung „des Rechts“ als komplexes theoretisches Phänomen, ähnlich den axiomatischen Bestrebungen von Ferrajoli (2007), unternommen werden soll.

Damit ist bereits ein wesentlicher Schritt zur Abgrenzung des Gegenstandes der Dissertation getan. Die Formalisierung im Kontext der Dissertation ist „im weiteren Sinn“[73], jedoch mit Ausnahme der Axiomatisierung, zu verstehen. Im Gegensatz zu den Überlegungen von Reisinger (1975) und Morscher (1988) steht jedoch keinesfalls ein primär logischer Ansatz, d.h. die Verwendung eines logischen Kalküls zur Formalisierung, im Vordergrund.

Das Festhalten an der deduktiven Schlussfolgerung zur Lösung juristischer Probleme erscheint, vergegenwärtigt man sich insbesondere die Ausführungen von Gordon in Kapitel 4.4.2. und die Erkenntnisse jahrzehntelanger Forschungen, nicht zielführend und wirkt, in Anbetracht des State of the Art, anachronistisch. Ebenso wenig soll ein linguistischer oder mathematischer Ansatz verfolgt werden (Reisinger, 1975, S. 37).

[72] Engl. „Necessary and sufficient conditions“.

[73] Siehe zur Definition Kapitel 4.3.2.

Die Darstellung der frühen Überlegungen hinsichtlich der Formalisierung ist jedoch keineswegs vergebene Mühe. Ohne sie wären die neueren Ansätze zur Formalisierung, wie z.B. Ontologien, nur schwer zu verstehen, da auch diesen ein logischer Kalkül zugrunde liegt. Nicht zu vergessen, dass auch heute noch die boolesche Logik[74] Grundlage moderner Computersysteme und Sprachen ist.

Ziel der Arbeit ist also weder die Übertragung eines Gesetzes in ein Modell der Prädikatenlogik etc., noch geht es primär um die Entwicklung einer neuen (logischen) Sprache. Vielmehr geht es um die Untersuchung vorhandener Technologien und Sprachen zur Erstellung von computergestützen Modellen des Rechts für die automatisierte Entscheidungsfindung. Das Gesetz soll demnach, im Sinne der Definition von Reisinger (siehe Kapitel 4.3.1.), in „Formeln" übersetzt werden. Der Fokus liegt aber dabei nicht auf der Abbildung der Normen durch logische Aussagen, daher ist der Begriff „Formel" eher im Sinn einer formalen Regel zu verstehen (Reisinger, 1975, S. 24).

Bei diesem Vorhaben soll die Ausdrucksstärke vorhandener Sprachen, insbesondere OWL 2, genutzt werden, um ein computergestütztes Modell des Gesetzes zu erstellen. Ein wesentlicher Teil der Forschungen befasst sich mit der Untersuchung der Eignung von Ontologien zur Formalisierung von Verwaltungsrecht.

Insbesondere die Eignung von Ontologien zur Formalisierung ist hier vorweg in diesem Zusammenhang als Gegenstand der Untersuchung zu nennen. In der Terminologie von Reisinger kann also die „Formalisierung" in einem weiten Sinn, als eine Anwendung formaler Methoden auf das Recht, verstanden werden (Reisinger, 1975, S. 28 f.).

Besonderes Augenmerk bei der Auswahl der Sprache gilt deren Eigenschaft, computerunterstützt verarbeitet werden zu können. Es muss möglich sein, das so erstellte Modell des Gesetzes zur automatisierten Lösung juristischer Probleme, also zur Entscheidungsfindung, verwenden zu können.

Das automatisiert ermittelte Ergebnis der rechtlichen Beurteilung muss jedenfalls wieder in die natürliche Sprache übersetzt werden. Man denke hier insbesondere an den Spruch und

[74] Bei dieser handelt es sich um eine zweiwertige Logik, d.h. eine Aussage kann entweder „wahr" oder „falsch" sein. Mithilfe der logischen Operatoren „UND", „ODER" und „NICHT" lassen sich einzelne Aussagen verknüpfen.

die rechtliche Begründung eines Bescheides, die für dessen Rechtmäßigkeit unabdingbar vorhanden und schlüssig sein müssen.

Obwohl das primäre Anliegen der Untersuchung demnach nicht die Schaffung einer neuen (logischen) Sprache zur Formalisierung ist, scheint diese doch eine vorzügliche Grundlage für die zukünftige Entwicklung einer „Domain Specific Language“ (Gosh, 2011) für die Modellierung von Verwaltungsrecht zu sein.

Auch wenn es verlockend erscheint, ein Gesetz in einem axiomatischen Modell abzubilden und vor allem auf seine Widerspruchslosigkeit zu prüfen, soll dieser Ansatz – wie bereits weiter oben ausgeführt – nicht weiter verfolgt werden. Es geht nämlich primär um eine Effizienzsteigerung von juristischen Expertensystemen, wofür eine „bloße“ Formalisierung vollkommen ausreichend ist.

Zusammengefasst kann also festgestellt werden, dass in dieser Dissertation die Formalisierung im *weiteren Sinn* zu verstehen ist. Nachdem primär die Eignung von Ontologien für die Erstellung eines computerunterstützten Modell des Gesetzes untersucht wird, soll von einer *ontologischen Formalisierung* gesprochen werden. Nachdem jedoch auch die neueren Sprachen auf irgendeiner Form der Logik beruhen, hat diese Art der Formalisierung jedenfalls auch eine logische Komponente.

4.6. Vorgehensweise

Nachdem nun die Bedeutung der Formalisierung im Kontext dieser Dissertation geklärt wurde, sollen im Folgenden die Schritte einer Formalisierung (im weiteren Sinn) dargestellt werden.

Eine Formalisierung glieder sich regelmäßig in folgende Phasen (Reisinger, 1975, S. 28):

1. Taxonomik: Es erfolgt eine erste Betrachtung des Rechtsbereichs, die diesen für eine weitergehende strukturelle Analyse vorbereiten soll. Dafür werden die Elemente isoliert und nach bestimmten Kriterien klassifiziert.
2. Strukturalistik: Die zweite Phase dient dazu, Beziehungen zwischen den Elementen, die im ersten Schritt identifiziert wurden, zu extrahieren und darzustellen.
3. Formalisierung im engeren Sinne: Im dritten Schritt erfolgt die Übersetzung der Elemente und Beziehungen in die formale Sprache. Zu diesem Zeitpunkt muss also bereits feststehen, welche Sprache für die Formalisierung verwendet werden soll.

4. Algorithmisierung: Abhängig von der gewählten Sprache wird ein entsprechendes Lösungsverfahren definiert, das einem genau festgelegten Ablauf folgt. Hier sei angemerkt, dass dieser Schritt unter Umständen entfällt, wenn eine vorhandene Sprache zur Formalisierung genutzt wird, da mit dieser grundsätzlich auch ein entsprechender Lösungsalgorithmus vorgegeben ist[75].
5. Automatisierung: In der letzten Phase wird das erstellte Modell schließlich auf einem Computersystem ausgeführt. Welche Hard- und vor allem Software verwendet wird, ist wiederum von der gewählten Sprache abhängig.

Die Schritte eins und zwei dienen der Vorbereitung der Formalisierung im weiteren Sinn (Schritte drei bis fünf).

Am wichtigsten ist wohl Phase drei, da in dieser das eigentliche formale Modell erstellt wird. Um eine eindeutige, formale Darstellung der Rechtsnormen zu erlangen, bedarf es der vorangehenden juristischen Interpretation des Rechtstexts. Für weitergehende Ausführungen zur Interpretation und Ursache semantischer Mehrdeutigkeiten, siehe Kapitel 4.8.3.

Bei der Formalisierung im engeren Sinne sind jedenfalls auch entsprechende Tests empfehlenswert, die das Modell fortlaufend schon während der Entwicklung auf seine Richtigkeit überprüfen.

Wie bereits erwähnt wurde, sind Überlegungen zur Algorithmisierung grundsätzlich nur dann notwendig, wenn mit der gewählten Sprache nicht bereits ein bestimmter Lösungsalgorithmus vorgegeben ist. Wird zum Beispiel eine Programmiersprache wie Java verwendet, ist es die Aufgabe des Entwicklers, die rechtlichen Regeln entsprechend abzubilden und auf dieser Basis einen Algorithmus zur Entscheidungsfindung zu implementieren[76].

Angemerkt sei, dass die einzelnen Schritte wohl kaum sequentiell abgearbeitet werden können. Vielmehr handelt es sich um einen iterativen Prozess, bei dem in jeder Iteration das Modell weiter verfeinert und verbessert wird. Insbesondere die Schritte eins bis drei

[75] Zum Beispiel die Beschreibungslogik $\mathcal{SROIQ}$ auf der OWL 2 basiert.

[76] Dies unterscheidet Java wesentlich von Prolog, einer logischen Programmiersprache, die den deklarativen Ansätzen angehört. Die Anweisungen eines Programms in Prolog sind Axiome, die vom Prologinterpreter mit einem bestimmten Algorithmus verarbeitet werden.

wiederholen sich üblicherweise in mehreren Entwicklungszyklen, bis das Modell vollständig und richtig ist.

4.7. Die Relativität der Formalisierung

In diesem Kapitel wird gezeigt, dass es nicht „die“ eine „richtige“ Formalisierung eines Rechtsbereichs gibt, sondern dass diese stets relativ und auf ein bestimmtes Problem gerichtet ist. Dazu empfiehlt es sich, den Diskurs mit den Begriffen „System“ und „Modell“ der allgemeinen Systemtheorie zu beginnen (Reisinger, 1975, S. 28 ff.).

Als „System“ ist die Form zu verstehen, in der beliebige Gegenstände der Wirklichkeit zu beschreiben und zu erklären sind. In der Wirklichkeit gibt es gegliederte Gebilde und Abläufe, deren Teile miteinander anscheinend enger zusammenhängen als mit anderen Erscheinungen. Dies ist zwar keine objektive Bestimmung, oft kann man aber den Zusammenhang zwischen Teilen eindeutig identifizieren oder doch mit beträchtlicher Plausibilität unterstellen. Solche zusammenhängenden Teile heißen *Ganzheiten*[77] (Ropohl, 2012, S. 52).

Das „Modell“ ist die weniger konkrete Abbildung eines Originals, die mit diesem nur in gewissen Zügen übereinstimmt, und demnach bloß bestimmte Eigenschaften des Originals wiedergibt. Modelle sind durch drei Merkmale charakterisiert (Ropohl, 2012, S. 53):

1. Das Abbildungsmerkmal: Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale.
2. Das Verkürzungsmerkmal: Modelle erfassen im Allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellschaffern und/oder Modellbenutzern relevant erscheinen.
3. Das pragmatische Merkmal: Modelle sind ihren Originalen nicht per se eindeutig zugeordnet, sondern erfüllen ihre Ersetzungsfunktion (1) für bestimmte – erkennende und/oder handelnde, modellbenutzende – Subjekte, (2) innerhalb bestimmter Zeitintervalle und (3) unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen.

[77] Hervorhebung im Original.

In diesem Sinne sind Systeme Modelle von Ganzheiten, die aber nur ausgewählte Eigenschaften des modellierten Objekts berücksichtigen.

Kommen wir nach diesem kurzen Exkurs in die Systemtheorie nun wieder zur Formalisierung zurück. Der betrachtete Rechtsbereich kann regelmäßig als „System“ im Sinne der allgemeinen Systemtheorie aufgefasst werden (Reisinger, 1975, S. 30). Der Aufbau der Taxonomik und die Offenlegung der Struktur des Rechtsbereichs können als die systemtheoretische Definition des „Originals“, und die Formalisierung im engeren Sinne als Erstellung eines Modells begriffen werden. Damit wird offenkundig, dass jede Formalisierung durch die oben genannten Merkmale eines Modells gekennzeichnet ist. Insbesondere die im dritten Punkt der oben stehenden Aufzählung dargestellten Kriterien, „wozu“ und „für wen“ ein Modell geschaffen werden soll, sind für die Formalisierung von größter Bedeutung. Aus dem beabsichtigten Zweck („wozu“) ergibt sich erst der notwendige Inhalt des Modells. Erst wenn das Modell diesen Zweck erfüllt, kann es als vollständig angesehen werden. Von wem das Modell verwendet werden soll, beeinflusst einerseits, eigentlich selbstredend, ebenfalls den Inhalt, aber andererseits auch die Höhe der noch akzeptablen Komplexität des Modells (Reisinger, 1975, S. 31 f.).

Diese Ausführungen haben gezeigt, dass jede Formalisierung stets problembezogen ist. Es scheint daher unwahrscheinlich, dass für einen größeren Bereich, wie z.B. das österreichische Verwaltungsrecht, ein einziges umfassendes Modell, z.B. in der Gestalt einer Ontologie, entwickelt werden kann, das sämtliche Zwecke und Anforderungen aller Benutzer erfüllt.

Aussichtsreicher erscheinen meines Erachtens Forschungen, die eine spezielle Sprache, im Sinne eines gemeinsamen Standards, zur Modellierung von Rechtsnormen entwickeln. Solch eine Sprache lässt ausreichend Raum für Flexibilität, sodass Modelle entwickelt werden können, die auf spezielle Anforderungen zugeschnitten sind.

4.8. Grundlagen der Rechtsanwendung

4.8.1. Einleitung

Das Gesetz als eine Anordnung des Gesollten schreibt den Rechtsunterworfenen vielfaltige Rechte und Pflichten vor. Der Tatbestand einer Norm legt dabei fest, auf welche

Sachverhalte diese anwendbar sein soll. Es liegt auf der Hand, dass es vollkommen unmöglich ist, für alle denkbaren Sachverhalte spezielle Normen zu erlassen.

Aus diesem Grund umschreibt der Tatbestand einer Norm abstrakt einen Sachverhalt. Der Sachverhalt wird durch ausreichend viele Tatbestandsmerkmale beschrieben, um diesen von anderen abgrenzbar zu machen. Diese abstrakte Anordnung des rechtlichen Gebotenen – oder genauer dessen Tatbestandsmerkmale – müssen vom Rechtsanwender schließlich auf einen konkreten Sachverhalt angewendet werden, um die Anwendbarkeit der Norm im Einzelfall feststellen zu können (Koller, 1997, S. 80 ff.).

Dieser Vorgang wird als Subsumtion bezeichnet. Die Rechtsfolge einer Norm darf ausschließlich dann die beteiligten Rechtssubjekte treffen, wenn deren Tatbestandsseite auf den Sachverhalt als zutreffend identifiziert wurde.

Um von der abstrakten, generellen Norm zur Anwendbarkeit derselben im Einzelfall zu gelangen, muss der Rechtsanwender demnach zuerst den intellektuellen Vorgang der Interpretation durchführen. Zu diesem Zweck steht ihm der gesamte Kanon der juristischen Interpretationsmethoden zur Verfügung (Bydlinski, 2005).

Dieser Abschnitt geht zuerst auf die Subsumtion und die Probleme, die beim Subsumtionsvorgang auftreten können, ein und endet sodann in einer Einführung in die juristische Interpretation.

4.8.2. Die Subsumtion

Allgemein bekannt ist, dass die Rechtsordnung generelle und individuelle Normen kennt. Die gedankliche Operation, die den Tatbestandsbedingungen einer Rechtsregel einen speziellen Sachverhalt, der diesen Bedingungen entspricht, zuordnet, wird Subsumtion genannt.

Diese Schlussform, die man als Subsumtionsschluss oder *juristischen Syllogismus* bezeichnet, hat die folgende (logische) Gestalt (Koller, 1997, S. 81):

(1) Für jede Person x: Wenn Person x den Tatbestand T erfüllt, dann soll für x die Rechtsfolge R eintreten,
(2) Person x_i erfüllt den Tatbestand T,
(3) für Person x_i soll Rechtsfolge R eintreten.

Die erste Prämisse (1) des Subsumtionsschlusses ist stets eine Rechtsregel oder eine Konjunktion (Verbindung) mehrerer Rechtsregeln. Die zweite Prämisse (2) ist die Beschreibung eines konkreten Sachverhalts. Aus beiden Prämissen zusammen folgt schließlich die Konklusion (3). Die Konklusion ist stets eine individuelle Norm, die zum Ausdruck bringt, dass die zugrundeliegende Rechtsregel für jene Person(en) eintreten soll.

Dieser auf den ersten Blick recht einfache Vorgang ist jedoch regelmäßig bei der Anwendung von Rechtsregeln auf einzelne Fälle alles andere als trivial. Die Probleme rühren dabei aus der Gewinnung der Prämissen her, oder sie ergeben sich, wenn man den Schluss bereits gezogen hat.

Engisch (1963, S. 15) hat die Vorgehensweise bei der Beurteilung von Sachverhalten einst sehr anschaulich mit seinem oft zitierten Satz als ein „Hin- und Herwandern des Blickes zwischen Obersatz und Lebenssachverhalt“ beschrieben (Koller, 1997, S. 190).

Koller (1997, S. 186 ff.) unterscheidet folgende Problemkreise, wobei die wichtigsten Aussagen und Erkenntnisse im Anschluss dargestellt werden.

4.8.2.1. Probleme betreffend die Rechtsregeln

Die Rechtsordnung präsentiert sich für den Rechtsanwender als ein komplexes Geflecht von Normen, deren Zusammenhang oftmals nicht sofort ersichtlich ist. Hat man dieses Gewirr entflochten, muss man zuerst ein hinreichend genaues Verständnis des Inhalts der Normen entwickeln, um daraus irgendwelche Schlussfolgerungen ableiten zu können. Es stellt sich also das Problem der Interpretation oder Auslegung des Rechts.

Zusammengefasst stellen sich im Hinblick auf die Anwendung von Rechtsvorschriften die folgenden drei Probleme (Koller, 1997, S. 186):

1. Die Auffindung der einschlägigen Rechtsvorschriften,
2. das Erkennen des Zusammenhangs dieser Rechtsvorschriften, und
3. ihre Interpretation bzw. Auslegung.

Diese drei Probleme sind jedoch nicht unabhängig voneinander zu sehen, sondern sind miteinander verstrickt – dies erschwert die Rechtsanwendung noch zusätzlich.

4.8.2.2. Probleme betreffend den Sachverhalt

Die Tatsachen und Abläufe des gesellschaftlichen Lebens treten von vornherein nicht als rechtliche Sachverhalte in Erscheinung. Die Tatsachen müssen zudem erst einmal erhoben und bewiesen werden, bevor sie rechtlich beurteilt werden können (Koller, 1997, S. 186 f.).

Es ergeben sich hinsichtlich des Sachverhalts bei der Rechtsanwendung zwei Probleme:

1. Die rechtliche Feststellung der behaupteten Tatsachen nach beweiswürdigenden Verfahrensschritten, und
2. die rechtliche Deutung und Qualifikation dieser Tatsachen.

4.8.2.3. Probleme betreffend die Rechtsfolge

Selbst wenn die beiden vorangehenden Probleme gelöst wurden, die relevanten Rechtsvorschriften also klar sind und der vorliegende Sachverhalt hinreichend festgestellt wurde, führt der Subsumtionsschluss nicht immer zu einem eindeutigen Ergebnis. In vielen Fällen räumt das Gesetz dem Rechtsanwender nämlich einen gewissen Ermessenspielraum ein, der im Hinblick auf den Umfang freilich variiert. Die anzuwendenden Normen geben damit also nur eine Art Rahmen vor, innerhalb dessen der Rechtsanwender im Einzelfall eine bestimmte Rechtsfolge setzen soll.

Nach Koller (1997, S. 187) ist demnach *„[…] das Hauptproblem der Rechtsanwendung hinsichtlich der Festlegung der Rechtsfolge […] die Bestimmung eines genauen Ergebnisses innerhalb eines durch die anzuwendenden Gesetze vorgegebenen Entscheidungsrahmens.“*

Obwohl zu diesen überblicksmäßig dargestellten Problemen noch einige weitere kommen, insbesondere wenn die Auslegung keine eindeutigen Ergebnisse liefert oder das Gesetz offensichtlich fehlerhaft, zweckwidrig oder ungerecht ist, soll hier lediglich die Interpretation näher dargestellt werden. Die Notwendigkeit der Interpretation, insbesondere von vagen Gesetzesbegriffen, stellt die Formalisierung im engeren Sinn und damit die automatisierte Rechtsanwendung vor große Schwierigkeiten (Koller, 1997, S. 187).

4.8.3. Die Interpretation von Rechtsvorschriften

Nicht immer sind Rechtsvorschriften so einfach und klar abgefasst, dass ohne Schwierigkeiten die Fakten des Sachverhalts unter sie subsumiert und daraus die Rechtsfolge

abgeleitet werden kann. Einfache Fälle[78] bereiten dem Juristen keine großen Probleme und können in der Regel relativ gut formalisiert werden. Es erscheint vertretbar, dass manch einer diesen einfachen Vorgang des Verstehens gar nicht als Interpretation bezeichnen möchte. Dies ist aber eine rein terminologische Frage, die hier nicht weiter von Bedeutung ist (Koller, 1997, S. 197 ff.).

Viel schwieriger sind jene Fälle, in denen das Verstehen einer Norm einen Prozess des Nachdenkens oder Überlegens erfordert, es also einer eingehenderen Interpretation bedarf. Die Definition der Interpretation lautet wie folgt (Koller, 1997, S. 200):

„Als Interpretation oder Auslegung kann man ganz allgemein jeden Vorgang bezeichnen, der auf das Verstehen der Bedeutung bzw. des Sinns bedeutungstragender Symbole gerichtet ist, gleichgültig, ob es sich dabei um sprachliche oder nichtsprachliche Symbole handelt."

Ein wesentliches Charakteristikum von Normen ist, dass sie den gedanklichen Bedeutungsinhalt sprachlicher Äußerungen oder kurz den Sinn von Sätzen bilden (Koller, 1997, S. 65). Es geht bei der Interpretation also um das Verstehen sprachlicher Ausdrücke. Dieses Problem stellt sich aber nicht nur in der Rechtswissenschaft, sondern in allen Wissenschaften, die mit symbolischen Artefakten zu tun haben. Vor allem die Geisteswissenschaften müssen sich mit der Auslegung intensiv auseinandersetzen. Daher wurde von den Geisteswissenschaften die Lehre der Hermeneutik entwickelt, die sich speziell mit der Frage beschäftigt, ob es eine Methode gibt, sinnhafte Symbole richtig zu deuten. Die Lehre der juristischen Interpretation wird demgemäß auch juristische Hermeneutik genannt (Koller, 1997, S. 200 f.).

Bisher wurde noch keine allgemeingültige Methode gefunden, die zielsicher zu eindeutigen und allgemein überzeugungskräftigen Ergebnissen führt. Das Dilemma besteht darin, dass das Verstehen eines Texts einerseits das Verstehen seiner einzelnen Komponenten voraussetzt, andererseits die Bedeutung dieser Komponenten aber wiederum vom Kontext abhängt, in dem sie stehen, sodass man diese nur dann richtig verstehen kann, wenn man bereits ein Verständnis des ganzen Textes hat. Dieses Problem wird aufgrund seiner Kreisform als hermeneutischer Zirkel bezeichnet.

[78] Die Unterscheidung zwischen „einfachen" Fällen, in denen die Fakten eindeutig von der Bedeutung der Begriffe des Tatbestands erfasst sind, und „schwierigen" Fällen, die einer Interpretation durch den Richter bedürfen, geht auf Hart (2009) zurück.

Der Zirkel wird daher von manchen Vertretern der Hermeneutik als Teufelskreis bezeichnet, aus dem es keinen Ausweg gibt. Ein richtiges Verstehen von Texten könne es nach dieser Ansicht überhaupt nicht geben (Koller, 1997, S. 201).

Diese Auffassung erscheint jedoch reichlich übertrieben. So ist es zwar richtig, dass für das Verstehen eines Textes immer ein gewisses Vorverständnis notwendig ist, um welche Art von Text es sich handelt. Dieses Vorverständnis ist jedoch keine unveränderbare Gegebenheit. Das Interpretieren von Texten ist vielmehr ein Prozess der Hypothesenbildung und Hypothesenprüfung, durch den man sich nach und nach von schlechteren zu angemesseneren Hypothesen weiter bewegt.

Damit ist jedoch keinesfalls gewährleistet, dass dadurch ein eindeutiges Ergebnis gefunden werden kann. Fast jeder Text lässt mehrere Deutungen zu. Zur Verteidigung einer bestimmten Rechtsauffassung ist es daher notwendig, dass bessere Argumente für eine bestimmte Interpretation des Textes gefunden werden können (Koller, 1997, S. 202).

Die Interpretation von Texten dient in erster Linie der Ermittlung der Bedeutung. Es soll also deren Semantik herausgearbeitet werden. Die Bedeutung von sprachlichen Äußerungen hängt dabei maßgeblich vom pragmatischen Kontext ab, in dem sie verwendet werden (Koller, 1997, S. 200 ff.).

Auch wenn es nicht eine einzige allgemein gültige Methode der Interpretation gibt, die zu einem eindeutigen und exakten Ergebnis gelangt, wurden von der Lehre anerkannte Interpretationsmethoden entwickelt. Die juristisch korrekte Anwendung dieser Methoden für die Interpretation von Rechtsvorschriften ist zielführender und überzeugender als das Streben nach der vermeintlich einzigen richtigen Lösung. Von besonderer Wichtigkeit sind diese Methoden bei der Anwendung von unbestimmten Gesetzesbegriffen, die auch Generalklauseln genannt werden, deren Anwendung eine juristisch und methodisch korrekte Interpretation im Einzelfall voraussetzt. Eine gute Einführung in die juristische Methodenlehre gibt Bydlinski (2005) in seinem Werk.

Wir können festhalten, dass die Interpretation von Rechtstexten eine unbedingte Voraussetzung für die Formalisierung ist, da sich deren Bedeutung nicht immer sofort erschließt. Freilich wäre es vermessen zu behaupten, es gäbe für einen bestimmten Text nur eine einzige richtige Interpretation. Vielmehr kann man in der Regel mehrere Interpretationen einer Norm für vertretbar halten. Dies zeigt sich vor allem in einem

Rechtsstreit, in dem die Parteien danach trachten, bessere, überzeugendere Argumente für die ihnen dienliche Interpretation der jeweiligen Norm zu finden (Koller, 1997, S. 202).

Überdies ist eine Interpretation in der Regel stark vom jeweiligen Kontext abhängig, in dem eine sprachliche Formulierung gebraucht wird. Aus dem Kontext heraus lässt sich oft erst der Sinn einer Regelung weiter ergründen.

Es zeigt sich auch hier wieder deutlich die bereits angesprochene Relativität der Formalisierung (Reisinger, 1975, S. 41 f.), die von der jeweiligen Interpretation der Normen abhängig ist.

4.9. Grenzen

4.9.1. Einleitung

In Unterkapitel 4.8. wurden die Grundlagen der Rechtsanwendung dargestellt. Es wurde gezeigt, dass der juristische Syllogismus, auf dem die Rechtsanwendung beruht, an sich kein komplexes Vorgehen ist. Dessen formale Abbildung, z.B. in einem Computerprogramm, ist grundsätzlich nicht sonderlich schwierig.

Die Probleme rühren vielmehr aus semantischen Mehrdeutigkeiten des Gesetzestextes her. Diese können mehrere Ursachen haben. An dieser Stelle soll auf die wichtigsten Probleme eingegangen werden. Zu diesem Zweck wird auf vage Gesetzesbegriffe als Problem auf Seiten der Rechtsregeln, und auf das behördliche Ermessen sowie die Rechtsfortbildung als Problem bei der Rechtsfolgenermittlung, eingegangen. Der Problemkreis betreffend den Sachverhalt[79] scheint in diesem Kontext eine eher untergeordnete Bedeutung zu haben, da erwartet wird, dass die Tatsachenfeststellungen bereits in strukturierter Form an das Programm übergeben werden (Koller, 1997, S. 207 ff.).

Vorausgeschickt sei, dass vage Formulierungen keineswegs immer ein Fehler des Gesetzes sind, sondern vielmehr die vielfältigen Sachverhalte erfassen sollen, wie Koller zutreffend formuliert (Koller, 1997, S. 210):

„Grundsätzlich aber ist die Existenz gewisser semantischer Unbestimmtheiten im Recht ganz und gar unvermeidlich, wenn die Rechtsvorschriften flexibel genug sein sollen, um auf eine Vielzahl verschiedener Sachverhalte Anwendung finden zu können.“

[79] Siehe Abschnitt 4.8.2.2.

Für die automatisierte Rechtsanwendung bedeutet das, dass solchen Unbestimmtheiten nicht einfach eine konkrete Bedeutung (willkürlich) gegeben werden darf, und auch, dass diese nicht gänzlich ignoriert werden dürfen. Die Bedeutung der Norm muss juristisch korrekt ermittelt werden und durch eine entsprechende formale Regel[80] repräsentiert werden.

4.9.2. Unbestimmte Gesetzesbegriffe

Für die automatisierte Rechtsanwendung ist das Problem der Interpretation von vagen Begriffen von besonderer Bedeutung (Franklin, 2012, S. 226 ff.; Morscher, 1988, S. 116 ff.; Reisinger, 1975, S. 42 ff.). Anders als bei sonstigen Interpretationsproblemen, bei denen sich in der Regel eine akzeptable Bedeutung ermitteln lässt, bietet in solchen Fällen das Gesetz keine eindeutige Antwort. Bei vagen Begriffen ist oft schwierig zu beurteilen, ob die Sachverhaltselemente unter den Tatbestand der jeweiligen Norm subsumiert werden können (Koller, 1997, S. 207 f.).

Dies bereitet erhebliche Schwierigkeit für die automatisierte Rechtsanwendung, da die logische Beziehung zwischen Tatbestand und Sachverhalt nicht eindeutig ist. Eine formale Darstellung verlangt nämlich, dass zuvor die Bedeutung einer Norm so weit klargestellt wurde, dass eindeutig ist, wann ein logischer Ausdruck, bei einer vorliegenden Interpretation, als wahr gilt.

Vage (unbestimmte) Begriffe sind deskriptive Begriffe, deren Inhalt und Umfang nicht klar umgrenzt ist, sodass sich nicht immer eindeutig sagen lässt, ob ihnen bestimmte Sachverhalte entsprechen oder nicht (Koller, 1997, S. 207). Solche Begriffe sind beispielsweise: „Dunkelheit“, „Gefahr“, „Wald“, „beeinträchtigen“ und viele mehr.

Nach der Lehre von Heck haben vage Begriffe einen „Begriffshof“ und einen „Begriffskern“ (Heck, 1914). Unproblematisch sind Begriffe, bei denen eindeutig festgestellt werden kann, ob ein Sachverhalt von diesen erfasst ist oder nicht. Man sagt, dass Fälle, die eindeutig von einem Begriff erfasst werden, in dessen Begriffskern fallen. Probleme bereiten Sachverhalte, die als Grenzfälle in den Übergangsbereich zwischen den beiden Extremen der Bedeutung

[80] Die formale Regel sollte dabei erkennen lassen, wie der jeweilige Bedeutungsinhalt ermittelt wurde.

vager Begriffe, in deren Begriffshof, fallen. Es kann bei diesen nur schwierig festgestellt werden, ob ein Fakt noch vom Tatbestand erfasst ist oder nicht (Koller, 1997, S. 208).

Ferner findet man in Gesetzen sogenannte Generalklauseln, wie z.B. § 879 Abs. 1 ABGB, die ihrer Art nach durch ein hohes Maß an Vagheit gekennzeichnet sind, und daher nicht bloß in manchen Fällen mehrdeutig bleiben.

4.9.3. Ermessen

Von den im vorherigen Abschnitt behandelten unbestimmten Gesetzesbegriffen ist eine andere Kategorie genereller Gesetzesbegriffe, die sogenannten „Ermessensklauseln", zu unterscheiden. Das Ausmaß der Rechtsfolge ist nicht genau festgelegt, weshalb deren Bestimmung zu einem erheblichen Teil dem Ermessen des Richters überlassen bleibt (Koller, 1997, S. 90). Ermessensklauseln kommen beispielsweise bei der Strafzumessung im Strafrecht zur Anwendung oder sollen in Fällen „besonderer Härte" einen sozial gerechten Ausgleich schaffen (bspw. § 76 KOVG BGBl. Nr. 152/1957 i.d.F. BGBl. I Nr. 150/2002). An den Rechtsanwender werden in solchen Fällen besonders hohe Anforderungen gestellt, was im Hinblick auf die Rechtssicherheit und das verfassungsrechtliche Legaliätsprinzip durchaus Probleme aufwerfen kann.

Solche Unschärfen des Gesetzes sind aber nicht immer ein Versehen des Gesetzgebers. Vielmehr dienen Ermessensklauseln der Erfassung von Sonderfällen und sollen nach der Vorstellung des Gesetzgebers im Einzelfall für Gerechtigkeit sorgen. Die Verwaltungsbehörde (oder das Gericht) ist bei der Ermessensausübung keinesfalls völlig ungebunden, sondern ist, neben der ohnehin obligatorischen Begründung, dem Gemeinwohl und den Zielen des jeweiligen Gesetzes verpflichtet. Man spricht daher gemeinhin von einem „gebundenen Ermessen" (Bader, 1955; Raschauer, 2009, S. 210).

4.9.3.1. Komparativregeln

Komparativregeln[81] sind Rechtsregeln, bei denen das *Ausmaß der Rechtsfolge* abstufbar ist und vom Grad der Erfüllung wenigstens einer Tatbestandsbedingung abhängig ist, was

[81] Auch genannt Ermessensregeln oder Maßstabnormen.

voraussetzt, dass jene Tatbestandsbedingung und die Rechtsfolge variabel sind (Koller, 1997, S. 90)[82].

Komparative Begriffe beziehen sich auf auf Eigenschaften oder Merkmale, die im Sinne eines Mehr oder Minder variabel sind. Beispiele für solche Begriffe sind „Ausmaß der Schuld", „Schadenshöhe", „Verschuldensgrad", „Gefährlichkeit", „Strafausmaß", „Sorgfalt", „Beeinträchtigung", „Verhältnismäßigkeit" und viele mehr (Koller, 1997, S. 209).

Die Subsumtion erfolgt in solchen Fällen derart, dass der Sachverhalt hinsichtlich der betreffenden Eigenschaft mit anderen Sachverhalten verglichen und sodann entsprechend dem Ausprägungsgrad der Eigenschaft in das Kontinuum von Abstufungsmöglichkeiten eingeordnet wird. Der Bestimmung der Rechtsfolge bleibt zu einem erheblichen Teil dem Ermessen des Richters überlassen, dieser ist dabei an die in der Norm angegebenen Gesichtspunkte oder Maßstäbe gebunden. Aus diesem Grund werden Komparativregeln auch Ermessensregeln oder Maßstabnormen bezeichnet (Koller, 1997, S. 90).

Sofern komparative Begriffe im Tatbestand von Rechtsregeln vorkommen, wie das bei Komparativregeln der Fall ist, können diese auf zwei Arten mit der Rechtsfolge verknüpft sein (Koller, 1997, S. 209):

Ist die Rechtsfolge selbst eine variable Größe, so kann das Ausmaß der Rechtsfolge vom jeweiligen Ausprägungsgrad des komparativen Merkmals abhängig gemacht werden. Beispiele für diese Variante sind die Höhe der Strafe oder des Schadenersatzes.

Falls die Rechtsfolge selbst keine variable Größe darstellt, kann lediglich deren Eintreten davon abhängig gemacht werden, wenn das betreffende Tatbestandsmerkmal einen bestimmten Ausprägungsgrad erreicht bzw. einen Schwellenwert überschreitet. Vertreter dieser Variante sind beispielsweise Normen, die den Eintritt der Rechtsfolge vom Ausmaß des Verschuldens des Täters abhängig machen[83]. Dem Rechtsanwender bleibt jedenfalls in beiden Fällen ein gewisser Ermessensspielraum, die komparativen Begriffe zu präzisieren.

Meines Erachtens handelt es sich im zweiten Fall nicht um eine Ermessensregel, da die Rechtsfolge selbst keine variable Größe darstellt, die einen Ermessenspielraum offen lässt.

---

[82] Ein Beispiel ist die Strafzumessungsregel des § 32 StGB, BGBl. Nr. 60/1974 i.d.F. BGBl. Nr. 762/1996.

[83] Siehe bspw. § 150 Abs. 3 Patentgesetz 1970, BGBl. Nr. 259/1970 i.d.F. BGBl. I Nr. 149/2004.

Solche Normen ähneln Rechtsvorschriften, deren Tatbestand unbestimmte Rechtsbegriffe enthält.

Die Abgrenzung zwischen unbestimmten Gesetzesbegriffen und komparativen Begriffen ist schwierig und nicht immer eindeutig durchführbar. Denn auch unbestimmte Gesetzesbegriffe, wie „Dunkelheit", „Wald" etc., beziehen sich auf Merkmale, die im Einzelfall mehr oder minder ausgeprägt sein können (Koller, 1997, S. 207 ff.; Raschauer, 2013, S. 233 ff.).

4.9.3.2. Ermessen im Verwaltungsrecht[84]

Diese bisherigen Ausführungen waren eher auf das Zivil- und Strafrecht bezogen. Vom Ermessensspielraum der den ordentlichen Gerichten eingeräumt wird, ist das Ermessen zu unterscheiden, zu dessen Ausübung die Verwaltungsbehörden vom Bundes-Verfassungsgesetz ermächtigt sind. Aus Art. 130 Abs. 3 B-VG geht hervor, dass der Gesetzgeber die Verwaltung zu sogenannten „Ermessensentscheidungen" ermächtigen darf, die Behörde muss aber, bei sonstiger Rechtswidrigkeit der Entscheidung, vom Ermessen im Sinne des Gesetzes Gebrauch machen. Man spricht in diesem Zusammenhang auch von der Einräumung einer Wahlmöglichkeit der Behörde. Der Behörde bleibt die Wahl zwischen mehreren rechtlich möglichen Entscheidungen offen (Wielinger, 2008, S. 105 ff.).

Der Begriff „Ermessen" wird selbst kaum vom Gesetz verwendet. Die Abgrenzung zwischen Ermessen und gebundener Entscheidung ist im Einzelfall oft schwierig. Gebundene Entscheidungen sind ebenso selten exakt vom Rechtstext vorherbestimmt wie Ermessensentscheidungen (Raschauer, 2009, S. 210). Es bedarf in der Regel der eingehenden Interpretation der fraglichen Bestimmung. Die Verwendung der Worte „kann" oder „darf" gilt als Indiz für die Einräumung von Ermessen. Das Ermessen im Gesetz tritt auf unterschiedliche Weise in Erscheinung. Als Unterscheidungskriterium wird auf den Inhalt des Ermessens abgestellt.

---

[84] Mit der Verwaltungsgerichtsbarkeits-Novelle 2012, BGBl. I Nr. 51/2012, die größtenteils mit 1.1.2014 in Kraft trat, erfolgte eine der umfangreichsten Änderungen der österreichischen Bundesverfassung. Die bisher nur einstufige Verwaltungsgerichtsbarkeit wurde zweitstufig organisiert, mit dem Verwaltungsgerichtshof und den untergeordneten elf Verwaltungsgerichten (9 Landesverwaltungsgerichte und das Bundesverwaltungsgericht sowie das Bundesfinanzgericht). Paragrafen ohne Angabe der Fassung sind solche des B-VG, BGBl. Nr. 1/1930 i.d.F. BGBl. I Nr. 100/2003, wie es zum 1.1.2014 in Kraft stand. Paragrafen des B-VG mit dem Zusatz „a.F." (alte Fassung) beziehen sich auf die Fassung der Rechtsvorschrift, die zum 31.12.2013 in Kraft stand.

Einen zentralen Bereich stellen Fälle dar, in denen die Verwaltung überhaupt nicht handeln „muss“, sondern lediglich „kann“[85]. Aus diesem Grund spricht man von **Handlungsermessen**. Ferner existieren Normen, die es der Verwaltung erlauben, eine aus mehreren alternativen Vorgangsweisen zu wählen. In solchen Fällen spricht man von **Auswahlermessen**. Im Bereich der Flächenwidmung hat der Begriff des **Planungsermessens** Eingang in die Rechtsprache gefunden. Gerade in diesem Bereich lässt sich das Verwaltungshandeln nur schwer vorherbestimmen, da es dabei um Auswahl, Zielbestimmung und Abwägung und in weiterer Folge um planerische Gestaltungen geht (Raschauer, 2013, S. 226 f.).

Exkurs zur Kontrolle von Ermessensentscheidungen: Die Unterscheidung zwischen „gebundenen Entscheidungen“ und Ermessensentscheidungen ist im Verwaltungsrecht auch heute noch von essentieller Bedeutung. Gewissermaßen spiegelbildlich ist die Unterscheidung zwischen unbestimmten Rechtsbegriffen und Ermessensklauseln. Diese Unterscheidung ist historisch[86] bedingt und wurde in der Lehre des ausgehenden 19. Jahrhunderts gebraucht, um die Einschränkung des der verwaltungsgerichtlichen Kontrolle damals entzogenen „Ermessens“ argumentativ zu begründen (Raschauer, 2013, S. 233).

Der österreichischen Tradition entspricht es (Art. 130 Abs. 2 a.F.), dass der VwGH bei kassatorischen Rechtsmittelentscheidungen darauf beschränkt ist zu prüfen, ob die kontrollierte Behörde Ermessen „im Sinne des Gesetzes“ geübt hat. Diese Bestimmung ist mit 1.1.2014 weggefallen (Raschauer, 2013, S. 231).

Nunmehr gilt für Rechtsmittel gegen administrative Bescheide Art. 130 Abs. 3 B-VG, der eine Beschwerde an das Verwaltungsgericht vorsieht. Nach dieser Bestimmung liegt „Rechtswidrigkeit nicht vor, soweit das Gesetz der Verwaltungsbehörde Ermessen einräumt und sie dieses im Sinne des Gesetzes geübt hat“. Die Kognitionsbefugnis des Verwaltungsgerichts bleibt demnach auf eine Art „Vertretbarkeitskontrolle“ beschränkt. Dies gilt nicht für Verwaltungsstrafsachen und in den zur Zuständigkeit des Verwaltungsgerichts des Bundes für Finanzen gehörenden Angelegenheiten. In diesen Angelegenheiten darf das

[85] In manchen Fällen bedeutet jedoch die Formulierung „kann“ sehr wohl eine Handlungspflicht der Behörde, was im Einzelfall durch Interpretation zu ermitteln ist.

[86] Für weitere Ausführungen zur Entwicklung der Ermessenslehre in Österreich und Deutschland sei auf die Artikel von Bullinger (1984) und Püttner (2008) verwiesen.

Verwaltungsgericht anstelle der kontrollierten Behörde eigenständig Ermessen üben. Eine ähnliche Bestimmung zur Kognitionsbefugnis des VwGH als Revisionsgericht findet sich in Art. 133 Abs. 3 B-VG (Raschauer, 2013, S. 231 ff.)[87]. Anders als die Verwaltungsgerichte ist der VwGH, unabhängig von der Rechtsmaterie, darauf beschränkt zu prüfen, ob das Verwaltungsgericht Ermessen „im Sinne des Gesetzes geübt" hat. Er ist daher nicht befugt, die Ermessensentscheidung des Verwaltungsgerichts zu ersetzen, wenn dieser das Ermessen anders geübt hätte.

Der VwGH ist auf Grundlage der alten Rechtslage davon ausgegangen, dass die mit der Interpretation und Anwendung von unbestimmten Gesetzesbegriffen verbundenen Fragen nicht als Ausübung von Ermessen zu sehen sind. Die seit 1.1.2014 in Kraft stehende Fassung des Art. 130 Abs. 3 B-VG verwendet ausschließlich den rechtstechnischen Begriff „Ermessen". Die Formulierung, dass die Gesetzgebung „von einer bindenden Regelung des Verhaltens der Verwaltung absehen" kann (Art. 130. Abs. 2 a.F.), wird nicht mehr gebraucht. Dies ist als verfassungsrechtliche Anerkennung der bisherigen Judikatur des VwGH zu verstehen, dass die Rechtskontrolle der Interpretation und Anwendung von unbestimmten Gesetzesbegriffen keinen Sonderregeln unterliegt (Raschauer, 2013, S. 233).

Hinsichtlich der Kategorisierung von Begriffen als „unbestimmter Gesetzesbegriff" ist anzumerken, dass bei genauerer Betrachtung jeder Rechtsbegriff, mit unterschiedlichem Ausmaß, in seinem Begriffskern bestimmt und im Begriffshof unbestimmt ist. Die traditionelle Hervorhebung der Kategorie der „unbestimmte Gesetzesbegriffe" erscheint in diesem Licht zumindest hinterfragbar (Raschauer, 2013, S. 234).

Es existieren Rechtsbegriffe, die rechtliche Abwägungen implizieren, womit in diesen eine Art der Ermessensübung gesehen werden kann. Die Formulierungen „die Bewilligung erteilt werden kann" und „die Bewilligung zu erteilen ist, wenn öffentliche Interessen nicht entgegenstehen" sind rechtlich gleichwertig. Folgt man dieser Meinung, dann kann Ermessen nicht nur durch die Verwendung des Wortes „kann", sondern auch durch andere

[87] Zur alten Rechtslage vor dem 1.1.2014 standen gegenteilige Aussagen der Senate des VwGH zur Überprüfung von Ermessensentscheidungen einander gegenüber. Es lag somit im richterlichen Ermessen des VwGH, in welcher Dichte und Intensität er die Prüfung, ob Ermessen im Sinn des Gesetzes geübt worden ist, vornahm (Raschauer, 2009, S. 213 ff.).

Formulierungen zum Ausdruck gebracht werden. Regelungen der skizzierten Art sind dann in weiterer Folge als Ermessensermächtigungen im Sinn von Art 130 Abs. 3 B-VG zu verstehen (Raschauer, 2013, S. 234 f.).

Es wurde herausgearbeitet, dass die Abgrenzung zwischen Ermessensbestimmungen und unbestimmten Gesetzesbegriffen nicht immer eindeutig durchführbar ist. Aus Sicht der Formalisierung bestehen zwischen beiden Kategorien, sofern man an diesen festhalten möchte, strukturelle Gemeinsamkeiten, die für einen einheitlichen Ansatz sprechen: Vage Gesetzesbegriffe und Ermessensklauseln können als ein System beweglicher Faktoren verstanden werden, die gegeneinander abgewogen werden müssen. Zukünftige Untersuchungen könnten sich der Formalisierung höchstgerichtlicher Entscheidungen annehmen, indem diese in einem beweglichen System aus „Dimensionen und Faktoren" abgebildet werden (Ashley, 1990; Sartor, 2006b).

4.9.3.3. Abschließende Bemerkungen

Zusammenfassend lässt sich feststellen, dass das Gesetz in der Regel danach trachtet, sämtliche Sachverhalte durch eine abstrakte Beschreibung – die Tatbestandselemente – zu erfassen. Mehrdeutigkeiten gehen oftmals auf Formulierungsmängel zurück, die sich durch Interpretation beseitigen lassen. Semantische Unbestimmtheiten sind jedoch auch eine legistische Methode, wenn Rechtsvorschriften flexibel genug sein sollen, um auf eine Vielzahl von Sachverhalten Anwendung finden zu können (Koller, 1997, S. 210). Mithilfe unbestimmter Gesetzesbegriffe und insbesondere von *Generalklauseln* fließen gesellschaftliche Wertungen und Prinzipien in die Rechtsanwendung mit ein.

Die in diesem Kapitel diskutierten unbestimmten Gesetzesbegriffe und Ermessensklauseln (im Straf-, Zivil- und Verwaltungsrecht) sind Beispiele für flexible Rechtsvorschriften, denn sie erlauben dem rechtsanwendenden Organ einen gewissen Spielraum bei seiner Entscheidung.

Es wurde gezeigt, dass das Ermessen in unterschiedlichsten Erscheinungsformen in der Rechtsordnung zu Tage tritt. Besonders schwierig ist die Bestimmung der Rechtsfolge, wenn deren Ausmaß nicht vom Ausprägungsgrad eines komparativen Merkmals des Tatbestands abhängig ist (Koller, 1997, S. 209 f.) oder es im Fall einer „Kann-Bestimmung" der Verwaltungsbehörde überhaupt freigestellt ist tätig zu werden (Raschauer, 2013, S. 226 f.).

Der Jurist löst derartige Fälle durch eine Abwägung aller Umstände des Einzelfalls und orientiert sich bei der Wahl der Rechtsfolge insbesondere am Ziel und Zweck des jeweiligen Gesetzes.

Unbestimmte Gesetzesbegriffe sind fester Bestandteil der Rechtsordnung und dienen der Zweckverwirklichung des jeweiligen Gesetzes. Ein Ansatz zur Formalisierung von Normen muss sich früher oder später mit vagen Begriffen auseinandersetzen. In welchem Ausmaß sich diese Problematik stellt, hängt letztendlich vom Ziel und Anspruch des jeweiligen Projekts ab. Es bleibt offen, ob vage Gesetzesbegriffe und Ermessensklauseln einer (vollständigen) Formalisierung zugänglich sind. Eine Möglichkeit wäre beispielweise, aus höchstgerichtlicher Judikatur Faktoren abzuleiten, mit denen sich das Ausmaß der Rechtsfolge, in Abhängigkeit von den Fakten des Einzelfalls, bestimmen lässt (Sartor, 2006b).

4.9.4. Rechtsfortbildung

Für den Rechtsanwender stellen sich Fälle als besondere Herausforderungen dar, wenn der äußerst mögliche Wortsinn einer Norm hinter dem Zweck bzw. Grund eines Gesetzes zurückbleibt und eine „planwidrige Unvollständigkeit“ vorliegt oder wenn Gesetze offensichtlich fehlerhaft, zweckwidrig oder ungerecht sind. In solchen Fällen kann die analoge Anwendung oder die teleologische Reduktion von Rechtsvorschriften geboten sein[88]. Neben einer stichhaltigen Begründung sind besonders hohe Anforderungen an den Rechtsanwender und dessen juristische Fähigkeiten gestellt. Methodisch wird dabei der Bereich der Interpretation verlassen und jener der Rechtsfortbildung betreten (Koller, 1997, S. 187):

„Rechtsfortbildung liegt vor, wenn konkrete Fälle auf eine Weise entschieden werden, die über den mögichen Wortsinn der positiv gegebenen Rechtsvorschriften hinausgeht oder diesen Vorschriften sogar widerspricht, sofern eine solche Entscheidung durch zureichende Gründe gerechtfertigt ist.“

Voraussetzung für die Rechtsfortbildung ist nach überwiegender Meinung das Vorliegen einer Gesetzeslücke, das heißt einer „planwidrigen Unvollständigkeit“ des positiven Rechts,

[88] Für das Privatrecht wird in § 7 ABGB die Möglichkeit von Gesetzeslücken anerkannt und ergänzende Rechtsfortbildung angeordnet. Darüber hinaus ist auch im öffentlichen Recht gemäß der Rechtsprechung des VwGH Analogie zulässig (VwGH 17.10.2012, 2012/08/0050).

gemessen am Maßstab der Zwecke und Werte der Rechtsordnung (Bydlinski, 2005, S. 59 ff.). Wenn das Gesetz über irgendeinen Sachverhalt schweigt, impliziert dies nicht zwingend das Vorliegen einer Rechtslücke. Nach ihrem Entstehungsgrund können folgende Arten von Rechtslücken unterschieden werden: Formulierungslücken, teleologische Lücken und Kollisionslücken (Koller, 1997, S. 228).

Die primäre, weil gesetzesnächste, Methode der Rechtsfortbildung ist der sogenannte Analogieschluss[89]. Die Rechtsfolge der Ausgangsnorm (Analogiegrundlage) wird auf einen von ihrem Tatbestand, auch im äußerst möglichen Sprachverständnis, nicht mehr erfassten Sachverhalt erstreckt, weil dieser gemäß dem maßgebenden Zweck der Ausgangsnorm gleich regelungsbedürftig erscheint (Bydlinski, 2005, S. 61). Man unterscheidet die *Gesetzes- oder Einzelanalogie* und die *Gesamt- oder Rechtsanalogie*. Der *Größenschluss* (argumentum a fortiori) ist mit dem Analogieschluss verwandt, unterscheidet sich jedoch von diesem dadurch, dass er statt einer teleologischen Gleichartigkeit eine wertmäßige Rangordnung der vorliegenden Fälle voraussetzt (Koller, 1997, S. 234).

Gleichsam spiegelbildlich zum Analogieschluss ist die *teleologische Reduktion* (Restriktion) zu sehen. Im Gegensatz zur Analogiesituation, in der die textliche Fassung des Tatbestandes hinter dem Zweck bzw. Grund des Gesetzes zurückbleibt, umfasst in diesen Fällen der Wortlaut im Begriffskern[90] der einschlägigen Regel mehr als ihrem Zweck entspricht. Die einschlägige Regel bedarf daher einer Einschränkung. Es liegt insofern eine, für die Rechtsfortbildung notwendige, „verdeckte“ (teleologische) Rechtslücke vor, als eine nach dem Gesetzeszweck notwendige Ausnahmeregel im Gesetz fehlt (Bydlinski, 2005, S. 69 f.).

Die Ausdehnung einer Regel durch Analogieschluss auf Sachverhalte, auf die sie streng genommen nicht anwendbar ist, kann in einem Computerprogramm durch zusätzliche Anweisungen repräsentiert werden. Selbiges gilt für den Fall der teleologischen Reduktion. Viel schwieriger wird die Angelegenheit, stellt man den Anspruch, dass die Maschine selbst den Analogieschluss im Einzelfall finden und anwenden soll. Die computergestützte juristische Entscheidungsfindung basierend auf Analogien wurde insbesondere in

[89] Dem Analogieschluss steht der Umkehrschluss (argumentum e contrario) direkt gegenüber und fordert, dass die in einer Rechtsvorschrift angeordnete Rechtsfolge nur unter den tatbestandlichen Voraussetzungen gelten soll.

[90] Bei einer „Überschreitung“ im Begriffshof kann die Regel einschränkend teleologisch interpretiert werden.

angloamerikanischen Rechtsordnungen schon früh untersucht (Ashley, 1990), da dort die Beurteilung eines Falles maßgeblich von der Ähnlichkeit zu Präzedenzfällen abhängig ist. Obwohl die Entscheidungsfindung basierend auf Fällen Gegenstand intensiver Forschungen war (Bench-Capon, 2012; Walton, 2010) und bis heute ist, konnte sich bislang kein Ansatz durchsetzen, der für die praktische Verwendung geeignet wäre.

4.9.5. Lösungsansätze

4.9.5.1. Fuzzy Logic

Schon in den 1970er-Jahren wurde untersucht, ob der Einsatz von „unscharfer" Logik, sogenannter „Fuzzy-Logik"[91], eine Lösung für das Problem der vagen Begriffe sein kann (Reisinger, 1975, S. 45 ff.). Diese ursprünglich von Zadeh (1965) entwickelte Theorie der „unscharfen Mengen" versucht die Übergänge zwischen einzelnen Begriffen zu berücksichtigen.

Die Idee der Fuzzy-Logik ist, Grenzfälle durch eine numerische Funktion zu repräsentieren, die den Grad der Zugehörigkeit eines Begriffes zu einem Tatbestandsmerkmal ausdrückt. Mithilfe der Mathematik lässt sich dies exakter definieren (Reisinger, 1975, S. 45):

Wir definieren zunächst eine beliebige Menge $X = \{x\}$. Die „unscharfe Menge" A in X ist bestimmt durch eine charakteristische Funktion $f_{\mathrm{A}}(x)$, welche jedem Element $x \in X$ einen Wert aus dem Intervall $[0,1]$ zuordnet. $f_{\mathrm{A}}(x)$ gibt dabei den „Grad der Mitgliedschaft" von x zu A an. Je näher dieser Wert bei eins liegt, desto höher ist der Grad der Mitgliedschaft von x zu A.

Es wird erkennbar, dass die traditionelle Definition der Menge jenen Fall bildet, bei dem $f_{\mathrm{A}}(x)$ nur die Werte null ($x \notin A$) oder eins ($x \in A$) annehmen kann. Die einfachen Fälle der juristischen Subsumtion lassen sich in diesem Sinne demnach als „zugehörig" (Wert eins) oder „nicht zugehörig" (Wert null) klassifizieren.

Auch wenn dieser Ansatz durchaus vielversprechend erscheint, vage Begriffe formal zu repräsentieren, gibt es einige Probleme, die im Folgenden diskutiert werden. Bevor auf diese eingegangen werden kann, seien an dieser Stelle die Schritte explizit dargestellt, die für die

[91] Engl. „fuzzy logic", für eine Definition sei auf Lackes (2013) verwiesen.

Formalisierung vager Begriffe nach dem Modell der Fuzzy-Logik notwendig sind (Reisinger, 1975, S. 47 f.):

1. Die Bestimmung der Bedeutungskomponenten, der Dimensionen, des Begriffs: Zuerst muss bestimmt werden, welche Eigenschaften eines Tatbestandsmerkmals dieses charakterisieren, um feststellen zu können, ob ein Sachverhaltselement[92] von diesem erfasst wird. In der Regel gestaltet sich das als schwierig, da Begriffe üblicherweise mehrdimensional sind. Beispielsweise hat der Begriff „Wald" nicht nur die Dimension „Anzahl der Bäume", sondern „durchschnittlicher Baumbestand", „durchschnittlicher Bewuchs mit Unterholz", „Verhältnis von Baum- und Unterholzbewuchs" und weitere.
2. Die operationale Definition der Bedeutungskomponenten: Zusammengefasst geht es darum, präzise Regeln zu finden, mit deren Hilfe sich eine Übersetzung von faktischen Gegebenheiten, also den beobachteten Tatsachen des Sachverhalts, in einen numerischen Wert x durchführen lässt.
3. Die Definition der charakteristischen Funktion $f_{\mathrm{A}}(x)$: Wie oben bereits ausgeführt wurde, muss eine mathematische Funktion definiert werden, durch die die Mitgliedschaft von x zu A ermittelt werden kann. Dies ist insbesondere bei mehrdimensionalen Begriffen schwierig.
4. Die Angabe des Schwellenwertes bzw. der Schwellenwerte: Zuletzt ist es erforderlich, dass entsprechende Schwellenwerte angegeben werden, die den Übergang vom einen zum anderen Begriff ausdrücken[93].

Das Modell der Fuzzy-Logik setzt voraus, dass präzise, objektiv nachvollziehbare Regeln gefunden werden können, mit deren Hilfe der Grad der Mitgliedschaft von x zu A festgestellt werden kann. Praktisch ist die Definition einer solchen Funktion allerdings ein schwieriges Unterfangen, wie nun gezeigt wird.

Erstens ist bereits eine objektiv nachvollziehbare Bestimmung der charakteristischen Funktion $f_{\mathrm{A}}(x)$ schwierig. Wie soll beispielsweise die Größe einer Person in einen

[92] Ein Sachverhaltselement kann hier ein Vorgang, ein Objekt, usw. sein.

[93] Vgl. dazu beispielsweise die Ausführungen bezüglich des Überganges von „kurzer" zu „mittlerer" Wartezeit bei Verkehrsunfällen (Joerden, 2010, S. 373 ff.).

Prozentwert, der den Grad der Mitgliedschaft an der Eigenschaft „groß“ ausdrückt, übersetzt werden?

Besondere Schwierigkeiten bereitet die Definition bei mehrdimensionalen Begriffen. So sind die einzelnen Dimensionen eines Begriffes in der Regel selbst wiederum vage und besitzen demnach ihre eigene charakteristische Funktion $f_{\mathrm{i}}(x)$. In einem zweiten Schritt sind also die einzelnen Funktionen $f_{\mathrm{i}}(x)$ zur einzigen Gesamtfunktion $f_{\mathrm{A}}(x)$ zu aggregieren. Wie Joerden (2010, S. 375) es ausdrückt sind *„…alle [...] gefundenen Zugehörigkeitsfunktionen in ein einheitliches Modell zu integrieren.“* Für diesen Zweck bieten sich verschiedene Mittelwerte an, wobei zusätzlich eine Gewichtung der einzelnen Dimensionen möglich ist.

Zweitens existieren keine präzisen Regeln, wie Fakten des Sachverhalts in die Elemente des Tatbestands, genauer in einen numerischen Wert des jeweiligen Bedeutungselements, zu übersetzen sind. Oftmals sind die einzelnen Bedeutungselemente selbst vage Begriffe, die ihre eigene charakteristische Funktion zur Bestimmung benötigen. Zum Problem der „Operationalisierung“ siehe Reisinger (1975, S. 48 f.).

Drittens sei angemerkt, dass charakteristische Funktionen in der Regel stark vom jeweiligen Kontext abhängig sind. Beispielsweise wird ein Mann, der so groß ist wie eine große Frau, nicht zwingend als „groß“ bezeichnet, oder ein großer Mosquito ist kleiner als ein kleiner Elefant etc. Dies ist wiederum kritisch zu sehen, da die Repräsentation des Kontexts selbst mit einer Reihe von ungelösten Problemen im Bereich der Künstlichen Intelligenz verbunden ist (Franklin, 2012, S. 227).

Juristen lösen die letzten beiden Probleme in der Regel nicht durch eine explizite Operationalisierung von Gesetzesbegriffen, sondern unter „Abwägung aller Sinnzusammenhänge“, gemäß der hermeneutischen Methode (Reisinger, 1975, S. 49).

Viertens ist relevant, welche Schwellenwerte gewählt werden, um auszudrücken, ob ein Sachverhaltselement noch unter einen Begriff des Tatbestands fällt oder nicht. Auch dafür gibt es allerdings keine verbindlichen Richtsätze, die objektiv nachvollziehbare Werte begründen. Ab wann gilt eine Person beispielsweise als „groß“? Diese Frage könnte z.B. mithilfe empirischer Studien beantwortet werden.

Besonders schwierig ist die Definition von Schwellenwerten bei mehrdimensionalen Begriffen, da hier sich der Gesamtprozentwert aus den aggregierten charakteristischen

Funktionen ergibt. Die Anforderungen an die Nachvollziehbarkeit der gewählten Schwellenwerte sind aus diesem Grund besonders hoch, sollen diese nicht willkürlich erscheinen.

Auch hier zeigt sich wieder deutlich die Relativität des formalen Modells, da keine präzisen Regeln für die Definition der charakteristischen Funktion $f_{\mathrm{A}}(x)$ etc. existieren. Die Definition von Schwellenwerten hängt damit von der subjektiven Wertung des Experten ab, der das formale Modell erstellt.

Zusammengefasst kann gesagt werden, dass es zwar möglich ist, ein Kontinuum durch Zahlen zu repräsentieren, dies aber ein eher ungeeigneter Ansatz ist. Es ist äußerst schwer, die „richtigen" Zahlen zu finden. Ausschließlich das menschliche Gehirn ist in der Lage, effizient mit unpräzisen und intuitiven Informationen umzugehen. Wie das Gehirn die Leistung bewerkstelligt, ist derzeit noch unbekannt (Franklin, 2012, S. 239 f.).

Fraglich ist, ob die Fuzzy-Logik auch für die oben diskutierten Fälle eine Lösung bietet, in denen der Behörde ein Ermessenspielraum eingeräumt wird. Eine derartige Lösung setzt voraus, dass die einzelnen Kriterien, die bei der Ermessensentscheidung im Einzelfall gegeneinander abgewogen werden müssen, durch Interpretation aus dem Gesetzestext gewonnen werden können. Sobald die einzelnen Dimensionen klargestellt, voneinander abgegrenzt und mit einer Gewichtung versehen wurden, ist denkbar, dass Fuzzy-Logik auch hier zur Lösung beitragen kann. Dafür ist erforderlich, dass der konkrete Einzelfall mit früheren Fällen verglichen wird, um eine sachgerechte Lösung zu finden. Neuere Entwicklungen im Bereich der Fuzzy-Logik deuten jedenfalls darauf hin, dass diese bei Abwägungsentscheidungen, z.B. bei der Beurteilung der Ähnlichkeit zu Präzedenzfällen, wichtige Hilfestellungen leisten kann (Joerden, 2010, S. 378; Sandro, 2011, S. 227 f.).

Kommt man im Einzelfall zum Ergebnis, dass der Einsatz von Fuzzy-Logik eine adäquate Lösung für ein rechtliches Problem ist, so muss man sich auch vergegenwärtigen, dass die Anforderungen an die rechtliche Begründung einer Entscheidung in diesem Fall besonders hoch sind. Aus der Begründung muss schlüssig hervorgehen, warum eine Rechtsregel für anwendbar erachtet wurde oder eben nicht.

4.9.5.2. Der pragmatische Ansatz

Die Ausführungen im vorherigen Abschnitt machen deutlich, dass trotz der anfänglichen Euphorie es (derzeit) meines Erachtens unwahrscheinlich ist, dass die Subsumtion von Sachverhalten unter vage Tatbestandselemente vollständig automatisiert werden kann.

Bereits Kinder sind relativ früh in der Lage, im Gegensatz zu Computern bei Vergleichen auch den jeweiligen Kontext zu berücksichtigen. Es scheint sogar, dass relative Wörter in der Regel einfacher gelernt werden können als absolute und endliche Begriffe, wie „50 cm lang“ (Franklin, 2012, S. 227). Diese Fähigkeit des menschlichen Gehirns, mit vagen Informationen umzugehen und trotzdem zu einer sachgerechten Lösung zu gelangen, wurde von Computern bisher nicht erreicht.

Die Sprache richterlicher Entscheidung ist hauptsächlich die Sprache der Logik. Hier sei auf die Arbeit von Schrems zur Rolle der Logik, insbesondere bei der Begründung von Urteilen, verwiesen (Schrems, 2001). Hinter jeder Entscheidung stehen jedoch Werturteile und Bewertungen der von den Parteien vorgebrachten Behauptungen (Franklin, 2012, S. 236), die nicht durch logische Aussagen repräsentiert werden können. Annahmen allgemeiner Art werden in einem Verfahren grundsätzlich nicht ausdrücklich vorgebracht, sondern der Richter setzt diese implizit bei seiner Entscheidung voraus. Die Entscheidungsfindung ist oftmals von einem sehr umfangreichen Allgemeinwissen über die Welt, wie sich Menschen verhalten, volkstümlichen Gebräuchen etc. abhängig (Franklin, 2012, S. 231 f.).

Ein recht einfacher und doch wirkungsvoller Ansatz zur Lösung dieses Dilemmas ist der Einsatz menschlicher (juristischer) Fähigkeiten. Die soeben überblicksmäßig und nicht abschließend dargestellten Problemkreise verdeutlichen, dass der „pragmatische Ansatz“ nicht bloß aus reiner Bequemlichkeit eine sinnvolle Lösungsmethode darstellt.

In Grundzügen skizziert funktioniert dieser Ansatz wie folgt: Sollten während der Durchführung eines Verfahrens Probleme der in diesem Abschnitt diskutierten Art auftauchen, legt das Computersystem den Fall einem Menschen zur Bearbeitung vor. Ein Jurist kann beispielsweise einen Fall sichten und beurteilen, ob ein vager Gesetzesbegriff des Tatbestands erfüllt ist oder nicht.

Als Beispiel sei das von Fellinger und Philipps entwickelte Programm „Sentencing Supporter“ erwähnt (Fellinger & Philipps, 2003). Dieses Programm soll den Richter bei der

Strafbemessung unterstützen. Das Ausmaß der Strafe ist von der Verwerflichkeit der Tat, insbesondere vom Vorsatz des Täters, abhängig.

Neben allgemeinen Angaben zum Fall werden in das System mehrere vorgegebene Kriterien zur Bewertung des Falls eingegeben. Der Richter gibt dabei den Erfüllungsgrad jedes Kriteriums aus dem Intervall [0;1] im vorliegenden Fall ein. Zusätzlich muss der Effekt jedes Kriteriums aus der Menge {-1,1} eingegeben werden. Die Erfüllungskriterien sind dabei an § 46 dStGB[94] angelehnt, es handelt sich unter anderem um die Art der Ausführung, die Pflichtwidrigkeit, die Beweggründe, das Vorleben und das Nachtatverhalten des Täters.

Jedes der Kriterien ist mit einer Gewichtung aus dem Intervall [0;2] versehen. Aus den eingegebenen Daten pro Kriterium, das sind Erfüllungsgrad, Effekt und Gewichtung, wird die *Erfüllung des Strafausmaßes* in Prozent [0;100] errechnet. Das Ergebnis wird schließlich in einem Sterndiagramm visualisiert, das recht anschaulich das Ausmaß der Strafe im Verhältnis zum maximalen Strafrahmen darstellt.

Der pragmatische Ansatz steht mit Bestrebungen zur Entwicklung einer vollautomatisierten Verfahrensdurchführung in einem gewissen Widerspruch und sollte daher nur wenn absolut notwendig eingesetzt werden. Das präsentierte Beispiel eines solchen Programms zeigt jedoch, dass dieser Ansatz gerade für schwierige Fälle von großer Wichtigkeit ist.

4.10. Diskussion

Kommen wir nun zu einer abschließenden Zusammenfassung und Diskussion der Erwägungen. Wir haben unseren Diskurs in den 1970er-Jahren begonnen und das damals vorherrschende primär logische Verständnis der Formalisierung behandelt. Auf dieser theoretischen Grundlage wurde die Axiomatisierung einer eingehenden Kritik unterzogen und gezeigt, dass die Anwendung deduktiver Schlussfolgerung als Modell des Denkens für die Lösung rechtlicher Probleme inadäquat erscheint.

Basierend auf den Erkenntnissen des ersten Teils dieses Kapitels wurde der Ansatz, den die Dissertation verfolgt, als „ontologische Formalisierung“ qualifiziert.

[94] Deutsches Strafgesetzbuch.

Der zweite Teil des Kapitels stellte die für die Automatisierung wichtigsten Grundlagen der Rechtsanwendung dar und zeigte im Anschluss daran die Probleme und potentiellen Lösungsmöglichkeiten auf. Insbesondere zwei Erkenntnisse sollen hervorheben werden:

1. In vielen Fällen sind mehrere Interpretationen eines Begriffes vertretbar. Die Notwendigkeit bessere Argumente zu finden, die eine bestimmte Interpretation des Textes unterstützen, zeigt, dass die juristische Entscheidungsfindung im Kern auf Argumentation beruht. Es setzt sich jene Partei in einem Prozess durch, der es gelingt die besseren Argumente für ihr Vorbringen zu finden.
2. Es ist kaum zielführend einen Begriff unabhängig von seinem Kontext zu betrachten. Oftmals kann die Bedeutung von Formulierungen erst aus dem jeweiligen Kontext erschlossen werden, in dem diese gebraucht wird.

Die präsentierten Lösungsansätze sind keinesfalls abschließend zu verstehen. Darüber hinaus wäre es denkbar, dass Modelle computergestützter Argumentation (Gordon & Walton, 2006; Gordon, 2010a) und auf Faktoren und Dimensionen basierende Systeme (Sartor, 2006b) zum Einsatz kommen, um unbestimmte Gesetzesbegriffe und Ermessensklauseln zu formalisieren. Zukünftige Forschungen könnten anhand einer prominenten Rechtsvorschrift die Tauglichkeit dieser Ansätze evaluieren.

Viele Probleme im Zusammenhang mit der Automatisierung juristsicher Entscheidungsfindung entspringen der Tatsache, dass Computer das Recht noch immer nicht wie Menschen verstehen können (Franklin, 2012; Schweighofer, 2011). Computer arbeiten mit uninterpretierten Symbolen, die sie gemäß fest definierten Regeln verarbeiten. Auch wenn die Verwendung der Logik in anderen Bereichen, wie beispielsweise der Mathematik, erfolgreich war, wird heute immer deutlicher, dass menschliches Verstehen viel mehr als bloß blindes Befolgen von Regeln ist.

Der menschliche Verstand ist in der Lage, die relevanten Eigenschaften eines Sachverhalts selbstständig zu abstrahieren und in Gedanken zu ordnen, um so auch in schwierigen Fällen eine sachgerechte Lösung zu finden. Computer sind von einer derartigen Leistungsfähigkeit weit entfernt. Aus diesem Grund ist es auch so schwierig, für die im Recht häufig vorkommenden Ungenauigkeiten, wie vage Gesetzesbegriffe oder Ermessensklauseln, adäquate Formalisierungen zu entwickeln.

Auch wenn dies vom heutigen Standpunkt aus mehr als unwahrscheinlich erscheint, könnte man dahingehend nach einer vermeintlichen Lösung suchen, dass der Gesetzgeber selbst, neben dem Gesetzestext, sozusagen eine formale Version desselben erlässt. Diesem Ansatz stehen Bedenken gegenüber, die in Abschnitt 4.4.2.5. schon der Forderung nach einem deduktiven Beweis von richterlichen Entscheidungen entgegen gehalten wurden. Durch ein solches formales Modell würde die Entscheidungsgewalt des Richters bzw. der Verwaltungsbehörde aus Sicht des Prinzips der Gewaltenteilung in nicht hinnehmbarer Weise beschränkt werden. Der Gesetzgeber ist nicht zur Entscheidung von Einzelfällen berufen, diese obliegt den Gerichten und Verwaltungsbehörden.

Ferner würde ein derartiger Ansatz wohl unweigerlich zur Anlassfallgesetzgebung führen, da ein formales Gesetzesmodell keinesfalls für alle Fälle eine sachgerechte Lösung von vornherein bieten kann. Man denke nur daran, dass zur Interpretation von vagen Begriffen auch gesellschaftliche Wertungen einfließen, die sich im Laufe der Zeit ändern können. Es ist demnach auch aus rechtspolitischer Sicht nicht völlig einwandfrei, nach einem vom Gesetzgeber selbst legitimierten formalen Modell des Gesetzes zu streben (Gordon, 1994a, S. 25 ff.).

Diese Ausführungen werfen die Frage auf, ob es überhaupt einen Sinn hat, weiter an einer Formalisierung festzuhalten?

Dazu ist zu bemerken, dass die Erfolgsaussichten einer Formalisierung maßgeblich vom jeweiligen Gesetz abhängig sind, das formalisiert werden soll. So ist zu erwarten, dass das Verwaltungsrecht aufgrund seiner höheren Regelungsdichte weniger Ungenauigkeiten wie beispielsweise das Zivilrecht aufweist. Die Formalisierung und Automatisierung (einfacher) Verwaltungsgesetze, die einen eher mathematischen Inhalt besitzen, scheint am erfolgversprechendsten.

Wie auch immer man die Erfolgssausichten einer Formalisierung beurteilen möge, sie ist jedenfalls ein recht aufwändiges Unternehmen. Wozu soll man daher Gesetze überhaupt formalisieren?

Die Antwort auf diese Frage ist zweigeteilt, da sich die Vorteile für die Verwaltung und die Bürger unterschiedlich darstellen.

Der erste Vorteil, insbesondere für die Verwaltung, liegt in der Effizienzsteigerung der Verfahren, da zumindest einfache Fälle von einer Software beurteilt werden können. Eine Effizienzsteigerung ist ein wesentliches Ziel von E-Government und gerade in Zeiten einer angespannten wirtschaftlichen Lage ein geeignetes Mittel, um die Staatsausgaben zu senken. Natürlich darf nicht vergessen werden, dass sich die anfänglichen Ausgaben für die Implementierung solcher Systeme erst amortisieren müssen, bis diese „gewinnbringend" arbeiten. Ein formales Modell kann überdies auch einfacher kommuniziert werden als ein Gesetzestext, da es grundsätzlich frei von Ungenauigkeiten ist. Damit sollte es einfacher erweitert und adaptiert werden können, sollte dies, z.B. im Fall von Gesetzesnovellen, erforderlich sein.

Für die Bürger ist der wohl wesentlichste Vorteil einer Formalisierung die Verkürzung der Verfahrensdauer. Sobald sämtliche Daten ins System eingegeben bzw. aus Registern abgefragt wurden, kann die Software automatisiert das Ergebnis, in der Regel einen Bescheid, erstellen. Es ist denkbar, dass der Antragsteller sich das Ergebnis „vorberechnen" lässt, um zu prüfen, ob dieses seinen Erwartungen entspricht, und um gegebenenfalls weitere zweckdienliche Angaben einbringen zu können.

Ein weiterer Vorteil ist die Nachvollziehbarkeit und Überprüfbarkeit der Entscheidung. Das System wendet vorgegebene Regeln an, um zu einer Entscheidung zu gelangen. Abgesehen davon, dass die jeweilige Entscheidung ohnehin eine Begründung enthalten muss, sind damit die Regeln jederzeit offen zugänglich für eine kritische Prüfung. Es wird überdies erwartet, dass die formalisierte Form des Gesetzes eine erhöhte Übersichtlichkeit bietet.

Zusätzlich ist denkbar, dass Entscheidungen anonymisiert in maschinenlesbarer Form veröffentlicht werden, um automatisierte Vergleiche zu ermöglichen. Damit könnte überprüft werden, ob eine Verwaltungsbehörde das durch Art. 7 B-VG postulierte Gebot, „gleiche Fälle gleich zu behandeln", ausreichend berücksichtigt hat. Der Gleichheitssatz erlangt insbesondere bei Ermessensentscheidungen eine herausragende Bedeutung (Raschauer, 2009, S. 212). Insgesamt führt die Formalisierung daher zu einer Steigerung der Transparenz.

Die Entscheidung, ob man einen Rechtsbereich formalisiert, wird maßgeblich vom Aufwand eines derartigen Unternehmens abhängen. Es gilt, den Aufwand gegen den erwarteten

Nutzen abzuwiegen. Praktisch wird die monetäre Komponente, d.h. das Einsparungspotential, wohl die größte Rolle bei derartigen Abwägungen spielen.

Hält man an der Formalisierung als solcher fest, so ist zu bemerken, dass es stets die Aufgabe der Rechtswissenschaft ist – nicht der Formalwissenschaften – zu überprüfen, ob eine Formalisierung juristisch vertretbar ist, insbesondere ob ein System sachgerechte Entscheidungen produziert. Gegebenenfalls ist ein entsprechendes Korrektiv vorzusehen, um „fehlerhafte“ Entscheidungen zu beheben. In der Regel wird ein solches Korrektiv auf eine Eingriffsmöglichkeit durch einen Sachbearbeiter hinauslaufen, dem die endgültige Entscheidungsbefugnis zukommt.

Im Zusammenhang mit der Automatisierung besteht die Gefahr, dass der Staat günstig, unter Einsatz geringer finanzieller Mittel, viele Entscheidungen erlassen kann, der Rechtsunterworfene diese aber im Gegensatz dazu relativ teuer im Instanzenzug bekämpfen muss.

Dies wirft die Frage nach einer angemessenen Risikobewertung für automatisierte Verfahren auf. Eine solche ist notwendig, um die Richtigkeit eines gewünschten Prozentsatzes der Entscheidungen zu gewährleisten. Diese Problematik umfasst insbesondere die Schaffung einer entsprechenden Methode zur Bewertung des Risikos, dass ein automatisiert durchgeführtes Verfahren eine fehlerhafte Entscheidung produziert. Für Untersuchungen zur Risikoanalyse im Recht sei auf die Arbeit von Preiß (2012) verwiesen.

Aufgrund der Relativität der Formalisierung ist es schwierig, einheitliche Theorien und Modelle für unterschiedliche Rechtsbereiche zu entwickeln. Schränkt man die Bemühungen jedoch auf einen konkreten Rechtsbereich ein, in diesem Fall das Verwaltungsrecht, so scheint ein einheitlicher Ansatz aufgrund der relativ hohen Einheitlichkeit der Rechtsmaterie durchaus im Bereich des Möglichen zu liegen.

Wie man auch immer zur Formalisierung und Automatisierung des Verwaltungsrechts stehen mag, eine eingehende wissenschaftliche Auseinandersetzung fehlt bislang. Eine solche ist meines Erachtens aber notwendige Voraussetzung, um einen einheitlichen Ansatz für die Formalisierung des Verwaltungsrechts in Zukunft entwickeln zu können. Damit könnten unnötige Mehrfachentwicklungen und Fehler bei der Formalisierung vermieden werden.

5. Wissensrepräsentation

5.1. Einleitung

Die Wissensrepräsentation kann als Kern des Forschungsgebiets der Künstlichen Intelligenz bezeichnet werden (Sowa, 1999). Bevor auf die Wissensrepräsentation als Forschungsdisziplin selbst eingegangen wird, scheinen überblicksmäßige Ausführungen zur Künstlichen Intelligenz angebracht.

5.1.1. Künstliche Intelligenz

Die „Künstliche Intelligenz“[95] ist ein Teilgebiet der Informatik, das sich mit Forschungen beschäftigt, die es Computern ermöglichen sollen, sich intelligent zu verhalten (Thomason, 2013). Bereits die Definition des Begriffs „Intelligenz“ ist schwierig.

Intelligenz zeigt sich in beinahe jeder Form menschlichen Handelns, z.B. die Fähigkeit sich anzupassen, zu lernen, zu verstehen, komplexe Beziehungen und Gesellschaften zu bilden und mit seiner Umwelt zu kommunizieren (Pradeesh Hosea, Harikrishnan, & Rajkuma, 2011).

Turing hat in den 1950er-Jahren eine – auch heute noch gültige – operationale Definition der „Intelligenz“ vorgeschlagen, den sogenannten „Turing Test“. Der Computer besteht den Test, wenn ein menschlicher Fragesteller, nachdem er ein paar schriftliche Fragen gestellt hat, nicht mehr unterscheiden kann, ob die schriftliche Antwort von einem Menschen oder Computer stammt.

Der Computer müsste demnach folgende Fähigkeiten besitzen, die im Wesentlichen die Teilbereiche der KI darstellen (Russel & Norvig, 2010, S. 2 f.):

- Natural Language Processing: Eine Sprachverarbeitung ist notwendig, damit der Computer (auf Englisch) kommunizieren kann.
- Knowledge Representation: Um zu speichern, was die Maschine weiß oder hört.
- Automated Reasoning: Um die gespeicherten Informationen verwenden und daraus neue Schlüsse ziehen zu können.
- Machine Learning: Die Maschine muss die Fähigkeit haben, sich an neue Umstände anzupassen, und in der Lage sein, Muster zu erkennen und vorauszusagen.

[95] Engl. „Artificial Intelligence“ (AI), im Folgenden wird die deutsche Abkürzung KI verwendet.

Für das Bestehen des erweiterten „Turing Tests“ muss die Maschine zusätzliche Fertigkeiten aufweisen:

- Computer Vision: Um Objekte wahrnehmen zu können.
- Robotics: Um Objekte verändern und bewegen zu können.

Der erweiterte Turing Test soll zusätzlich die Wahrnehmungsfähigkeit der Maschine testen, indem ein visuelles Signal verwendet wird. Überdies hat der Fragesteller die Möglichkeit, ein Objekt durch eine Klappe an die Maschine zu übergeben, um deren Fähigkeit zur Handhabung physischer Objekte zu testen. Es wurde jedoch bewusst darauf verzichtet, dass der Fragesteller direkt mit der Maschine physisch interagiert. Die physische Stimulation einer Person ist nämlich kein Wesensmerkmal der Intelligenz.

Die sechs Fähigkeiten repräsentieren im Wesentlichen das, was das Forschungsgebiet Künstliche Intelligenz ausmacht. Turing verdient Respekt für einen Test, der auch nach mehr als 60 Jahren noch relevant ist. Dennoch wurde bisher kaum Aufwand betrieben, den Test zu bestehen. Die Forscher konzentrieren sich vielmehr auf die Untersuchung der zugrundeliegenden Prinzipien der Intelligenz, nicht so sehr darauf, intelligentes Verhalten nachzuahmen. Luftfahrtingenieure streben beispielsweise nicht danach Maschinen so zu bauen, dass diese wie Tauben fliegen und diesen täuschend ähneln, sodass man sie für echte Tauben halten könnte.

Das Ziel der KI-Forschung ist die Simulation oder Annäherung von Computern an die menschliche Intelligenz. Zu einem Großteil besteht dies aus der Entwicklung von computergestützten Denkprozessen, die es Maschinen ermöglichen Probleme zu lösen. Roboter sind ein gutes Beispiel dafür. Man stelle sich vor, was ein Roboter wissen muss, bevor dieser in der Lage ist, mit seiner Umgebung wie ein Mensch zu interagieren?

Der Roboter benötigt zunächst eine sehr exakte interne Repräsentation der Wirklichkeit. Er muss auf Wahrnehmungen entsprechend reagieren, wissen wie er seine Ziele erreichen kann, welche Gegenstände er dafür verwenden kann, welche Arten von Gegenständen existieren, u.v.m.

Die maschinelle Problemlösung findet nicht im menschlichen Gehirn, sondern in einem Computer statt. Aus diesem Grund ist sie anderen Einschränkungen unterworfen, wie der

Speicherkapazität, der Prozessorgeschwindigkeit und Operationen der symbolischen Mathematik.

Wo ein Mensch auf ein breites Spektrum von Problemlösungsstrategien zurückgreifen kann, ist die Maschine auf unergiebiges und unvollständiges Wissen beschränkt, das bloß für die Lösung bestimmter Probleme ausreicht. Jede Form der künstlichen Intelligenz ist demnach von Wissen und einer entsprechenden Repräsentation dieses Wissens abhängig. Die Wissensrepräsentation nimmt sich dieses grundlegenden Problems der Künstlichen Intelligenz an.

5.1.2. Wissensrepräsentation

Die Forschungen auf dem Gebiet der Wissensrepräsentation versuchen die Probleme zu lösen, die mit der Implementierung von Wissen in ein Computersystem verbunden sind (Sowa, 1999). Das Ziel dieser Forschungen ist die Strukturierung und Aufbereitung von Wissen, um automatisierte intelligente Entscheidungen zu ermöglichen. In diesem Sinne ist die Wissensrepräsentation *das* grundlegende Forschungsgebiet der Künstlichen Intelligenz (Schweighofer, 1999, S. 29 ff.).

Nach Davis, Shrobe und Szolovits (1993, S. 17) kann *Wissensrepräsentation* am besten anhand von fünf Rollen verstanden werden, die diese bei der Modellierung von Wissen spielt:

„First, a knowledge representation is most fundamentally a surrogate, a substitute for the thing itself, that is used to enable an entity to determine consequences by thinking rather than acting, that is, by reasoning about the world rather than taking action in it.

Second, it is a set of ontological commitments, that is, an answer to the question, in what terms should I think about the world?

Third, it is a fragmentary theory of intelligent reasoning expressed in terms of three components: (1) the representation's fundamental conception of intelligent reasoning, (2) the set of inferences that the representation sanctions, and (3) the set of inferences that it recommends.

Fourth, it is a medium for pragmatically efficient computation, that is, the computational environment in which thinking is accomplished. One contribution to this pragmatic efficiency

is supplied by the guidance that a representation provides for organizing information to facilitate making the recommended inferences.

Fifth, it is a medium of human expression, that is, a language in which we say things about the world."

Eng in Zusammenhang mit der Wissensrepräsentation steht die Entwicklung von „wissensbasierten Systemen" (WBS). Derartige Systeme sind in der Lage, durch Anwendung von Methoden der KI mit dem modellierten Wissen Schlussfolgerungen durchzuführen. Sofern diese mit rechtlichem Wissen arbeiten, werden sie „juristische wissensbasierte Systeme" (JWBS) genannt[96]. Wissensbasierte Systeme benötigen für den Problemlösungsprozess eine explizite Repräsentation des erforderlichen Wissens.

Die Forschung, die sich mit der Konstruktion von juristischen wissensbasierten Systemen beschäftigt, nennt man *Legal Knowledge Engineering*[97], das ein Teil des Forschungsgebiets Künstliche Intelligenz und Recht[98] (KI und Recht) ist. Das Gebiet *Künstliche Intelligenz und Recht* ist eine Teildisziplin der Künstlichen Intelligenz (KI) und beschäftigt sich mit der Entwicklung von Systemen, die zu „intelligentem" Verhalten fähig sind.

Ein WBS besteht im Wesentlichen aus zwei Komponenten. Erstens aus einer (juristischen) Wissensbasis, die die symbolische Repräsentation des Wissens darstellt, das zur Lösung eines bestimmten Problems oder einer Klasse von Problemen verwendet wird. Die Wissensbasis ist die Summe aller Wissensrepräsentationen, die von einem WBS verwendet werden (Boer, 2009, S. 18). Der zweite Teil ist die Inferenzmaschine, die Schlussfolgerungen mit dem in der Wissensbasis gespeicherten Wissen durchführt.

Dieser Aufbau unterscheidet sich wesentlich von der Struktur „gewöhnlicher" Computerprogramme, bei denen das Wissen zur Problemlösung nicht vom im Algorithmus oder Code abgebildeten Problemlösungsprozess getrennt ist.

Die Entwicklung von WBS ist mit einer Reihe von Problemen verbunden, die im Wesentlichen eine der beiden Komponenten betreffen. Die Erforschung der Methoden und

---

[96] Angemerkt sei, dass im Kontext der Dissertation stets letztere gemeint sind und die Begriffe „wissensbasiertes System" und „juristisches wissensbasiertes System" synonym verwendet werden.

[97] Eher unüblich ist die deutsche Übersetzung „juristische Wissensmodellierung".

[98] Engl. „Artificial Intelligence and Law".

Ansätze zur Repräsentation von Wissen in Computersystemen ist Gegenstand des Gebiets *Wissensrepräsentation*. Der zweite Problemkreis betrifft die Erforschung von Methoden zur Entwicklung von Programmen, die mit diesem Wissen Schlussfolgerungen (Inferenzen) durchführen, und wird *automatisierte Entscheidungsfindung* (auch *automatisiertes Schließen*) genannt.

Die Wissensrepräsentation ist eine wichtige Aufgabe, die es bei der Konstruktion von WBS zu lösen gilt, und steht demnach in engem Zusammenhang mit dem „Knowledge Engineering", einem Prozess zur Konstruktion solcher Systeme.

In dieser Dissertation wird der Schwerpunkt der Untersuchung auf die Konstruktion von Modellen des Rechts gelegt. Dies bedeutet jedoch keinesfalls, dass die praktische Dimension – die Implementierung von wissensbasierten Systemen – dabei außer Acht gelassen wird. Insbesondere für das E-Government ist die praktische Umsetzbarkeit theoretischer Überlegungen von größter Bedeutung, da in Zukunft ein noch stärkerer Bedarf an „intelligenten" Softwaresystemen zur Unterstützung der Verwaltung bestehen wird. Der theoretische Diskurs kann daher nur der erste – jedoch sehr wichtige – Schritt zur Implementierung neuer verbesserter Systeme sein.

Zusammengefasst ist das zentrale Thema der KI-Forschungen die Wissensrepräsentation, ein Forschungsgebiet, das sich mit der Modellierung von Wissen zur Problemlösung befasst (Gordon, 1994a, S. 19):

„Artificial intelligence is the study of computer models of individual and group problem solving processes."

Die bis heute andauernde Diskussion im Bereich der KI – ob es eines Tages möglich sein wird Maschinen zu konstruieren, die dem Menschen ähnlich sind – kann weitgehend vermieden werden, wenn man für die Definition der Intelligenz einen modellbasierten Ansatz verfolgt:

„My standpoint is that artificial intelligence is about modeling, not constructing, intelligence. As models of this kind have enormous practical and theoretical utility, there is no need to engage in a debate about the possibility of intelligent machines." (Gordon, 1994a, S. 20)

Nach der Meinung von Gordon geht es demnach nicht um die *Konstruktion*, sondern um die *Modellierung* von Intelligenz. Die Erstellung von adäquaten Repräsentationen des Wissens,

die es Maschinen ermöglichen, bestimmte Probleme zufriedenstellend zu lösen, ist jedoch keinesfalls eine triviale Aufgabe.

5.2. Über Modelle

Im vorherigen Abschnitt wurde eine kurze Einführung in das Forschungsgebiet der Künstlichen Intelligenz gegeben und die Wichtigkeit von adäquaten Wissensrepräsentationen – den Modellen – betont. Aus diesem Grund scheinen an dieser Stelle ein paar Ausführungen zur Rolle von Modellen im Bereich der KI angebracht. Die folgende Darstellung folgt den Ausführungen von Gordon (1994a, S. 19 ff.).

Ein Modell ist im Wesentlichen ein beschränktes Abbild der Wirklichkeit, das in der Regel nicht alle Attribute des Originals wiedergibt. Als Einstieg seien zunächst die Dimensionen in Erinnerung gerufen, anhand derer Modelle unterschieden werden können: (1) Der Zweck, (2) das modellierte Objekt, (3) die analytischen, empirischen und normativen Aussagen.

Modelle können theoretischen oder praktischen Zwecken dienen. Ein theoretisches Modell dient der Verifizierung einer Theorie. Dies geschieht aber nicht etwa durch den Beweis von Theoremen, sondern das Modell wird modifiziert, um zu beobachten, wie es sich verhält. Praktische Modelle dienen hingegen einem bestimmten Anwendungsfall wie der Planung, dem Entwurf oder der Diagnose.

Im Hinblick auf den zweiten Aspekt, den Entwurf, gibt es keine Einschränkung bezüglich des modellierten Objekts, egal wie abstrakt oder konkret dieses auch sein mag. Beispielsweise könnten die Struktur von Moralvorstellungen oder die aerodynamischen Eigenschaften eines Motorrads modelliert werden.

Wichtig ist, dass das Modell nur bestimmte Eigenschaften des Originals wiedergibt. So ist das in einer aerodynamischen Simulation verwendete Modell eines Motorrads (selbstverständlich) kein echtes Motorrad.

Die analytischen Behauptungen eines Modells sind seine Aussagen über die Struktur und die Beziehungen zwischen den Komponenten des modellierten Objekts. Das aerodynamische Modell eines Motorrads im vorherigen Beispiel gibt keinen Aufschluss über die Eigenschaften des Motors oder des Getriebes.

Ob die Ansprüche an ein Modell empirisch oder normativ sind, hängt davon ab, welcher Standard für die Beurteilung des Modells angewendet wird. Im Fall von empirischen

Modellen wird dieser Standard durch das tatsächliche Verhalten des modellierten Objekts gesetzt. Bei normativen Modellen sind die Rollen vertauscht – das Verhalten des modellierten Objekts wird beurteilt, indem es mit dem idealen Verhalten, repräsentiert durch das Modell, verglichen wird. Aus diesem Grund müssen normative Modelle auf andere Weise validiert werden.

Es gibt verschiedene Typen von empirischen Modellen. Soll ein Verhalten eines Objekts simuliert werden, reicht es nicht aus, wenn das Modell dasselbe Verhalten wie das Objekt zeigt, es muss dieses auf vergleichbare Art tun. Diese Eigenschaft wird bei mentalen Modellen „kognitive Adäquatheit" genannt. Beispielsweise simuliert ein Flugzeugmodell nicht einen Vogel. Ein Simulationsmodell ist also ein empirisches Modell, das starke analytische Aussagen über die interne Struktur des modellierten Objekts macht.

Die frühen Modelle der *Jurimetrics*[99] in den Anfängen der Rechtsinformatik waren vollständig verhaltensbezogen und hatten weder einen normativen noch analytischen Anspruch.

Es sei noch angemerkt, dass der Zweck, das Objekt und die Aussagen eines Modells nicht zu dessen inhärenten Eigenschaften gehören. Sie sollten besser aus den Erwartungen heraus verstanden werden, die ein Agent[100] an das Modell stellt. Daher scheint es besser, vom Zweck zu sprechen, den ein bestimmter Benutzer des Modells verfolgt.

Mithilfe von KI-Modellen lassen sich Rechtstheorien sehr gut durchleuchten. Gordon hat beispielsweise derartige Modelle verwendet, um verschiedene Theorien zum Umfang richterlichen Ermessens einer Prüfung zu unterziehen. Gordon hat gezeigt, dass die Einschränkung richterlichen Ermessens durch die Anwendung deduktiver Logik, wie sie von Rödig und Klug vorgeschlagen wurde, nicht zielführend ist. Die Vertreter des Amerikanischen

---

[99] Der Begriff „Jurimetrics" entstand in den 1960er Jahren, als die rechtliche Praxis immer stärker vom Einsatz von Computer durchdrungen wurde. Durch den Computer wurden die rechtliche Forschung, die Analyse von Beweisen und das Datenmanagement revolutioniert. Dabei wurde ein streng empirischer Zugang zum Recht verfolgt, wobei der Schwerpunkt auf die logische Schlussfolgerung gelegt wurde. Es herrscht weitgehend Einigkeit, dass der Begriff „Jurimetrics" ein Neologismus ist, der aus den englischen Wörtern „Jurisprudence" und „Measurement" besteht (The Gale Group, 2008). Für eine weitergehende Einführung in das Forschungsgebiet der Jurimetrics sei auf die einschlägige Literatur verwiesen (de Mulder et al., 2010; Loevinger, 1962).

[100] In der Philosophie wird unter einem Agenten eine Entität verstanden, die in der Lage ist zu handeln oder mit anderen Worten, eine Aktion auszuführen (Wilson & Shpall, 2012).

Realismus hingegen gingen zu weit, indem sie jede Form der Bindung des Richters bei der Ermessensausübung ablehnten (Gordon, 1994a).

Dies sind nur einige Beispiele, die zeigen, wie die KI zur Untersuchung und Entwicklung von Theorien beitragen kann. Von diesen bereichsübergreifenden Forschungen profitieren jedoch keinesfalls nur die Rechtstheorie und Philosophie. Vielmehr bieten die theoretischen Ansätze für die KI eine solide Basis für die Entwicklung und Verbesserung von computergestützten Modellen, die beispielsweise der automatisierten Entscheidungsfindung dienen. Ferner führten die Forschungen zur Entwicklung eines computergestützten juristischen Argumentationsmodells zu einer Zusammenarbeit von Experten aus den Bereichen des Rechts und der Künstlichen Intelligenz. Insbesondere der kanadische Philosoph Walton, der als einer der führenden Experten auf dem Gebiet der Argumentationstheorie gilt, begann sich für die KI zu interessieren. Dieses Interesse führte zu einer Zusammenarbeit mit Gordon und zur Entwicklung eines Argumentationssystems, genannt „Carneades“[101] (Boer et al., 2007, S. 8 f.).

Die theoretischen Erkenntnisse und Einsichten der Forschungen auf dem Gebiet der Künstlichen Intelligenz und Recht können genutzt werden, um computergestützte Modelle des Rechts zu verbessern. Die Verankerung auf einem soliden theoretischen Fundament ist überdies notwendig, um die Modelle zu legitimieren und attraktiv für potentielle Anwender zu machen.

5.3. Wissen und Wissensarten

5.3.1. Was ist Wissen?

In dieser Dissertation sowie in der Literatur wurde der Begriff „Wissen“ viele Male verwendet. Die Bedeutung des Begriffs ist jedoch in vielen Publikationen keineswegs eindeutig und hängt vom Hintergrund des jeweiligen Autors ab (Schweighofer, 1999, S. 11 ff.).

Beispielsweise liegt der Erkenntnistheorie (Epistemologie) als philosophische Disziplin das traditionelle geisteswissenschaftliche Verständnis des Wissens als „wahre und gerechtfertigte Meinung“ (engl. justified true believe) zugrunde (Steup, 2014).

[101] *Gordon, T.*, Carneades Argumentation System. http://carneades.github.io/ aufgerufen: 28.11.2013 (2012).

Es gibt jedoch auch Fälle, in denen eine wahre gerechtfertigte Meinung nicht als Wissen qualifiziert werden kann. Eine wahre gerechtfertigte Meinung ist daher keine *hinreichende* Bedingung für Wissen. Dieser Einwand geht auf den amerikanischen Philosophen Edmund Gettier zurück und wird demnach als Gettier-Problem bezeichnet. Gettier (1963) hat in seinem nur drei Seiten umfassenden Paper gezeigt, dass weder der Besitz von Beweisen noch von zuverlässigen geistigen Fähigkeiten ausreicht um auszuschließen, dass eine Aussage nicht bloß aufgrund eines Zufalls wahr ist. Dieser philosophische Streit ist für die gegenständliche Arbeit jedoch nicht von Belang und wird nicht weiter vertieft. Die Ausführungen machen hinreichend deutlich, dass die Definition des Wissens schwierig ist.

Newell postuliert die Existenz eines *knowledge level*, innerhalb dessen Wissen definiert wird, als Hypothese (Newell, 1982, S. 99):

*„**The Knowledge Level Hypothesis.** There exists a distinct computer systems level, lying immediately above the symbol level, which is characterised by knowledge as the medium and the principle of rationality as the law of behaviour."*

Vernunft und Wissen sind demnach eng miteinander verbunden. Systeme, denen vernünftiges Verhalten postuliert werden kann, besitzen demnach Wissen. Wissen ist dabei ein kompetenzorientierter Begriff, nämlich das Potential, eine Aktion ausführen zu können.

Van Engers (2001, S. 11) und andere Autoren unterscheiden Daten, Informationen und Wissen:

- Daten sind symbolische Repräsentationen.
- Informationen sind interpretierte Daten, die mit einer Bedeutung verknüpft sind.
- Wissen ist die Fähigkeit, mit Informationen umzugehen (Know-how, Know-why, Know-where und Know-when).

Die Definition des Wissens als „die Fähigkeit mit Informationen umzugehen" deckt sich im Wesentlichen mit der Definition des Wissens nach Newell, ist jedoch nicht frei von Interpretationsproblemen. Trotzdem ist der Versuch einer Definition des Wissens nicht wertlos. Sie hilft, den Begriff „Wissen" näher einzugrenzen. Wichtig ist hervorzuheben, dass das Wissen unabhängig von seiner symbolischen Repräsentation auf dem *symbol level* nach Newell ist.

5.3.2. Wissensarten

Häufig unterscheidet man zwischen „deklarativem" (declarative) und „prozeduralem" (procedural) Wissen. Letzteres bezeichnet Prozeduren und Techniken („Rezepte") und wird gelegentlich auch als „präskriptives" Wissen (prescriptive knowledge) bezeichnet (Mokyr, 2002, S. 2). Darüber hinaus existieren jedoch weitere detailliertere Qualifikationen des Wissens. In der Literatur sind diverse Ansätze, wie Wissen klassifiziert werden kann, vorhanden. Nicht selten haben diese zu einer Ausuferung der Begrifflichkeit geführt.

Ein sinnvoller Ansatz zur Bestimmung der Wissensarten ist deren Klassifikation aus Sicht der Verwendung des Wissens. Die Lösung von Aufgaben formt die Grundlage für die Identifikation der relevanten Aspekte des Wissens. Diese Kategorisierung des Wissens vom Standpunkt der Problemlösung erscheint im gegenständlichen Kontext besonders geeignet, da das zentrale Anliegen dieser Dissertation die Formalisierung und Lösung rechtlicher Probleme ist.

Die folgende Darstellung folgt de Jong und Ferguson-Hessler (1996), die das Wissen anhand der Dimensionen „Art" und „Qualität" klassifizieren. Diese beiden Dimensionen sind grundsätzlich voneinander unabhängig. Es wird systematisch zwischen Charakteristika, die die Art des Wissens, die Eigenschaften oder die Qualität des Wissens spezifizieren, unterschieden. Mithilfe dieser zweidimensionalen Beschreibung, die zwischen Wissensart und Qualität unterscheidet, ist es möglich Wissen zu klassifizieren, ohne neue Wissensarten einführen zu müssen.

	Types			
Qualities	*Situational*	*Conceptual*	*Procedural*	*Strategic*
Level Surface ↔ deep	Case-based reasoning ↔ translation into domain concepts	Symbols and formulae ↔ concepts and relations	Rules/recipes/algebraic manipulation ↔ meaningful action	Symbol-driven search for formula ↔ analysis and planning
Structure Isolated elements ↔ structured knowledge	Isolated features ↔ grouped together (i.e. models of situations)	Independent concepts and laws ↔ meaningful (hierarchical) structure	Isolated algorithms ↔ action related to concept or principle	Isolated actions ↔ Coherent set of sequential actions
Automation Declarative ↔ compiled	Conscious and stepwise ↔ automatic translation to domain concepts	Verbalizable principles, definitions, etc. ↔ intuitive, tacit understanding	Conscious choice and step by step execution ↔ automatic access and routine execution	Step by step choices and planning ↔ automatic analysis and planning; parallel checking
Modality Verbal ↔ pictorial	Words and symbols ↔ pictures and diagrams	Propositions and formulae ↔ pictures, diagrams	Sets of production rules ↔ pictorial (diagrams, figures, graphs)	Sets of production rules ↔ pictorial (diagrams, figures, graphs)
Generality General ↔ domain specific	General properties (e.g., homogeneous, time independent) ↔ domain specific characteristics	General structures of domains ↔ a specific domain, and also: conservation laws ↔ specific cases thereof	Define system for application of conservation laws ↔ check points of contact for forces	General steps (analysis, planning, etc.) ↔ specific steps (thermodynamics: system, interaction, process, etc.)

Abbildung 2: Wissen als eine Funktion von Typ und Qualität (de Jong & Ferguson-Hessler, 1996)

Abbildung 2 zeigt eine Beschreibung des Wissens als Funktion von Typ und Qualität anhand von Beispielen aus der Physik. Wie ersichtlich, unterscheiden de Jong und Ferguson-Hessler vier Wissensarten, auf die nun näher eingegangen werden soll.

Situatives Wissen: Wissen über Situationen, wie sie typischerweise in einer bestimmten Domäne auftreten. Dieses Wissen erlaubt es dem Problemlöser, relevante Eigenschaften der Problemstellung zu erkennen und zu selektieren.

Begriffliches Wissen: Unter begrifflichem Wissen ist statisches Wissen über Fakten, Begriffe und Prinzipien zu verstehen, die innerhalb einer bestimmten Domäne anzuwenden sind. Diese Wissensart wird zuweilen auch als „deklaratives Wissen“ bezeichnet.

Prozedurales Wissen: Diese Kategorie umfasst Aktionen und Manipulationen des Wissens, die in einer bestimmten Domäne zulässig sind. Es dient dem Problemlöser als Anleitung, um den Übergang von einem Problemzustand zum nächsten zu bewerkstelligen. Wie oben bereits angemerkt wurde, wird diese Wissensart gelegentlich auch „präskriptives Wissen“ genannt.

Strategisches Wissen: Strategisches Wissen erlaubt es, den Problemlösungsprozess in Schritten zu organisieren mit dem Ziel, eine bestimmte Lösung zu erreichen. Eine Strategie kann allgemein als Aktionsplan verstanden werden, in dem die Reihenfolge der Problemlösungsaktivitäten festgelegt ist.

5.3.3. Qualitäten des Wissens

Um für die vorgestellten Wissensarten eine weitergehende Klassifikation zu erreichen, unterscheiden de Jong und Ferguson-Hessler fünf Qualitäten des Wissens. Die Ausprägungen dieser zweiten Dimension zur Klassifikation des Wissens werden im Folgenden in Grundzügen dargestellt.

Niveau des Wissens: Hinsichtlich des Niveaus des Wissens muss zwischen oberflächlichem (künstlichem) und tiefem Wissen unterschieden werden. Dabei wird üblicherweise tief mit gut und oberflächlich mit schlecht assoziiert.

Wissen ist „tief“, wenn es fest im Gedächtnis einer Person verankert ist und externe Informationen in grundlege Begriffe, Prinzipien und Prozeduren des jeweiligen Bereichs übersetzt wurden. Tiefes Wissen steht mit Begriffen wie Abstraktion, Vollständigkeit, kritische Beurteilung und Bewertung etc. in Verbindung.

Oberflächliches Wissen kennzeichnet sich durch bloße Wiedergabe und Auswendiglernen, Trial-and-Error und das Fehlen von kritischer Beurteilung. Dieses Wissen ist im Gedächtnis mehr oder weniger als bloße Kopie der externen Informationen gespeichert.

Struktur des Wissens: Studien über die Problemlösung von Experten und Anfängern haben gezeigt, dass die große Menge an Wissen, über die ein Experte verfügt, nicht durch eine größere Speicherkapazität ermöglicht wird, sondern durch die Stückelung von Informationen in größere sinnvolle Einheiten. Den meisten Anfängern fehlt es an einer solchen Organisation des Wissens.

Hierarchische Strukturen scheinen am besten geeignet zu sein um Wissen zu behalten, schnelle Suchprozesse zu ermöglichen und sich neues Wissen aneignen zu können, ohne vorhandenes Wissen neu strukturieren zu müssen.

Überdies wird Wissen in einer höheren Organisationsform, dem sogenannten Schema, organisiert. Ein Schema besteht aus verschiedenen Wissensarten und ist zugeschnitten auf die Lösung einer bestimmten Aufgabe. Ein Experte zeichnet sich dadurch aus, dass er mehrere solcher Schemas für die Lösung von Problemen eines Bereichs in seinem Gehirn parat hält.

Automatisches (kompiliertes) vs. nicht-automatisches Wissen: Anfänger lösen Probleme, indem sie Wissen bewusst abrufen und schrittweise anwenden. Dabei handelt es sich um einen langsamen Prozess, der auf der Anwendung genereller Methoden beruht.

Diese Schwerfälligkeit wandelt sich bei einem Experten in einen kontinuierlichen Fluss, in Form eines automatischen Prozesses, der stark auf situationsabhängigen und bereichsspezifischen Methoden aufbaut. Der Experte verfügt dabei über eine tiefe Repräsentation der jeweiligen Aufgabe und gut strukturiertes Wissen über Abläufe und Prinzipien des Bereichs.

Die Wissensbasis des Experten ist demnach „kompiliert“, das heißt auf eine bestimmte Art der Anwendung trainiert. Dem Experten stehen durch diese Routine mehr Ressourcen zur Verfügung, sodass dieser parallel zur Lösung einer Aufgabe auch das Ergebnis der einzelnen Schritte überprüfen kann. Dies führt insgesamt zu einer schnellen und zuverlässigen Lösung der Aufgabe.

In engem Zusammenhang mit der Unterscheidung zwischen automatischem und nicht-automatischem Wissen steht jene zwischen *stillem* und *explizitem* Wissen. Unter stillem Wissen wird Know-how verstanden, das nicht offen geäußert oder spezifiziert wird. Dieses Wissen wird üblicherweise durch formloses Lernen aus eigener Erfahrung oder der Erfahrung eines Mentors angeeignet. Stilles Wissen ist nach Ansicht mancher Autoren einer expliziten Verbalisierung nicht zugänglich (de Jong & Ferguson-Hessler, 1996, S. 109).

Modalität des Wissens: Wissen kann im Langzeitgedächtnis entweder in einer analytischen Repräsentation als eine Reihe von Aussagen oder in einer analogen Repräsentation in Form von Bildern gespeichert werden.

Demnach können zwei Modalitäten des Wissens – verbal (analytisch) und bildlich (analog) – unterschieden werden.

Generelles vs. domänenspezifisches Wissen: Die Qualitäten „generell“ und „domänenspezifisch“ werden oft in Zusammenhang mit Wissen über Strategien oder Prozeduren behandelt. Problemlösungsstrategien werden oft auf einem generellen Level beschrieben. Heuristiken, die oftmals Teil einer Problemlösungsstrategie sind, können grundsätzlich generell oder domänenspezifisch sein, sind jedoch häufig auf einen bestimmten Bereich zugeschnitten. Studien zeigen, dass spezifische, auf einen bestimmten Bereich ausgerichtete Problemlösungsstrategien in der Regel erfolgversprechender sind.

Begriffliches Wissen kann ebenso generell oder domänenspezifisch ausgerichtet sein. Spezielles begriffliches und auch situatives Wissen kann oftmals abstrahiert und dadurch generalisiert werden. Aus einer konkreten Problemstellung können beispielsweise die Charakteristika abgeleitet und durch generelle Begriffe beschrieben werden. Diese Art der Generalisierung dient üblicherweise der Erstellung von hierarchischen Strukturen, die die Speicherung und den Abruf des Wissens erleichtern.

5.3.4. Abschließende Bemerkungen

Die Ausführungen in den vorherigen Unterabschnitten machen deutlich, dass es schwierig ist, den Begriff des Wissens eindeutig zu beschreiben. Im Kontext der Dissertation geht es insbesondere um die Formalisierung von prozeduralem Wissen zur Problemlösung. Die im Gesetzestext zum Ausdruck gebrachten Normen werden als eine „Anleitung“ zur Lösung rechtlicher Probleme verstanden.

Die verschiedenen Wissensarten und Qualitäten deuten an, dass abhängig vom Ziel einer Formalisierung eine adäquate Repräsentation des Wissens zweckmäßig erscheint. Im Recht sind vor allem begriffliches und prozedurales Wissen („Regeln“) von zentraler Bedeutung. Diese beiden Wissensarten müssen angemessen modelliert werden können. Es gilt stets zu prüfen, ob mit einer einzigen Sprache eine adäquate Repräsentation von Begriffen und Regeln erreicht werden kann, oder ob dafür verschiedene Formalismen zweckmäßiger erscheinen.

Angemerkt sei, dass im Kontext der Dissertation das geschriebene Recht in Form von Gesetzen und Verordnungen die zentrale Quelle des Wissens darstellt. Die Probleme, die mit der Extraktion und Formalisierung von stillem Expertenwissen einhergehen, sind daher nur von untergeordneter Bedeutung.

5.4. Ansätze zur juristischen Wissensrepräsentation

5.4.1. Überblick

Die Idee der Fomalisierung von Rechtswissen, um die Suche und Abfrage zu verbessern sowie automatisierte Entscheidungen zu ermöglichen, ist nicht neu. Bereits in den frühen 1950er-Jahren versuchte die amerikanische Schule der Jurimetrics (de Mulder, van Noortwijk, & Combrink-Kuiters, 2010; Loevinger, 1962) das Recht zu computerisieren. Mitte der 1980er-Jahre haben Forschungen zum fallbasierten Schließen (Case-based Reasoning, kurz CBR) maßgeblich zur Geburt des Forschungsgebiets Künstliche Intelligenz und Recht (Artificial Intelligence and Law) beigetragen. Damit wurde eine tiefgehende wissenschaftliche Auseinandersetzung mit dem Thema eingeläutet (Rissland, Ashley, & Loui, 2003). So wurde 1987 die erste International Conference on Artificial Intelligence and Law (ICAIL) in New York veranstaltet, die unlängst ihr 25. Jubiläum feierte (Bench-Capon et al., 2012).

Die Versuche, rechtliches Wissen auf Computersysteme zu übertragen, reichen damit mehr als 60 Jahre zurück. Im Lauf der Zeit haben die Forschungen verschiedenste Ansätze zur (juristischen) Wissensrepräsentation hervorgebracht, die nun zur Systematisierung in Grundzügen dargestellt werden.

Aufgrund der Tatsache, dass es sich um einen interdisziplinären Forschungsbereich handelt, ist es relativ schwierig, eine Systematik für diese Darstellung zu finden. Denkbare Kriterien

für eine Kategorisierung sind unter anderem die Aktivität (Vertragsgestaltung, Rechtsstreit), das Objekt, das formalisiert wird (Gesetze, Verträge etc.), der verwendete Formalismus (z.B. Java), die Art der Entscheidungsfindung (basierend auf Gesetzen oder Fällen) und die Art, wie Rechtstexte interpretiert werden (statisch oder dynamisch).

Die folgende Kategorisierung der Ansätze in der KI und Recht stellt darauf ab, wie juristisches Wissen *repräsentiert* wird. Basierend auf diesem Kriterium ist es möglich, die folgenden Hauptansätze zu differenzieren, unterschieden nach der Art des Formalismus, der verwendet wird, um juristisches Wissen zu repräsentieren:

- Der regelbasierte Ansatz: Darunter ist die Anwendung von regelbasierter Schlussfolgerung und der logischen Programmierung auf das Recht zu verstehen.
- Der fallbasierte Ansatz: Bei diesem Ansatz wird das Recht in Form von Fällen repräsentiert und werden fallbasierte Schlussfolgerungsmethoden angewendet.
- Der logische Ansatz: Diese Kategorie umfasst sämtliche Ansätze, das Recht mithilfe der Logik zu repräsentieren und Inferenzen daraus abzuleiten.
- Die Argumentation: Ansätze der computergestützten Argumentation befassen sich mit der Repräsentation juristischer Argumente.

Die Ansätze entsprechen im Wesentlichen der Repräsentation von juristischem Wissen durch (1) Regeln, (2) Fälle, (3) logische Aussagen und (4) Argumente.

Für die Unterscheidung der einzelnen Ansätze wird maßgeblich auf die Struktur abgestellt, wie juristisches Wissen repräsentiert wird. Es sei darauf hingewiesen, dass die Abgrenzung nicht immer eindeutig möglich ist und fließende Übergänge bestehen. Argumente können beispielsweise aus Regeln oder Fällen gebildet werden.

5.4.2. Der regelbasierte Ansatz

Die ersten Ansätze, rechtliches Wissen zu repräsentieren, basierten auf Regeln. Dieser Ansatz ist gekennzeichnet durch die Verwendung von regelbasierten Formalismen, um rechtliches Wissen zu repräsentieren und Entscheidungen daraus abzuleiten[102].

[102] Beispielsweise sei die von Sergot et al. (1986) durchgeführte Implementierung des britischen Staatsbürgerschaftsgesetzes in Prolog erwähnt.

Im Kern beruht dieser Ansatz auf der Annahme, dass das Recht als Produktionsregeln abgebildet werden sollte. Die verwendeten Regeln werden als „Produktionsregeln“ bezeichnet, da sie verwendet werden, um neue Informationen zu erzeugen. Die Vertreter dieses Ansatzes bekennen sich dabei in der Regel nicht explizit zu der diesem Ansatz innewohnenden theoretischen Annahme, dass das Recht als eine Menge von Regeln begriffen werden kann. Diese Annahme ist jedoch eine wesentliche Voraussetzung für diesen Ansatz und wird von Susskind (1987), als einem der wenigen, ausdrücklich verteidigt (Valente, 1995, S. 9). Als weiterer prominenter Verteidiger des regelbasierten Ansatzes sei Bench-Capon (1994a) genannt.

Der Kategorie der regelbasierten Ansätze kann auch die logische Programmierung[103] zugezählt werden[104]. Logische Programme sind theoretisch mächtiger als regelbasierte Systeme. Sie besitzen eine fundierte theoretische Grundlage und eine formale Semantik. Ferner sind sie nicht auf die üblichen rückwärts- oder vorwärtsverkettenden Strategien regelbasierter Systeme beschränkt, sondern können auf andere Schlussfolgerungstrategien wie Abduktion oder das Schließen mit Constraints zurückgreifen (Valente, 1995, S. 10).

Trotz dieser theoretischen Stärken logischer Programmierung weisen die in der KI und Recht präsentierten Ansätze keine wesentlichen Unterschiede zu regelbasierten System auf. In sämtlichen logikbasierten Systemen kommt Prolog zum Einsatz. Es besteht kein grundlegender Unterschied zwischen den Horn-Klauseln, die ein Programm in Prolog ausmachen, und den Fakten und Regeln eines regelbasierten Systems. Beide repräsentieren das Modell eines Gesetzes in Form von Regeln (Kowalski & Sergot, 1990; Kowalski, 1995). Überdies kann die Schlussfolgerung, die Prolog verwendet, als Rückverkettung mit Backtracking klassifiziert werden, ein Algorithmus, der in regelbasierten Expertensystemen oftmals zum Einsatz kommt.

5.4.2.1. Kritische Würdigung

Die Verwendung von Regeln ist ein etablierter Ansatz im Bereich des Knowledge Engineerings. Wie bereits weiter oben ausgeführt wurde, werden solche Regeln

[103] Ein bekannter Vertreter dieser Kategorie ist die Programmiersprache Prolog, die Anfang der 1970er-Jahre entwickelt wurde.

[104] Einen ausgezeichneten Überblick über die Geschichte der logischen Programmierung gibt Kowalski (2014).

üblicherweise als Produktionsregeln verstanden, da sie neues Wissen produzieren. Zum Beispiel entsteht aus der Regel „Wenn A dann B“ und dem Fakt „A ist wahr“ die Konklusion „B ist wahr“, die zur Wissensbasis hinzugefügt wird.

Regeln können einerseits zur Modellierung von Gesetzen verwendet werden und andererseits um wissensbasierte Systeme zu implementieren. Die im zweiten Fall implementierten Expertensysteme werden in Englisch als „expert system shells“ bezeichnet, als „shell“[105] deshalb, weil ihre Wissensbasis anfänglich leer ist und sie damit gewissermaßen als leere Hülle begriffen werden können.

Die Verwendung von Regeln zur Modellierung von Gesetzen ist selbst prinzipiell nicht weiter zu beanstanden. Der Ansatz ist etabliert und es gibt einige gut entwickelte Systeme. Problematischer ist jedoch die Frage, ob Regeln das Recht epistemologisch adäquat repräsentieren.

Oftmals wir das Recht als eine Sammlung von Regeln begriffen. Diese Ansicht ist im Allgemeinwissen so stark verankert, dass die Begriffe „Norm“ und „Regel“ oftmals als Synonyme verstanden werden.

Ein Verfechter des regelbasierten Ansatzes ist Susskind (1987), ein Forscher auf dem Gebiet der KI und Recht[106]. Selbst dieser erkennt jedoch, dass es keine einheitliche Meinung zwischen den Theoretikern gibt, was Regeln tatsächlich sind. Es besteht zwar weitgehend Einigkeit über den grundsätzlichen Aufbau, dass Regeln aus Antezedens und Konsequenz bestehen, wie letztere jedoch aufgebaut sind, ist strittig.

Nach Valente scheint das Problem daher zu rühren, dass der von den Rechtstheoretikern verwendete Begriff der „Regel“ zu sehr dem entspricht, was im allgemeinen Sprachgebrauch darunter verstanden wird. Nach diesem Verständnis wird der Begriff „Regel“ als Synonym für „Norm“ oder „rechtliches Wissen“ verwendet (Valente, 1995, S. 11). Natürlich ist es sinnvoll, rechtliches Wissen in dieser Form von Antezedens und Konsequenz zu strukturieren, wenn man daraus Entscheidungen ableiten möchte. Dies trifft jedoch praktisch auf jede Form des Wissens zu.

---

[105] Deutsch „Schale“ oder „Hülle“.

[106] Weitere Verteidiger des regelbasierten Ansatzes sind Bench-Capon (1994a) und Kowalski (1995).

In der KI hat der Begriff „Regel“ eine speziellere Bedeutung. Als eine Regel wird im Wesentlichen eine logische Implikation verstanden, die neues Wissen durch eine Form der Deduktion erzeugt. Auch wenn manche Rechtstheoretiker Normen als logische Aussagen betrachten, ist die Verwendung einer deduktiven Schlussfolgerung heftig umstritten .

Auf den Punkt gebracht bedeutet die Kritik, dass das rechtstheoretische Verständnis einer Regel wesentlich von jenem der KI abweicht.

Es besteht jedoch auch noch ein zweites, eher rechtstheoretisches Problem. Viele Rechtstheoretiker sind der Meinung, dass es nicht nur eine Art von Regeln gibt. Kelsen beispielsweise identifiziert in seinem Werk mehrere Arten von Normen (Kelsen, 1979). Überdies unterscheidet auch Hart zwischen primären und sekundären Regeln (Hart, 2009, S. 99 ff.). Aus theoretischer Sicht liegt es auf der Hand, dass es einer differenzierten Schlussfolgerung und Modellierung der verschiedenen Arten von Normen bedarf. Die üblichen Ansätze unterscheiden jedoch nicht zwischen den verschiedenen Arten der Normen.

Zusammengefasst kann gesagt werden, dass Regeln jedenfalls ihre Berechtigung für die Repräsentation von juristischem Wissen haben. Die Annahme, Regeln seien ein epistemologisch adäquates Modell um Normen zu repräsentieren, ist jedoch mit zwei Problemen behaftet, wie soeben dargestellt wurde. Die zuerst augenscheinlich vorhandene theoretische Unterstützung wird aufgeweicht, vergegenwärtigt man sich die Vielzahl an unterschiedlichen rechtstheoretischen Bedeutungen des Begriffs „Regel“ und die Tatsache, dass die Verwendung der Deduktion als Methode zur Schlussfolgerung im juristischen Bereich höchst umstritten ist. Zweitens ist die bloße Existenz verschiedener Arten von Rechtsregeln ein Indiz dafür, dass es zumindest einer Anpassung oder Erweiterung der üblichen regelbasierten Ansätze bedarf. Regeln scheinen die Normen demnach nur oberflächlich zur repräsentieren, da sie wichtige epistemologische Unterschiede des Rechts nicht erfassen.

Abschließend sei noch auf ein praktisches Problem des regelbasierten Ansatzes hingewiesen. Das Recht besteht zu einem wesentlichen Teil aus Legaldefinitionen in einer mehr oder weniger hierarchischen Anordnung. Diese Struktur ist jedoch mit Regeln nur schwer repräsentierbar, sie ist jedoch für die Erzeugung von Erklärungen – insbesondere der rechtlichen Begründung – von großer Wichtigkeit.

Regeln eignen sich daher zur Darstellung eines Teiles des rechtlichen Wissens, sind jedoch alleine nicht ausreichend, um die Wissensbasis eines wissensbasierten Systems zu bilden.

5.4.3. Der fallbasierte Ansatz

Fallbasierte Methoden der Schlussfolgerung (case-based reasoning, kurz CBR) haben weite Verbreitung in der KI erlangt (Ashley, Desai, & Levine, 2002; Ashley, 1990; Rissland & Ashley, 1987). Diese Ansätze verwenden fallbasierte Schlussfolgerungsmodelle, um die rechtliche Entscheidungsfindung zu modellieren.

Die am häufigsten verwendete Art der fallbasierten Schlussfolgerung ist die sogenannte „interpretative fallbasierte Schlussfolgerung“ (*interpretive CBR*) (Kolodner, 1992). Dabei wird ein neuer Fall im Licht vorheriger Fälle beurteilt. Ein bekanntes Beispiel für ein fallbasiertes System ist das von Ashley entwickelte HYPO (Ashley, 1990).

HYPO analysiert einen neuen Fall und sucht nach relevanten Faktoren, um basierend auf diesen ähnliche Fälle zu selektieren. Danach werden die ausgewählten Fälle, je nachdem ob sie die jeweilige Rechtsansicht unterstützen, in zwei Gruppen geteilt. Diese Zweiteilung wird daraufhin verwendet, um Argumente für oder gegen die jeweilige Rechtsansicht zu erzeugen. Dieser Ansatz imitiert die Taktik von Strafverteidigern in den U.S.A. und wurde in einer Reihe von praktischen Fällen erfolgreich angewendet.

Die Forschungen haben eine Reihe von fallbasierten Systemen hervorgebracht, die beginnend mit Ashleys HYPO (Ashley, 1990) bis in jüngste Zeit reichen (Araszkiewicz, Łopatkiewicz, & Zienkiewicz, 2013; Atkinson, Bench-Capon, Prakken, & Wyner, 2013).

5.4.3.1. Kritische Würdigung

Unzweifelhaft hat der fallbasierte Ansatz einige Vorteile, verglichen mit den anderen Ansätzen. Insbesondere wird versucht, eine Balance zwischen Theorie und Praxis zu finden. So wurden neben den Modellen zur rechtlichen Entscheidungsfindung auch tatsächlich Systeme und Architekturen entwickelt, um die theoretischen Konzeptionen zu testen.

Trotz dieses wünschenswerten Gleichgewichts zwischen Theorie und Praxis ist der fallbasierte Ansatz mit einigen Problemen behaftet. Das wohl wichtigste Problem ist, dass die Adäquanz dieses Ansatzes für rechtliche Probleme über weite Strecken als selbstverständlich angenommen wird.

Dieser Standpunkt rührt von der Tatsache her, dass im Common Law Präzedenzfälle die wichtigste Rechtsquelle darstellen. Dieser Umstand alleine könnte bereits als Rechtfertigung für den fallbasierten Ansatz dienen.

Die Rolle des fallbasierten Ansatzes in der KI und Recht ist nicht eindeutig und es besteht Diskussionsbedarf hinslichtlich zweier zugrundeliegender Annahmen (Valente, 1995, S. 12 f.).

Die erste Annahme bezeichnet Valente als „structural validity assumption“. Darunter ist die Annahme zu verstehen, dass das gesamte rechtliche Wissen in Fällen organisiert ist. Dies kann als Gegenteil zum regelbasierten Ansatz gesehen werden, in dem die Annahme vorherrschend ist, dass das Recht in Form von Regeln abgebildet werden soll. Das erste Problem dieser Annahme ist, dass sie die Anwendbarkeit dieses Ansatzes auf Rechtssysteme des Common Law beschränkt. Damit wird der Anschein erweckt, als wären die Forschungszweige der KI, die sich mit der rechtlichen Entscheidungsfindung in Common Law Systemen und jenen, die sich mit dem kontinentaleuropäischen Rechtskreis[107] befassen, getrennt. Diese strikte Trennung ist jedoch keinesfalls uneingeschränkt haltbar. So werden selbst in den U.S.A. – als ein Paradebeispiel für das Common Law – regelmäßig auch andere Rechtsquellen, wie Gesetze und Verordnungen, für die Beurteilung von Fällen herangezogen. Umgekehrt sind aber auch in kontinentaleuropäischen Rechtssystemen frühere Fälle nicht völlig unbedeutend, insbesondere dann nicht, wenn diese von einem Höchstgericht entschieden wurden. Es gilt im Wesentlichen das Prinzip, dass gleiche Fälle gleich entschieden werden müssen. Weicht die Entscheidung eines Gerichts ungerechtfertigt von der höchstgerichtlichen Rechtsprechung ab, so kann das Höchstgericht diese Entscheidung aufheben.

Die Ausführungen machen deutlich, dass unabhängig vom jeweiligen Rechtssystem Fälle nicht die einzige Rechtsquelle für Entscheidungen sind.

Zweitens hinterfragt Valente die Annahme, dass die fallbasierte Schlussfolgerung ein kognitiv valides Modell rechtlicher Entscheidungsfindung ist. Diese Annahme wird „cognitive validity assumption“ genannt. Sie wird von der Vorstellung getragen, dass in der Praxis Fälle durch

[107] Valente verwendet die Bezeichnung „Romano-Germanic legal system“. Oftmals wird das kontinentaleuropäische Rechtssystem in Abgrenzung zum Common Law auch „Civil Law“ genannt.

Anwendung von Analogien und Feststellung der Ähnlichkeit zu vorherigen Fällen beurteilt werden.

Diese Behauptung wurde jedoch bisher nicht empirisch nachgewiesen und scheint daher auf Beobachtungen des augenscheinlichen Verhaltens von Rechtsanwälten zu beruhen. Die daraus abgeleitete Vermutung, dass Fälle für die Unterstützung von rechtlichen Argumenten verwendet werden, muss nicht zwingend richtig sein. Menschen sind nämlich in der Lage, die wesentlichen Aspekte durch Abstraktions- und Generalisierungsprozesse aus mehreren Fällen abzuleiten. Dies bedeutet jedoch nicht zwingend, dass die einzelnen Fälle direkt im Denkprozess für die rechtliche Beurteilung herangezogen werden.

Selbst wenn man davon ausgeht, dass die fallbasierte Schlussfolgerung ein kognitiv valides Modell der rechtlichen Beurteilung ist, bedeutet das nicht, dass dieser Ansatz auch zwingend für die Konstruktion von wissensbasierten Systemen geeignet ist. Bei der Entwicklung von Methoden zur automatisierten Problemlösung müssen nicht zwingend die Problemlösungsstrategien von Menschen imitiert werden. Das menschliche Gehirn ist nämlich, insbesondere wenn es um die Anwendung von Analogien und die Beurteilung von Ähnlichkeiten geht, dem Computer weit überlegen. Maschinen sind weit von derartigen Fähigkeiten entfernt. Es fehlt insbesondere an geeigneten Algorithmen, um Maschinen den menschlichen Fähigkeiten anzunähern.

Diese Ausführungen sollen verdeutlichen, dass die Rolle der fallbasierten Schlussfolgerung in der KI nicht ganz klar ist. Es ist schwer, die Verwendung dieses Ansatzes als kognitiv valides Modell oder als „natürliche“ Art der rechtlichen Schlussfolgerung zu verteidigen. Der fallbasierte Ansatz orientiert sich oftmals an praktischen Anforderungen. Aus diesem Grund ist fraglich, ob die fallbasierten Modelle geeignet sind, eine epistemologisch adäquate Wissensbasis zu konstruieren.

5.4.4. Der logische Ansatz

Die Anwendung der Logik im Recht zielt einerseits darauf ab, das Recht durch logische Aussagen zu repräsentieren und andererseits Schlussfolgerungen abzuleiten. Der Anfang dieser Bestrebungen war die Verwendung von Aussagenlogik und der darauf aufbauenden Prädikatenlogik. Die Bedeutung des logischen Ansatzes hat stetig zugenommen, und bis heute spielt die Logik im Bereich der Formalisierung von Rechtsnormen eine wichtige Rolle.

Das Semantic Web und die damit forcierte Verwendung von Beschreibungslogik zum Aufbau von Ontologien ist dafür ein Beispiel (vgl. dazu die Ausführungen in Kapitel 10).

5.4.4.1. Die Rolle der Logik in der Künstlichen Intelligenz

Bevor auf die einzelnen Ausprägungen des logischen Ansatzes näher eingegangen wird, soll zuerst die Rolle der Logik selbst in der KI beleuchtet werden. Die anschließende Darstellung folgt Valente (Valente, 1995, S. 15 ff.).

Die Möglichkeiten der Anwendung von logischen Formalismen im Bereich der KI sind (1) die Schlussfolgerung, (2) die Begründung und (3) die Repräsentation von Wissen.

Bei der Verwendung der Logik zur *Schlussfolgerung* gibt es folgende Ausprägungen:

- Die Modellierung der Inferenz: Die Logik kann eingesetzt werden, um Modelle der Schlussfolgerung zu verfeinern, indem die modellierten Ableitungen als logische Ableitungsregeln dargestellt werden. Auf diese Weise wurden nicht nur die Deduktion, sondern auch die nicht-monotone Schlussfolgerung und die Schlussfolgerung mit Standardwerten (*default logic*) etc. formalisiert.
- Die Implementierung der Inferenz: Logische Modelle der Schlussfolgerungen können schließlich in einem Programm implementiert werden, um die rechtliche Beurteilung durchzuführen. Die Schlussfolgerungen werden von einer „logischen Maschine“ abgeleitet. Die Programmiersprache Prolog ist ein Beispiel für diese Verwendung.
- Die Verwendung zur Problemanalyse: Die Logik kann verwendet werden, um die Komplexität von Algorithmen oder Problemen, die in logischen Aussagen formuliert sind, zu analysieren.

Die Verwendung der Logik zur *Begründung* von Argumenten wurde bereits eingehend im Kapitel zur „Formalisierung“ diskutiert. Eine deduktive logische Ableitungsregel kann nicht nur neue Informationen produzieren, sondern auch Argumente überprüfen, die behaupten, eine Information sei wahr. Im zweiten Fall wird die Logik also nicht zur Schlussfolgerung verwendet, sondern um zu überprüfen, ob die Konklusion gemäß den Regeln des jeweiligen Modells gültig ist. Hinsichtlich der gewichtigen Bedenken gegen Verwendung der Logik zur Begründung von Argumenten sei auf Kapitel 4 verwiesen.

Abschließend sei an dieser Stelle auf die Verwendung der Logik zur *Repräsentation von Wissen* eingegangen. Nach Valente gibt es zumindest zwei Arten, wie die Logik in dieser Hinsicht eingesetzt werden kann:

- Die Repräsentation eines Bereichs: Ein logischer Formalismus kann als Sprache zur Wissensrepräsentation verwendet werden. Dabei ist es möglich, das Wissen eines Bereichs, z.B. der Medizin, des Rechts etc. oder einer bestimmten speziellen Anwendung zu repräsentieren. Überdies ist es möglich, ganz allgemeine Konzepte wie Zeit, Aktion und Ursache zu formalisieren.
- Die Repräsentation von ontologischen Verpflichtungen („ontological commitments"): Sofern logische Formalismen verwendet werden, um einen Bereich oder ein grundlegendes Konzept wie die Zeit zu repräsentieren, muss man sich der Tatsache bewusst sein, dass diesen selbst einige ontologische Verpflichtungen innewohnen. Auch wenn ein Formalismus verwendet werden kann, um z.B. die Zeit zu repräsentieren, ist seine axiomatische Struktur selbst eine Repräsentation von ontologischen Verpflichtungen. Eine „ontologische Verpflichtung" kann als eine Entscheidung zugunsten jener Begriffe verstanden werden, die für die Beschreibung eines Phänomens verwendet werden sollen (Valente, 1995, S. 34).

Grundsätzlich kann ein logischer Formalismus für alle drei Verwendungszwecke eingesetzt werden, in der Praxis werden jedoch in der Regel logische Sprachen auf bestimmte Verwendungszwecke abgestimmt und optimiert.

Im Bereich der KI und Recht wurde die Logik vor allem für die Modellierung der juristischen Schlussfolgerung[108], für die Repräsentation rechtlichen Wissens[109] und die Abbildung von rechtlichen Argumenten eingesetzt. Die Modellierung von Argumenten kann als eine Kombination der Anwendung der Logik für die rechtliche Schlussfolgerung und für die Begründung von Aussagen verstanden werden.

5.4.4.2. Arten logischer Formalismen

Nachdem nun die einzelnen Verwendungsarten der Logik im Recht in Grundzügen dargestellt wurden, soll nun auf die wichtigsten Arten logischer Formalismen eingegangen werden.

[108] Hier ist der Einsatz von nicht-monotonen Logiken hervorzuheben.

[109] Für die Repräsentation von rechtlichem Wissen wurde beispielsweise die deontische Logik eingesetzt.

Deontische Logik: Die deontische Logik[110] versucht, die deontischen Begriffe – Erlaubnis, Verbot und Verpflichtung – zu formalisieren. Das Hauptargument für die Verwendung der deontischen Logik ist, dass in Rechtstexten die soeben genannten deontischen Begriffe verwendet werden. Ein stärkeres Argument ist die Aussage, dass die deontische Logik als Werkzeug verwendet werden kann, um den Unterschied zwischen dem „was gesollt und was tatsächlich ist", und zwischen dem „Ideal und dem Istzustand" zu repräsentieren.

Nicht-monotone Logik: Eine charakteristische Eigenschaft der monotonen Logik ist, dass einmal abgeleitete Konklusionen nicht wieder zurückgezogen werden können. Mit anderen Worten, sobald eine Konklusion wahr oder falsch ist, kann dieses Ergebnis nicht wieder revidiert werden, wenn neue Informationen verarbeitet werden, die für ein anderes Ergebnis sprechen. Nicht-monotone Logik unterscheidet sich in dieser Hinsicht wesentlich, da neue Informationen zu einer Änderung bereits abgeleiteter Schlüsse führen können. Nicht-monotone Formalismen werden daher verwendet, um die sogenannte „anfechtbare Schlussfolgerung"[111] zu implementieren. Dabei handelt es sich um eine Form der Schlussfolgerung, in der Schlüsse durch weitere Informationen „angefochten" werden können.

Das wichtigste Argument für die Verwendung nicht-monotoner Logik ist die Tatsache, dass rechtliche Normen in der Regel Ausnahmen enthalten und die aus ihnen abgeleiteten Rechtsfolgen nicht zwingend anwendbar sind, selbst wenn der Tatbestand erfüllt ist. Diese sogenannte „Anfechtbarkeit" von Rechtsnormen kann sich auf ein solides rechtstheoretisches Fundament stützen und findet weitgehend Zustimmung.

Weitere Argumente für die Verwendung nicht-monotoner Logik sind die Repräsentation der Dynamik des Rechtssystems, die Konkurrenz von Rechtsquellen und die semantische Unbestimmtheit. Überdies wurde die Verwendung nicht-monotoner Logik für die Modellierung rechtlicher Argumentation und der kontradiktorischen Schlussfolgerung vorgeschlagen.

Es wurden verschiedene Methoden der nicht-monotonen Schlussfolgerung entwickelt, als Beispiel sei die Abduktion (Gordon, 1991) erwähnt.

---

[110] Eine aktuelle Mongrafie über die deontische Logik findet sich in Gabbay et al. (2013).

[111] Auch „anfechtbares Schließen", engl. „defeasible reasoning".

Beschreibungslogik: Durch das Semantic Web hat eine bestimmte Art von Computermodell besondere Bedeutung erlangt, die Ontologie. Für die Modellierung von Ontologien wird üblicherweise die Web Ontology Language (OWL) oder die aktuelle Version OWL 2 eingesetzt. OWL, so wie auch OWL 2, basiert auf Beschreibungslogik und damit im Grunde auf Prädikatenlogik erster Ordnung. Die Beschreibungslogik ist demnach der Kategorie der monotonen Logik zuzuordnen.

Charakteristisch für die Beschreibungslogik ist die Unterscheidung zwischen der „Terminological Box" (TBox) und der „Assertional Box" (ABox). Die TBox enthält terminologische Axiome, also das Wissen über die Begriffe einer Domäne und deren Beziehungen. Die ABox enthält Instanzen dieser Begriffe und Beziehungen zwischen diesen. Damit repräsentiert letztere das in einem Bereich vorhandene Wissen (vgl. Valente, 1995, S. 117).

5.4.4.3. Kritische Würdigung

Nachdem im vorherigen Abschnitt die Ausprägungen des logischen Ansatzes und dessen unterschiedliche Rollen in der KI überblicksmäßig dargestellt wurden, soll nun eine kritische Auseinandersetzung folgen.

In diesem Sinn stellt sich die erste Frage, ob logische Formalismen überhaupt geeignet sind, das Recht und die rechtliche Entscheidungsfindung zu modellieren. Die Rechtstheorie stand dem Einsatz der Logik zur Lösung juristischer Probleme schon immer kritisch gegenüber. Die Argumente reichen von Zweifeln über die Anwendung der Deduktion als Grundlage für die rechtliche Entscheidungsfindung bis zum Fehlen von geeigneten Wahrheitswerten für Normen.

Die von Valente vorgebrachte Kritik betrifft weniger die Adäquanz der Logik und formaler Sprachen im Allgemeinen, um das Recht zu repräsentieren, sondern konzentriert sich auf die Frage, wie gut die Logik die an sie gestellten Anforderungen erfüllt. Die Anforderungen resultieren dabei aus dem jeweiligen Verwendungszweck der Logik in der KI (Valente, 1995, S. 15 ff.).

Das grundsätzliche Problem des Einsatzes der Logik für die *Repräsentation von rechtlichem Wissen* ist, dass den epistemologischen Konsequenzen, die durch den Einsatz eines Formalismus zwingend folgen, zu wenig Beachtung geschenkt wird. Der wichtigste

epistemologische Aspekt eines Formalismus sind die mit ihm zwingend verbundenen ontologischen Verpflichtungen. Die Ausdrucksstärke einer Sprache bildet die Grenze der Darstellungsmöglichkeiten. Durch die Bestimmung einer Sprache erfolgt daher schon eine gewisse Vorauswahl der Entitäten und deren Eigenschaften, die im Modell darstellbar sind. Durch die Anwendung eines Formalismus auf einen bestimmten Bereich entsteht eine Repräsentation einer Konzeptualisierung dieses Bereichs. Fehler der Konzeptualisierung führen zu unerwünschten Folgerungen oder logischen Inkonsistenzen. Die Ursache solcher Fehler kann jedoch nicht nur in einer fehlerhaften Konzeptualisierung liegen, sondern auch der verwendete Formalismus kann inkonsistent sein. Nach Valente beschäftigen sich die Diskussionen überwiegend mit Problemen der zweiten Art, also der Qualität der Logik, und nicht mit deren Eignung zur Lösung von bestimmten Problemen. Die praktische Dimension – die Modellierung des Rechts – wird nur unzureichend berücksichtigt.

Im Hinblick auf die Verwendung der Logik zur *Schlussfolgerung* ist fraglich, ob ein logisches Modell rechtlicher Entscheidungsfindung überhaupt geeignet sein kann. Sämtliche logische Formalismen verwenden ein bestimmtes Schema zur Ableitung von Aussagen (Inferenzen), wie beispielsweise ein nicht-monotones Ableitungssystem, das für sämtliche Schlussfolgerungen homogen verwendet wird. Damit ist es unzweifelhaft möglich, bestimmte Aussagen korrekt zu modellieren. Die Entwicklung von Problemlösungsstrategien erfordert jedoch eine differenzierte Betrachtung und Sprache. Es ist notwendig, dass Aufgaben, Ziele und Methoden auf einer höheren Abstraktionsebene beschrieben werden können (Chandrasekaran, Johnson, & Smith, 1992). Eine Inferenzmaschine mag in der Lage sein manche dieser Anforderungen zu erfüllen, jedoch nicht alle. Die verwendeten Methoden zur Schlussfolgerung sind oftmals zu allgemein und können daher nicht effizient in der Praxis eingesetzt werden. Aus diesem Grund kommen in wissensbasierten Systemen oftmals mehrere Inferenzmaschinen zum Einsatz, um das vorhandene Wissen bestmöglich zu nutzen.

Wie bereits erwähnt, vertritt Valente die Ansicht, dass die Forschungen oftmals die praktische Anwendung der logischen Formalismen und deren Implementierung in Computersystemen nur unzureichend berücksichtigen. Obwohl unter anderem die deontische Logik als Werkzeug zur Modellierung des Rechts eine gut entwickelte Forschungslinie hervorgebracht hat, fehlt es bislang an einer Anwendung, die die deontische

Logik zur Schlussfolgerung einsetzt. Es sei jedoch darauf hingewiesen, dass State-of-the-Art-Engines, wie SPINdle (Lam & Governatori, 2009), in der Lage sind, deontische Begriffe mit ausreichender Präzision zu repräsentieren.

Trotz aller Kritik, die dem Einsatz der Logik im Recht entgegenstehen mag, darf nicht übersehen werden, dass im Recht durchaus logische Strukturen erblickt werden können, die einer angemessenen Repräsentation bedürfen, wie Philipps (2012) gezeigt hat.

5.4.5. Argumentation

Eine wichtige Lektion, die aus den Forschungen auf dem Gebiet Künstliche Intelligenz und Recht gelernt wurde, ist, dass juristische Entscheidungsfindung primär nicht als die deduktive Anwendung von Logik gesehen werden kann. Tatsächlich hat niemand in diesem Fachgebiet ernsthaft behauptet, dass juristische Entscheidungsfindung ein gänzlich deduktiver Prozess ist. Kritiker der Anwendung Künstlicher Intelligenz auf juristische Problemstellungen haben dies falsch verstanden oder fehlinterpretiert, indem sie annahmen, dass dies der Fall sei, wie von Rissland et al. (2003, S. 4) ausgeführt wird:

„Contrary to some popular notions, law is not a matter of simply applying rules to facts via modus ponens, for instance, to arrive at a conclusion. Mechanical jurisprudence, as this model has been called, is somewhat of a strawman. It was soundly rejected by rule skeptics like the realists. As Gardner puts it, law is more 'rule-guided' than 'rule-governed'."

Der Verweis auf Gardner bezieht sich auf deren Dissertation, die von Rissland (1988) diskutiert wird. Juristische Entscheidungsfindung ist nicht nur deduktiv, da juristische Begriffe nicht durch notwendige und hinreichende Bedingungen definiert werden können. Zwar können solche Versuche unternommen werden, besser werden jedoch juristische Definitionen als Hypothesen oder Theorien betrachtet, denen nicht „blind" oder „mechanisch" gefolgt wird, indem Deduktion bei der Subsumtion von Einzelfällen angewendet wird. Juristische Begriffe besitzen eine „offene Struktur" (Hart, 2009). Ob ein juristischer Begriff im Einzelfall anzuwenden ist, bedarf der Interpretation der Gesetze im Lichte früherer Fälle, dem Zweck der Bestimmung und sozialen Wertungen.

Der Prozess zu bestimmen, ob die Fakten eines Falles unter einen Begriff subsumiert werden können wird *Argumentation* genannt, wobei auch *informale Logik* gebräuchlich ist (Walton, 2005). Juristische Argumentation ist ein Dialog, der vom formellen Recht vorgegeben und

angeleitet wird. Je nach Verfahrensart und Situation sind unterschiedliche Verfahrensregeln anwendbar.

Obwohl demnach immer schon die Ansicht verbreitet war, dass es bei der juristischen Entscheidungsfindung im Wesentlichen um Argumentation geht, dauerte es bis zur Veröffentlichung zweier grundlegender Papers, in einer speziellen Ausgabe des International Journal of Man-Machine Studies über KI und Recht (Gordon, 1991; Routen & Bench-Capon, 1991), dass Argumentation ein heißes Thema wurde. Danach folgten ernsthafte Anstrengungen, Argumentationstheorie mit fallbasierten, regelbasierten und anderen Ansätzen der juristischen Entscheidungsfindung zu integrieren. Im Lauf der Zeit kamen die prozeduralen Aspekte in den Fokus, die Argumentation als einen Dialog sehen und nicht bloß Vor- und Nachteile vergleichen (Amgoud, Maudet, & Parsons, 2000; Gordon, 1993; Prakken & Sartor, 1998; Walton, 2011). Ebenso wurde ein dialogischer Ansatz entwickelt, der Argumente aus Regeln und Fällen kombiniert (Prakken & Sartor, 1998).

Nach dieser kurzen Einführung in die Historie der Argumentation blieb bislang die Frage offen, was unter einem „Argument“ im Forschungsgebiet KI und Recht zu verstehen ist?

Vorausgeschickt sei, dass in der Literatur über Argumentationstheorie keine einheitliche Definition des Begriffs „Argument“ existiert. Nach Walton (2009, S. 2), einem kanadischen Philosophen, der als führender Argumentationstheoretiker gilt, kann ein Argument wie folgt definiert werden:

“An argument is a set of statements (propositions), made up of three parts, a conclusion, a set of premises, and an inference from the premises to the conclusion. An argument can be supported by other arguments, or it can be attacked by other arguments, and by raising critical questions about it.”

Die aus einem Argument abgeleitete Konklusion ist nicht zwingend wahr, sondern kann von anderen Argumenten attackiert werden. Es ist sogar möglich, dass die Konklusion unrichtig ist, obwohl die Prämissen des Arguments wahr sind. Im Gegensatz dazu ist ein logischer Beweis zwingend wahr, wenn seine Prämissen wahr sind.

Zusammengefasst bestehen vier charakteristische Unterschiede zwischen Argumenten und Beweisen (Bench-Capon & Prakken, 2006):

- Das Ziel eines Arguments ist zu überzeugen, wohingegen ein Beweis Akzeptanz erfordert.
- Argumente lassen bestimmte Prämissen implizit, in einem Beweis müssen sämtliche Prämissen explizit sein. Neue Prämissen können in einem Argument herangezogen werden, sofern dies notwendig ist, um den Zuhörer zu überzeugen, die Konklusion zu akzeptieren.
- Weitere Informationen können in einem Argument verwendet werden, sobald diese verfügbar sind, Beweise setzen voraus, dass sämtliche Informationen im Vorfeld vorhanden sind.
- Daraus folgt, dass Argumente immanent anfechtbar sind[112].

Bei der Argumentation handelt es sich primär um eine Methode der juristischen Entscheidungsfindung. Dennoch können Argumente auch als Ansatz zu Strukturierung von Wissen auf höherer Ebene gesehen werden, der beispielsweise auf nicht-monotoner Logik, Fällen oder Regeln aufbaut (Gordon, 2008a, S. 182, 2010a, S. 5 ff.). Darüber hinaus können Argumente aus Analogien und aus der Ähnlichkeit von Fällen konstruiert werden (Araszkiewicz, 2011; Walton, 2010, 2013a, 2013b).

5.4.5.1. Kritische Würdigung

Die Erforschung und Entwicklung von computergestützten Argumentationsmodellen hat sich zum wohl wichtigsten Zweig des Forschungsgebiets Künstliche Intelligenz und Recht entwickelt. Die Bedeutung dieses Forschungsbereichs wird deutlich, vergegenwärtigt man sich die zahlreichen Papers der ICAIL-[113] (Bench-Capon, Prakken, Wyner, & Atkinson, 2013; Prakken & Sartor, 2007; Wyner et al., 2011) und JURIX-Konferenzen[114] (Al-Abdulkarim, Atkinson, & Bench-Capon, 2014; Prakken & Sartor, 2013; Sartor, Walton, Macagno, & Rotolo, 2014), die sich mit dem Thema „Argumentation“ in einer Weise befassen[115]. Erstmals wurde die International Conference on AI and Law 1995 ganz im Zeichen der Argumentation

[112] Die Argumentation im Forschungsgebiet der KI und Recht ist daher eine Form der anfechtbaren Schlussfolgerung (defeasible reasoning).

[113] International Conference on Artificial Intelligence and Law.

[114] International Conference on Legal Knowledge and Information Systems.

[115] Bei den zitierten Arbeiten handelt es sich bloß um eine exemplarische Auswahl, es existieren zahlreiche weitere Werke, die sich mit dem Thema „Argumentation“ befassen.

ausgerichtet. Die seit 2006 stattfindende Konferenz Computational Models of Argument (COMMA) widmet sich sogar ausschließlich computergestützten Argumentationsmodellen.

Das gesteigerte Interesse an computergestützter Argumentation hat zahlreiche Varianten hervorgebracht, von Dungs abstraktem Argumentationsframework (Dung, 1995) über Prakkens ASPIC+ Framework für strukturierte Argumentation (Modgil & Prakken, 2014; Prakken, 2010) bis zu Gordons Carneades Argumentation System (Gordon & Walton, 2006; Gordon, 2010a, 2011a, 2011b). Einen recht aktuellen Überblick über vorhandene Ansätze computergestützter Argumentation geben Bench-Capon und Dunne (2007), und insbesondere Bentahar, Moulin und Bélanger (2010).

Argumentation und Argumentationstheorie sind an sich sehr komplexe Materien. Das Problem wird durch die Pluralität verfügbarer Argumentationsmodelle zusätzlich verstärkt. Beide Faktoren tragen wohl dazu bei, dass Argumentationssysteme in der juristischen Praxis und im E-Government nicht eingesetzt werden. Die verfügbaren Modelle hätten meines Erachtens jedoch bereits einen Reifegrad erreicht, der die Entwicklung von Prototypen zur Entscheidungsunterstützung ermöglichen würde.

Um Argumentationsmodelle einsetzen zu können, ist noch immer erforderlich, dass das juristische Wissen (Fälle, Normen etc.) von einem Experten aufbereitet und modelliert wird. Methoden zur Extraktion des relevanten Wissens aus juristischen Dokumenten (Natural Language Processing, kurz NLP) haben noch nicht die notwendige Präzision erreicht, um diese Aufgabe zu übernehmen. Der Engpass bei der automatisierten Akquisition von Wissen (knowledge acquisition bottleneck) ist nach wie vor vorhanden und bleibt ungelöst[116] (Ashley & Walker, 2013). Es sei jedoch angemerkt, dass in jüngster Zeit auch im deutssprachigen Raum vermehrt Anstrengungen zur automatisierten Analyse und Extraktion von Argumenten unternommen werden (Houy, Niesen, Calvillo, Fettke, & Loos, 2014). Spannend bleibt die Frage, ob sich in naher Zukunft Werkzeuge zur automatisierten Extraktion von Argumenten durchsetzen können und einen Reifegrad erreichen, der eine praktische Verwendung zulässt.

[116] Der Engpass bei der Akquisition von Wissen ist für alle Ansätze der juristischen Wissensrepräsentation eine Herausforderung, nicht nur für computergestütze Argumentationsmodelle.

In Zukunft wird eine Integration von Argumentationssystemen mit „einfachen“, unter Umständen bereits vorhandenen, regelbasierten Systemen notwendig sein, um das volle Potential wissensbasierter Systeme zu entfesseln.

5.4.6. Sonstige Ansätze

Neben den dargestellten Ansätzen zur juristischen Wissensrepräsentation gibt es noch weitere Forschungslinien, die hier überblicksmäßig dargestellt werden.

Begriffsbasierte Formalismen: Zu dieser Familie gehören Ansätze, deren Schwerpunkt die Darstellung von Begriffen ist, wie semantische Netze und begriffliche Graphen. Valente ordnet auch die objektorientierte Programmierung dieser Kategorie zu (Valente, 1995, S. 18).

Neurale Netzwerke: Solche Systeme emulieren die assoziativen Fähigkeiten von Neuronen im menschlichen Gehirn. Obwohl dieser Ansatz ein ernstzunehmendes Gebiet für wissenschaftliche Forschungen ist, hat er sich im Bereich des Rechts nicht durchgesetzt. Neurale Netzwerke repräsentieren das gespeicherte Wissen nicht unmittelbar, und es lässt sich nicht nachvollziehen, wie ein Ergebnis zustande gekommen ist. Die Begründung von Ergebnissen ist jedoch gerade im Recht eine unverzichtbare Notwendigkeit, vgl. (Boer, 2009, S. 24).

Information Retrieval: Bei diesen Forschungen geht es vor allem um die Entwicklung von juristischen Informationssystemen. Dazu gehören beispielsweise Datenbanken, die Rechtstexte und Dokumente speichern. Die gespeicherten Daten können dann mithilfe einer booleschen Suche abgefragt werden. Juristische Informationssysteme leisten wertvolle Hilfestellungen für den Rechtsanwender für die Lösung rechtlicher Probleme, da sie rasch große Mengen an Textkorpora durchsuchen können, um die relevanten Informationen zu finden. Forschungen in diesem Bereich beschäftigen sich insbesondere mit der automatisierten Analyse von Rechtstexten und der Verbesserung von Suchergebnissen (Ferber, 2003; Schweighofer, 1999). In jüngster Zeit wurden Anstrengungen unternommen, das Information Retrieval mit Technologien des Semantic Web zu integrieren, um Suchergebnisse zu verbessern und semantische Informationen aus Dokumenten zu extrahieren (Buey, Garrido, & Ilarri, 2014; Jothilakshmi, Shanthi, & Babisaraswathi, 2013; Schweighofer & Liebwald, 2007; Schweighofer, 2010). Der Trend geht weg von einer rein

syntaktischen Suche von Übereinstimmungen hin zur Auffindung semantischer Strukturen, wie beispielsweise von Argumenten (Ashley & Walker, 2013).

5.4.7. Abschließende Bemerkungen

In diesem Kapitel wurde ein Überblick über die wichtigsten Ansätze der juristischen Wissensrepräsentation gegeben, wobei auf die Art des verwendeten Formalismus zur Unterscheidung abgestellt wurde. Grundsätzlich gelangt man nahezu zum selben Ergebnis, zieht man die Struktur heran (Regeln, Fälle, logische Aussagen, Argumente), die zur Repräsentation von juristischem Wissen verwendet wird. Die Übergänge sind jedenfalls fließend, wie eingangs bereits angemerkt wurde.

Offen blieb bislang, welchem Ansatz Ontologien zuzurechnen sind. Dazu ist anzumerken, dass für die Repräsentation von Ontologien verschiedene Sprachen mit unterschiedlichem Formalisierungsgrad zur Verfügung stehen. Eine ziemlich vollständige Auflistung von Sprachen zur Modellierung von Ontologien findet sich in Casellas (2011, S. 80 ff.). Valentes „Functional Ontology of Law“ wurde beispielsweise in Ontolingua modelliert.

Heute werden Ontologien üblicherweise mithilfe von Beschreibungslogik erstellt, hier ist insbesondere OWL 2 hervorzuheben. Es gibt aber auch Ansätze, die die Verwendung alternativer Sprachen, wie z.B. F-Logic, untersuchen (Angele, Kifer, & Lausen, 2009).

Eine Ontologie, als ein spezielles computergestütztes Modell, ist demnach unabhängig vom zur Modellierung verwendeten Formalismus. Dies zeigt auch der Umstand, dass Probleme bei der Modellierung von Ontologien solche des „knowledge level“ und nicht des „symbol level“ nach Newell sind (Newell, 1982; vgl. Valente, 1995, S. 36). Der Weg über den zur Modellierung gewählten Formalismus bringt uns damit einer Kategorisierung der Ontologie nicht näher.

Zielführender scheint es, die Struktur näher zu betrachten, die in Ontologien verwendet wird, um Wissen zu modellieren. In der Informatik ist eine Ontologie eine Repräsentation der Begriffe und deren Relationen. Sie besteht aus Klassen und Instanzen dieser Klassen, die gemeinsam das Vokabular einer Domäne repräsentieren. Instanzen werden durch die Zugehörigkeit zu bestimmten Klassen typologisch in Subklassen organisiert.

Diese Ausführungen bringen uns dem Ziel der Kategorisierung des ontologischen Ansatzes schon deutlich näher. Sie zeigen, dass Ontologien vor allem die Strukturierung und

Kategorisierung von terminologischem Wissen zum Ziel haben. Ontologien lassen sich daher in die Kategorie der begriffsbasierten Formalismen einordnen, wobei diese üblicherweise in einer entscheidbaren Untermenge der Prädikatenlogik erster Stufe, wie der Beschreibungslogik, modelliert werden. Sie weisen damit eine solide logische Grundlage auf, womit diese auch den logischen Ansätzen zugerechnet werden könnten. Der Schwerpunkt der Modellierung von Ontologien liegt jedoch nicht in der Formulierung logischer Aussagen, die Zuordnung zu den begriffsbasierten Formalismen erscheint daher treffender.

5.5. Aktuelle Probleme

Die Forschungen auf dem Gebiet der Künstlichen Intelligenz reichen nunmehr bereits etwa 50 Jahre zurück. Dabei wurden bei der Anwendung von KI-Modellen auf rechtliche Probleme sowohl Erfolge als auch Misserfolge erzielt. Das hehre Ziel, die vollautomatisierte Entscheidungsfindung, wurde bisher jedoch (noch) nicht erreicht. Obwohl diese Aussage in ihrer Allgemeingültigkeit sicherlich hinterfragt werden kann, behält sie doch weitgehend ihre Gültigkeit, insbesondere dann, wenn die Lösung eines Falls die Ausübung richterlichen Ermessens erfordert[117]. Im Verwaltungsrecht gibt es jedoch seit den 1970er-Jahren Programme, die eine weitgehend vollautomatisierte Durchführung einfacher Verfahren bewerkstelligen. Die Leistungsfähigkeit aktueller Technologien im Hinblick auf die Formalisierung von Verwaltungsrecht und die automatisierte Verfahrensdurchführung wird später noch erörtert.

Trotz aller Bemühungen verstehen Maschinen das Recht noch immer nicht wie Menschen. Im Folgenden werden aktuelle Probleme im Zusammenhang mit der Formalisierung juristischer Entscheidungsfindung diskutiert. Die Darstellung folgt dabei Franklin (2012), der die Ansicht vertritt, dass die Probleme, die eine Formalisierung allgemeiner Denkvorgänge so schwierig machen, exakt dieselben sind, die auch der Formalisierung juristischer Entscheidungsfindung entgegen stehen. Dies wird besonders augenscheinlich, wenn für die rechtliche Beurteilung auf menschliches Allgemeinwissen zurückgegriffen werden muss.

[117] Fälle, in denen der Richter Ermessen ausüben muss, werden „schwierige“ Fälle in der Terminologie von Hart genannt, im Gegensatz zu „einfachen“ Fällen, die weitgehend „mechanisch“ gelöst werden können (Hart, 2009).

Unterstützung findet diese Meinung unter anderem durch Boer, der in seiner Dissertation die Ansicht vertritt, dass die rechtliche Entscheidungsfindung sich nicht von allgemeinen menschlichen Denkprozessen unterscheidet, jedoch rechtliches Wissen benötigt (Boer, 2009, S. 9):

„Legal reasoning is straightforward common sense reasoning, but with legal knowledge."

Franklin macht keine näheren Ausführungen, was genau unter „common sense knowledge" zu verstehen ist, verweist jedoch in diesem Zusammenhang auf das Cyc-Projekt von Douglas Lenat, einer maschinenauswertbaren Wissensdatenbank des Allgemeinwissens. Aus diesem Grund versteht Franklin unter „common sense knowledge" wohl das, was Lenat und Feigenbaum (1991) als *consensus reality* bezeichnet haben: Das allgemeine Wissen, von dem angenommen wird, dass es jeder Mensch besitzt. Dies umfasst triviales Wissen wie zum Beispiel, dass Wasser den Berg hinunter fließt, Lebewesen krank werden können und der Rasen nass wird, wenn es regnet.

Kommen wir nun zurück zur Darstellung der Probleme, die es bei einer Formalisierung zu bewältigen gilt. Vorweggenommen sei, dass der gemeinsame Kern dieser Probleme im Unterschied zwischen der Eindeutigkeit der symbolischen Mathematik[118] und der fließenden Natur der unpräzisen natürlichen Sprache, der Ähnlichkeit und Analogie, und von Wahrscheinlichkeiten besteht.

Franklin hat mehrere Problemkreise identifiziert, die im Anschluss dargestellt werden. Angemerkt sei, dass Franklin in seinem Beitrag vorwiegend Beispiele aus dem rechtlichen Bereich präsentiert, trotzdem sind – wie bereits ausgeführt – die Probleme grundsätzlich dieselben wie bei der Formalisierung von Allgemeinwissen. Die diskutierten Probleme stehen eher im Zusammenhang mit der Repräsentation von Wissen und nicht so sehr mit den Methoden der Schlussfolgerung (der Inferenz). Die aufgezeigten Schwierigkeiten sind vor dem Hintergrund zu verstehen, dass juristische Probleme stets aus Sachverhalten des täglichen Lebens resultieren, und das Recht als Schnittstelle zwischen juristischen Experten und Laien gesehen werden kann.

[118] Engl. „symbol manipulation".

5.5.1. Die Ungenauigkeit der natürlichen Sprache und von rechtlichen Begriffen

Allgemein bekannt ist, dass Begriffe der natürlichen Sprache oftmals mehrere Bedeutungen haben können. Das Problem rührt aus der Unschärfe (auch genannt Vagheit) von sprachlichen Formulierungen her. Man spricht in diesem Zusammenhang auch von Grenzfällen, da es oftmals nicht eindeutig ist, ob eine bestimmte Gegebenheit noch von einem Begriff erfasst wird oder nicht.

Begriffe wie „groß“, „teuer“, „gebildet“, „sinnvoll“ etc. sind oftmals nur auf den ersten Blick eindeutig zu verstehen. Bei genauerer Betrachtung wird jedoch deutlich, dass diese Begriffe einen Teil eines Kontinuums mit fließenden Übergängen beschreiben. Aus diesem Grund können Grenzfälle auftreten, bei denen die Zuordnung zu einem Begriff schwierig ist. Diese Problematik kommt deutlich zum Ausdruck, versucht man beispielsweise die Größe einer Person zu bestimmen.

Eine Person kann beispielsweise „eindeutig groß“, „gerade noch groß“ aber auch „weder groß noch klein“ sein. Wie das menschliche Gehirn diese Bewertung exakt durchführt, verbleibt bislang ungeklärt. Es spielt wohl das Allgemeinwissen einer Person, das sich auch aus Erfahrungswerten zusammensetzt, eine entscheidende Rolle bei der Bewertung der Größe einer Person.

Die automatisierte Entscheidungsfindung wird dadurch jedenfalls vor erhebliche Probleme gestellt. Es ist in diesem Fall schwierig, die logische Relation zwischen einem Begriff des Tatbestands und einem Sachverhaltselement zu beschreiben. Für die logische Schlussfolgerung ist es aber unabdingbar, dass der Wahrheitsgehalt einer Aussage eindeutig mit „wahr“ oder „falsch“ beantwortet werden kann.

Zur Lösung dieser Probleme wurde die Verwendung von Fuzzy-Logik vorgeschlagen. Der Einsatz von Fuzzy-Logik sowie weitere Details im Hinblick auf die Formalisierung unscharfer Begriffe wurden bereits in Abschnitt 4.9.5.1. behandelt. Für weitergehende Ausführungen sei daher auf diesen verwiesen.

Ein Problem, das bisher noch nicht behandelt wurde, ist die „offene Struktur“ von Begriffen. Die oben angesprochenen Grenzfälle sind ein Resultat davon. Nicht so offensichtlich ist jedoch, dass es sich dabei exakt um das Problem der Unschärfe von sprachlichen Formulierungen handelt.

In Harts (2009) berühmtem Beispiel „No vehicles are allowed in the park" geht es im Wesentlichen darum, die Kluft zwischen "vehicle" und einem konkreten Fortbewegungsmittel, z.B. einem Motorrad, zu überwinden. Der Begriff „Fahrzeug" hat gewissermaßen eine Bedeutungspenumbra, d.h. seine Bedeutung hat „dunkle Flecken". Dies deutet darauf hin, dass – wie bei den Unscharfen Begriffen – eine mathematische Funktion hilfreich sein könnte, die ausdrückt, in wie weit ein bestimmtes Fortbewegungsmittel vom Begriff des Fahrzeugs in diesem Fall erfasst wird. Diese Funktion könnte beispielsweise den Wert 1 für Autos, 0,5 für Motorräder und 0,1 für Rollschuhe liefern.

Eine derartige mathematische Funktion weist jedoch nur einen sehr beschränkten Nutzen für die automatisierte Entscheidungsfindung auf. Oftmals ist es für die Bestimmung der Anwendbarkeit einer Norm im Einzelfall erforderlich, deren Sinn zu ermitteln[119]. Im gegenständlichen Beispiel könnte insbesondere das Risiko, dass Fahrzeuge Besucher des Parks verletzen, relevant sein, sodass Motorräder und Autos – nicht jedoch Rollschuhe – verboten sind. Der Sinn des Verbots ist viel wichtiger für die Entscheidung im Einzelfall als das abstrakte Ergebnis einer Mitgliedschaftsfunktion. Man könnte auch sagen, der Grad der Mitgliedschaft eines Sachverhaltselements an einem Begriff ist vom jeweiligen Kontext abhängig, in den Sinnzusammenhänge einfließen.

Das Problem der offenen Struktur des Rechts kann jedenfalls nicht zufriedenstellend gelöst werden, indem man sich auf eine Fuzzy-Mitgliedschaftsfunktion stützt.

5.5.2. Ähnlichkeit und Analogie

Ähnliche Probleme treten auf, wenn die Ähnlichkeit von Präzedenzfällen zu einem aktuellen Fall beurteilt werden soll, um auf dieser Basis eine Entscheidung zu treffen. Die Ähnlichkeit zwischen Rechtsfällen könnte ebenso mit einer Fuzzy-Funktion beschrieben werden, analog zur vorhergehenden Darstellung zur Behandlung vager Begriffe. Die Ähnlichkeit zwischen Rechtsfällen wird aber eher durch argumentative Begründung festgestellt. Die beste Begründung geht schlechteren, quasi in Form einer Alles-oder-nichts-Entscheidung, vor.

Dieses Problem wurde in der KI oftmals im Zusammenhang mit dem Fall *Popov vs. Hayashi* diskutiert (Franklin, 2012, S. 228). In diesem Fall geht es um die Erlangung von Besitz an

[119] Zur Ermittlung des Sinns einer Norm wendet der Jurist die teleologische Interpretationsmethode an.

einem Baseball, der sogleich wieder verloren geht. Dem Zuschauer Popov gelang es, einen Baseball zu fangen, der sich auf die Zuschauertribüne verirrt hatte. Bevor er den Ball jedoch vollständig sichern konnte, wurde er von einer Menschenmenge niedergestoßen, die ebenfalls versuchte, den Ball zu fangen. Der Ball rollte auf den Boden und Hayashi, der selbst kein Teil der Menschenmenge war, gelang es den Ball an sich zu nehmen.

Bei den Präzedenzfällen, die zur Beurteilung herangezogen wurden, ging es um die Jagd. In diesen Fällen wurden Probleme des (teilweisen) Besitzes an der erlegten Beute diskutiert. Die Schwierigkeit besteht darin herauszufinden, welche Fälle für den gegenständlichen Fall relevant sind und einer Betrachtung unterzogen werden müssen. Auf den ersten Blick hat beispielsweise ein gefangener Fuchs nichts mit dem Baseball im gegenständlichen Fall zu tun. Durch Abstraktionsvorgänge zeigt sich jedoch, dass es im Fall des Fuchses und des Baseballs im Kern um dasselbe Problem geht, nämlich jenes des (teilweisen) Besitzes, der wieder verloren geht. Damit ein Computersystem Präzedenzfälle selbstständig auffinden kann, bedarf es einer Theorie, um die relevanten und irrelevanten Dimensionen eines Falles zu identifizieren (Franklin, 2012, S. 228 f.).

Die Komplexität der Ambitionen eines KI-Projekts hängt maßgeblich davon ab, ob das System selbstständig in der Lage sein soll, Fälle nach relevanten Faktoren zu durchsuchen, die eine bestimmte Rechtsmeinung unterstützen, oder ob diese Aufgabe vom Menschen übernommen wird.

5.5.3. Die Repräsentation des Kontexts

Die Wichtigkeit der Berücksichtigung des Kontexts ist in der KI, ebenso wie in der Linguistik und der Kognitionspsychologie, allgemein anerkannt. In diesem Zusammenhang sei der Hermeneutische Zirkel in Erinnerung gerufen. Neuer Text wird im Kontext des gesamten Dokuments interpretiert, und neues Wissen führt zu einer Veränderung des Verständnisses des ganzen Texts.

Trotz der Wichtigkeit des Kontexts ist es schwierig, einen so vagen Begriff näher zu definieren, um ihn einer weiterführenden Diskussion zugänglich zu machen. Franklin diskutiert das Problem des Kontexts am Beispiel der Klassifizierung von Tieren (Franklin, 2012, S. 229 ff.). Franklin wirft die Frage auf, wie die Klassifizierung von Tieren automatisiert

durchgeführt werden kann? Die wichtigsten Ausführungen werden an dieser Stelle überblicksmäßig dargestellt.

Die Klassifizierung von Tieren im rechtlichen und biologischen Bereich ist vollkommen verschieden. Im rechtlichen Bereich geht es im Hinblick auf Schäden, die von Tieren verursacht werden, darum, ob es sich um zahme Haus- oder Nutztiere handelt, die einem Halter zugerechnet werden können, oder um Wildtiere (*ferae naturae*) (vgl. § 1320 ABGB). In der Biologie geht es hingegen insbesondere um die Zuordnung von Tieren zu einer bestimmten Gattung nach biologischen Merkmalen.

Manchmal ist es schwierig zu beurteilen, ob ein Tier wild oder bereits gezähmt ist und einem Halter zugerechnet werden kann. Eindeutig können beispielsweise Kühe oder Löwen zugeordnet werden. Bei anderen Tieren ist die Bestimmung weit schwieriger und es ist ausschlaggebend, in welchem Kontext, insbesondere in welcher Umgebung, diese angetroffen werden. Bienen sind grundsätzlich Wildtiere, werden jedoch zum Eigentum desjenigen, der diese fängt. Sobald sie ausschwärmen, werden sie jedoch wieder zu *ferae naturae*[120]. Vernünftigerweise kann ein gefangener Bär nicht als Haustier bezeichnet werden, ebenso wenig wie eine gezähmte Möwe.

Für die rechtliche Zurechnung von Tieren muss damit der Kontext berücksichtigt werden, in dem das Tier angetroffen wird. Es geht um die Beantwortung der rechtlichen Frage, ob ein Tier vernünftigerweise als das Eigentum eines Halters angesehen werden kann. Im rechtlichen Kontext ist daher sowohl der eben erwähnte gefangene Bär als auch die zahme Möwe als das Eigentum eines Halters zu betrachten und diesem daher zurechenbar. Insofern können die erwähnten Tiere als „Haus-“ oder „Nutztiere“ im *rechtlichen* Sinn betrachtet werden.

Besonders augenscheinlich wird das Problem der Repräsentation des Kontexts bei der Übersetzung von natürlicher Sprache. Diese Schwierigkeiten übertragen sich auf andere Bereiche, wie beispielsweise die Repräsentation des Hintergrundwissens, das notwendig ist, um allgemeine und rechtliche Aussagen zu interpretieren.

[120] Zur österreichischen Rechtslage vgl. § 384 ABGB: Erst wenn der Eigentümer des Mutterstockes den Schwarm über zwei Tage nicht verfolgt hat, können sich andere den Schwarm aneignen.

Folgende drei Probleme können identifiziert werden, die es bei der Repräsentation des Kontexts zu bewältigen gilt:

(1) Erstens müssen der Kontext und neue Informationen zusammenwirken. Franklin nennt diese Anforderung das „meshing problem“. Die Informationen werden dabei grundsätzlich im jeweiligen Kontext interpretiert. Allerdings ist es möglich, dass neue, gewichtige Informationen zu einer Änderung des Kontexts führen. Es bedarf einer sinnvollen Abstimmung, da auf der Hand liegt, dass nicht jede neue Information zu einer Änderung des Kontexts führen darf.

(2) Zweitens ist es schwierig herauszufinden, in welchem Kontext sich ein Begriff befindet. Die Information um festzustellen, in welchem Kontext man sich befindet, ist jedoch selbst wiederum in einen Kontext eingebettet. Es handelt sich demnach um ein zirkuläres Problem, das schwer gelöst werden kann.

(3) Drittens kann es vorkommen, dass für die Interpretation einer Aussage mehrere Kontexte, wie beispielsweise der nachfolgende oder verschachtelte, berücksichtigt werden müssen. Es ist schwierig für einen Computer, alle relevanten Kontexte zu finden, gegebenenfalls müssen zuerst Widersprüche gelöst werden. Überdies stellt sich das erste Problem verstärkt, wenn neue Informationen mit den vorhandenen Kontexten zusammenwirken müssen.

Der Untersuchung der Repräsentation des Kontexts wurde beachtlicher Aufwand im Bereich der KI gewidmet. Den Bemühungen zum Trotz blieb eine zufriedenstellende Lösung bislang aus (Schweighofer, 1999).

5.5.4. Das Problem der Symbolverankerung

Beim Problem der Symbolverankerung[121] geht es im Wesentlichen um die Frage, wie Begriffe ihre Bedeutung erlangen.

Wie im vorherigen Abschnitt ausgeführt wurde, muss jeder Begriff in einem bestimmten Kontext verstanden werden. Der umgebende Kontext ist jedoch selbst wiederum in einem anderen Kontext eingebettet. Es droht demnach ein unendlicher Regress.

[121] Engl. „symbol grounding problem“.

Wenn jeder Begriff in einem Kontext verstanden werden muss, so stellt sich letztendlich die Frage, wie denn der Kontext selbst verstanden werden soll? Anders formuliert: Was verleiht einem Begriff, der in einem Kontext erwähnt wird, letzten Endes seine Bedeutung?

Im Kern dieser Frage verbirgt sich das bereits erwähnte Problem der Symbolverankerung. Wörter können nicht bloß durch die Verwendung anderer Wörter definiert werden. Aus demselben Grund kann die Bedeutung von Wörtern der chinesischen Sprache nicht durch Auswendiglernen eines Wörterbuchs erlernt werden (Franklin, 2012, S. 233).

Die Verwendung des Wortes „Katze“ beispielsweise muss irgendwie mit den Erfahrungen, die ein Mensch über Katzen gesammelt hat, verbunden werden. Üblicherweise tritt das Problem der Symbolverankerung im Zusammenhang mit Robotern auf. Diese Maschinen müssen ihre interne Repräsentation von Symbolen mit den Eingabesensoren in Verbindung bringen, und andererseits mithilfe dieser ihre Effektoren, z.B. ihre Arme, steuern.

Menschen bereiten solche Anforderungen – im Gegensatz zu Maschinen – keine großen Schwierigkeiten. Diese allgemeine Intelligenz ermöglicht es Menschen, ad hoc auf äußere Einflüsse zu reagieren und sich zum Beispiel in einer fremden Stadt zu orientieren oder sich vor einem heranfliegenden Stein zu ducken.

Für wissensbasierte Systeme[122], die beispielsweise die rechtliche Entscheidungsfindung unterstützen, ist die Relevanz des Problems der Symbolverankerung nicht so augenscheinlich. Es tritt jedoch in zweifacher Weise in Erscheinung.

Einerseits enthält die Wissensbasis eines wissensbasierten Systems *Termini technici*, wie beispielsweise „Besitz“. Die Bedeutung dieser Begriffe sollte dieselbe sein, die z.B. die Autoren und Leser eines Texts mit diesen verbinden. Andererseits müssen solche Rechtsbegriffe auch mit den in der Datenbank des Systems gespeicherten Fällen in Verbindung gesetzt werden.

Es stellt sich die Frage, wie diese Verbindungen korrekt hergestellt werden können? Je nach Anspruch des Projekts können solche Verknüpfungen entweder manuell oder automatisiert erfolgen. Dementsprechend stellt sich das Problem der Symbolverankerung insbesondere

[122] Franklin spricht von „Expertensystemen“.

dann, wenn ein System von selbst, sozusagen „intelligent“, Rechtstexte durchsuchen und derartige Verbindungen herstellen soll.

5.5.5. Die Repräsentation der Kausalität, von Bedingungen und Alternativen

Experten auf den Gebieten der Philosophie, der KI und der Rechtsphilosophie stimmen überein, dass die Berücksichtigung von Ursachen, Bedingungen und Alternativen (Kontrafaktuale) bei der Entscheidungsfindung äußerst schwierig ist.

Das Problem zeigt sich deutlich bei der Beurteilung der Kausalität. So wird insbesondere im Straf- und Zivilrecht (aber auch in der Philosophie) die allgemein bekannte Formel „conditio sine qua non“ angewendet, um die Kausalität einer Tathandlung zu beurteilen.

War beispielsweise das Feuer, das der Beklage gelegt hat, ursächlich für die Zerstörung des Hauses des Klägers?

Um zu einer Entscheidung zu gelangen, ist die Vorstellung einer alternativen Realität, in der der Täter die Tathandlung nicht gesetzt hat, notwendig. Wäre der jeweilige Erfolg auch ohne der Handlung des Täters eingetreten? Von der Beantwortung dieser Frage ist abhängig, ob die Tathandlung kausal für den Schadenseintritt war oder nicht.

Noch komplizierter wird die Beantwortung der Frage, wenn es sich um einen Fall der alternativen oder überholenden Kausalität handelt. Wie ist die Frage der Verursachung zu beantworten, wenn beispielsweise ein zweites, sich rascher ausbreitendes Feuer, das Haus erfasst und zerstört (überholende Kausalität)? Wie verhält es sich, wenn zwei, von unterschiedlichen Tätern gelegte, Feuer das Haus gleichzeitig in Brand setzen (alternative Kausalität)?

Die Beantwortung derartiger Fragen ist freilich aus juristischer Sicht längst gelöst und gehört zur alltäglichen Arbeit. Für ein Computersystem bereitet aber hingegen schon die Beurteilung der „einfachen“ Kausalität erhebliche Schwierigkeiten. Das System benötigt dafür eine formale Darstellung alternativer Realitäten, die es für die Beurteilung heranziehen kann. Im Wesentlichen geht es also um die Formalisierung des „gewöhnlichen Laufs der Dinge“ (Franklin, 2012, S. 233 f.).

Alleine wenn man die unglaubliche Vielzahl an Alternativen betrachtet, die man für beliebige Fallkonstellationen konstruieren müsste, erscheint dieses Unterfangen aussichtslos. Überdies existiert noch kein Ansatz, wie man die erheblichen Unterschiede zwischen der

realen Welt – in der eine Tathandlung gesetzt wurde – und der alternativen Realität – in der diese Handlung nicht gesetzt wurde – repräsentieren soll. Zusammengefasst kann gesagt werden, dass die Repräsentation der Normalität, damit ist der gewöhnliche Lauf der Dinge gemeint, einen enormen Aufwand bedeutet und derzeit kein adäquater Lösungsansatz existiert.

5.5.6. Die Gewichtung von Argumenten

Die bisher diskutierten Probleme betrafen eher die Formalisierung von Fakten und weniger die Frage, wie mit diesen die eigentliche Entscheidungsfindung durchgeführt werden kann.

In diesem Zusammenhang ist vor allem die Gewichtung von Argumenten von größter Bedeutung. Argumente unterstützen die Entscheidung eines Falles in eine bestimmte Richtung. Die besseren Argumente setzen sich dabei gegen schlechtere durch. Die Gewichtung von Argumenten erfolgt oftmals anhand von Präzedenzfällen.

Es soll jedoch nicht der Eindruck entstehen, die Gewichtung von Argumenten sei nur im Common Law von Bedeutung. Vielmehr spielen Abwägungsentscheidungen auch in kontinentaleuropäischen Rechtssystemen eine wichtige Rolle. Man denke insbesondere an die Verhältnismäßigkeitsprüfung beim Eingriff in verfassungsgesetzlich gewährleistete Rechte (Grundrechte), an Ermessensentscheidungen oder die Anwendung unbestimmter Rechtsbegriffe. In solchen Fällen müssen entgegengesetzte Argumente gegeneinander abgewogen werden, um zu einer Entscheidung zu gelangen.

Die Formalisierung von Argumenten und deren Verwendung bei der automatisierten (rechtlichen) Entscheidungsfindung ist nach wie vor eines der bedeutendsten Themen im Bereich der KI und Recht.

Abschließend sei angemerkt, dass in manchen Bereichen bereits signifikante Fortschritte erzielt wurden. Es verbleiben jedoch noch einige ungelöste Probleme, insbesondere im Hinblick auf die Formalisierung von Wahrscheinlichkeiten und Annahmen, auf die an dieser Stelle bloß hingewiesen sei (Franklin, 2012, S. 236 ff.).

5.5.7. Das grundlegende Problem der Kontinuität

Der Grund für viele der oben diskutierten Probleme ist nach Franklin der Gegensatz zwischen dem Diskreten und dem Kontinuierlichen. Diese Trennung wird in der Mathematik des 17. bis 19. Jahrhunderts und jener des 20. Jahrhunderts sichtbar.

Die Gedankenwelt der Mathematik des 17. bis 19. Jahrhunderts beschäftigte sich vor allem mit Differenzialgleichungen, sanften Flüssen, schrittweisen Übergängen und Glockenkurven. Gegenstand der Mathematik des 20. Jahrhunderts sind hingegen ganze Zahlen, Kryptografie, mathematische Logik, diskrete Annäherungen und Computercode (Franklin, 2012, S. 239).

Nach Franklin sind die Forscher auf dem Gebiet der KI überwiegend dem zweiten Lager – der Mathematik des 20. Jahrhunderts – zuzurechnen. Überraschenderweise ist er der Ansicht, dass auch Rechtsanwälte – trotz ihres linguistischen Hintergrunds – dieser Richtung verhaftet sind.

Franklin plädiert für die Formalisierung der Schlussfolgerung, in der Unschärfen berücksichtigt werden müssen, durch die Verwendung von Zahlen, die – analog zur Fuzzy-Logik – die Grade des Kontinuums repräsentieren. Grundsätzlich wäre dies ein gangbarer Weg – ähnlich dem Einsatz der Fuzzy-Logik bei vagen Gesetzesbegriffen. Es stehen diesem Ansatz jedoch ähnliche Probleme wie dem Einsatz der Fuzzy-Logik zur Formalisierung vager Gesetzesbegriffe entgegen.

Abschließend seien die wichtigsten Aussagen Franklins zusammengefasst (Franklin, 2012, S. 240):

(1) Für Franklin ist es essentiell, die Theoretiker zu überzeugen, dass das Festhalten an diskreten Ansätzen der falsche Weg ist. Als Begründung führt Franklin den (fehlgeschlagenen) Versuch ins Treffen, die probabilistische Schlussfolgerung durch den Einsatz nicht-monotoner und Default-Logik, mit hierarchisch angeordneten Standardannahmen, durchzuführen. Es gilt, Argumente zu gewichten und nicht bloß zu zählen. Die Verwendung bayesscher Logik ist seines Erachtens der einzig richtige Ansatz für die Durchführung probabilistischer Schlussfolgerungen.

(2) Stimmt man grundsätzlich mit dem Ansatz überein, die Grade eines Kontinuums mithilfe von Zahlen zu repräsentieren, so stellt sich doch die Frage, woher diese Zahlen kommen sollen? Wie sollen beispielsweise die Grade ermittelt werden, die angeben, inwieweit ein

bestimmtes Beweismittel für die Schuld des Täters spricht? Ein ähnliches Problem wurde bereits bei der Verwendung von Fuzzy-Logik-Mitgliedschaftsfunktionen diskutiert.

(3) Grundsätzlich ist es möglich, ein Kontinuum durch Zahlen zu repräsentieren. Dies ist jedoch nicht die Arbeitsweise des menschlichen Gehirns. Nach Franklin mangelt es aber an einer adäquaten Alternative, sodass kein Weg an der Verwendung von Zahlen vorbei führt. Das Hauptproblem betrifft die Genauigkeit dieses Ansatzes. Die Ähnlichkeit zwischen zwei alternativen Realitäten und die durch einen bestimmten Beweis ausgedrückte Wahrscheinlichkeit sind, im Gegensatz zu Nummern, unpräzise. Es ist beispielsweise wenig zielführend, den Beweismaßstab für einen „Beweis ohne vernünftige Zweifel“ mit einer Wahrscheinlichkeit von 0,937 zu repräsentieren.

5.5.8. Menschliche Intelligenz

Wie bereits eingangs erwähnt wurde, war es die Bestrebung früher KI-Ansätze, menschliche Intelligenz durch die Verarbeitung uninterpretierter Symbole zu simulieren.

Nach 50 Jahren Erfahrung herrscht weitgehend Einigkeit, dass Intelligenz weit mehr ist als das bloße Befolgen von Regeln. Obwohl ein Computer ohne Probleme 3 x 2 und 2 x 3 rechnen kann, versteht die Maschine nicht, warum in beiden Fällen das Ergebnis gleich sein muss. Ein Mensch versteht hingegen sehr schnell, dass eine Anordnung mit zwei Reihen mit jeweils drei Elementen auch als eine Figur mit drei Spalten mit jeweils zwei Elementen begriffen werden kann.

Auch nach 50 Jahren fehlt es Maschinen, wie in den Anfängen der KI, an solch einer echten Intelligenz. Die Fortschritte in dieser Hinsicht halten sich stark in Grenzen und tendieren gegen Null (Franklin, 2012, S. 241).

Nach Ansicht Franklins ist die Fähigkeit zu verstehen die grundlegende Voraussetzung für allgemeine Denkprozesse und die rechtliche Entscheidungsfindung. Zum Beispiel ist es für die Lösung des Falls Popov vs. Hayashi wichtig zu begreifen, dass es sich um das rechtliche Problem des (teilweisen) Besitzes handelt. Weiter, dass der teilweise Besitz auf einer kontinuierlichen Skala eingeordnet werden muss. Der menschliche Verstand ist in der Lage, in solchen Fällen den Prozess der Abstraktion durchzuführen. Relevante Eigenschaften des Falles werden vom Gehirn abstrahiert und geordnet, um schließlich die rechtliche

Beurteilung durchführen zu können. Computer sind von solchen Fähigkeiten noch weit entfernt.

Die Sinnhaftigkeit der Debatte über die Simulation menschlicher Intelligenz wurde bereits am Anfang dieses Kapitels in Frage gestellt. Trotz der Tatsache, dass Computer bei Weitem nicht in der Lage sind, menschliche Denkprozesse durchzuführen, ist auch Franklin der Ansicht, dass Formalisierungsprojekte durchaus ein brauchbares Ergebnis liefern können. Entscheidend ist, welche Ziele man mit einem solchen Projekt verfolgt (Franklin, 2012, S. 242):

- Der in den Anfängen der KI gestellte Anspruch an Software, die Daten selbstständig zu interpretieren, den Entscheidungsprozess durchzuführen und das Ergebnis zu formulieren, ist ein hehres Ziel, jedoch – selbst mit heutiger Technologie – noch in weiter Ferne.
- Die Analyse und Aufbereitung der Daten, zum Beispiel von Rechtsfällen, kann durch menschliche Benutzer geschehen. Damit stehen die für die Entscheidung notwendigen Daten der Maschine bereits in strukturierter Form zur Verfügung. Die Maschine muss „bloß“ die Schlussfolgerung durchführen.
- Eine weitere Möglichkeit ist, dass weiterhin der menschliche Benutzer eines Systems die Entscheidungsfindung durchführt. Die Software soll dabei den Beurteilungsprozess lediglich unterstützten, beispielsweise indem komplexe Argumente grafisch aufbereitet werden.
- Schließlich ist es möglich, den Einsatz von Software bloß auf die Sammlung der Ergebnisse menschlicher Entscheidungen zu beschränken. Dies ist der Ansatz, den diverse Rechtsinformationssysteme, aber auch Google im Hinblick auf Webseiten, verfolgen.

Die Implementierung von Systemen der dritten und vierten Art war bisher am erfolgreichsten und gehört zum Stand der Technik. Im Bereich des Rechts jedoch konnten sich Expertensysteme der ersten und zweiten Art bislang nicht durchsetzen, auch Entscheidungsunterstützungssysteme (der dritten Art) sind kaum verbreitet. Ein Grund dafür könnte deren geringer praktischer Nutzen sein (Conetta & Schafer, 2013, S. 63). Juristische Informationssysteme, wie Rechtsdatenbanken, sind hingegen etabliert und ein unverzichtbares Recherchewerkzeug für Rechtsanwälte und Behörden geworden.

5.5.9. Abschließende Bemerkungen

Dieses Unterkapitel gab einen recht guten Überblick über verschiedene Probleme, die im Zusammenhang mit KI und Recht existieren. Es wurde gezeigt, dass diese Probleme bisher verhindert haben, dass Computer juristische Entscheidungen fällen können.

Von derartigen Hindernissen sollte man sich jedoch nicht zu schnell entmutigen lassen. Viele der Probleme, die Franklin aufzeigt, können umgangen werden, indem die Analyse und Dateneingabe durch einen Menschen erfolgt. Dieser Ansatz hat sich bei der Automatisierung von Verwaltungsverfahren (in Österreich) schon früh bewährt.

Die automatisierte Extraktion und Erstellung von Computermodellen – wie beispielsweise Ontologien – steht hingegen erst am Anfang und kann bestenfalls unterstützend herangezogen werden.

Der Fokus der gegenständlichen Untersuchung ist insbesondere auf Systeme der dritten Kategorie gerichtet, die im letzten Abschnitt dargestellt wurden. Es geht also um Software, deren Wissensbasis von menschlichen Experten erstellt wird und die anhand dieses Wissens (einfache) rechtliche Entscheidungen treffen kann. Aufgrund dieser Zielbestimmung, insbesondere durch die massive Unterstützung durch den Menschen, werden nicht alle der diskutierten Probleme schlagend.

Für die Lösung komplexerer Rechtsprobleme ist vor allem die Repräsentation von Argumenten von besonderer Wichtigkeit. Die Formalisierung von Argumenten ist eines der wichtigsten Themen auf dem Gebiet der KI und Recht. Es wurde viel Aufwand in die Erforschung und Formalisierung juristischer Argumentation investiert, was bereits zu guten Ergebnissen geführt hat. So ist es möglich, diverse Arten von Argumenten – basierend auf Regeln, Fällen und Analogien etc. – sowie deren Beziehungen untereinander abzubilden (Gordon, 2010a; Walton & Gordon, 2012). Offen bleibt hingegen deren automatisierte Gewichtung, diese muss von einem Experten durchgeführt werden.

6. Semantic Web

6.1. Einleitung

Bevor nun auf die Relevanz von Ontologien für die Formalisierung des KOVG näher eingegangen wird, soll an dieser Stelle zum besseren Verständnis ein Exkurs in das „Semantic Web“ erfolgen. Freilich kann hier keine weitergehende Einführung in das „Semantic Web“ und seine Konzepte gegeben werden, sondern es soll lediglich ein kurzer Überblick erfolgen.

Das Semantic Web, auch genannt Web 3.0, ist eine Weiterentwicklung des Web 2.0 und ein Standard des World Wide Web Consortium (W3C). Die Idee des Semantic Web wurde 2001 von Berners-Lee et al. (2001) vorgestellt.

Der Standard soll gemeinsame Datenformate für das World Wide Web etablieren. Durch die Einbindung von semantischen Inhalten in Webseiten zielt das Semantic Web darauf ab, das derzeitige Web, in dem unstrukturierte und semi-strukturierte Dokumente vorherrschen, in ein „Web der Daten“ zu verwandeln. Maschinen sollen die Daten in gewisser Weise verstehen und automatisiert verarbeiten können.

Der ursprüngliche Artikel von Berners-Lee et al. (2001) beschreibt diese Entwicklung, die jedoch erst vollzogen werden muss. Im Jahr 2006 stellten Berners-Lee und seine Kollegen fest, dass die recht simple Idee des Semantic Web über weite Strecken noch nicht realisiert wurde (Shadbolt, Berners-Lee, & Hall, 2006).

An dieser Feststellung von Berners-Lee et al. hat sich bis heute – 2014 – nicht grundlegend etwas verändert. Zwar stellen insbesondere Behörden im Rahmen von Open Data ihre Daten im Web zur Vefügung, der Einsatz von semantischen Technologien ist jedoch noch nicht sehr verbreitet. Es bleibt abzuwarten, ob die vorhandenen Datensammlungen im Zuge von Linked (Open) Data (Bizer et al., 2009) alsbald mit semantischen Auszeichnungen versehen werden.

6.2. Bausteine des Semantic Web

Wesentlich für den Begriff eines Netzwerks ist der Gedanke, dass jeder sein Wissen beitragen kann und dieses dann (auch) für andere nutzbar wird. Das Semantic Web richtet sich dabei nicht nur an technisch versierte Benutzer, sondern an die Allgemeinheit schlechthin.

Die Grundlage des Semantic Web bildet das „Resource Description Framework“ (RDF). Die Modellierung von Wissen in RDF[123] erfolgt in der Form von Tripeln. RDF ermöglicht es, in Form der Tripel Aussagen über beliebige Dinge der Welt zu treffen. Auf diese Weise lassen sich die Beziehungen der Dinge zueinander beschreiben sowie Aussagen über Aussagen treffen. Jedes Tripel setzt sich, wie in Abbildung 3 zu sehen ist, entsprechend den grundlegenden Regeln der Grammatik aus Subjekt, Prädikat und Objekt zusammen (Allemang & Hendler, 2008, S. 35).

Abbildung 3: RDF-Tripel

RDF ermöglicht eine einheitliche Modellierung von Wissen unabhängig von der jeweiligen Fachdomäne. Die so entstehenden „Wissenssammlungen“ des „Semantic Web“ bezeichnet man als Ontologien. Eine Ontologie besteht aus einer Vielzahl von RDF-Tripeln. Um Wissen aus einer Ontologie wieder extrahieren zu können, wurde eine standardisierte Abfragesprache namens SPARQL[124] entwickelt.

Der Schwerpunkt der Modellierung mit RDF wird nicht auf die reine Repräsentation der Daten gelegt, sondern auf die formale Beschreibung derselben (Allemang & Hendler, 2008, S. 6). Ebenso werden Beziehungen zwischen den Daten explizit definiert. Dies erlaubt es Anwendungen, nahezu beliebige Repräsentationen der Daten, z.B. eine einfache Webseite, zu generieren.

Wenn von Semantik im Zusammenhang mit der menschlichen Sprache gesprochen wird[125], dann wird darunter oft verstanden, was es bedeutet, eine bestimmte Aussage zu verstehen. Um die Wörter und Sätze verstehen zu können, ist üblicherweise ein intellektueller Vorgang notwendig, der aus diesen eine bestimmte Bedeutung ableitet. Ein relativ einfacher Teil dieses Vorgangs des Verstehens ist die Verknüpfung eines Begriffes mit einem

[123] Obwohl es richtigerweise „das RDF“ lauten müsste, wird in diesem Beitrag der Kürze wegen öfters der Ausdruck „in RDF“, als Synonym für „in der Sprache RDF“ verwendet. Freilich handelt es sich beim RDF um einen Standard, der mehr als nur die Syntaxdefinition einer Sprache umfasst.

[124] *W3C*, SPARQL Query Language for RDF. http://www.w3.org/TR/rdf-sparql-query/ aufgerufen: 9.6.2011.

[125] Genannt „linguistische Semantik“.

entsprechenden Objekt in der Wirklichkeit. Diese einfache Auffassung des Begriffs „Semantik“ ist jene, die dem „Semantic Web“ zugrunde liegt und es „antreibt“ (Allemang & Hendler, 2008, S. 1–13, 31 ff.).

Mit Hilfe der „Tripel“ lässt sich also vorhandenes Wissen modellieren, indem bestimmte Aussagen über Dinge (und andere Aussagen) getroffen werden. Das „Semantic Web“ geht jedoch noch einen Schritt weiter und ermöglicht es, aus den vorhandenen Daten neues Wissen abzuleiten (Allemang & Hendler, 2008, S. 79 ff.). Dieser Vorgang wird „Inferenz“[126] genannt. Mit Hilfe von formalen Regeln der Logik kann eine Inferenzmaschine neue Informationen aus dem Modell ableiten (Hitzler, Krötzsch, Rudolph, & Sure, 2008, S. 105 ff.). Einfache Ontologien, die solche Regeln enthalten, lassen sich bereits mit RDF erstellen. Ausgereiftere Regeln werden von RDF Schema[127] und vor allem OWL 2 (fortan nur OWL) definiert. Mit Hilfe von OWL und deren formaler Logik lassen sich komplexere Ontologien erstellen.

Die genannten Standards definieren formale Regeln, die mit bestimmten syntaktischen Sprachkonstrukten verknüpft sind[128]. Durch die vom Standard vorgegebenen Regeln wird es überhaupt erst möglich, dass Inferenzmaschinen verschiedener Hersteller einheitliche und vorhersehbare Ergebnisse liefern (Hitzler et al., 2008, S. 91).

Welche Rolle spielt nun das „Semantic Web“ bei der Entwicklung von „intelligenten“ Anwendungen?

Eine Software kann ihren Zweck nur so gut erfüllen, so gut die Qualität der ihr zur Verfügung stehenden Daten ist. Ein Problem traditioneller Datensammlungen ist die fehlende Verbindung zwischen den Daten. Widersprüchliche Daten können verhindern, dass eine Software sinnvolle und richtige Ergebnisse liefert. Die Daten, die in Ontologien modelliert sind, sind stärker miteinander verknüpft und daher konsistenter. Das „Semantic Web“ ermöglicht es Maschinen, die Daten auf eine gewisse Weise (im Sinne der vorherigen Ausführungen) zu verstehen.

---

[126] Auch Schlussfolgerung, engl. „Inference“.

[127] *W3C*, RDF Schema. http://www.w3.org/TR/rdf-schema/ aufgerufen: 12.11.2011.

[128] Wie z.B. `rdf:type`, `rdfs:range` und `owl:inverseOf`, u.v.m.

Aufgabe der Dateninfrastruktur des „Semantic Web“ ist es nicht komplexe Vorgänge auszuführen, die üblicherweise von einer Software durchgeführt werden, sondern verknüpfte, konsistente Daten bereit zu stellen, die es einer Anwendung ermöglichen, ihr volles Potential auszuschöpfen.

6.3. Schichten des Semantic Web

Die vom Semantic Web (Berners-Lee et al., 2001) beabsichtigte Evolution soll das Information Retrieval hin zum Data Retrieval entwickeln. Das Information Retrieval liefert die Dokumente, die den Abfragekriterien entsprechen, im Gegensatz dazu ist das Data Retrieval in der Lage, auf eine Frage die korrekte Antwort zu finden.

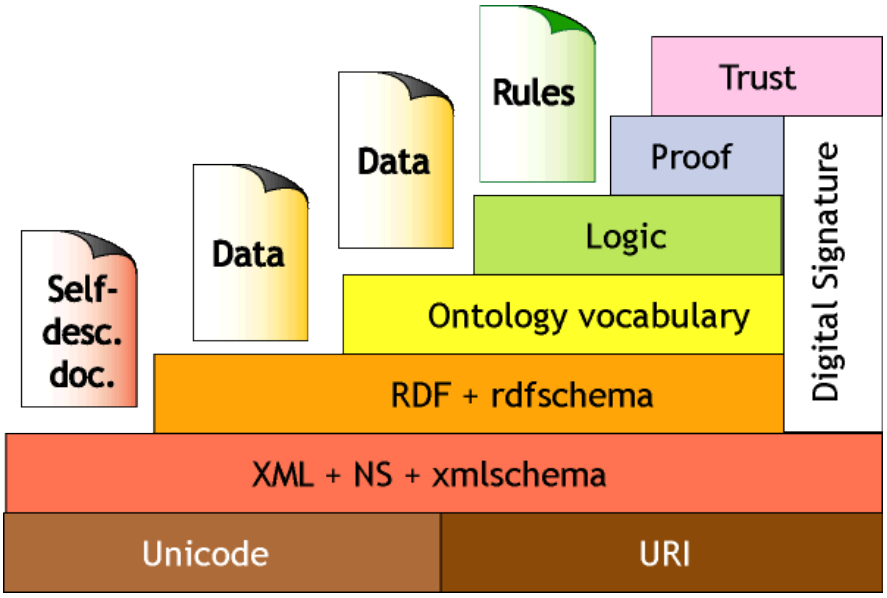

Abbildung 4: Semantic Web Stack (Berners-Lee, 2000)

Die Verwirklichung des Semantic Web wird es uns erlauben, die Limitierungen des derzeitigen Web zu überwinden, um den Benutzern den Zugang zu Informationen zu erleichtern und verschiedenste Verwendungen der Daten, von E-Commerce bis E-Government, zu ermöglichen.

Die Technologien, die benötigt werden, um diese Aufgabe zu bewerkstelligen, wurden im Semantic Web Stack nach steigender Komplexität angeordnet (siehe Abbildung 4). In diesem Stack sind höhere Schichten von niedrigeren abhängig, wobei die untersten Schichten (URI, Unicode, XML etc.) vom „gewöhnlichen“ World Wide Web übernommen wurden. Der Semantic Web Stack wurde im Lauf der Zeit aktualisiert, um voranschreitenden

technologischen Entwicklungen und Anforderungen Rechnung zu tragen (siehe Abbildung 5 oder (Horrocks, Parsia, Patel-Schneider, & Hendler, 2005)).

Das Fundament des Semantic Web Stacks wird von zwei Standards gebildet:

- Unicode: Der Unicode-Standard ermöglicht die Darstellung und den Austausch beliebiger Zeichen.
- URI: URIs (Uniform Resource Identifiers) ermöglichen die eindeutige Benennung von Ressourcen, die im Web verfügbar sind.

Auf diesen Technologien bauen die komplexeren höheren Schichten des Semantic Web Stacks auf:

- XML: Der XML-Standard stellt eine universelle Sprache (XML) zur Verfügung, die es erlaubt, die Struktur von Dokumenten mithilfe von Tags zu beschreiben und weitere Metadaten in die Dokumente zu inkludieren. Mithife der Struktur und der Metadaten können solche Dokumente automatisiert verarbeitet werden.
- RDF: RDF (Resource Description Framework) erlaubt die semantische Anreicherung von Dokumenten und ist der grundlegende Baustein des Semantic Web.
- Ontologien: Mithilfe von Ontologien können Begriffe und Beziehungen zwischen diesen modelliert werden. Die formale Semantik ermöglicht die deduktive Ableitung von neuen Schlüssen aus den vorhandenen Daten.
- Logik: Durch den Einsatz von Logik können komplexe Informationen beschrieben werden, die von Maschinen zur Bildung von Argumenten (Beweisen) verwendet werden können.
- Trust: Der Trust-Layer beinhaltet Technologien, um die Geheimhaltung, Authentizität, Integrität und Zuverlässigkeit der Daten sicherzustellen.

Komplementiert werden die Layer von Kryptographie und digitalen Signaturen.

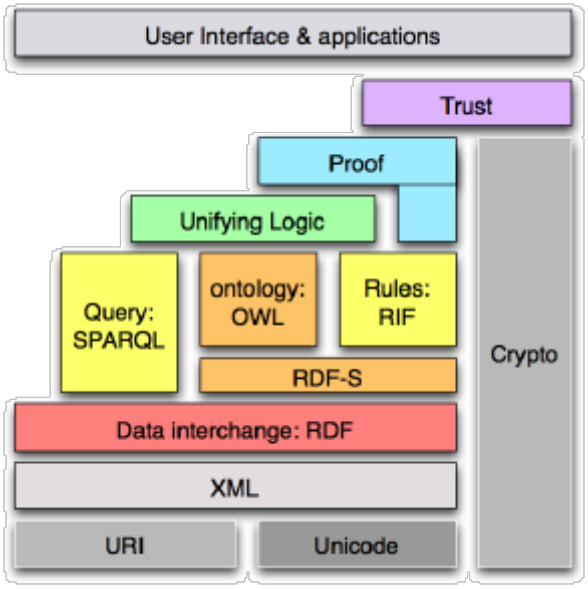

Abbildung 5: Aktualisierter Semantic Web Stack (Bratt, 2008)

Um den juristischen Anforderungen gerecht zu werden, wurde juristisches Wissen in das Web integriert und die Idee des komplexeren *Legal Semantic Web* geboren (siehe Abbildung 6). Das Legal Semantic Web wurde erstsmals von Benjamins, Casanovas, Breuker und Gangemi (2005) erwähnt, und von Biasiotti, Francesconi und Sartor (2008) näher spezifiziert.

In der Vision des Legal Semantic Web stehen elektronische juristische Dokumente im Web zur Verfügung und sind mit maschinenlesbaren Inhalten angereichert, die eine automatisierte Verarbeitung ermöglichen.

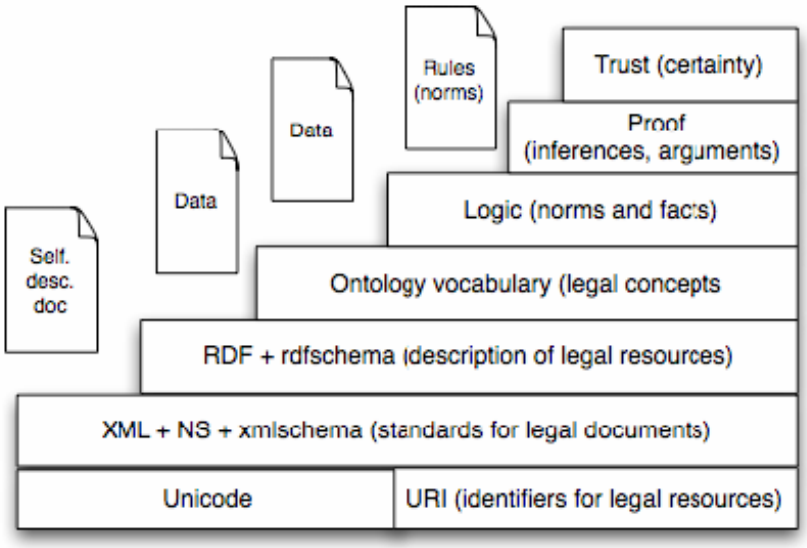

Abbildung 6: Legal Semantic Web Stack (Biasiotti et al., 2008)

6.4. Was ist eine Ontologie?

Der Begriff „Ontologie“ stammt ursprünglich aus der Philosophie und hat in diesem Bereich folgende Bedeutung: *„Ontology as a branch of philosophy is the science of what is, of the kinds and structures of objects, properties, events, processes and relations in every area of reality.“* (Smith, 2004)

In der Disziplin „Wissensrepräsentation“, die zum Bereich der Künstlichen Intelligenz gehört, bezeichnet „Ontologie“ wiederverwendbare Vokabulare der identifizierten Begriffe und Beziehungen von bestimmten Phänomenen der Welt, die in einem maschinenlesbaren Format explizit dargestellt werden.

Ontologien können als erweiterte taxonomische Strukturen begriffen werden, in denen Begriffe als Klassen formalisiert, durch Axiome definiert, mit der Beschreibung von Attributen und Restriktionen angereichert und mit anderen Klassen durch Properties verknüpft werden.

Gruber (1993, S. 2) führt zu Ontologien aus: *„In the context of AI systems using human knowledge we identify the ontology with the set of formal terms with which one represents knowledge since the representation completely determines what exists for the system.”*

Der aus der Philosophie geliehene Begriff Ontologie wird demnach in der Informatik und der Künstlichen Intelligenz in einem mehr operativen Sinn gebraucht. Die bekannteste Definition der *computergestützten Ontologie* in der Künstlichen Intelligenz ist jene von Gruber, der *Ontologie* als eine „explicit specification of a conceptualization“ (Gruber, 1993, 1995) definiert.

Basierend auf Grubers Definition wurde von Struder et al. später eine vollständigere Definition eingeführt, die heutzutage am häufigsten zitiert wird: *„an ontology is a formal, explicit specification of a shared conceptualization.“* (Studer, Benjamins, & Fensel, 1998, S. 184)

Das in der Informatik vorherrschende Verständnis der *computergestützten Ontologie* ist, dass diese als spezielles Objekt oder computergestütztes Artefakt begriffen wird. Eine *computergestützte Ontologie* ist eine Spezifikation eines gemeinsamen repräsentativen Vokabulars zur Beschreibung des Untersuchungsgegenstandes, das Begriffe und Beziehungen zwischen diesen enthält (Gruber, 1993).

In Kontext der gegenständlichen Forschungen ist ausschließlich die computergestützte Ontologie im Sinne der oben stehenden Ausführungen von Bedeutung.

Es sei jedoch erwähnt, dass zahlreiche weitere Definitionen einer Ontologie von den Forschungen auf dem Gebiet der Künstlichen Intelligenz hervorgebracht wurden. Guarino, Oberle und Staab (2009) beispielsweise geben, im Gegensatz zu vielen anderen Begriffsdefinitionen, eine formale Definition der Begriffe *Ontologie* und *Konzeptualisierung*.

Für eine vollständige Darstellung *computergestützter Ontologien*, der unterschiedlichen Definitionen des Begriffs *Ontologie* und deren Konsequenzen sowie der existierenden Ontologien, sei auf Casellas (2011) verwiesen.

Aus der oben genannten Definition alleine kann ein recht weites Spektrum an Interpretationen abgeleitet werden, sodass nicht nur Taxonomien, sondern auch Glossare und Thesauri als Ontologie klassifiziert werden könnten.

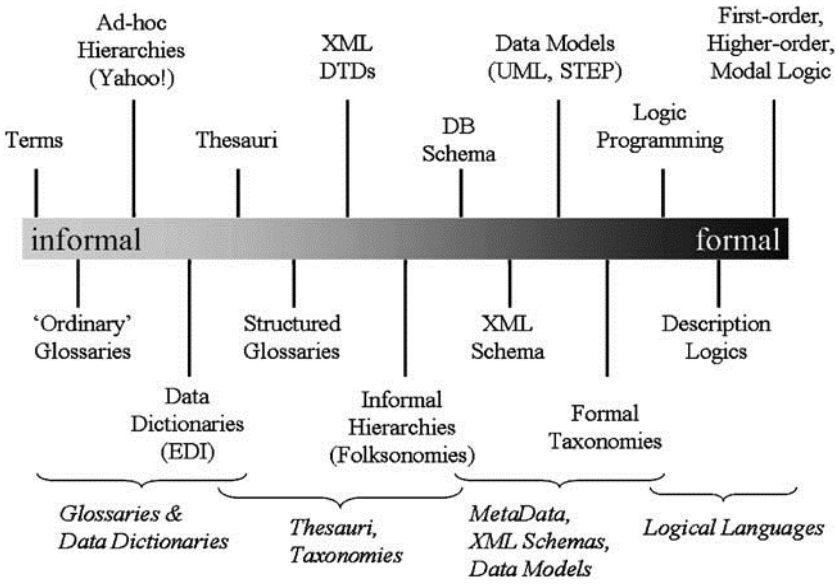

Abbildung 7: Ausdrucksstärke verschiedener Sprachen (Guarino et al., 2009, S. 13)

Abbildung 7 zeigt eine Skala der Ausdrucksstärke verschiedener Sprachen, beginnend mit informalen Begriffen am linken Ende, bis zu logischen Sprachen am anderen Ende der Skala, die einen hohen Grad an Formalität aufweisen. Gut erkennbar ist, dass Thesauri und Taxonomien einen eher niedrigen Formalisierungsgrad aufweisen und in der Mitte der Skala angesiedelt sind. Sprachen, die einen hohen Formalisierungsgrad aufweisen, werden verwendet um Modelle zu erstellen, die automatisiert verarbeitet werden können.

Für die Erstellung von computergestützten Ontologien im Kontext der Dissertation wird davon ausgegangen, dass diese mit Beschreibungslogik (OWL 2) modelliert werden. Computergestützte Ontologien im Sinne der Dissertation weisen damit einen hohen Formalisierungsgrad auf, der diese einer automatisierten Verarbeitung zugänglich macht.

6.5. Die Relevanz von Ontologien für die Formalisierung

Bevor in einem späteren Kapitel das zu entwickelnde Modell näher dargestellt wird, soll an dieser Stelle der erwartete Nutzen des Einsatzes von semantischen Technologien (RDF und OWL) für die Formalisierung der Normen des KOVG dargestellt werden. Aus dieser Darstellung ergibt sich letztendlich die Relevanz von semantischen Technologien und mit diesen erstellten Ontologien für die gegenständliche Arbeit.

Allgemein dienen Modelle (wie Ontologien) der Organisation von Wissen in Form von Erklärungen. Dies ermöglicht es Menschen, das modellierte Wissen besser zu verstehen. Eine im Modell enthaltene Erklärung (oder Regel) führt ein bestimmtes Phänomen auf die ihm zugrundeliegenden Prinzipien zurück (Allemang & Hendler, 2008, S. 20). Eine Funktion des Modells ist also die Erklärung von Phänomenen[129]. Durch die im Modell enthaltenen Regeln lassen sich aber auch Vorhersagen treffen[130].

Damit ein Modell überhaupt diese Funktionen erfüllen kann, müssen dessen Grundprinzipien sowie die Regeln, die diese in Beziehung setzen, in einer objektiven Form definiert sein, damit diese nicht einer Interpretation durch den Verwender des Modells bedürfen. Eine solche objektive Form und die Regeln, die diese beherrschen, werden „Formalismus“ genannt (Allemang & Hendler, 2008, S. 29).

Die Normen des KOVG sollen in einem solchen formalen Modell (der Ontologie) modelliert werden. Die Ontologie soll sämtliche Daten beinhalten, die für die automatisierte Erstellung eines entsprechenden Bescheides notwendig sind. Dazu gehören in erster Linie Daten für die Berechnung der Beschädigtenrente. Im Wesentlichen sind dies diverse Grundbeträge sowie bestimmte Erhöhungen in Abhängigkeit vom Grad der Erwerbsminderung. Neben dieser monetären Leistung muss die Ontologie auch Informationen über die zu gewährenden Sachleistungen beinhalten. Außerdem muss die Ontologie mehrere Versionen einer

[129] Engl. „Explanation“.

[130] Engl. „Prediction“.

bestimmten Norm abbilden können. Aufgrund von Novellen und Betragsanpassungen durch Verordnungen müssen unterschiedliche Versionen der Normen von der Ontologie der Software zur Verfügung gestellt werden können. Die Ontologie muss also auch bestimmte Metadaten enthalten, aufgrund derer die Software die anwendbaren Normen zum jeweiligen Zeitpunkt aus der Ontologie ableiten kann.

Die Architektur der zu erstellenden E-Government-Software wird noch in einem späteren Kapitel ausführlich dargestellt werden. Hier sei nur erwähnt, dass sich die Software aus mehreren Schichten zusammensetzen wird, wobei die unterste Schicht für die Ableitung der Daten aus der Ontologie zuständig ist. Diese Schicht wird sich im Wesentlichen einer bestehenden Programmbibliothek zur Verarbeitung von OWL sowie einer zu dieser Bibliothek kompatiblen Inferenzmaschine bedienen, um die benötigten Daten aus der Ontologie ableiten zu können.

Mit Hilfe der Inferenzmaschine sollen aus der Ontologie (auch) implizit vorhandene Daten abgeleitet werden, um z.B. bestimmte Klassifizierungen durchzuführen. So könnte beispielsweise die Inferenzmaschine, die von der E-Government-Software verwendet werden wird, anhand des Grades der Minderung der Erwerbsfähigkeit einer Person die Kategorisierung der Person in die Gruppe der „Schwerbeschädigten“[131], „erwerbsunfähigen Schwerbeschädigten“[132] usw. durchführen.

Ausgehend von der Klassifizierung von Personen nach der Minderung der Erwerbsfähigkeit, die als eine Art Einstiegspunkt in die Feststellung der Leistungen angesehen werden kann, müssen die Formel für die Berechnung sowie die zugehörigen Beträge und Prozentsätze aus der Ontologie abgeleitet werden können. Wie bereits im vorherigen Kapitel ausgeführt, ist es mit Hilfe von RDF (und OWL) möglich, diverse Beziehungen zwischen Objekten zu modellieren. So sollte es auch möglich sein, die einzelnen Elemente der Formel für die Berechnung, ausgehend von einem Objekt, das die oben dargestellte Klassifizierung repräsentiert, in Form von miteinander in Beziehung stehenden Tripeln zu modellieren. Der so entstehende Graph[133] repräsentiert schließlich die für die Feststellung der gebührenden

[131] Minderung der Erwerbsfähigkeit größer gleich 50 %.

[132] Minderung der Erwerbsfähigkeit größer gleich 90 %.

[133] Ein RDF Graph besteht aus mehreren Tripeln. Vgl. *W3C*, Graph Definitions. http://www.w3.org/TR/rdf-mt/#graphdefs aufgerufen: 14.11.2011.

Leistungen notwendigen Informationen. Es muss ein Ansatz gefunden werden, wie die einzelnen Leistungen des KOVG, insbes. deren Anspruchsvoraussetzungen, am besten mit Hilfe der Ontologie modelliert werden können.

Die dynamische Natur des „Semantic Web“ sollte es ermöglichen, jederzeit Änderungen in Form von Tripeln in das Modell einpflegen zu können. Damit wird es möglich, für den jeweiligen Zeitpunkt das korrekte Berechnungsschema in derselben Ontologie zu modellieren. Jährliche Anpassungen von Beträgen und Prozentsätzen – sei es durch Gesetz oder Verordnung – könnten in die Ontologie eingefügt werden und stünden damit sofort der Software zur Verfügung. Die Ontologie bildet damit eine zuverlässige und einheitliche Quelle, die sämtliche zur Berechnung erforderlichen Informationen enthält.

Ein weiteres Problem, das es zu lösen gilt, ist das Mapping der Tripel aus der Ontologie in Regeln die, das Programm verarbeiten kann. Die Rechenoperationen, die sich aus der abgeleiteten Formel ergeben, werden nämlich von der Software selbst durchgeführt und nicht von der Ontologie bzw. der Programmbibliothek, die auf die Ontologie zugreift. Wünschenswert wäre es, wenn die Daten aus der Ontologie möglichst direkt in Regeln übersetzt werden könnten, die von der Software verarbeitet werden können. Diese Regeln dienen als Ersatz für die in bisherigen Ansätzen verwendeten fest codierten Bedingungen in der Software. Letzten Endes werden diese Bedingungen aber in einer Form benötigt die erlaubt, dass das Programm aufgrund der eingegebenen Daten die korrekten Leistungen ermitteln kann.

Prinzipiell könnte das korrekte Berechnungsschema direkt, je nach Minderung der Erwerbsfähigkeit, aus der Ontologie abgeleitet werden, sodass das Programm ohne weitere Logik der Formel folgen und die einzelnen Rechenoperationen nacheinander ausführen kann.

Falls sich eine derartige Modellierung als zu komplex erweist oder nicht möglich sein sollte, sollen aus der Ontologie zumindest die einzelnen Bedingungen abgeleitet werden, die dann auf eine dynamische Art und Weise übersetzt werden müssen, sodass sie vom Programm verarbeitet werden können. Zu diesem Zweck könnte z.B. eine eigene „Domain Specific Language“ (im Folgenden DSL) entwickelt werden, die schließlich von der Software direkt verarbeitet werden kann. Mit Hilfe einer Softwarekomponente, die Teil der gesamten Anwendung ist, müsste dann eine Übersetzung der Tripel in die DSL erfolgen.

Neben einer möglichst direkten Verwertbarkeit der ontologisch modellierten Normen durch die Software wird erwartet, dass sich die Normen des KOVG mit Hilfe der Ontologie transparenter und für Menschen besser lesbar formalisieren lassen als dies mit bisherigen Ansätzen möglich war. Zu diesem Zweck soll die Ontologie möglichst einfach gehalten werden, damit diese auch für menschliche Benutzer gut lesbar und damit besser verständlich ist.

Zusammengefasst stellen sich die wichtigsten Anforderungen an die zu modellierende Ontologie wie folgt dar. Die Ontologie muss (1) sämtliche, für die Berechnung der Beschädigtenrente sowie (2) für die Ermittlung sonstiger Leistungsansprüche relevanten Daten enthalten. (3) Die Ontologie muss dabei unterschiedliche Versionen der Normen (die einer bestimmten Gesetzeslage entsprechen) unterstützen, um auf Gesetzesnovellen oder Betragsanpassungen durch Verordnung reagieren zu können. (4) Letztendlich sollen die modellierten Normen möglichst direkt von der Software verwendet werden können.

6.6. Sonstige Einsatzmöglichkeiten von Ontologien

6.6.1. Integration heterogener Daten

Eine weitere Aufgabenstellung, zu deren Lösung sich Ontologien eignen könnten, ist die Integration heterogener Daten.

Sämtlichen bisherigen Datensammlungen[134] ist gemein, dass sie eine spezielle starre Struktur aufweisen, die nur von Programmen verarbeitet werden kann, die exakt auf diese Struktur zugeschnitten sind. Die Programme können aber die Bedeutung der Daten (die Semantik) nicht verstehen. So ist es schwierig, automatisiert mehrere Datensammlungen zu einem neuen Bestand zusammen zu fügen oder neue Schlüsse aus einer Kombination von Daten abzuleiten. Die Aufgabe, heterogene Daten zu integrieren, bleibt damit weitgehend dem Menschen überlassen, der entsprechende Lade- und Transformationsroutinen entwickeln muss, um einen neuen Datenbestand zu erzeugen.

Es besteht die Gefahr, dass sich in einzelnen Behörden, Unternehmen etc. Datensilos entwickeln, die gerade im öffentlichen Verwaltungsbereich – datenschutzrechtliche Probleme nun einmal ausgeklammert – eine sinnvolle Kombination von Daten ergeben

[134] In den meisten Fällen werden relationale Datenbanken eingesetzt.

würden (Hadrbolec, 2011). So könnten Behörden die Daten einer anderen Behörde bei der Durchführung eines Verfahrens nutzen. Dabei müssen selbstverständlich die datenschutzrechtlichen Bestimmungen eingehalten werden, worauf an dieser Stelle jedoch nicht weiter näher eingegangen werden soll.

Ontologien können im Kontext der Wissensrepräsentation im Wesentlichen als standardisierte Vokabulare verstanden werden, mit deren Hilfe das in einem Bereich vorhandene Wissen beschrieben werden kann. Eine Ontologie in diesem Sinne ist eine strukturierte Sammlung von Begriffen und Beziehungen zwischen diesen, ähnlich einem Thesaurus, jedoch mit erweiterter Funktionalität und formaler Semantik.

Ontologien stellen die Grundlage für die Integration von Daten aus heterogenen Quellen zur Verfügung und wirken so den typischen Siloeffekten entgegen, die üblicherweise durch die getrennte Datenhaltung entstehen. Die in der Ontologie modellierten Begriffe und deren Beziehungen stellen dabei gewissermaßen einen Standard für die Beschreibung von Daten zur Verfügung. Daten aus unterschiedlichen Quellen können so mithilfe der modellierten Begriffe beschrieben und schließlich integriert werden. Die Ontologie repräsentiert gewissermaßen das gemeinsame Vokabular, auf das sich die Anbieter der Daten verständigen müssen.

Die Kombination vorhandener Daten ist die Grundlage, um diese wiederverwenden zu können. Für den Bürger kann dies eine Beschleunigung von Verfahren bedeuten, da die erneute Erfassung der Daten entfällt. So könnte beispielsweise der Benutzer bei der elektronischen Antragsstellung durch vorausgefüllte Formulare unterstützt werden.

Vorstellbar ist, dass die jeweiligen Daten und Schnittstellen eines E-Government-Systems einer Behörde mit Hilfe einer Ontologie beschrieben werden, und so die in der Ontologie modellierten Konzepte wiederum auf andere Ontologien, z.B. einer anderen Behörde, verweisen könnten. Diese Verweise könnten mit entsprechenden Metadaten versehen werden, die beschreiben, wie die Daten zu transformieren sind, damit diese mit der anderen Datensammlung kombiniert werden können.

Die Verwendung von semantischen Technologien könnte demnach maßgeblich zur Steigerung der Interoperabilität von Services und E-Government-Systemen beitragen.

6.6.2. Automatische Verwendung von (neuen) Services

Eng mit der Integration von heterogenen Daten ist das automatische Auffinden und Verwenden von Services verwandt. Eine E-Government-Software könnte prinzipiell so entworfen werden, dass an bestimmten Stellen im Programm die Möglichkeit für eine zukünftige Erweiterung vorgesehen wird. Diese „extension points“ könnten mit Hilfe einer Ontologie semantisch beschrieben werden. Sobald ein Service, dessen Schnittstelle und Funktionalität ebenfalls semantisch beschrieben werden muss, zur Verfügung steht, könnte dieses von der Software automatisch verwendet werden, falls es zu einem verfügbaren „extension point“ passt. Freilich ist für diese Funktionalität auch ein Mechanismus erforderlich, der ein neues Service der Software zur Kenntnis bringt. Denkbar ist hier ein ereignisgesteuerter Ansatz, der neu hinzukommende Services dem Softwaresystem meldet. Weitere Probleme, die in diesem Zusammenhang auftreten können, sind die Feststellung der Authentizität des Services sowie Haftungsfragen für die Richtigkeit der Daten.

Der Ansatz hätte den Vorteil, dass ein bestehendes E-Government-System, dessen „extension points“ von vornherein wohl überlegt werden müssen, im Laufe der Zeit „mächtiger“ wird, sobald neue Services mit entsprechender Funktionalität zur Verfügung stehen. Das bestehende Programm müsste nicht mehr verändert werden, sondern könnte neu hinzukommende Services transparent einbinden.

Im gegenständlichen Kontext muss z.B. der Antragsteller bei der Beantragung einer Rente nach dem KOVG seine Bankdaten dem System bekannt geben. Es könnte ein „extension point“ in der Anwendung vorgesehen werden, der die Bankdaten eines Benutzers abfragen kann. Solange kein Service existiert, das diese Funktionalität anbietet, muss der Antragsteller seine Bankdaten manuell in ein Webformular eintragen. Sobald ein entsprechendes Service aktiv wird, könnte die Anwendung über dieses Ereignis informiert werden und das Service, mithilfe des zuvor erwähnten semantischen Markups, mit dem definierten „extension point“ zur Abfrage der Bankdaten verglichen und gegebenenfalls verwendet werden.

Das Service könnte schließlich mit den im vorherigen Kapitel beschriebenen ontologisch gestützten Transformationsroutinen eingebunden werden. Die Bankdaten des Antragstellers könnten dann vom Service abgefragt und im Webformular bei der Antragstellung bereits vorgeschlagen werden.

7. Die Anforderungen an Sprachen zur Formalisierung von Verwaltungsrecht

7.1. Einleitung

Dieses Kapitel dient der Darstellung der allgemeinen Anforderungen an formale Sprachen zur Modellierung von Normen (des Verwaltungsrechts). Damit sind jedoch nicht konkrete Anforderungen eines bestimmten Gesetzes gemeint, sondern vor allem solche, die sich abstrakt aus den Eigenschaften der Rechtsnormen und den Eigenheiten juristischer Entscheidungsfindung ergeben. Der Fokus der Untersuchung ist dabei auf das österreichische Verwaltungsrecht gerichtet.

Diese Analyse ist ein wesentlicher Teil der Dissertation und dient als Grundlage für die Untersuchung der vorhandenen Sprachen, wie beispielsweise LKIF und LegalRuleML, hinsichtlich deren Tauglichkeit zur Formalisierung von Verwaltungsrecht.

Die Formalisierung von Gesetzen ist notwendig, um diese computerunterstützt verarbeiten zu können und so eine automatisierte Entscheidungsfindung zu ermöglichen. Die Schaffung eines einheitlichen Standards für die Formalisierung von Verwaltungsgesetzen scheint auf lange Sicht unabdingbar, um eine Effizienzsteigerung und höhere Transparenz im E-Government zu erreichen. Die Entwicklung einer soliden konzeptuellen Grundlage für die Modellierung von Normen und die automatisierte Entscheidungsfindung ist ein wesentlicher Faktor für den Erfolg derartiger Unternehmungen.

Obwohl es schon seit längerem Softwarelösungen im Bereich der Verwaltungsautomatisierung gibt, die die Durchführung von Verfahren unterstützen, fehlt bislang eine eingehende theoretische Auseinandersetzung mit diesem Thema. Den Forschungen auf dem Gebiet der Künstlichen Intelligenz und Recht wurde bisher kaum Beachtung geschenkt. Dies zeigt sich unter anderem im Fehlen einheitlicher Standards und der Diversität der vorhandenen Implementierungen.

Die gängigen Lösungen sind zudem mit einer Reihe von Problemen behaftet. Zu diesen gehören insbesondere:

- Das Fehlen einer geeigneten „Domain Specific Language“ zur Modellierung von Verwaltungsrecht. Die bisherigen Ansätze sind zu sehr technisch orientiert, sodass Fachexperten nicht das eigentliche Wissen – die Rechtsnormen – modellieren

können. Die derzeitigen gesetzlichen Modelle im Programmcode weisen ein sehr niedriges Abstraktionsniveau auf.

- Die Erstellung von E-Government-Systemen ist fehleranfällig, es mangelt an etablierten Standards für die Implementierung.
- Juristische Informatiksysteme müssen, insbesondere aufgrund von Gesetzesnovellen, mit häufigen Änderungen des modellierten Wissens umgehen können. Die derzeitigen Lösungen sind dafür meines Erachtens nur unzureichend geeignet. Gesetzliche Änderungen führen zu einem hohen Anpassungsaufwand der Softwaresysteme.
- Die Implementierung und Anpassung solcher Systeme ist aus diesen Gründen nach wie vor sehr teuer.

Dieses Kapitel ist der erste Schritt, um eine solide theoretische Grundlage für die Implementierung sogenannter juristischer wissensbasierter Systeme – insbesondere für das Verwaltungsrecht – zu entwickeln. Charakteristisch für wissensbasierte Systeme (WBS) ist die Trennung zwischen der Wissensbasis und der Inferenzmaschine, die aus dem vorhandenen Wissen Schlussfolgerungen ableitet. Diese Trennung unterscheidet WBS von bisherigen Lösungen, in denen das rechtliche Wissen zur Problemlösung und der Lösungsalgorithmus direkt im Programmcode eingebettet sind.

Bei der Erhebung der Anforderungen für WBS muss insbesondere zwischen jenen unterschieden werden, die an die formale Sprache zur Modellierung des Wissens gestellt werden, und solchen, die an die Softwarearchitektur gerichtet sind. Überdies wird durch die Auswahl der Sprache zur Wissensmodellierung nicht auch bereits die Struktur der Wissensrepräsentation – des formalen Modells – vollständig vorgegeben. Demnach kann man zusätzlich Anforderungen definieren, die das Modell des Wissens selbst betreffen.

Die folgenden Ausführungen in diesem Kapitel beschränken sich auf die Anforderungen, die an formale Sprachen zur Wissensrepräsentation gestellt werden.

7.2. Die Besonderheiten des Rechts

Das Recht ist ein komplexes Phänomen und tritt in unterschiedlichen Erscheinungsformen – wie Gesetzen, Verordnungen und Entscheidungen – zu Tage. Die einzelnen Rechtsnormen

können insbesondere nach ihrem Adressaten, der normsetzenden Autorität und dem Erzeugungsverfahren unterschieden werden.

Trotz dieser Diversität des Rechts – man denke nur an die unglaubliche Vielfalt an Verwaltungsvorschriften – hat die Rechtstheorie einige gemeinsame Eigenschaften von Rechtsnormen identifiziert. Aus diesen Eigenschaften ergeben sich im Wesentlichen die Anforderungen an Sprachen zur Formalisierung von Rechtsnormen.

Bevor die Arten der Rechtsnormen und die soeben bereits erwähnten Eigenschaften diskutiert werden, sei an dieser Stelle zunächst auf die Besonderheiten des Rechts eingegangen. Bei der Implementierung von wissensbasierten Systemen und der damit verbundenen Formalisierung von Rechtsnormen müssen diese Besonderheiten ausreichend berücksichtigt werden, um eine adäquate Repräsentation zu gewährleisten.

Die folgende Darstellung orientiert sich an Boer (2009, S. 16 ff.), der sich in seiner Dissertation unter anderem mit den Besonderheiten des Rechts beschäftigt hat, die es bei der Entwicklung von juristischen wissensbasierten Systemen zu berücksichtigen gilt. Die Ausführungen von Boer sind aus der Sicht der Rechtstheorie und Informatik zu verstehen.

(1) Die in einem förmlichen Gesetzgebungsverfahren zustande gekommenen formellen Gesetze[135] spielen als geschriebene Rechtsquellen eine zentrale Rolle für die Extraktion von Wissen. Das Recht regelt explizit, wie sich die Rechtsunterworfenen zu verhalten haben, legt den Ablauf von Verfahren fest und darüber hinaus auch, wie Rechtsvorschriften zu interpretieren sind[136]. Es scheint auf den ersten Blick so, als wäre das Recht vorzüglich für die Formalisierung geeignet. Die Forschungen und praktischen Erfahrungen haben jedoch gezeigt, dass die Implementierung von juristischen Expertensystemen oder die Bildung allgemeiner Theorien für das Design solcher Systeme durch die Tatsache erschwert werden, dass das Recht spezielle Regeln vorgibt, „wie“ die rechtliche Beurteilung zu erfolgen hat.

Angemerkt sei, dass implizites („stilles“) Wissen von Experten eine eher untergeordnete Rolle, im Vergleich zu anderen Disziplinen, zu spielen scheint. Rechtliches Wissen tritt vielmehr explizit, insbesondere in Form von Gesetzen und Entscheidungen, in Erscheinung.

---

[135] Damit ist das vom Gesetzgeber erlassene positive Recht gemeint, das auch gelegentlich gesatztes Recht genannt wird.

[136] Es sei auf den bekannten § 6 ABGB verwiesen.

Das rechtliche Wissen steht damit relativ kompakt und vollständig zur Verfügung, um in juristischen wissensbasierten Systemen implementiert zu werden.

(2) Charakteristisch für das Recht ist, dass die Rechtsquellen selbst häufigen Änderungen unterworfen sind. Derartige Anpassungen werden durch gesetzliche Novellen, aber auch durch gerichtliche Entscheidungen bedingt, die oftmals das Abgehen von der bisher vertretenen Rechtsansicht und damit die Änderung des im WBS repräsentierten Wissens erforderlich machen. Neues Wissen entsteht damit ebenso explizit in Form von Änderungen der Rechtsquellen. Eine weitere Besonderheit in diesem Zusammenhang ist, dass das Recht selbst festlegt, wann neues Wissen anzuwenden ist. Man denke insbesondere an das in Novellen explizit angeführte Datum des Inkraftretens der neuen Bestimmungen.

Hoekstra (2009, S. 120) führt dazu aus: *„In fact, the meaning of terms in law imposes an ordering on entities in reality that can change over time, but stays applicable to older cases. In short, law adopts an intricate versioning scheme.“*

Diese Dynamik des Rechts unterscheidet sich maßgeblich von anderen Gebieten, in denen das vorhandene Wissen oftmals über einen längeren Zeitraum relativ konstant bleibt und nur das neueste Wissen relevant ist.

(3) Bei sämtlichen Arten der Schlussfolgerung – nicht nur im Recht – unterscheidet man zwischen *vorschreibendem* und *beschreibendem* Wissen. Im Recht spielt jedoch die Spezifikation und Kategorisierung von vorschreibendem Wissen die gewichtigere Rolle.

Entscheidungen benötigen allgemein eine große Menge von unveränderlichem, beschreibendem Wissen. Die Entscheidungsfindung wird dabei von Zielvorgaben, Beschränkungen, Anforderungen und Regeln bestimmt, die die angestrebte Lösung in Grundzügen vorgeben. Die Rechtswissenschaft beschäftigt sich oftmals mit der Organisation von *vorschreibendem* Wissen – *beschreibendes* Wissen wird aus anderen Bereichen, falls notwendig, herangezogen.

Das Recht regelt Sachverhalte des alltäglichen Lebens und muss sich aus diesem Grund dem Vokabular des jeweiligen Fachbereichs oder allgemeiner Begriffe bedienen. Für die Entscheidungsfindung muss in der Regel oftmals auf weitreichendes Allgemeinwissen über die Welt und die darin existierenden Entitäten zurückgegriffen werden (Franklin, 2012).

(4) Die Begründung von Entscheidungen ist im Recht eine unabdingbare Notwendigkeit. Die korrekte Durchführung des Verfahrens (der Methode) zur Entscheidungsfindung ist wichtiger als die Qualität der Entscheidung selbst. Für die Bewertung der Qualität der Entscheidung existiert nämlich kein unabhängiger Standard, der zur Beurteilung herangezogen werden könnte.

In anderen Disziplinen hingegen werden oftmals etablierte wissenschaftliche Methoden angewendet, um ein entsprechendes Ergebnis zu erlangen. Die Forschungen sind primär an korrekten Ergebnissen interessiert. Zur Kontrolle werden externe Standards herangezogen, die unabhängig von der verwendeten Methode sind. Rechtliche Entscheidungen hingegen sind jedoch mangelhaft und können angefochten werden, sofern das vorgeschriebene Verfahren (die vorgeschriebene Methode) nicht formell- und materiellrechtlich korrekt durchgeführt wurde.

Von der (rechtlichen) Begründung einer Entscheidung ist deren bloße Erklärung zu unterscheiden. Für die bloße Erklärung einer Entscheidung ist nicht erforderlich, eine wertende Position einzunehmen und zu erklären, ob diese „gut“ oder „schlecht“ ist. Bei der Begründung einer Entscheidung geht es hingegen um die Erklärung, warum diese im juristischen Sinne „richtig“ oder „gut“ ist.

(5) Des Weiteren ist im Recht die Unterscheidung zwischen *Fakten* und *Behauptungen* notwendig. Diese Unterscheidung hat im Bereich der KI und Recht zu diversen Ansätzen und Theorien geführt, die sich mit Konfliktsituationen auseinandersetzen, in denen die Parteien gegenteilige Ansichten über einen Fakt oder die rechtliche Beurteilung vertreten.

Ein juristisches wissensbasiertes System muss also grundsätzlich in der Lage sein, mit widersprüchlichen Informationen wie abweichende Rechtsansichten über dieselben Fakten umgehen zu können. Im Gegensatz dazu müssen wissensbasierte Systeme in anderen Bereichen, wie beispielsweise der Medizin, in der Regel nicht zwischen vom Benutzer des Systems (dem Arzt) vorgebrachten Behauptungen und Fakten unterscheiden.

(6) Rechtsnormen sind voneinander abhängig und miteinander verknüpft. In Gesetzen findet man eine Vielzahl von Verknüpfungen zwischen den Normen, die auch auf andere Gesetzestexte verweisen. Normative Konflikte werden dabei durch Metaregeln wie *lex specialis*, *lex superior* und *lex posterior* gelöst (van de Ven, Hoekstra, Breuker, Wortel, & El Ali, 2008, S. 1).

Solche Verknüpfungen bestehen nicht nur zwischen Normen derselben Stufe, sondern (zumindest implizit) zwischen einfachen Gesetzen und dem Verfassungsrecht, darüber hinaus auch zwischen nationalen Gesetzen und Bestimmungen des Europarechts.

7.3. Der Begriff „Regel“

Im vorherigen Abschnitt wurden die besonderen Eigenschaften des Rechts im Überblick dargestellt. Dieses noch recht vage Grundverständnis soll nun vertieft und in den folgenden Abschnitten die Anforderungen an Sprachen zur Formalisierung von Normen erhoben werden.

Seit Langem gibt es auf dem Gebiet der KI und Recht Anstrengungen, die Charakteristika von Normen und Gesetzen zu modellieren. Dies hat zur Entwicklung von XML-Standards zur Repräsentation von Rechtsdokumenten wie beispielsweise CEN MetaLex (Boer, Hoekstra, & Winkels, 2002) oder LKIF (Gordon, 2008b, 2010b) geführt. Viele dieser Ansätze sind gut entwickelt und praktisch relevant, beschränken sich jedoch oftmals auf die Repräsentation von *Rechtsdokumenten* und nicht der *Rechtsnormen* selbst (Gordon, Governatori, & Rotolo, 2009, S. 283)[137].

Viele Nachteile und Probleme der verfügbaren Sprachen resultieren möglicherweise aus der Tatsache, dass es vor den Bemühungen von Gordon et al. (2009) keinen ganzheitlichen und systematischen Versuch gab, die allgemeinen Anforderungen an Sprachen zur Modellierung von Normen zu erheben. Überdies scheint es bislang noch keine (vollständige) Übereinstimmung zwischen den Forschern hinsichtlich der Minimalanforderungen an solche Sprachen zu geben.

Der eben erwähnte Beitrag von Gordon et al. ist der erste systematische Überblick über die bisherigen Forschungen auf diesem Gebiet und stellt die Minimalanforderungen an Sprachen zur Modellierung von Rechtsnormen dar. Vorausgeschickt sei, dass die Autoren von „rule interchange languages“ – also Sprachen zum Austausch von Regeln – sprechen.

Der folgenden Darstellung liegt das Normverständnis von Kelsen (1960) zugrunde. Eine Norm ist demnach der Sinn eines menschlichen Willensaktes, der intentional auf das

[137] In diesem Zusammenhang sei auf die Arbeiten von Routen und Bench-Capon (1991) sowie Bench-Capon (1989, 1994b) verwiesen, die die ideale Repräsentation von Normen aus Sicht der logischen Programmierung diskutieren.

Verhalten anderer gerichtet ist, und so anderen Personen ein Verhalten gebietet, verbietet oder erlaubt.

In der Rechtstheorie werden Rechtsvorschriften meistens in die Kategorien konstituierende Normen und vorschreibende Normen eingeordnet (Athan et al., 2013b, S. 7). Es besteht in der Rechtstheorie und auf dem Gebiet der KI und Recht weitgehend Einigkeit, dass Normen grundsätzlich folgende konditionale Struktur haben (Gordon et al., 2009, S. 284; Kelsen, 1979):

$$\text{wenn } A_1, \ldots, A_n \text{ dann } B$$

„$A_1, \ldots, A_n$“ ist der Tatbestand der Norm und B die Rechtsfolge, die anzuwenden ist, wenn die Bedingungen erfüllt sind. Diese Grundstruktur einer Norm deutet auf eine direkte Verbindung zwischen den Begriffen Norm und Regel hin. Die Bedeutungen des Begriffs „Regel“ (engl. „rule“) variiert jedoch abhängig vom jeweiligen Bereich.

Die allgemeine Bedeutung des Begriffs „Regel“ ist (Jewell & Abate, 2001): *„One of a set of explicit or understood regulations or principles governing conduct within a particular sphere of activity.“*

In der Logik wird unter Regel eine Inferenzregel oder eine materiale Implikation verstanden. In der Computerwissenschaft werden mit dem Begriff der Regel mehrere Bedeutungen verbunden, wie unter anderem Produktionsregel, Grammatikregel oder Umschreibregel (rewrite rule). Oftmals werden im Bereich der Programmierung unter einer Regel beliebige konditionale Strukturen verstanden (Sartor, 2009b, S. 218).

Indessen wird im Recht eine Regel üblicherweise im regulatorischen Sinn – also als Synonym zum Begriff der Norm – verstanden. Normen drücken jedoch nicht nur Verhaltensregeln aus, wie Gordon et al. (2009, S. 284) ausführen.

Normen werden üblicherweise durch Schrift in Form von textuellen Klauseln wie Sätzen, Artikeln und Paragrafen, in Gesetzen zum Ausdruck gebracht. Im *logischen* Sinn ist eine Regel in diesem Kontext demnach eine Formalisierung des Rechtstexts in der Form von Prämisse und Konklusion. In Zusammenhang mit einem Konditional spricht man von Antezedens und Konsequenz (Sukzedens) einer Regel. Jeder Ansatz der Formalisierung von Rechtsnormen muss – soll dieser von Erfolg gekrönt sein – die Struktur von Rechtsnormen ausreichend berücksichtigen.

Angemerkt sei, dass zwischen Prämisse und Konklusion der formalen Regel sowie Tatbestand und Rechtsfolge nicht zwingend eine direkte Beziehung bestehen muss. Unter Umständen werden durch eine Regel mehrere Normen formalisiert. Umgekehrt ist ebenso denkbar, dass es zweckmäßig sein kann, eine Norm durch mehrere Regeln zu formalisieren. Aus diesem Grund werden auch die Begriffspaare Prämisse/Konklusion und Tatbestand/Rechtsfolge spiegelbildlich gegenübergestellt.

Die anschließende Erhebung der Anforderungen an Sprachen zur Modellierung von Normen, abgeleitet aus deren Charakteristika, folgt dem Beitrag von Gordon et al. (2009).

7.4. Charakteristika von Rechtsnormen

Im folgenden Abschnitt sollen die Charakteristika von Rechtsnormen diskutiert werden. Aus diesen lassen sich Anforderungen an Sprachen zur Formalisierung ableiten. Die Charakteristika lassen sich in drei Gruppen unterscheiden: Semantische Eigenschaften, logische Eigenschaften und rechtsprozessuale Eigenschaften.

7.4.1. Semantische Eigenschaften

Isomorphismus: Zwischen den Rechtsnormen und deren formaler Repräsentation muss eine Beziehung hergestellt werden. Idealerweise sollte eine direkte Beziehung (1:1) zwischen den in natürlicher Sprache formulierten Normen des Rechtstexts und deren Repräsentation im Modell hergestellt werden.

Reifikation: Normen können als Objekte mit bestimmten Eigenschaften verstanden werden. Solche Eigenschaften sind:

- Die *Jurisdiktion*, innerhalb deren Hoheitsbereich die Norm verbindlich ist und Rechtswirkungen entfaltet.
- Die normsetzende *Autorität*, die die Norm erlassen hat. Damit wird auch die Rangordnung einer Norm innerhalb der Rechtsordnung bestimmt. Beispielsweise stehen Verfassungsgesetze, die im Verfassungsgesetzgebungsverfahren erlassen wurden, im Stufenbau der Rechtsordnung über einfachen Gesetzen.
- Die Repräsentation von *zeitlichen Eigenschaften* von Normen. Zu diesen gehören:
    - Der Zeitpunkt, an dem die Norm erlassen wurde und deren Inkrafttreten.
    - Der Zeitraum, in dem Sachverhalte vom Tatbestand der Norm erfasst werden (Bedingungsbereich), und der Zeitraum, in dem das vollziehende Organ die

Rechtsfolgen verhängen darf (Rechtsfolgenbereich). Diese beiden Zeiträume decken sich nicht zwingend und können auseinanderfallen.

Formale Semantik: Sprachen zur Modellierung von Normen sollten eine präzise formale Semantik besitzen. Eine solche erlaubt die korrekte automatisierte Ableitung der Rechtsfolgen einer Norm.

Normative Effekte: Wenn eine Norm auf einen Sachverhalt anwendbar ist, folgen aus diesem Umstand diverse „normative Effekte“ (Rechtsfolgen). Zu diesen gehören Verpflichtungen, Erlaubnisse und Verbote. Eine Sprache zur Formalisierung sollte zwischen diesen normativen Effekten differenzieren und deontische Operatoren für die Modellierung von Verpflichtungen, Erlaubnissen und Verboten zur Verfügung stellen.

Darüber hinaus gibt es aber auch „ausdrucksstärkere“ Rechtsfolgen, wie beispielsweise im Fall von Beweislastregeln, die an einen Fakt eine bestimmte Vermutung knüpfen, oder Definitionen, die die Bedeutung eines Begriffs spezifizieren.

Werte: Üblicherweise werden durch Normen auch moralische und soziale Werte transportiert. Derartige Wertungen können insbesondere bei der Auflösung von Normkonflikten herangezogen werden. Aus diesem Grund ist es hilfreich, wenn eine Sprache auch die Modellierung von Werten und deren Rangordnung ermöglicht.

7.4.2. Logische Eigenschaften

Anfechtbarkeit: Selbst wenn der Tatbestand einer Rechtsnorm, formalisiert durch den Bedingungsteil einer Regel, erfüllt ist, bedeutet das nicht, dass auch die Rechtsfolge zwingend auf einen Sachverhalt anwendbar ist. Bei der in der Literatur so genannten „Anfechtbarkeit“ von Rechtsnormen geht es im Wesentlichen um folgende Probleme:

- **Konflikte:** Rechtsnormen können miteinander in *Konflikt* stehen, wenn sie auf denselben Sachverhalt anwendbar, aber widersprüchlich sind. Derartige Konflikte können entstehen, wenn eine Norm spezieller ist, eine höhere Rangordnung hat oder später beschlossen wurde. Solche scheinbaren Widersprüche werden durch Anwendung der *Lex-specialis-*, *Lex-Superior-* und *Lex-Posterior*-Regeln gelöst.
- **Ausnahmen:** Andererseits können Normen auch so formuliert sein, dass diese *explizite Ausnahmen* für andere Rechtsvorschriften enthalten. Die betroffene Norm

wird nicht mehr angewendet, wenn die Ausnahmebestimmung ebenso anwendbar ist.

Kontraposition: In der Logik versteht man unter Kontraposition ein Gesetz, das besagt, dass eine Implikation logisch äquivalent zu ihrem Umkehrschluss ist. Die Aussage „Wenn A, dann B“ ist logisch äquivalent zu ihrer Kontraposition „Wenn nicht B, dann nicht A“. Wie ersichtlich, wird die Kontraposition einer Aussage gebildet, indem Prämisse und Konklusion invertiert und vertauscht werden.

Bei Rechtsnormen ist die Kontraposition jedoch grundsätzlich nicht zulässig. Wenn die Konklusion einer Regel falsch ist, lässt dies keine Schlüsse über die Wahrheit ihrer Prämissen zu (Prakken & Sartor, 1996).

Caminada (2008) hat in seinem Paper jedoch gezeigt, dass die Kontraposition nicht zwingend unzulässig sein muss. Lehnt man das Prinzip der Kontraposition kategorisch ab, muss man konsequenter Weise auch Gegenbeispiele für die Gültigkeit anderer Prinzipien der anfechtbaren Schlussfolgerung (defeasible reasoning) in Betracht ziehen. Caminada stellt die These auf, dass zwei grundsätzlich verschiedene Formen der Schlussfolgerung existieren und bezeichnet diese als „epistemologische Schlussfolgerung“ und „konstitutive Schlussfolgerung“.

Im Fall der konstitutiven Schlussfolgerung ist das Prinzip der Kontraposition zulässig. Dies gilt jedoch grundsätzlich nicht für Fälle der epistemologischen Schlussfolgerung. Rechtliche Regeln beschreiben die Welt nicht bloß, sondern konstruieren diese bis zu einem gewissen Grad. Die rechtliche ist daher eine Form der konstitutiven Schlussfolgerung.

Faktisch existiert für den Prozess der konstitutiven Schlussfolgerung kein objektives externes Kriterium, das eine bestimmte Form des Prozesses bedingen würde. Im Wesentlichen ist es die Aufgabe des Problemlösers, anhand seiner Intuition und Meinung eine geeignete Form des Problemlösungsprozesses zu wählen und Eigenschaften der Schlussfolgerung festzulegen. Das Kartenexperiment von Wason (1968) hat jedoch gezeigt, dass viele Menschen mit dem Prinzip der Kontraposition Schwierigkeiten haben. Es ist daher wenig verwunderlich, dass Menschen bei der Schlussfolgerung nicht zwingend von der Zulässigkeit der Kontraposition ausgehen, wenn kein externes Kriterium vorhanden ist, das die Gültigkeit der Kontraposition fordert.

Für die Formalisierung von Rechtsnormen bleibt noch anzumerken, dass grundsätzlich von einer Unzulässigkeit der Kontraposition auszugehen ist. Wenn ein Formalismus die Kontraposition zulässt, muss geprüft werden, ob dieser für die Formalisierung rechtlicher Regeln im jeweiligen Anwendungsfall geeignet ist.

Unterstützende Gründe oder Faktoren: Im Recht ist es nicht immer möglich, präzise Regeln zu formulieren, wann ein Tatbestandselement vom Sachverhalt erfüllt wird. Man denke in diesem Zusammenhang vor allem an vage Gesetzesbegriffe. Es wäre nützlich, wenn eine Sprache die Möglichkeit bietet, auch „unscharfe" Regeln, wie beispielsweise sogenannte „Faktoren", zu formulieren. Ein Faktor kann dabei als ein „Argument" für oder gegen eine bestimmte Rechtsansicht verstanden werden. Durch Abwägung der Faktoren lässt sich so insbesondere feststellen, ob ein vager Gesetzesbegriff des Tatbestands erfüllt ist oder nicht.

Sartor (2006b) präsentiert eine Form der juristischen Argumentation mithilfe sogenannter „Faktoren" als einen wichtigen Aspekt juristischer Argumentation. Die Berücksichtigung eines Faktors unterscheidet sich dabei maßgeblich von der Anwendung einer Regel. Aus der Anwendung einer Regel folgt eine bestimmte Schlussfolgerung. Relevante Faktoren sind hingegen bei der Entscheidungsfindung bloß zu berücksichtigen und lenken diese gewissermaßen in eine bestimmte Richtung. Dabei werden binäre und skalierbare Faktoren unterschieden, wobei letztere „Dimensionen" genannt werden. Ein binärer Faktor ist im Einzelfall entweder vorhanden oder nicht. Je ausgeprägter ein skalierbarer Faktor vorhanden ist, desto mehr wird eine Entscheidung in eine bestimmte Richtung favorisiert.

Es sei noch angemerkt, dass der Ansatz von Sartor und dessen skalierbare Faktoren dem Konzept der „Fuzzy-Logik" ähnlich sind, das von Zadeh (1965) in den frühen 1960er-Jahren vorgestellt wurde. Sartors skalierbare Faktoren sind jedoch bidirektional und unterstützten die Argumentation von zwei entgegengesetzten Entscheidungen. Überdies können Faktoren gewichtet werden, sodass der Rang von Argumenten adäquat repräsentiert werden kann.

Gültigkeit von Normen: Normen können insbesondere durch Novellen oder höchstgerichtliche Urteile ungültig werden. Nun können in der Regel Rechtsvorschriften nicht einfach aus der Wissensbasis eines Systems entfernt werden, da unter Umständen ältere Fälle noch nach einer früheren Rechtslage beurteilt werden müssen. Vom theoretischen Standpunkt aus betrachtet sind zwei Fälle zu unterscheiden:

- Die Nichtigerklärung von Normen wirkt *ex tunc*, als sei die Norm nie Bestandteil der Rechtsordnung gewesen.
- Die Aufhebung einer Norm wirkt hingegen *ex nunc*. Damit bleibt die aufgehobene Norm auf Fälle weiterhin anwendbar, die vor der Aufhebung der Norm verwirklicht waren.

Die Abbildung von komplexen Versionierungsmechanismen in computerunterstützen Modellen des Rechts ist demnach eine notwendige Anforderung, die jedoch mit einigen erheblichen Schwierigkeiten verbunden ist (Klarman, Hoekstra, & Bron, 2008).

7.4.3. Rechtsprozessuale Eigenschaften

Verfahrensregeln: Im Recht existieren nicht nur Regeln, die der Auflösung normativer Konflikte dienen, sondern auch solche, die überprüfen, ob eine Handlung oder ein Zustand mit materiellen Bestimmungen übereinstimmt.

Das formelle Recht umfasst Bestimmungen, die ein Verfahren zur Prüfung, ob eine bestimmte Handlung oder ein Zustand mit dem materiellen Recht vereinbar ist, vorgeben. Mithilfe solcher Regeln sollen Verletzungen von Normen festgestellt und auch die entsprechenden Rechtsfolgen zur Kompensation der Normverletzung verhängt werden.

Dabei kann es vorkommen, dass durch die Verletzung einer Norm mehrere kompensatorische Rechtsfolgen ausgelöst werden, die in Folge die Anwendung weiterer Regeln bedingen und so fort.

Abhängig von den Zielen des jeweiligen Projekts ist auch eine Formalisierung solcher Verfahrensregeln erforderlich. Aus diesem Grund ist es zweckmäßig, wenn eine formale Sprache die Modellierung von Prozessen samt den beteiligten Parteien etc. unterstützt.

Dauer von normativen Effekten: Hinsichtlich der Dauer von Rechtsfolgen kann folgende Unterscheidung getroffen werden:

- Wiederkehrende Rechtsfolgen werden solange angewendet, bis diese durch ein späteres Ereignis beendet werden. In diese Kategorie fallen beispielsweise wiederkehrende Schadenersatzzahlungen aufgrund entgangenen Verdienstes infolge einer Körperverletzung (§ 1325 ABGB).
- Andere Rechtsfolgen sind hingegen nur solange anwendbar, wie der Tatbestand einer Norm erfüllt ist. Beispielsweise gilt die Regel „In Räumen für Unterrichts- und

Fortbildungszwecke ist das Rauchen verboten“ nur solange man sich in derartigen Räumlichkeiten aufhält.

7.5. Zusätzliche Anforderungen

Im vorherigen Abschnitt wurden die aus den Charakteristika der Rechtsnormen abgeleiteten, Anforderungen an Sprachen zur Formalisierung von Normen präsentiert. Die Ausführungen stützten sich dabei auf den Beitrag von Gordon et al. (2009), dessen Schwerpunkt die Erhebung der Anforderungen an Sprachen zum Austausch von Rechtsnormen ist. Der Fokus der gegenständlichen Untersuchung ist jedoch etwas anders ausgerichtet: Es geht nicht bloß um eine möglichst adäquate Modellierung der Charakteristika von Normen mit dem Ziel des Austausches formaler Repräsentationen von Gesetzen, sondern insbesondere um die Erstellung von computergestützten Modellen, die eine automatisierte Entscheidungsfindung ermöglichen. Die von Gordon et al. erhobenen Anforderungen müssen daher dieser Ausrichtung entsprechend ergänzt werden.

Dieser Abschnitt präsentiert die Ergebnisse der Analyse des Kriegsopferversorgungsgesetzes (KOVG) zur Ableitung zusätzlicher Anforderungen. Für die Untersuchung wird das Kriegsopferversorgungsgesetz BGBl. Nr. 152/1957 in der Fassung BGBl. I Nr. 81/2013 herangezogen, wie es am 31. Dezember 2013 in Kraft stand. Angemerkt sei, dass die erhobenen Anforderungen nicht unbedingt Charakteristika von Rechtsnormen sind, sondern sich eher auf die erforderlichen Eigenschaften beziehen, die eine Sprache zur Modellierung von Normen aufweisen sollte.

Mathematische Berechnungen und Mengen: Für die Vollziehung von Normen ist es oftmals erforderlich, mathematische Berechnungen, insbesondere die Summierung von Geldbeträgen, durchzuführen. Die Notwendigkeit von Berechnungen kann dabei sowohl aus dem Tatbestand als auch der Rechtsfolge einer Norm folgen.

Beispielsweise ist für die Feststellung der Höhe der Beschädigtenrente nach dem Kriegsopferversorgungsgesetz der Grundbetrag aus einem Prozentsatz zu errechnen und sodann gegebenenfalls um eine altersabhängige Zulage zu erhöhen (§ 11 KOVG). Die berechneten Beträge sind sodann auf volle zehn Cent zu runden (§ 67 KOVG).

Zur Beschädigtenrente selbst sind unter Umständen noch weitere Zulagen, wie beispielsweise eine Familienzulage für jeden Angehörigen eines Schwerbeschädigten gemäß § 16 KOVG, hinzuzurechnen.

Es ließen sich nur unschwer weitere Beispiele für derartige Berechnungen finden. Man denke insbesondere an das Einkommenssteuergesetz, dessen zentrales Anliegen die Feststellung der Steuerschuld bzw. -gutschrift des Steuerpflichtigen ist.

Überdies ist die Spezifikation von Mengen Inhalt zahlreicher Rechtsvorschriften (Hoekstra et al., 2007). Man denke beispielsweise an das Kraftfahrgesetz BGBl. Nr. 267/1967 (KFG), dessen Bestimmungen oftmals auf Eigenschaften von Kraftfahrzeugen wie Hubraum, Geschwindigkeit und Leistung Bezug nehmen.

Es lässt sich feststellen, dass Mengen und mathematische Berechnungen in vielen, wenn nicht allen, Rechtstexten eine Rolle spielen (Hoekstra, Breuker, Di Bello, & Boer, 2009, S. 44 f.).

Eine Sprache zur Modellierung von Rechtsnormen muss demnach die Möglichkeit bieten, **mathematische Berechnungen**, **Mengen** und **Rundungsregeln** zu spezifizieren.

Obwohl diese Anforderungen keinesfalls die Ausnahme, sondern die Regel sind, bieten viele Sprachen zur Wissensmodellierung nicht die Möglichkeit, mathematische Formeln und Mengen abzubilden. Dies liegt wohl daran, dass viele dieser Sprachen, wie beispielsweise die Web Ontology Language (OWL), auf einer Form der Logik beruhen. Die Sprache OWL und auch deren aktuelle Version OWL 2 basiert auf Beschreibungslogik, sodass Berechnungen – zumindest ohne Erweiterungen – nicht durchgeführt werden können (Iannone & Rector, 2008). Hinsichtlich der Repräsentation von Mengen in OWL sei auf die Ausführungen von Parsia und Smith (2008) verwiesen, die die Einführung neuer Datentypen zur Lösung des Problems vorschlagen.

Diese Limitierungen werden an sämtliche Formalismen weitergegeben, die auf OWL basieren. Zu diesen Ansätzen gehört beispielsweise das Legal Knowledge Interchange Format (LKIF), das unter anderem aus einer in OWL modellierten Ontologie besteht.

Abschließend sollen noch weitere Anforderungen diskutiert werden, die vor allem aus der Praxis der Verwaltungsautomatisierung abgeleitet werden.

Computerunterstützung: Damit das formale Modell vom Computer verarbeitet werden kann, wird eine geeignete Inferenzmaschine (engl. reasoner) benötigt, die aus diesem Schlüsse ableiten kann. Diese Anforderung ergänzt die bereits weiter oben angeführte „formale Semantik“, die von einer Sprache zur Modellierung von Normen gefordert wird. Die Inferenzmaschine ist selbstverständlich kein Teil und kein Charakteristikum der Sprache selbst, sondern ein Programm.

Angemessenes Abstraktionsniveau: Die Sprache muss ein angemessenes Abstraktionsniveau aufweisen. Gemeint ist damit, dass die Struktur und Semantik von Rechtsnormen ausreichend repräsentiert werden muss. Ein Jurist oder Sachbearbeiter soll sich nicht mit technischen Details der Syntax einer Programmiersprache, wie dies bei geläufigen Ansätzen der Fall ist, bemühen müssen.

Werkzeugunterstützung: Für die Erstellung einer formalen Repräsentation des Gesetzes ist es sehr zielführend, wenn die Modellierung durch ein entsprechendes Werkzeug grafisch unterstützt wird. So wäre es möglich, dass eine Entwicklungsumgebung die Annotation von Gesetzestexten erlaubt, um diese für die automatisierte Extraktion des Modells vorzubereiten.

Angemerkt sei, dass diese Anforderung nicht unmittelbar an die jeweilige Sprache selbst gerichtet ist, sondern die Entwicklungsumgebung als eine Sammlung von Werkzeugen insgesamt betrifft.

7.6. Die Gewichtung der Anforderungen

Die erhobenen Anforderungen sollen nun einer Gewichtung unterzogen werden. Die Einteilung erfolgt dabei in die Kategorien „hoch“, „mittel“ und „niedrig“. Eine Anforderung wird mit „hoch“ kategorisiert, wenn sie für die Erreichung der Ziele eines „typischen“ Projekts der Verwaltungsautomatisierung zwingend notwendig ist und „niedrig“, wenn das nicht der Fall ist. Die Erfüllung von Anforderungen, die als „mittel“ eingestuft werden, ist wünschenswert, jedoch für die Erreichung der Ziele nicht zwingend. Die Erfüllung der als „hoch“ eingestuften Anforderungen ist daher stets *zwingend*, als „mittel“ und „niedrig“ eingestufte Anforderungen können *optional* umgesetzt werden.

Zusätzlich werden die Anforderungen in optionale und zwingende kategorisiert.

Als Maßstab für die Bewertung wird ein fiktives Projekt zugrunde gelegt, dessen Ziel die Formalisierung und semi-automatische Vollziehung des KOVG ist. Der menschliche Benutzer wird zur Unterstützung herangezogen, wenn vage Gesetzesbegriffe angewendet oder Ermessen ausgeübt werden soll.

Tabelle 1 zeigt die Gewichtung der erhobenen Anforderung unter Verwendung des soeben skizzierten Maßstabes.

Anforderung	**Gewichtung**
Anfechtbarkeit	hoch
Computerunterstützung	hoch
Formale Semantik	hoch
Gültigkeit von Normen	hoch
Isomorphismus	hoch
Mathematische Berechnungen	hoch
Reifikation	hoch
Angemessenes Abstraktionsniveau	mittel
Normative Effekte	mittel
Unterstützende Faktoren	mittel
Verfahrensregeln	mittel
Werkzeugunterstützung	mittel
Dauer von normativen Effekten	niedrig
Kontraposition	niedrig
Werte	niedrig

Tabelle 1: Gewichtung der Anforderungen

Wie ersichtlich, wurden viele Anforderungen als „hoch“ kategorisiert. Auf den ersten Blick mag verwundern, warum die Anforderung „Normative Effekte“ als mittel kategorisiert wurde. Dies erklärt sich daraus, dass die Modellierung von deontischen Normen, also von Verboten, Geboten und Erlaubnissen, für die Implementierung der Berechnung von Leistungen eine eher untergeordnete Rolle spielt. An der jeweiligen Stelle im Programm ist üblicherweise bekannt, welche Bedeutung das Ergebnis hat. Dennoch wird der Erfüllung dieser Anforderung üblicherweise bei der Implementierung juristischer Expertensysteme eine entscheidende Rolle zukommen.

7.7. Diskussion

Das Kapitel thematisierte die Charakteristika von Rechtsnormen und gab einen Überblick über die daraus resultierenden Anforderungen an Sprachen zur Modellierung von Normen. Diese Darstellung umfasste die grundlegenden, aus der Rechtstheorie abgeleiteten Anforderungen an Sprachen zum Austausch von Normen. Dieses Grundverständnis wurde

vor dem Hintergrund dieser Dissertation mit Blick auf die automatisierte Vollziehung von Rechtsnormen ergänzt. Anhand ausgewählter Normen des Verwaltungsrechts wurden weitere Erfordernisse ermittelt.

Gemäß ihrem Ursprung lassen sich die erhobenen Anforderungen in rechtliche, technische und praktische unterteilen. Der Großteil der Anforderungen ist rechtlicher Natur und resultiert aus der gesetzlichen Grundlage selbst. Die Grenzen sind jedoch nicht immer eindeutig und fließend.

Eine interessante Frage ist, ob eine einzige Sprache geeignet ist, sämtliche Anforderungen abzudecken. Für die Modellierung von Arbeitsabläufen existiert zum Beispiel bereits die sogenannte Business Process Model and Notation (BPMN), eine Sprache zur Modellierung von Prozessen (Workflows).

Meines Erachtens ist es zielführender spezielle Sprachen zu entwickeln, die ein angemessenes Abstraktionsniveau zur Modellierung bestimmter juristischer Probleme aufweisen. Wichtig ist jedoch, dass die einzelnen Formalismen miteinander kompatibel sind, damit die erstellten Modelle integriert werden können. Für die automatisierte Vollziehung von Rechtsnormen wäre beispielsweise eine geeignete Sprache zur Modellierung einfacher Syllogismen hilfreich. Wenn notwendig, könnten spezielle Sprachen zur Formalisierung juristischer Argumentation oder von vagen Begriffen mithilfe von „Faktoren und Dimensionen“ unterstützend hinzugezogen werden.

Es erscheint nicht sinnvoll, bereits ausgereifte Formalismen neu zu erfinden und eine Sprache zu entwickeln, die sämtliche Anforderungen erfüllt. Selbst wenn dies theoretisch möglich ist, wäre die Komplexität einer solchen Sprache wohl zu hoch für den praktischen Einsatz und die Weiterentwicklung des Sprachumfangs.

Eine weitere Frage, die es zu beantworten gilt, ist, ob sämtliche Anforderungen auf Ebene der formalen Sprache erfüllt werden sollten. Nicht alle der erhobenen Anforderungen adressieren die formale Sprache selbst. Möglicherweise ist es zielführender, die Bewertung von Argumenten, die aus Normen und anderen Quellen erzeugt werden, auf Ebene der Entscheidungsfindung zu behandeln.

Wie oben gezeigt wurde, müssen abhängig vom jeweiligen Projekt nicht sämtliche Anforderungen erfüllt werden. Es bedarf einer individuellen Gewichtung. Die Kriterien, die

für die Beurteilung der einzelnen Anforderungen herangezogen werden, sind von der konkreten Rechtsmaterie und den angestrebten Zielen des jeweiligen Projekts abhängig.

Mithilfe der gewichteten Anforderungen können neue Technologien und Sprachen, wie insbesondere die LegalRuleML oder das Carneades Argumentation System, hinsichtlich ihrer Eignung evaluiert werden.

Für die Beurteilung der vorhandenen Ansätze auf dem Gebiet der KI und Recht bleibt offen, welche Bewertungsmethoden herangezogen werden können. Als erstes Kriterium kann jedenfalls die Bewertung dienen, ob eine Sprache sämtliche zwingende Anforderungen erfüllt. Darüber hinaus müssen jedoch speziellere Kriterien gefunden werden.

Ein solches Kriterium ist beispielsweise die Effizienz neuer Technologien. Damit ist insbesondere die Reduktion des Implementierungs- und Wartungsaufwands gegenüber gängigen Lösungen gemeint. Auch hier gilt es, eine objektive Methode zur Messung der Komplexität zu finden. Dies ist durchaus anspruchsvoll, da der Programmcode (in Java etc.) und die überwiegend in XML[138] erstellten Modelle miteinander verglichen werden müssen.

Es wird sich in Zukunft zeigen, ob die Schaffung einer theoretischen Grundlage für die Implementierung von wissensbasierten Systemen im öffentlichen Bereich zu einer Verbesserung der formalen Modelle des Rechts führt. Spannend ist die Frage, ob durch neue Technologien tatsächlich eine Effizienzsteigerung erreicht werden kann.

Neue Technologien, die im Bereich der KI und Recht entwickelt wurden, scheinen sehr mächtig zu sein, weisen aber gleichsam eine relativ hohe Komplexität auf, die deren praktische Verwendung erschwert. Darüber hinaus sind Modelle zur automatisierten Entscheidungsfindung oftmals nicht über ihre theoretische Konzeption hinausgewachsen.

Möglicherweise sind dies Gründe dafür, dass bisher zwischen der an praktischen Ergebnissen interessierten Praxis der Verwaltungsautomatisierung und den Forschungen auf dem Gebiet der KI und Recht faktisch bisher kein Austausch stattgefunden hat. Obwohl die Anfänge der Verwaltungsautomatisierung in Österreich und die ersten Forschungen hinsichtlich der Formalisierung von Gesetzen bis in die 1970er Jahre zurückverfolgt werden können

[138] Die Ansätze im Bereich KI und Recht, wie LegalRuleML (Athan et al., 2013b), basieren häufig auf XML-Dialekten.

(Reisinger, 1975), scheinen sich diese beiden Bereiche vollkommen unabhängig voneinander entwickelt zu haben.

Dieses Kapitel repräsentiert den ersten Schritt einer systematischen wissenschaftlichen Aufarbeitung des Themas. Die weiteren Forschungen werden zeigen, ob die Verwaltungsautomatisierung von den Forschungen auf dem Gebiet der KI und Recht profitieren kann. Des Weiteren wird auch die entgegengesetzte Blickrichtung untersucht, nämlich inwiefern die auf dem Gebiet der Theorie entwickelten Ansätze geeignet sind, die automatisierte Vollziehung von Verwaltungsrecht zu unterstützen.

8. Systemdesign und Architektur

8.1. Einleitung

Dieses Kapitel widmet sich einer Darstellung der Funktionsweise des Expertensystems aus fachlicher Sicht. Der skizzierte Ablauf entspricht nicht exakt dem Verfahren, wie es in der Praxis derzeit durchgeführt wird, sondern wie dieses unter Einbeziehung neuer Technologien durchgeführt werden könnte.

Im Anschluss werden das Potential semi-automatisierter Systeme im Kontext der Verwaltung thematisiert und die wichtigsten Aspekte diskutiert.

8.2. Software zur semi-automatisierten Vollziehung des KOVG

An dieser Stelle sollen das Expertensystem zur semi-automatisierten Vollziehung des KOVG und dessen Funktionsweise im Überblick behandelt werden. Das System soll zur Demonstration die Beschädigtenrente berechnen. Von einer vollständigen Implementierung des KOVG wird Abstand genommen. Die folgende Darstellung skizziert in Grundzügen den Ablauf des semi-automatisierten[139] Verfahrens.

Das Expertensystem wird als Webanwendung konzipiert, die sich aus mehreren Modulen zusammensetzt. Der modulare Aufbau der Software gewährleistet eine bessere Wartbarkeit und erleichtert die Durchführung (automatisierter) Tests während der Entwicklung. Die Webanwendung kann mit einem herkömmlichen Browser genutzt werden. Die Authentifizierung des Antragstellers erfolgt ausschließlich mit der Bürgerkarte[140]. Danach findet die Erhebung der Daten für den Antrag statt.

Die Software leitet den Antragsteller bei der Einbringung des Antrags mit einem textbasierten Dialogsystem an, um die relevanten Fakten zu erheben. Dem Antragsteller werden dabei bereits im System vorhandene Daten vorgeschlagen, der diese annehmen oder bei Bedarf ändern kann. Die Erhebung der Daten geschieht durch einen Dialog zwischen Mensch und Maschine, wobei der Antragsteller stets die Möglichkeit hat, vorgeschlagene

---

[139] „Semi-automatisiert“ bedeutet, dass die rechtliche Beurteilung durch das Zusammenwirken zwischen Mensch und Maschine erfolgt.

[140] *Bundeskanzleramt Österreich*, Bürgerkarte. http://bka.gv.at/site/5268/default.aspx aufgerufen: 11.12.2011.

Daten zu ändern und besondere Vorkommnisse anzumerken. Ebenso besteht die Möglichkeit, Nachweisdokumente elektronisch beizustellen[141].

Um den Antragsteller bestmöglich zu unterstützen, werden nicht nur Daten vorgeschlagen, die im eigenen System bereits vorhanden sind, sondern auch Register (Services) anderer Behörden, z.B. das „Zentrale Melderegister", abgefragt. Die Authentifizierung mittels Bürgerkarte schafft dafür die notwendigen Voraussetzungen, da mit Hilfe des „bereichsspezifischen Personenkennzeichens" des Antragstellers auch Daten aus einem anderen Verwaltungsbereich abgefragt werden können. Zusätzlich könnten auch Services von privaten Unternehmen abgefragt werden, um z.B. die Höhe des Einkommens des Antragstellers zu ermitteln. Diese Zusammenführung von Daten aus vertrauenswürdigen Quellen wird unter dem Begriff „Federated Identity" verstanden (Rountree, 2013; Schweighofer & Hötzendorfer, 2012). Die Verwendung von bereits vorhandenen Daten führt nicht nur zu einer wesentlichen Effizienzsteigerung von E-Government-Systemen, sondern bietet dem Antragsteller die Möglichkeit, seinen Antrag vollständig und korrekt in kürzerer Zeit einzubringen.

Nach der Erfassung der Daten wird der Antrag einem fachkundigen Sachbearbeiter (in der Regel dem zuständigen Beamten) vorgelegt, damit dieser grundsätzliche Plausibilitätsprüfungen vornehmen kann. Die Authentifizierung des Sachbearbeiters erfolgt – ebenso wie jene des Antragstellers – mittels Bürgerkarte. Je nachdem, welche Daten der Software aufgrund der vorherigen Schritte zur Verfügung stehen, könnte auch eine automatisierte Überprüfung der eingegebenen Daten stattfinden, um anschließend Widersprüche oder Fehler dem Sachbearbeiter automatisch anzuzeigen. Nachdem der Sachbearbeiter die grundsätzliche Prüfung des Antrags durchgeführt hat, entscheidet er, ob der Antrag begründet ist oder abgewiesen wird. Falls der Antrag unbegründet ist, wird vom System ein entsprechender Bescheid, der das Begehren abweist, erstellt und dem Antragsteller (elektronisch) zugestellt.

Wenn der Antrag begründet ist, wird ein ärztlicher Sachverständiger mit der Erstattung eines Gutachtens über den Gesundheitszustand des Antragstellers beauftragt. Zu diesem Zweck

[141] Dafür bestehen gemäß § 17 E-GovG prinzipiell drei Möglichkeiten: Bürger übermittelt Dokumente elektronisch (Upload), Datenbeschaffung durch die Behörde auf Basis einer gesetzlichen Ermächtigung, Datenbeschaffung durch die Behörde nach Zustimmung des Bürgers.

wählt die Software automatisch einen Arzt aus, wobei Faktoren wie die Nähe der Praxis zum Wohnort des Antragstellers etc. automatisch berücksichtigt werden könnten. An den Antragsteller verschickt das System nach Auswahl des Arztes automatisch die Aufforderung, in einer bestimmten ärztlichen Praxis an einem festgelegten Termin zu erscheinen, damit die Untersuchung durchgeführt werden kann.

Der beauftragte Arzt meldet sich mit der Bürgerkarte im System an und erhält Einsicht in den elektronisch geführten Akt. Das Gutachten wird entweder direkt über die Weboberfläche eingegeben oder könnte auch mit Hilfe einer speziellen Software erstellt und anschließend an das System über eine definierte Schnittstelle übermittelt werden. Dafür ist es notwendig, dass die Datenstruktur und das Format des Gutachtens im Voraus definiert werden. Die Erfassung der einzelnen Gesundheitsschädigungen sowie die damit verbundene Minderung der Erwerbsfähigkeit werden strukturiert erfasst und eingebracht. Sofern eine Gesundheitsschädigung direkt mit einem bestimmten Grad der Minderung der Erwerbsfähigkeit verknüpft ist, kann die Software diese automatisch feststellen, womit eine Feststellung derselben durch den Arzt entfällt. Eine Verarbeitung des vom Arzt erfassten Gutachtens mit Technologien wie „Natural Language Processing“ ist daher nicht notwendig. Freilich kann der formulierte Text des Gutachtens sowie die dazugehörigen Bilder etc. in den elektronisch geführten Akt eingebracht werden.

Das Gutachten ist ein wesentlicher Bestandteil des Ermittlungsverfahrens, da die vom Arzt festgestellte Minderung der Erwerbsfähigkeit ausschlaggebend ist für die Versorgungsleistungen, die dem Antragsteller gewährt werden. Die Minderung der Erwerbsfähigkeit ist, wie bereits ausgeführt, nach Richtsätzen einzuschätzen, die durch Verordnung festgelegt werden. Die vom Dialogsystem im ersten Schritt erfassten Daten des Antragstellers sowie das ärztliche Gutachten stellen die faktische Grundlage für die rechtliche Beurteilung dar.

Die für die rechtliche Beurteilung erforderlichen Regeln sind in der Wissensbasis des Expertensystems hinterlegt. Zu diesen Regeln gehört unter anderem das Schema zur Klassifizierung des Antragstellers zwecks Einteilung in die eingangs genannten Personengruppen[142] sowie Regeln zur Bestimmung der zustehenden Versorgungsleistungen

[142] „Beschädigte“, „Schwerbeschädigte“, usw.

und deren Betrag. Das System muss in der Lage sein, die vorhandenen Fakten (Alter, Geschlecht, Minderung der Erwerbsfähigkeit usw.) unter die normativen Tatbestände zu subsumieren, um die rechtliche Beurteilung durchzuführen.

Nachdem die soeben skizzierte rechtliche Beurteilung von der Software durchgeführt wurde, werden die Ergebnisse dem Sachbearbeiter zur Kontrolle vorgelegt. Dieser kann gegebenenfalls Korrekturen vornehmen oder die vom System erstellte Entscheidung bestätigen.

Ausgehend von der rechtlichen Qualifikation des Einzelfalls muss anschließend ein rechtskonformer Bescheid mit dem erforderlichen Inhalt, insbesondere dem Spruch, der Begründung und der Rechtsmittelbelehrung, erstellt werden. Im Spruch eines stattgebenden Bescheides werden die Gesundheitsschädigung als Dienstbeschädigung anerkannt sowie die Minderung der Erwerbsfähigkeit festgestellt und die Versorgungsleistungen zugesprochen. Freilich muss das System auch in der Lage sein, falls es an bestimmten Voraussetzungen mangelt, einen Bescheid zu erstellen, der den Antrag insgesamt oder teilweise abweist. Abschließend wird der Bescheid elektronisch signiert und an ein Zustellservice weitergeleitet, das die elektronische Zustellung[143] an den Empfänger (den Antragsteller) vornimmt.

Mit der Zustellung ist das Verfahren grundsätzlich beendet, sofern binnen offener Frist kein Rechtsmittel eingebracht wird. Die Einbringung von Rechtsmitteln und die Durchführung des Rechtsmittelverfahrens sollen allerdings nicht weiter behandelt werden. Weitere Automatismen, wie die Ausstellung eines Schwerkriegsbeschädigtenausweises[144], sind denkbar, sollen aber hier ebenfalls nicht weiter erörtert werden.

Der rechtskräftige Bescheid, in dem Leistungen und deren Höhe zugesprochen werden, ist die Grundlage für die monatliche Auszahlung von Renten. Dabei kommt es unter Umständen zu einer Rück- und Nachverrechnung, sollte sich herausstellen, dass die in der Vergangenheit bezogenen Renten zu hoch bzw. zu niedrig waren.

---

[143] Vgl. *Bundeskanzleramt Österreich*, Elektronische Zustellung. http://www.bka.gv.at/site/5532/default.aspx aufgerufen: 23.12.2011.

[144] § 77 KOVG.

Zusammengefasst besteht die rechtliche Beurteilung zur Feststellung der zustehenden Leistungen aus folgenden Schritten: (1) Die Klassifizierung des Antragstellers anhand der festgestellten Minderung der Erwerbsfähigkeit. (2) Ermittlung der zustehenden Versorgungsleistungen. (3) Berechnung des Betrags der monatlichen Rente bei monetären Versorgungsleistungen.

8.3. Das Potential semi-automatisierter Vollziehung

Im Anschluss an die Darstellung der grundlegenden Funktionen der Software soll an dieser Stelle auf das Potential des semi-automatisierten Ansatzes eingegangen werden.

Die semi-automatisierte Unterstützung der Vollziehung des KOVG durch das Expertensystem birgt großes Potential, das Verfahren rascher, bei niedrigerem Ressourcenverbrauch, durchführen zu können. Wenn möglich, erfolgt die Durchführung einzelner Verfahrensschritte automatisiert durch die Software, nur falls notwendig, ist eine Aktion des Sachbearbeiters erforderlich. Daher wird auch von „semi-automatisierter Vollziehung“ gesprochen.

Wie in den vorangehenden Kapiteln gezeigt, sind Computer bei Weitem noch nicht in der Lage, die rechtliche Beurteilung wie ein Jurist (oder Richter) durchzuführen. Es besteht eine Reihe von ungelösten Problemen, die insbesondere aus der offenen Textur des Rechts nach Hart (2009) entspringen. Für die Lösung von schwierigen Rechtsfragen sind oftmals Wertungen und Abwägungen erforderlich, zu denen nur ein Mensch fähig ist. Der Dialog zwischen Mensch und Maschine ist eine vorzügliche Möglichkeit, um viele dieser Probleme zu umgehen. Immer dann, wenn ein Problem ansteht, dass die Maschine nicht lösen kann, wird der fachkundige Benutzer des Systems um Hilfe gebeten. Dies schließt nicht aus, dass das System, im Sinne einer Entscheidungsunterstützung für den Richter, bereits mehrere Entscheidungsmöglichkeiten vorschlägt.

Im Detail besteht großes Diskussionspotential, beispielsweise ist jeder Rechtsbegriff nur in seinem Begriffskern eindeutig und im Begriffshof vage. Sogenannte vage Gesetzesbegriffe kennzeichnen sich durch einen schmalen Begriffskern und einen weiten Begriffshof. Wo die Grenze zwischen vagen und eindeutigen Begriffen zu ziehen ist und das System damit eine Rechtsfrage dem Benutzer vorlegen soll, ist damit selbst diskussionsbedürftig. Es existiert keine Methode für die Kategorisierung von Rechtsbegriffen. Weiters ist eine formale

Repräsentation von vagen Begriffen erforderlich, um diese (automatisiert) von eindeutigen unterscheiden zu können. Nicht besonders schwierig ist die Klassifikation von Rechtsnormen, die im Tatbestand auf Zahlen und mathematische Vergleiche abstellen.

Ein weiterer Vorteil semi-automatisierter Systeme ist rechtlicher Natur. Der § 49 Datenschutzgesetz BGBl. I Nr. 165/1999 i.d.F. BGBl. I Nr. 133/2 (DSG) normiert nämlich ein grundsätzliches Verbot automatisierter Einzelentscheidungen. Die Entscheidung darf nicht ausschließlich aufgrund der automatisierten Verarbeitung ergehen. Da § 49 DSG nicht im Verfassungsrang steht, kann die Bestimmung freilich von einfachen Gesetzen verdrängt werden, die automatisierte Entscheidungen im Einzelfall zulassen. Die Entscheidung eines Expertensystems kann jedenfalls rechtlich im Sinne von § 49 DSG legitimiert werden, indem die Letztentscheidung durch eine natürliche Person – den zur Entscheidung berufenen Organwalter – stattfindet.

9. Die Formalisierung des Kriegsopferversorgungsgesetzes in Java

9.1. Einleitung

In diesem Kapitel wird das Kriegsopferversorgungsgesetz im Überblick dargestellt, und im Anschluss genauer auf die Beschädigtenrente als monetäre Leistung eingegangen. Diese Ausführungen sind die Grundlage für eine exemplarische Formalisierung der Beschädigtenrente in Java. Das Programm dient der Darstellung der Vorgehensweise bei der Implementierung von Normen. Anhand ausgewählter Beispiele werden verschiedene Möglichkeiten für die Modellierung von Normen in Java diskutiert.

Das Kapitel gibt Aufschluss, welche Aspekte der Normen modelliert werden müssen, damit die Vollziehung durch die Behörde unterstützt werden kann. Insofern kann das erstellte Programm als Referenzmodell verstanden werden, an dem andere Ansätze gemessen werden.

Die abschließende Diskussion fasst die Ergebnisse zusammen und gibt einen Ausblick auf zukünftige Forschungen.

9.2. Das Kriegsopferversorgungsgesetz

Dieser Abschnitt widmet sich der Darstellung des Kriegsopferversorgungsgesetzes 1957 (KOVG 1957), BGBl. Nr. 152/1957, in der Fassung BGBl. I Nr. 81/2013. Paragrafen des Allgemeinen Sozialversicherungsgesetzes (ASVG) beziehen sich auf die Fassung BGBl. Nr. 189/1955, geändert durch BGBl. II Nr. 434/2013.

Die im Folgenden verwendeten Paragrafen, ohne Angabe des Gesetzes, sind solche des KOVG in der oben angegebenen Fassung. Die genannten Fassungen der Gesetze repräsentieren die Rechtslage zum 31. Dezember 2013, auf die sich die nachfolgenden Ausführungen beziehen.

Das Kriegsopferversorgungsgesetz ist die zentrale Rechtsvorschrift im „Versorgungsrecht“. So verweisen unter anderem das Heeresversorgungsgesetz BGBl. Nr. 27/1964 (HVG), das Impfschadengesetz BGBl. Nr. 371/1973 und das Verbrechensopfergesetz BGBl. Nr. 288/197 (VOG) für die Prüfung und Berechnung verschiedener Ansprüche auf das KOVG.

Nach diesem kurzen systematischen Überblick soll auf das Kriegsopferversorgungsgesetz selbst eingegangen werden. Zentrales Anliegen des KOVG ist es, für eine erlittene

Dienstbeschädigung einen entsprechend Ausgleich zu gewähren. Um dieses Ziel zu erreichen, normiert es diverse Versorgungsleistungen für Kriegsbeschädigte und zur Absicherung der Hinterbliebenen.

Anspruch auf Leistungen haben österreichische Staatsbürger, die für die Republik Österreich, die vormalige österreichisch-ungarische Monarchie oder deren Verbündete, oder nach dem 13. März 1938 als Soldat der ehemaligen deutschen Wehrmacht *militärische Dienste* geleistet und hierdurch oder durch die vormilitärische Ausbildung eine *Gesundheitsschädigung* (Dienstbeschädigung) erlitten haben. Hat das schädigende Ereignis den Tod verursacht, so sind deren Hinterbliebene versorgungsberechtigt.

Diesen Personen, die *Kriegsbeschädigte* genannt werden, sind Personen gleichgestellt, die durch sonstige Dienstverrichtung (zum Beispiel Krankenschwester, Reichsarbeitsdienst), Kriegsgefangenschaft, unverschuldete Kriegseinwirkung (zum Beispiel Bombenangriff, aufgefundene Sprengkörper) oder unverschuldete Gewaltakte der Besatzungsmächte Österreichs eine Gesundheitsschädigung erlitten haben. Damit sind insbesondere auch Personen erfasst, die *unverschuldet* durch Blindgänger eine Gesundheitsverletzung erlitten haben.

Den Kriegsbeschädigten gebühren folgende Versorgungsleistungen: Eine Beschädigtengrundrente, wenn die Erwerbsfähigkeit in Folge der Dienstbeschädigung um mindestens 25 % (ab 1.1.2008: mind. 20 %) gemindert ist, Zusatzrente und Familienzulagen (garantiertes Mindesteinkommen für Schwerbeschädigte), Schwerstbeschädigtenzulage, Pflegezulage oder Blindenzulage, Blindenführzulage, Pflegegeld nach dem Bundespflegegeldgesetz, pauschalierter Ersatz für Mehrverbrauch an Kleidern und Wäsche, Diätkostenzuschüsse, Heilfürsorge und orthopädische Versorgung sowie berufliche und soziale Rehabilitation.

Zu den Leistungen für Hinterbliebene gehören unter anderem die Hinterbliebenenrente für hinterbliebene Ehegatten und waisenversorgungsberechtigte Kinder, das Sterbegeld und die Gebührnisse für das Sterbevierteljahr.

Die Versorgungsleitungen können gemäß ihrem Inhalt in monetäre, medizinische und soziale Leistungen eingeteilt werden. Die Auszahlung der Beschädigtenrente erfolgt monatlich und ist jeweils am Monatsersten im Voraus fällig (§ 66). Am 1. Mai und am 1. Oktober ist zusätzlich eine Sonderzahlung zu gewähren (§ 109).

Überdies werden immer wieder Sonderbestimmungen in das Gesetz aufgenommen, die beispielsweise unter gegebenen Voraussetzungen einen Zuschuss zu den Energiekosten, verteilt über mehrere Monate, gewähren (§ 113d i.V.m. § 638 ASVG). Zu diesen Sonderbestimmungen gehören auch die Einmalzahlungen für bestimmte Jahre (§§ 113f, 113g), die einmalig in einem bestimmten Monat des jeweiligen Jahres auszuzahlen sind.

Systematisch unterscheidet das Gesetz zudem zwischen „einkommensabhängigen“ und „einkommensunabhängigen“ Leistungen. Muss das Einkommen für die Berechnung einer Leistung berücksichtigt werden, handelt es sich um eine „einkommensabhängige“ Leistung nach der Terminologie des Gesetzes. Diese Kategorisierung ist Anknüpfungspunkt für die Gewährung weiterer Leistungen wie die erwähnten Einmalzahlungen, die unter anderem einen Anspruch auf eine einkommensabhängige Leistung voraussetzen.

9.3. Die Beschädigtenrente

Die im 3. Abschnitt des KOVG in den §§ 7 ff. normierte Beschädigtenrente ist die grundlegende monetäre Versorgungsleistung des KOVG. Ihr Zweck ist es, einen Beschädigten finanziell für eine erlittene Dienstbeschädigung zu entschädigen, sofern und solange dessen Erwerbsfähigkeit um mindestens 20 von Hundert (v.H.) vermindert ist. Grundsätzlich gilt, je höher die Minderung der Erwerbsfähigkeit desto höher ist der Anspruch auf Beschädigtenrente.

Vereinfacht dargestellt, setzt sich die Beschädigtenrente aus der Grund- und Zusatzrente zusammen. Zur Beschädigtenrente sind dann, sofern die Voraussetzungen dafür vorliegen, die Pflegezulage oder alternativ die Blindenzulage sowie die Kleider- und Wäschepauschale zu addieren. Daneben gibt es noch diverse Erhöhungen der Grund- und der Zusatzrente.

9.3.1. Die gesetzliche Grundlage

Die folgende Darstellung widmet sich den wichtigsten Bestimmungen der Beschädigtenrente, die im 3. Abschnitt des KOVG (§§ 7 ff.) zu finden ist. Es wird der für die Formalisierung wesentliche Gehalt der Bestimmungen zusammengefasst wiedergegeben, dabei wird auf die Angabe der konkreten Absätze weitgehend verzichtet, außer wenn dies aufgrund der Komplexität der jeweiligen Bestimmung zweckmäßig erscheint. Falls zweckdienlich, werden erläuternde Kommentare hinzugefügt. Die folgenden Formulierungen weichen daher unter Umständen vom ursprünglichen Gesetzestext ab.

§ 7 KOVG (Anspruchsgrundlage): Der Antragsteller, vom Gesetz *Beschädigter* genannt, hat Anspruch auf Beschädigtenrente, wenn und solange seine Erwerbsfähigkeit infolge der Dienstbeschädigung um mindestens 20 v.H. vermindert ist. Unter Minderung der Erwerbsfähigkeit ist die durch Dienstbeschädigung bewirkte körperliche Beeinträchtigung in Hinsicht auf das allgemeine Erwerbsleben zu verstehen.

Die Minderung der Erwerbsfähigkeit ist nach Richtsätzen einzuschätzen, die den wissenschaftlichen Erfahrungen entsprechen. Das Bundesministerium für soziale Verwaltung ist ermächtigt, hierfür nach Anhörung des Bundesbehindertenbeirates verbindliche Richtsätze aufzustellen.

§ 9 KOVG (Klassifizierung des Beschädigten): Beschädigte mit einer Minderung der Erwerbsfähigkeit von 50 v. H. oder darüber heißen *Schwerbeschädigte*. Als *erwerbsunfähig* gelten Schwerbeschädigte mit einer Minderung der Erwerbsfähigkeit von 90 v.H. und 100 v.H. (*erwerbsunfähige Schwerbeschädigte*).

§ 10 KOVG (Beschädigtenrente): Die Beschädigtenrente wird als *Grundrente* und als *Zusatzrente* geleistet.

§ 11 KOVG (Höhe der Grundrente): (1) Die Grundrente für erwerbsunfähige Schwerbeschädigte beträgt monatlich 511,00 €. Für die Beschädigten mit einer Minderung der Erwerbsfähigkeit von 20 v.H. bis 80 v.H. ist die Grundrente aus den folgenden Hundertsätzen des jeweiligen Betrages der Grundrente für erwerbsunfähige Schwerbeschädigte zu berechnen:

1. bei einer Minderung der Erwerbsfähigkeit von 20 v.H. aus 10 v.H.,
2. bei einer Minderung der Erwerbsfähigkeit von 30 v.H. aus 20 v.H.,
3. bei einer Minderung der Erwerbsfähigkeit von 40 v.H. aus 30 v.H.,
4. bei einer Minderung der Erwerbsfähigkeit von 50 v.H. aus 40 v.H.,
5. bei einer Minderung der Erwerbsfähigkeit von 60 v.H. aus 50 v.H.,
6. bei einer Minderung der Erwerbsfähigkeit von 70 v.H. aus 60 v.H. und
7. bei einer Minderung der Erwerbsfähigkeit von 80 v.H. aus 80 v.H.

(2) Der festgestellte Betrag erhöht sich für männliche und weibliche Schwerbeschädigte in Abhängigkeit vom Alter.

Die Grundrente ist vom Ersten des Monates an, der im Falle von männlichen Schwerbeschädigten auf die Vollendung des 60., im Falle von weiblichen Schwerbeschädigten auf die Vollendung des 55. Lebensjahres folgt, um 20,90 € zu erhöhen.

(3) An Stelle des in Abs. 2 angeführten Betrages gebührt Schwerbeschädigten eine Erhöhung der Grundrente vom Ersten des Monates an, der auf die Vollendung des 65., 70., 75., beziehungsweise 80. Lebensjahres folgt, in folgendem Ausmaß:

nach Vollendung des	bei einer Minderung der Erwerbsfähigkeit von				
	50 v.H.	60 v.H.	70 v.H.	80 v.H.	90/100 v.H.
65. Lebensjahres	22,90 €	38,20 €	46,30 €	61,30 €	76,60 €
70. Lebensjahres	46,40 €	76,40 €	86,80 €	102,30 €	122,50 €
75. Lebensjahres	84,70 €	114,90 €	128,00 €	143,00 €	158,50 €
80. Lebensjahres	122,50 €	153,50 €	168,70 €	184,00 €	199,40 €

§ 11a KOVG (Schwerstbeschädigtenzulage): Erwerbsunfähige Schwerbeschädigte erhalten zur Beschädigtenrente eine Schwerstbeschädigtenzulage, wenn die Summe der Hundertsätze, die nach den Richtsätzen auf die einzelnen Dienstbeschädigungen entfallen, die Zahl 130 erreicht.

Bei der Addition der Hundertsätze der einzelnen Dienstbeschädigungen gilt es einige Besonderheiten zu berücksichtigen, die an dieser Stelle nicht dargestellt werden.

Die Schwerstbeschädigtenzulage ist gemäß § 11a Abs. 4 nach der Summe der ermittelten Hundertsätze zu bemessen und aus den folgenden Hundertsätzen des jeweiligen Betrages der Grundrente für erwerbsunfähige Schwerbeschädigte zu berechnen:

1. bei einer Summe von mindestens 130 aus 30 v.H.,
2. bei einer Summe von mindestens 160 aus 40 v.H.,
3. bei einer Summe von mindestens 190 aus 50 v.H.,
4. bei einer Summe von mindestens 220 aus 60 v.H.,
5. bei einer Summe von mindestens 250 aus 70 v.H.,
6. bei einer Summe von mindestens 280 aus 80 v.H..

Überdies ist für Empfänger einer Pflege- oder Blindenzulage in § 11a Abs. 4 KOVG eine Sonderbestimmung vorgesehen, die eine gewisse Mindesthöhe der

Schwerbeschädigtenzulage gewährleistet. Dabei werden die Pflegestufen des § 18 Abs. 3 den Hundertsätzen des § 11a Abs. 4 gleichgestellt. Ergibt sich aus dieser Sonderbestimmung eine höhere Schwerstbeschädigtenzulage, ist diese zu gewähren, sonst der übliche nach Abs. 1 bis 4 ermittelte Betrag.

§ 12 KOVG (Zusatzrente): (1) Schwerbeschädigte, die das 18. Lebensjahr vollendet haben, erhalten zur Sicherung ihrer Lebenshaltung zur Grundrente eine Zusatzrente, wenn sie kein Einkommen haben, das die Gewährung einer Zusatzrente ausschließt.

(2) Die Zusatzrente beträgt monatlich 266,90 €. Sie ist auf Antrag und nur insoweit zu zahlen, als das monatliche Einkommen des Schwerbeschädigten ohne Berücksichtigung der Grundrente und einer allfälligen Schwerstbeschädigtenzulage die Höhe der ihm bei Erwerbsunfähigkeit zustehenden Beschädigtenrente - Grundrente und Zusatzrente, jedoch ohne Berücksichtigung der Erhöhungen nach Abs. 3 und § 11 Abs. 2 und 3 – nicht erreicht. Diese Grenze erhöht sich, falls Familienzulagen gebühren, um je 40,60 €.

Zweck dieser Bestimmung ist es, die Höhe der Zusatzrente in Abhängigkeit vom Einkommen des Schwerbeschädigten zu begrenzen. Zunächst ist die Höhe der Beschädigtenrente für erwerbsunfähige Schwerbeschädigte ohne die Erhöhungen der Grundrente aufgrund des Alters (§ 11 Abs. 2 und 3) und der Zusatzrente durch die Ausgleichszulage (§ 12 Abs. 3) zu ermitteln. Zu diesem Betrag ist die Familienzulage in der Höhe von 40,60 € für jedes Familienmitglied zu addieren. Die Höhe der Zusatzrente ergibt sich sodann aus der ermittelten Einkommensgrenze abzüglich des Einkommens des Schwerbeschädigten, wobei höchstens ein Betrag von 266,90 € geleistet wird.

Definiert man die Einkommensgrenze EG samt allfälligen Erhöhungen, das Einkommen EK und die Zusatzrente ZR, ergibt sich daraus folgende Formel: $ZR = \min(266{,}90, \max(EG - E, 0))$.

Der innere Ausdruck $\max(EG - E, 0)$ berechnet die maximale Höhe der Zusatzrente als Differenz zwischen Einkommensgrenze und Einkommen. Die Maximumfunktion „max“ sorgt dafür, dass bei einer negativen Differenz – bei einem Einkommen größer als die Einkommensgrenze – das Ergebnis gleich 0 ist. Durch die äußere Minimumfunktion „min“ wird schließlich die Zusatzrente mit dem Höchstbetrag von € 266,90 begrenzt.

Für die Berechnung der Einkommensgrenze EG seien die Grundrente GR und die Zusatzrente ZR für erwerbsunfähige Schwerbeschädigte sowie der Erhöhungsbetrag EB definiert. Die Berechnung der Einkommensgrenze EG lässt sich sodann mit der Funktion $f(x) = GR + ZR + x * EB$ beschreiben, wobei x die Anzahl der Familienangehörigen repräsentiert.

Die Grundrente GR ist dabei mit dem Höchstwert für erwerbsunfähige Schwerbeschädigte gemäß § 11 Abs. 1 in der Höhe von € 511,00 anzusetzen. Die altersabhängigen Erhöhungen gemäß § 11 Abs. 2 und 3 bleiben dabei außer Betracht. Für die Zusatzrente ZR ist sodann der maximale Wert von € 266,90 gemäß § 12 Abs. 2 heranzuziehen. Auch hier ist die vom Einkommen abhängige Erhöhung gemäß § 12 Abs. 3 nicht zu berücksichtigen.

Der Umstand, dass weder die altersabhängigen Erhöhungen der Grundrente noch die einkommensabhängige Erhöhung der Zusatzrente für die Bestimmung der Einkommensgrenze maßgeblich sind, erleichtert deren Berechnung erheblich. Es muss für die Berechnung nicht auf individuelle Eigenschaften des Beschädigten Bezug genommen werden, sondern es wird vom Höchstbetrag der Grundrente ausgegangen.

Der Erhöhungsbetrag EB für jeden Familienangehörigen x, für den eine Familienzulage gebührt, ist gemäß § 12 Abs. 2 fest mit einem Wert von € 40,60 bestimmt. § 12 Abs. 2 definiert, welche Personen als Familienangehörige zu verstehen sind. Zum Kreis der Familienangehörigen gehören insbesondere der Ehegatte und die Kinder des Beschädigten bis zum vollendeten 18. Lebensjahr – gegebenenfalls auch länger, sofern die Voraussetzungen des § 41 gegeben sind. Wichtig ist, dass nur solche Familienangehörige zu berücksichtigen sind, denen auch eine Familienzulage gemäß § 16 gebührt.

Hinweis: Die Differenz zwischen der aufgrund von Familienzulagen erhöhten Zusatzrente und der Zusatzrente ohne der Erhöhung ist von etwaigen Familienzulagen abzuziehen – siehe dazu die Ausführungen zu § 16 weiter unten.

(3) Die Zusatzrente ist insoweit zu erhöhen, als das monatliche Einkommen, ohne Berücksichtigung der Grundrente und einer allfälligen Schwerstbeschädigtenzulage, 66 v.H. des jeweiligen Betrages des Richtsatzes für Pensionsberechtigte auf Witwen/Witwerpension, gemäß § 293 Abs. 1 erster Satz lit. b des Allgemeinen Sozialversicherungsgesetzes, BGBl. Nr. 189/1955, nicht erreicht. Diese Beträge sind in sinngemäßer Anwendung des § 63 Abs. 3 zu runden.

Mit dieser Bestimmung wird analog zur ASVG-Witwen/Witwerpension eine *Ausgleichszulage* gewährt, falls das monatliche Einkommen des Beschädigten 66 v.H. des Richtsatzes nicht erreicht.

§ 14 KOVG (Diätzuschuss): Schwerbeschädigten ist wegen der ihnen erwachsenden außergewöhnlichen Ausgaben für eine ihnen verordnete Diätverpflegung ein Zuschuss zu gewähren, wenn die Diätverpflegung wegen einer der aufgezählten Erkrankungen erforderlich ist.

Der Zuschuss gebührt auf Antrag zur Zusatzrente und beträgt

1. bei Zuckerkrankheit 31,90 € monatlich;
2. bei Zuckerkrankheit entsprechend einer Minderung der Erwerbsfähigkeit von 50/60 v.H. 63,90 € monatlich;
3. bei Zuckerkrankheit entsprechend einer Minderung der Erwerbsfähigkeit von mindestens 70 v.H. 96,00 € monatlich;
4. bei chronischen Erkrankungen des Magens, des Darmes, der Gallenblase, der Leber und der Nieren entsprechend einer Minderung der Erwerbsfähigkeit von insgesamt mindestens 50 v.H. 31,90 € monatlich;
5. bei chronischen Nierenerkrankungen mit Dialysebehandlung 96,00 € monatlich.

§ 16 KOVG (Familienzulage): Schwerbeschädigten gebührt auf Antrag zur Zusatzrente für jeden Familienangehörigen monatlich eine Familienzulage in doppelter Höhe des gemäß § 12 Abs. 2 letzter Satz jeweils festgesetzten Betrages.

In § 12 Abs. 2 KOVG wird ein Betrag von 40,60 € genannt. Damit beträgt die Familienzulage für jeden Familienangehörigen 81,20 €.

Die Familienzulage ist um jenen Betrag zu kürzen, um den die Zusatzrente gemäß § 12 Abs. 2 wegen des Anspruches auf die Familienzulage erhöht wird. Besteht Anspruch auf zwei oder mehr Familienzulagen, so sind diese zu gleichen Teilen zu kürzen. Gebührt eine Zusatzrente lediglich auf Grund der Erhöhung der Einkommensgrenze wegen des Anspruches auf Familienzulage, so beträgt die Familienzulage monatlich 40,60 €.

Die Intention des Gesetzgebers ist, dass ein Schwerbeschädigter nicht gleichzeitig eine Erhöhung der Zusatzrente aufgrund von Familienzulagen und die Familienzulagen in voller Höhe erhalten soll. Wichtig dabei ist zu vergegenwärtigen, dass eine Familienzulage nicht

direkt zur Erhöhung der Zusatzrente führt, sondern die Einkommensgrenze nach oben verschiebt, was sich in Folge auf die Höhe der Zusatzrente auswirkt.

Um den Betrag zu ermitteln, um den die Familienzulagen zu gleichen Teilen gekürzt werden sollen, muss zuerst die Zusatzrente *samt* der Erhöhung der Einkommensgrenze durch die Familienzulagen ermittelt werden. Von dieser erhöhten Zusatzrente ist die „gewöhnliche" Zusatzrente *ohne* der Erhöhung der Einkommensgrenze abzuziehen. Die *Differenz* repräsentiert jenen Betrag, um den die Familienzulagen zu gleichen Teilen zu kürzen sind.

§ 18 KOVG (Pflegezulage): Zur Beschädigtenrente wird eine Pflegezulage gewährt, wenn der Beschädigte infolge der Dienstbeschädigung so hilflos ist, dass er für lebenswichtige Verrichtungen der Hilfe einer anderen Person bedarf.

Die Höhe des Pflegegeldes ist von der jeweiligen Stufe abhängig, wobei auf die Einstufung selbst hier nicht näher eingegangen werden soll.

Die Pflegezulage beträgt monatlich

1. in der Stufe I 671,60 €,
2. in der Stufe II 1.006,80 €,
3. in der Stufe III 1.342,90 €,
4. in der Stufe IV 1.678,90 €,
5. in der Stufe V 2.014,00 €.

§ 19 KOVG (Blindenzulage): Blinden im Sinne des Gesetzes ist zur Beschädigtenrente an Stelle der Pflegezulage eine Blindenzulage zu leisten.

§ 20 KOVG (Blindenführzulage): Blinde erhalten zur Beschädigtenrente auf Antrag eine Blindenführzulage. Die Blindenführzulage beträgt monatlich 149,90 €.

§ 20a KOVG (Kleider- und Wäschepauschale): Auf Antrag zur Beschädigtenrente sind monatliche Pauschbeträge für außergewöhnlichen Kleider- und Wäscheverbrauch zu leisten. Dabei gibt es drei Kategorien mit einem Betrag von 22,70 €, 36,00 € und 60,20 €. Die zu gewährende Leistung hängt unter anderem von der Art der Amputation und der Höhe der Minderung der Erwerbsfähigkeit ab. Sind mehrere Voraussetzungen erfüllt, werden die Beträge nebeneinander geleistet.

Auf Details zu den einzelnen Gesundheitsschädigungen, die einen Anspruch begründen, soll an dieser Stelle nicht eingegangen werden.

9.3.2. Das Berechnungsschema

Nach der Systematik des Gesetzes setzt sich die Beschädigtenrente aus Grund- und Zusatzrente zusammen (§ 10). Daneben besteht Anspruch auf diverse Zulagen, Zuschüsse und Pauschalen, die hier allesamt unter dem Begriff „Zulagen“ zusammengefasst werden. Die einzelnen Zulagen werden dabei nach dem Wortlaut des Gesetzes entweder zur Zusatzrente oder zur Beschädigtenrente geleistet. Die Grundrente wird unter Umständen erhöht, es werden jedoch keine Zulagen im oben definierten Sinn zu dieser direkt gewährt.

Basierend auf dieser Kategorisierung ergibt sich nun folgendes Schema zur Berechnung der *Beschädigtenrente* samt allfälligen Zulagen:

	Grundrente (§ 11)
+	Zusatzrente i.w.S.
Σ	**Beschädigtenrente i.e.S.**
+	Schwerstbeschädigtenzulage (§ 11a)
+	Pflegezulage (§ 18)
+	Blindenzulage (§ 19)
+	Blindenführzulage (§ 20)
+	Kleider- und Wäschepauschale(n) (§ 20a)
Σ	**Beschädigtenrente i.w.S.**

Die Berechnung der *Zusatzrente i.w.S.* samt allfälligen Zulagen stellt sich wie folgt dar:

	Zusatzrente i.e.S. (§ 12)
+	Diätzuschuss (§ 14)
+	Familienzulage(n) (§ 16)
Σ	**Zusatzrente i.w.S.**

Zur Erläuterung dieses, auf den ersten Blick recht einfachen, Berechnungsschemas scheinen einige weitergehende Ausführungen angebracht.

Die Beschädigtenrente besteht nach der Terminologie des Gesetzes streng genommen nur aus Grundrente und Zusatzrente. Die bloß aus Grund- und Zusatzrente bestehende Beschädigtenrente wird „Beschädigtenrente im engeren Sinn“ genannt. Die in der oben stehenden Berechnung ersichtlichen Zulagen werden „zur“ Beschädigtenrente gewährt, sind aber selbst nicht Teil derselben. Die Summe der Beschädigtenrente i.e.S. und der Zulagen wird als „Beschädigtenrente im weiteren Sinn“ bezeichnet.

Durch diese begriffliche Bündelung der einzelnen Leistungen wird eine adäquate Bezeichnung für die Beschädigtenrente samt allfälligen Zulagen geschaffen. Um eine gewisse Vereinfachung zu erreichen, wurde die Berechnung der Zusatzrente i.w.S. separat dargestellt. Auch hier wird zwischen der „Zusatzrente i.e.S.“ ohne Zulagen und der „Zusatzrente i.w.S.“ samt allfälligen Zulagen differenziert.

Angemerkt sei, dass die gewählte Darstellung bewusst auf die Präsentation der Voraussetzungen der einzelnen Leistungen verzichtet. Ebenso wenig werden die Abhängigkeiten der einzelnen Leistungen und Ausschlussrelationen zwischen diesen behandelt. Beispielsweise erhalten Blinde anstatt einer allfälligen Pflegezulage eine entsprechende Blindenzulage, nicht jedoch beide Leistungen. Durch die Abhängigkeiten wird die Reihenfolge gewissermaßen vorgegeben, in der die einzelnen Leistungen berechnet werden müssen.

Auch wenn bestimmte Aspekte zu Gunsten einer einfacheren Darstellung bewusst vernachlässigt wurden, zeigt das Berechnungsschema recht anschaulich die Systematik des Gesetzes und gibt einen guten Überblick über den Funktionsumfang, den eine Software zur Berechnung der Beschädigtenrente implementieren muss.

9.4. Systembeschreibung

9.4.1. Architektur

Ein bewährter Ansatz in der Softwareentwicklung ist die Unterteilung eines Softwaresystems in Module. Die Grenzen zwischen diesen Modulen werden dabei üblicherweise anhand der einem Modul zugewiesenen Funktionalität gezogen. Jedes Modul soll eine möglichst abgegrenzte Aufgabe im Gesamtsystem übernehmen. Dies entspricht dem Separation-of-Concerns-Prinzip, das in Abschnitt 9.5.3. näher dargestellt wird.

Die einzelnen Module kommunizieren über Schnittstellen, sogenannten „Interfaces“, miteinander. Ein Interface repräsentiert eine abstrakte Schnittstelle, die konkreten Details der Implementierung des Moduls bleiben dahinter verborgen.

Abstrakt handelt es sich bei einer Schnittstelle um eine Beschreibung der Eigenschaften eines Systems als Blackbox. Die Schnittstelle repräsentiert die sichtbare „Oberfläche“ des Systems, die Kommunikation ist nur über diese möglich. Die Spezifikationen von Schnittstellen können sich im Hinblick auf den Grad der Formalisierung unterscheiden. Eine rein textuelle Beschreibung einer Benutzeroberfläche beispielsweise weist im Gegensatz zu einer Programmierschnittstelle (API) einen eher geringen Formalisierungsgrad auf.

Ein Modul, das eine allgemein verbindliche Schnittstelle und seine externen Abhängigkeiten definiert, wird auch als „Komponente“ bezeichnet. Nicht jedes Modul ist aber auch zwingend eine Komponente in diesem Sinn. Oftmals wird der Begriff „Modul“ sehr vage zur Bezeichnung diverser Programmeinheiten verwendet. In Java hat der Begriff Modul keine spezielle Bedeutung und bezeichnet beispielsweise ein Package (vgl. Gosling, Joy, Steele, Bracha, & Buckley, 2013, S. 3) oder eine Programmbibliothek als ein Bündel von Klassen, Interfaces und anderen Ressourcen. Eine solche Programmbibliothek wird üblicherweise als „Java Archive“, umgangssprachlich auch „JAR-Datei“ genannt, zur Verfügung gestellt.

Die Schnittstelle einer Komponente muss dabei nicht immer eine Programmierschnittstelle (API) sein. Auch eine grafische Benutzeroberfläche (GUI) kann als Schnittstelle einer Komponente verstanden werden. In diesem Kontext wird mit Schnittstelle in der Regel eine API bezeichnet.

Im Bereich der objektorientierten Programmierung spricht man von einem Interface – das ist die eigentliche Programmierschnittstelle des Moduls (API) – und dessen Implementierung – das ist eine Klasse, die das Interface implementiert. Um nicht an die objektorientierte Terminologie gebunden zu sein, wird der abstrakte Begriff des Moduls verwendet.

Selbstverständlich kann sich ein Modul weiterer Module bedienen, um eine bestimmte Aufgabe zu lösen. So ist es nicht unüblich, dass ein zentrales Modul eine einheitliche Schnittstelle für den Aufrufer zur Verfügung stellt, die den Aufruf an weitere Hilfsmodule delegiert. Auf diese Weise lassen sich komplexe Lösungsalgorithmen in überschaubare Teile zerlegen, die wiederverwendet werden können.

Der Begriff „Modul“ wird im Folgenden stets im Sinne einer funktional definierten und durch Schnittstellen begrenzten Einheit verwendet. Die Begriffe „Modul“ und „Komponente“ sind daher in diesem Kontext gleichbedeutend, sofern nicht explizit anders angegeben. Da jedes Modul über seine Schnittstelle ein bestimmtes „Service“ bereitstellt, spricht man oftmals schlicht von „Service“, und meint dabei die technische Programmierschnittstelle. Ein Webservice ist beispielsweise ein Dienst, der über eine definierte Schnittstelle angesprochen werden kann, und über das „Web“ bereitgestellt wird.

Der Begriff der „Komponente“ wird gerade im Kontext der „komponentenbasierten Entwicklung“ auch als eine Einheit höherer Abstraktionsstufe begriffen, die eine Reihe von Funktionen als „Services“ über Schnittstellen bereitstellt. Eine Komponente in diesem Sinne ist auch für die Konfiguration der Implementierungen der jeweiligen Schnittstellen verantwortlich, und kann als selbstständige Einheit auf einem Server ausgeliefert werden.

Es existieren keine verbindlichen Regeln, wie der Aufgabenbereich eines Moduls und dessen Schnittstellen zu definieren sind. Die Aufgabe des Softwarearchitekten ist es, unwesentliche Details zu erkennen und bei der Definition der Schnittstellen tunlichst zu ignorieren. Welche Details wesentlich sind, bestimmt sich nach der Kernaufgabe des jeweiligen Moduls. Es gilt zu vermeiden, dass die Grenzen der Module zu eng gezogen werden, da dies zu einer unüberschaubaren Anzahl an Modulen führt. Gutes Design von Softwaresystemen erfordert daher mangels fester Regeln viel Wissen und Erfahrung des Softwarearchitekten.

Die Qualität eines Softwaresystems spiegelt sich maßgeblich in dessen Eigenschaft wieder, auf neue Anforderungen möglichst flexibel reagieren zu können. Je stärker sich die Änderung einer Komponente auf andere Komponenten auswirkt, desto höher ist der Grad der Koppelung des Gesamtsystems.

Durch die Modularisierung wird die Strategie verfolgt, ein Gesamtproblem in mehrere kleine, besser überschaubare Teilprobleme zu zerlegen. Falls aufgrund neuer Anforderungen Änderungen der Software notwendig sind, beschränken sich diese nur auf einzelne Module und betreffen nicht das gesamte System. Dies reduziert den Wartungsaufwand, der durch neue Anforderungen bedingt wird, erheblich. Neue Module können, falls notwendig, hinzugefügt werden. Das Gesamtsystem wird daher durch die Modularisierung resistenter gegen Änderungen. Überdies wird die Wiederverwendbarkeit einzelner Teile der Software

durch die Modularisierung gewährleistet. Im Bereich der Informatik spricht man in dieser Hinsicht von „loser Koppelung“ von Softwarekomponenten.

Lose Koppelung und Wiederverwendbarkeit haben jedoch ihren Preis. Das Design der Architektur modularer Systeme ist aufwändiger als bei monolithischer Software. Auf lange Sicht überwiegen jedoch die Vorteile eines modularen Systems, insbesondere im Hinblick auf die Kosten für Anpassungen. Obwohl im Bereich der Verwaltung die implementierten Systeme oft jahrzehntelang in Betrieb sind, wird der Faktor der Nachhaltigkeit aufgrund kurzfristiger finanzieller Einsparungen während der Entwicklung in der Praxis vernachlässigt. Dieser Umstand läuft dem Ziel des E-Governments, eine (längerfristige) Effizienzsteigerung herbeizuführen, entgegen.

9.4.2. Das Subsumtionsmodul

In diesem Unterkapitel wird die Architektur des „Subsumtionsmoduls“ dargestellt. Diese Ausführungen sollen insbesondere zeigen, wo das Subsumtionsmodul im Gesamtsystem einzuordnen und insbesondere wie dessen Schnittstelle definiert ist.

Zunächst soll eine abstrakte Beschreibung des Subsumtionsmoduls erfolgen, die bewusst auf Details hinsichtlich der verwendeten Technologie und des formalisierten Gesetzes verzichtet. Damit wird eine solide Grundlage für die Formalisierung des KOVG mit einer beliebigen Sprache geschaffen. Eine Sprache ist in diesem Sinne beispielsweise Java oder die Web Ontology Language für die Erstellung einer Ontologie. Die Darstellung der Architektur wird dabei möglichst generell gehalten, sodass diese auch für die Formalisierung anderer Gesetze Relevanz hat.

Was versteht man nun unter dem bereits erwähnten „Subsumtionsmodul“? Das Subsumtionsmodul ist als ein abstrakt durch Schnittstellen abgegrenzter Softwareteil zu verstehen, der für die automatisierte Vollziehung von Normen – also für die Rechtsanwendung – zuständig ist.

Wie aus dem Namen des Modus hervorgeht, ist dessen wichtigste Funktion die automatisierte Anwendung von Rechtsnormen. Das Modul soll also die gesetzlichen Vorschriften auf die Fakten des Einzelfalles anwenden, um so zu einem rechtlichen Ergebnis – dem Bescheid – zu gelangen. Dabei ist die rechtlich korrekte Erstellung der Erledigung von größter Wichtigkeit. Die Software muss nicht nur die Normen korrekt anwenden, sondern

auch die Entscheidungen und deren Gründe nachvollziehbar dokumentieren, damit eine entsprechende Begründung generiert werden kann.

Abstrakt beschrieben, muss das Subsumtionsmodul eine Menge von Eingabedaten entgegen nehmen, diese verarbeiten und ein entsprechendes Ergebnis liefern. Im Folgenden sollen diese noch relativ vagen Begrifflichkeiten näher untersucht werden. Dabei gilt es, stets den rechtlichen Hintergrund dieser Ausführungen in Erinnerung zu behalten.

Die Eingabedaten: Damit das Subsumtionsmodul seine Aufgabe erfüllen kann, benötigt es diverse Daten. Man unterscheidet zwischen den Fakten des Einzelfalls und den Regeln, die die Rechtsnormen repräsentieren.

Unter den Fakten sind sämtliche Umstände des Falls zu verstehen, die rechtlich relevant sind. Sie repräsentieren also die Elemente des Sachverhalts, die unter die Elemente des Tatbestands der Normen subsumiert werden sollen. Die Fakten werden über eine definierte Schnittstelle in strukturierter Form zur weiteren Verarbeitung an das Modul übergeben.

Die Regeln repräsentieren die normative Grundlage der Subsumtion. Bei diesen handelt es sich also um eine formale Darstellung der Rechtsnormen, deren Quelle Gesetze und Verordnungen sind. Sie werden in der Regel nicht über die externe Schnittstelle des Moduls übergeben sondern sind Teil des Systems selbst. Demzufolge werden sie aus der internen Wissensbasis des Systems gelesen.

Die Verarbeitung: Unter der „Verarbeitung“ versteht man in diesem Kontext die Durchführung der juristischen Subsumtion. Die generelle, abstrakte Wissensbasis muss mit dem individuellen konkreten Lebenssachverhalt in Verbindung gebracht werden. Das Programm muss demnach den allgemein bekannten *juristischen Syllogismus* zur Anwendung bringen. Aus den Prämissen wird mithilfe logischer Operatoren die Schlussfolgerung *deduktiv* abgeleitet. Das Modul führt demnach grundsätzlich eine Art deduktiver Schlussfolgerung aus, berücksichtigt dabei jedoch „Konflikte“ zwischen Rechtsnormen.

Um zu einem Ergebnis zu gelangen, ist es in der Regel erforderlich, dass die Software in der Lage ist, diverse Berechnungen durchzuführen. Einerseits kann der Tatbestand von Normen numerische Vergleiche enthalten, sodass unter Umständen eine Berechnung notwendig ist, um überhaupt die Bedingung prüfen zu können. Andererseits sind Berechnungen oftmals für

die Bestimmung der Rechtsfolgen – man denke beispielsweise an die Höhe der Steuerschuld – notwendig. Die logische Prüfung von Bedingungen alleine ist daher oftmals unzureichend.

Das Ergebnis: Das Subsumtionsmodul liefert abstrakt sämtliche Daten, die für die Erstellung eines rechtskonformen Bescheides notwendig sind. Dies umfasst vor allem die Informationen über die Regeln, die angewendet wurden, inklusive der Gründe für deren Anwendung sowie die Rechtsfolgen, die den normativen Inhalt des Bescheids widerspiegeln.

Wichtig ist, dass das „Ergebnis“ in diesem Sinn nicht bereits ein fertiger Bescheid ist, der sämtliche rechtlich relevanten Angaben und Formulierungen enthält. Vielmehr handelt es sich um eine Menge roher Daten, die dazu geeignet ist weiterverarbeitet zu werden, um letztlich einen fertigen Bescheid im Sinne einer rechtskonformen Entscheidung zu erstellen. Die Aufgabe der Erstellung des Bescheides selbst ist damit nicht die Aufgabe des Subsumtionsmoduls.

9.4.2.1. Technische Kategorisierung

Bisher wurden das Subsumtionsmodul, seine Funktionalität und seine Schnittstelle beschrieben. Offen blieb hingegen die Frage, wo dieses Modul aus technischer Sicht im Gesamtsystem einzuordnen ist?

Die Funktionalität des Subsumtionsmoduls kann als Geschäftslogik qualifiziert werden. Bei der Geschäftslogik handelt es sich üblicherweise um jenen Programmteil, der zur Unterstützung der täglichen Arbeit eines „Unternehmens“ – in diesem Fall der Behörde – dient. In einer klassischen Dreischichtenarchitektur wäre das Subsumtionsmodul damit in die mittlere Schicht, die Serviceschicht, einzuordnen. Dem Subsumtionsmodul werden die benötigten Daten in strukturierter Form übergeben.

Das Subsumtionsmodul ist die „Kernkomponente“ der Anwendung, die durch zusätzliche Module für das GUI und den Datenzugriff ergänzt wird.

9.4.3. Die Funktionsweise des Systems

In diesem Abschnitt soll die Funktion des Subsumtionsmoduls innerhalb eines Systems betrachtet werden. Als System wird dabei eine vollständige Anwendung verstanden, in der das Subsumtionsmodul eingebettet ist. Weitgehend lässt sich die erwartete Funktionsweise des Moduls bereits aus der in den vorherigen Abschnitten skizzierten Funktionalität ableiten.

Trotzdem darf nicht vergessen werden, dass die Software einen bestimmten Routineprozess in der Behörde unterstützen soll, der sich an rechtlichen Anforderungen orientiert.

Um diesen Prozess anschaulicher zu machen, sei an dieser Stelle ein möglicher Ablauf skizziert:

1. Der Antragsteller bringt einen entsprechenden Antrag auf Gewährung einer Beschädigtenrente elektronisch oder in Papierform ein, wobei im zweiten Fall das ausgefüllte Formblatt eingescannt wird. Die elektronische Datenerfassung könnte von einem Dialogsystem unterstützt werden.
2. Das verwaltungsrechtliche Ermittlungsverfahren beginnt. Sofern verfügbar, werden Daten, wie beispielsweise das Einkommen oder die Anzahl und das Alter der Familienangehörigen, aus externen Registern abgefragt. Bereits im System der Behörde vorhandene Daten können bei Bedarf ebenso verwendet werden. Der Antragsteller wird eventuell aufgefordert, weitere Unterlagen vorzulegen und erhält eine Vorladung zur ärztlichen Untersuchung. Der medizinische Sachverständige ergänzt die bereits vorhandenen Daten um das medizinische Attest, das insbesondere die Beurteilung der Dienstbeschädigung enthält.
3. Der Sachbearbeiter führt eine grundlegende rechtliche Beurteilung des Falles durch. Dabei entscheidet er aufgrund des medizinischen Gutachtens, ob beispielsweise eine Pflegezulage gewährt wird. Die rechtliche Beurteilung des Falles könnte dabei von der Software durch einen intelligenten Assistenten und entsprechende Fragestellungen unterstützt werden. Auf diese Weise können auch kompliziertere Rechtsfragen, wie beispielsweise die Interpretation vager Begriffe, vom menschlichen Benutzer des Systems beantwortet werden.
4. Die Beschreibung des Sachverhalts wird samt der rechtlichen Beurteilung in strukturierter Form an das Subsumtionsmodul übergeben. Die Software führt aufgrund dieser Daten die Berechnung der Beschädigtenrente durch und liefert das Ergebnis zurück. Im Ergebnis sind nicht nur der Gesamtbetrag, sondern auch die einzelnen Beträge der Teilleistungen enthalten. Das Ergebnis muss ausreichend Daten zur Verfügung stellen, die in weiterer Folge für die Erstellung eines rechtskonformen Bescheids als Grundlage dienen.

5. Der Sachbearbeiter kann das vom System vorgeschlagene Ergebnis samt Begründung sichten und gegebenenfalls korrigieren. Falls notwendig, kann die Berechnung auch erneut angestoßen werden. An dieser Stelle ist es theoretisch auch möglich, das automatisiert erstellte Ergebnis vollständig zu verwerfen und eine Ermessensentscheidung zu fällen.
6. Zur Minimierung des Risikos falscher Entscheidungen und zum Schutz vor Missbrauch kann noch eine automatisierte Plausibilitätsprüfung angeschlossen werden. Dabei könnte beispielsweise Alarm ausgelöst werden, sofern die zu gewährende Leistung außerordentlich hoch ist. Eine solche ist gerade bei semi-automatisierten Verfahren ratsam, in denen ein menschlicher Sachbearbeiter kaum mehr in den Beurteilungsprozess involviert ist und „lediglich" das Ergebnis durch seine Bestätigung legitimiert.
7. Sobald die rechtliche Beurteilung abgeschlossen ist, kann aus dem vorhandenen Ergebnis ein rechtskonformer Bescheid erstellt werden, der sodann elektronisch oder wie bisher im Postweg zugestellt wird. Sofern dem Antragsteller eine Beschädigtenrente zusteht, kann die zuständige Stelle über eine Schnittstelle beauftragt werden, den ermittelten monatlichen Betrag anzuweisen.

Wie aus diesem fiktiven Ablauf hervorgeht, ist die Aufgabe des Subsumtionsmoduls klar abgegrenzt. Es ist für einen kleinen, jedoch sehr wichtigen Teil des Verwaltungsverfahrens zuständig und kommt nach dem Abschluss des Ermittlungsverfahrens zum Einsatz.

Überdies ist denkbar, dass eine Behörde den Bürgern die Vorberechnung des Ergebnisses in Form eines interaktiven Services online anbietet, damit diese das Ergebnis „vorberechnen" können.

9.4.4. Abschließende Bemerkungen

In der Praxis hat es sich sehr bewährt, Softwaresysteme nach funktionalen Gesichtspunkten zu untergliedern. Im gegebenen Zusammenhang könnte beispielsweise jede Leistungsart in Form einer Komponente implementiert und durch ein gemeinsames Modul, das das KOVG repräsentiert, gebündelt werden. Auf diese Art und Weise entstünde ein Modul je Gesetz, zu dessen Vollziehung die jeweilige Behörde berufen ist. Diese Kernsystemmodule würden sodann durch diverse Hilfssystemmodule unterstützt, die beispielsweise die Generierung und Zustellung des Bescheides übernehmen.

Wie aus den obenstehenden Ausführungen hervorgeht, müssen an das Subsumtionsmodul strukturierte Daten übergeben werden. Es ist insbesondere nicht dessen Aufgabe, einen Text in natürlicher Sprache zu analysieren, um daraus strukturierte Daten wie beispielsweise den Sachverhalt zu extrahieren.

Die Extraktion strukturierter Daten aus Dokumenten, Sprachaufzeichnungen etc. muss daher dem Aufruf des Subsumtionsmoduls vorgeschaltet werden. Unstrukturierte Dokumente, wie sie in einem elektronischen Aktensystem geführt werden, sind daher einer automatisierten oder manuellen Vorverarbeitung zu unterziehen mit dem Ziel, die relevanten Daten strukturiert zu erfassen. Systeme wie der „ELAK im Bund“, deren Anliegen es ist, den Verfahrensablauf in Papier elektronisch nachzubilden, sind daher für die Erreichung des Ziels der automatisierten Entscheidungsfindung *ungeeignet*.

9.5. Die Formalisierung in Java

9.5.1. Das Programm im Überblick

In diesem Abschnitt wird ein simples Programm, geschrieben in Java, zur automatischen Berechnung der Beschädigtenrente des KOVG vorgestellt[145].

Für die Implementierung des Programms wird ein möglichst pragmatischer Ansatz verfolgt, der auf bewährte Entwurfsmuster und Vorgehensweisen zur Softwareentwicklung setzt. Die Idee ist, das Gesetz als einfaches Java-Programm zu repräsentieren. Im Gegensatz zu einem wissensbasierten System ist der Lösungsalgorithmus nicht von den formalen Regeln (der Wissensbasis) separiert. Eine eigene Wissensbasis, die ausschließlich die formalisierten Normen enthält, existiert demnach nicht.

Das Programm dient dem Zweck einer möglichst simplen Demonstration der automatisierten Vollziehung von Verwaltungsrecht. Aus diesem Grund wird auf eine grafische Oberfläche zur Eingabe und Anzeige von Daten verzichtet und die Berechnung anhand fest definierter Fakten mithilfe sogenannter Modultests durchgeführt.

Ein weiterer Umstand, der die Implementierung des Prototyps erleichtert, ist die Tatsache, dass es sich um ein „Einzelbenutzersystem“ handelt. Damit ist jedoch nicht gemeint, dass

[145] Das Programm ist auf Anfrage beim Autor und bei der Arbeitsgruppe Rechtsinformatik an der Universität Wien verfügbar.

nur ein Benutzer zur selben Zeit das System benutzen kann, sondern dass die Software lediglich den Antrag eines Antragstellers einer rechtlichen Beurteilung unterziehen muss. Die Bewertung von Anträgen verschiedener Antragsteller, die unter Umständen gegenläufige Argumente enthalten, ist hingegen nicht erforderlich.

Die Berechnung der Beschädigtenrente nach dem KOVG erfolgt monatlich. Entsprechend der Fälligkeit der Beschädigtenrente gemäß § 66 erfolgt die Berechnung zum Monatsersten. Dem Programm werden dafür über eine definierte Schnittstelle die zu beurteilenden Fakten samt dem gewünschten Datum der Berechnung übergeben. Aufgrund der übergebenen Daten wird sodann die für den jeweiligen Monat zum Monatsersten zustehende Beschädigtenrente berechnet und ein entsprechendes Objekt, das das Ergebnis repräsentiert, zurückgeliefert.

Soll die Beschädigtenrente über einen längeren Zeitraum berechnet werden, muss die Berechnung mit entsprechenden Daten öfters aufgerufen und die monatlichen Teilbeträge addiert werden.

Um den Berechnungsvorgang transparent und nachvollziehbar zu machen, gibt das Programm bei jedem Rechenschritt entsprechende Meldungen aus. Das Ergebnis der automatisierten Beurteilung wird ebenso in anschaulicher Form am Ende ausgegeben. Das Ergebnis, repräsentiert durch ein entsprechend strukturiertes Objekt, könnte ohne großen Aufwand auch in eine andere Darstellungsform wie beispielsweise XML transformiert oder zur Erstellung eines Bescheides verwendet werden.

Der Fokus liegt demnach bei diesem Versuch auf der Implementierung von Programmcode zur Repräsentation von Normen mit Java. Es soll jedoch kein vollständiges Programm samt Benutzeroberfläche etc. erstellt werden.

Dieser Ansatz kann durchaus als Fortschritt zu den üblichen Ansätzen im öffentlichen Sektor gesehen werden. So gibt es zwar E-Government-Webanwendungen, die mit „modernen“ Programmiersprachen wie Java implementiert wurden, Programme, die „Subsumtionsregeln“ implementieren, laufen jedoch bis heute oftmals auf Großrechnern und sind in veralteten Sprachen wie COBOL, PL/I (auch genannt PL/1 oder PLI) oder Assembler geschrieben. Teilweise wurde der Großrechner durch entsprechende Emulatoren abgelöst, die die alten Programme auf neuer Hardware ausführen können. Ein anderer Ansatz ist die Implementierung von Weboberflächen, die im Hintergrund auf den

Großrechner zugreifen, um alte Programme über das Internet zugänglich zu machen. Solche Ansätze ändern freilich nichts an der Tatsache, dass die Programme mit einer veralteten Technologie implementiert sind.

Die Programmiersprache Java wurde nicht nur im öffentlichen Bereich in einer Vielzahl von Projekten (erfolgreich) eingesetzt und ist wohl eine der populärsten objektorientierten Sprachen zur Entwicklung von Software. Die Java-Plattform bietet überdies Unterstützung für die Sprachen Groovy, JRuby und Closure, die allesamt in der Java-Laufzeitumgebung ausgeführt werden. Auf diese Sprachen soll an dieser Stelle nicht weiter eingegangen werden. Es sei jedoch erwähnt, dass diese Java um zusätzliche Features wie beispielsweise dynamische Typen und Metaprogrammierung erweitern.

Der Vorteil von Java ist die Plattformunabhängigkeit, die es ermöglicht, dass ein Programm auf einem beliebigen System ausgeführt werden kann. Voraussetzung dafür ist, dass für das jeweilige Betriebssystem eine Java-Laufzeitumgebung – die sogenannte „Java Virtual Machine“ – vorhanden ist. Damit kann ein Programm beispielsweise unter Windows implementiert und auf einem Server, auf dem Linux läuft, eingesetzt werden.

Zudem steht eine Vielzahl von Open-Source-Programmbibliotheken zur Verfügung, die für die Implementierung verwendet werden können. Der Einsatz von Open-Source-Software hat sich in der Praxis bewährt und stellt oftmals eine bessere Alternative zur Verwendung proprietärer Software dar. Gerade für kleinere Verwaltungsressorts sind die Kosten für Implementierung und den Betrieb bei der Verwendung von proprietären Bibliotheken und Servern nicht tragbar.

Ohne hier einen breiteren Diskurs über die Vor- und Nachteile von Open-Source-Software und dem proprietären Ansatz beginnen zu wollen, sei darauf hingewiesen, dass stets der geeignetste Ansatz für ein konkretes Problem gewählt werden sollte. Überdies ist zu bedenken, dass durch die Verwendung proprietärer Software oftmals eine Bindung an die Werkzeuge für die Entwicklung und die Server eines Anbieters erfolgt. Diese als „Vendor-Lock-in“ bezeichnete Bindung an einen Hersteller sollte stets bedacht und entsprechend bewertet werden.

9.5.2. Anforderungen

Berechnung der Beschädigtenrente: Die entwickelte Softwarekomponente muss die monatliche Berechnung der Beschädigtenrente samt allfälligen Zulagen gemäß den §§ 7 ff. KOVG durchführen.

Aus technischer Sicht muss ein Objekt zurückgeliefert werden, das den Gesamtbetrag der Beschädigtenrente sowie alle Zulagen und Erhöhungen im Einzelnen enthält, damit die Daten insbesondere für die automatisierte Erstellung der rechtlichen Begründung herangezogen werden können.

Protokollierung der Aktionen: Sämtliche Aktionen des Programmes, insbesondere die Berechnung von Leistungen, müssen in textueller Form protokolliert werden. Dies umfasst die Ausgabe der Berechnungsschritte, der Rechenoperationen und sämtlicher Beträge, die für die Berechnung verwendet werden.

9.5.3. Designprinzipien

Für die Implementierung von Software gibt es keine einheitliche und allgemein gültige Vorgehensweise. Jedes Problem ist auf seine Art und Weise einzigartig. Dies gilt umso mehr für die Repräsentation von Normen durch Programmcode. Für eine Darstellung der verschiedenen Repräsentationsmöglichkeiten von Normen siehe Bench-Capon und Gordon (2009).

Es haben sich jedoch Ansätze für den Aufbau der Grundstruktur von Programmen herausgebildet, die sich durchaus bewährt haben. Diese Erfahrungen manifestieren sich in der Softwareentwicklung in Entwurfsmustern und Designprinzipien. Überdies existieren fertige Programmbibliotheken (oft genannt „Frameworks“) für die Lösung von gängigen Problemen, die es bei der Entwicklung jedes Softwaresystems zu lösen gilt.

Durch die Vorgabe von Designprinzipien kann der Rahmen für die Entwicklung in Grundzügen abgesteckt werden.

Leichtgewichtige Architektur: Für die Implementierung mit Java wird ein möglichst leichtgewichtiger Ansatz gewählt. Der Code soll auf einer herkömmlichen Java 7 Laufzeitumgebung (Java Runtime Environment, kurz JRE) ausgeführt werden können. Damit kann der Code in einer Programmbibliothek (ein herkömmliches Java Archive) verpackt und beispielweise in einer Webanwendung eingebettet werden.

Um dies zu erreichen, wird durchgehend mit sogenannten „Plain Old Java Objects“ (POJOs) gearbeitet. Dieser Begriff wurde von Fowler und anderen auf einer Konferenz im Jahr 2000 erfunden mit der Absicht, einfache Objekte von Objekten mit zahlreichen externen Abhängigkeiten unterscheiden zu können (Fowler, 2000).

Loose coupling: Unter „Koppelung“ von Klassen und Subsystemen versteht man den Grad der Abhängigkeit zwischen diesen Klassen und Subsystemen. Enge Koppelung bedeutet, dass zusammengehörige Klassen interne Details voneinander kennen müssen. Änderungen einer Klasse wirken sich auf andere Klassen aus, sie „plätschern“ sozusagen durch das System, und die Software ist üblicherweise auch schwerer zu verstehen (J. Miller, 2008).

In lose gekoppelten Systemen hingegen haben die einzelnen Komponenten gegenseitig keine Kenntnis von den internen Strukturen der anderen Komponenten und interagieren über wohldefinierte Schnittstellen miteinander. Ein solches Design kann flexibel auf Änderungen reagieren, da Wechselwirkungen zwischen den einzelnen Komponenten minimiert werden (Freeman & Freeman, 2004, S. 53).

Um die gewünschte lose Koppelung zu erreichen, wird bei der konkreten Implementierung durchgängig gegen Abstraktionen – den Interfaces – gearbeitet.

Single Responsibility: Das Single-Responsibility-Prinzip besagt, dass es nur einen Grund geben sollte, eine Klasse ändern zu müssen (Freeman & Freeman, 2004, S. 339). Je mehr Zuständigkeiten – im Sinne von Funktionalitäten – eine Klasse hat, desto wahrscheinlicher ist es, dass die Klasse angepasst werden muss.

Das Prinzip dient als Anleitung zur Erreichung des Ziels, dass jede Klasse nur eine einzige Zuständigkeit hat. Die Abgrenzung, wo eine Zuständigkeit endet und die nächste beginnt, ist jedoch oftmals keine einfache Aufgabe.

In diesem Zusammenhang wird oft der Begriff „Cohesion“ (Kohäsion) verwendet. Darunter versteht man das Maß, wie stark die Zuständigkeiten, Daten und Methoden einer Klasse miteinander verwandt sind (J. Miller, 2008). Eine Klasse hat einen hohen Grad an Kohäsion, wenn sie Methoden mit ähnlicher Funktionalität beinhaltet, einen niedrigen Grad, wenn deren Methoden nicht miteinander verwandt sind (Freeman & Freeman, 2004, S. 339).

Es ist schwer, formale Regeln zu definieren. Ein guter Leitsatz lässt sich dennoch finden. In einer Klasse sollte man nur jene Methoden, Daten etc. finden, die man nach ihrem Namen sinnvoll erwarten darf.

Separation of Concerns: „Separation of Concerns“ (SoC) ist ein Designprinzip, um Software in abgetrennte Sektionen (Module) zu teilen. Jeder Teil kümmert sich dabei um bestimmte „Belange“. Ein Belang ist gleichbedeutend mit einem Feature des Systems. Die Features der anderen Teile sind dabei für den jeweiligen Programmteil irrelevant. Es geht darum, Informationen zu kapseln und die Implementierung zu verbergen.

Der Begriff wurde möglicherweise von Dijkstra in seinem Paper im Jahr 1982 zuerst gebraucht (Dijkstra, 1982) und von diversen Autoren analysiert und weiterentwickelt (Hürsch & Lopes, 1995). Das SoC-Prinzip ist mittlerweile ein etabliertes Designprinzip in der Softwareentwicklung und oftmals Grundlage für architektonische Entscheidungen.

Ein Systemdesign, das das SoC-Prinzip befolgt, wird „modular“ genannt. Die Trennung eines Programms in Schichten – „presentation layer“, „business logic layer“ etc. – ist beispielsweise ein Ausfluss dieses Prinzips. Die aspektorientierte Programmierung, deren Ziel es ist, generische Features wie Sicherheit, Transaktionen etc. über mehrere Klassen und Module hinweg zur Verfügung zu stellen, kann ebenso als ein Anwendungsfall dieses Designprinzips begriffen werden.

Im Verhältnis zum Single-Responsibility-Prinzip ist das Separation-of-Concerns-Prinzip genereller und richtet sich eher an größere Programmteile. Wendet man das SoC-Prinzip jedoch auf das Design von Klassen an, so entstehen in der Regel Klassen, die das Single-Responsibility-Prinzip befolgen.

Dependency Injection: Wie bereits erwähnt wurde, wird eine lose Koppelung der Komponenten durch die Verwendung von Interfaces angestrebt. Ein Interface stellt jedoch bloß eine abstrakte Funktionalität dar, ohne dass diese bereits implementiert ist. Es bedarf einer Implementierung des jeweiligen Interfaces, damit dessen abstrakt definierte Methoden auch aufgerufen werden können. Grundsätzlich muss sich das jeweilige Programm um die Erzeugung der Objekte einer konkreten Klasse selbst kümmern. Das Programm entscheidet daher, wann und wie ein Objekt einer bestimmten Klasse erzeugt wird.

Das Programm wird dadurch – trotz der Verwendung von Interfaces – an konkrete Klassen gebunden. Dies läuft insbesondere dem Prinzip der losen Koppelung und wohl auch dem Single-Responsibility-Prinzip zuwider.

Einen Ausweg aus dem Dilemma bietet das Prinzip der „Dependency Injection“. Dabei wird die Aufgabe, Objekte zu erzeugen, ausgelagert an ein „externes“ Framework. Die Erzeugung und Bereitstellung von Objekten erfolgt nicht mehr im eigenen Code.

Im konkreten Fall wird das Spring Framework verwendet, um Objekte zu erzeugen und diese zu injizieren, wo sie benötigt werden. Dadurch lässt sich ein hoher Grad an loser Koppelung erreichen. Der Code wird dadurch besser verständlich sowie besser zu testen, und das System kann flexibel auf Änderungen reagieren.

Das Dependency-Injection-Prinzip ist eine Implementierung des generelleren Inversion-of-Control-Prinzips (IoC). Der Begriff „Inversion of Control“ ist keinesfalls neu und wurde bereits 1988 von Johnson und Foote in ihrem Beitrag verwendet (R. E. Johnson & Foote, 1988). Es geht dabei um eine Umkehrung der Steuerung des Programmflusses. Der Aufruf von Methoden wird dabei von einem externen Framework übernommen. Aus diesem Grund spricht man auch vom „Hollywood-Prinzip“ und dem damit verbundenen Slogan „Don’t call us, we’ll call you“ (Freeman & Freeman, 2004, S. 296). Das IoC-Prinzip erlangte in den letzten zehn Jahren große Bekanntheit durch IoC-Container wie dem Spring Framework.

Fowler hat den Ursprung des IoC-Prinzips zurückverfolgt und betont, dass IoC nicht mit Dependency Injection verwechselt werden darf. Die Popularität von IoC-Containern hat offenbar dazu beigetragen, dass die Begriffe „Inversion of Control“ und „Dependency Injection“ als Synonyme verstanden werden (Fowler, 2005).

Safe by default: Webanwendungen wie moderne E-Government-Portale, die Services über das Internet zur Verfügung stellen, bieten eine relativ große Angriffsfläche für Hacker. Einen guten Überblick über die Schwachstellen, die bei Webanwendungen auftreten können, gibt der Artikel von Ollmann (2007). Anwendungssicherheit ist ein Aspekt, der nicht erst am Ende der Entwicklung überprüft werden sollte, sondern stetig schon während der Implementierung von Software.

Das Prinzip „Safe by default“, auch genannt „fail securely“, besagt, dass wenn ein Fehler auftritt, die konkrete Aktion „sicher“ fehlschlagen sollte. Dies umfasst mehrere Dinge wie

insbesondere ein sicheres Standardverhalten und Standardwerte. Im Kontext der Anwendungssicherheit beispielsweise wird grundsätzlich der Zugriff auf bestimmte Ressourcen verweigert und es bedarf einer expliziten Authentifizierung und Autorisierung, bevor dieser zugelassen wird. Es darf nicht passieren, dass ein Angreifer einen Fehler produzieren kann, durch den dieser Zugriff auf sensitive Ressourcen erhält (Gegick & Barnum, 2013).

Inspiriert durch das Buch von Nygard (2007) und jahrelanger Erfahrung des Autors auf dem Gebiet der Softwareentwicklung lässt sich dieses Prinzip auch auf nicht unmittelbar der Anwendungssicherheit zugehörige Belange übertagen.

Im aktuellen Kontext bedeutet dies, dass grundsätzlich dem Antragsteller keine Leistungen zustehen, bis das Programm explizit eine entsprechende Überprüfung der Fakten vorgenommen hat, die gegebenenfalls zu einer positiven Bewilligung führt. Dies ist aus rechtlicher Sicht eine Selbstverständlichkeit, bedeutet aber, dass Standardwerte und Algorithmen im Programm als grundsätzliches Ergebnis „kein Anspruch“ liefern. Aus Sicht der Programmierung ist es ein Leichtes, fälschlicherweise bestimmte Variable mit Werten zu initialisieren, die dem Antragsteller einen Anspruch zugestehen würden.

Ein weiterer Aspekt der vom Safe-by-default-Prinzip inspiriert wird, ist es, sensible Klassen – genauer deren Objekte – als „unveränderbar“ (engl. immutable) zu implementieren. Dies betrifft insbesondere jene Objekte, deren Attribute Geldwerte repräsentieren. Nach der Erzeugung ist es grundsätzlich unerwünscht, dass diese Geldwerte verändert werden können. Einem Angreifer, der möglicherweise eigenen Code über eine Schwachstelle in das System eingeschleust hat, wird so die nachträgliche Manipulation von Objekten erheblich erschwert.

Isomorphismus: Zwischen den im Gesetz zum Ausdruck gebrachten Rechtsnormen und deren formaler Repräsentation im Programmcode sollte idealerweise eine direkte Beziehung bestehen. Aus dem implementierten Code im Programm sollte dessen normative Grundlage erkennbar sein.

Aus diesem Grund sollen die Struktur und Terminologie des Gesetzes soweit als möglich beibehalten und für die Strukturierung des Codes und die Benennung von Klassen etc. herangezogen werden.

9.5.4. Das Programm in Aktion

Bevor in den folgenden Abschnitten auf weitere technische Details eingegangen wird, sei hier die Funktionsweise des Programms an einem exemplarischen Programmlauf erläutert. Für die Berechnung des Beispiels werden statische Werte verwendet und das Objekt mit den Daten des Antragstellers direkt im Code konstruiert.

```
1  Daten des Antragstellers:
2  Name:                          Max Mustermann
3  Geschlecht:                    MAENNLICH
4  Geb. am:                       1936-07-26
5  Einkommen:                     EUR 550.00
6  Angehörige:                    2
7  Pflegestufe:                   IV
8  Diätverpflegung - Zf.:         II
9  Außergewöhnlicher Kleiderverbrauch:
10   [1] Zf.:                     I
11   [2] Zf.:                     III
12 Dienstbeschädigungen:
13   [1] MdE:                     60
14   [2] MdE:                     80
```

Abbildung 8: Die Daten des Antragstellers

Zu Beginn des Testlaufs gibt das Programm die Daten des Antragstellers aus, wie in Abbildung 8 ersichtlich ist. Unter anderem ist erkennbar, dass der Antragsteller zwei Angehörige (Zeile 6) und zwei Dienstbeschädigungen erlitten hat (Zeilen 12 bis 14).

Im Anschluss daran wird das vom Subsumtionsmodul gelieferte Objekt in tabellarischer Form ausgegeben, um das Ergebnis zu veranschaulichen. Wie Abbildung 9 zeigt, ist die Gliederung der Ausgabe des Programms an das in Abschnitt 9.3.2. beschriebene Berechnungsschema angelehnt. Die Ausgabe in Zeile 1 verdeutlicht, dass die Berechnung der Beschädigtenrente monatlich erfolgt. Das Programm gibt dazu das Jahr und den Monat der Berechnung – in diesem Fall „2014-02“ – aus. Die einzelnen Teilleistungen werden gruppiert mit Summe und Angabe des Paragrafen dargestellt. In Zeile 29 ist schließlich der Gesamtbetrag der Beschädigtenrente ersichtlich.

Auch wenn die Ausgabe bewusst nicht alle Details wie beispielsweise Kürzungen wiedergibt, lässt sich doch einigermaßen der Gang der Berechnung nachvollziehen. Es sei darauf hingewiesen, dass die Präsentation des Ergebnisses nicht auf die hier gewählte, recht einfache textuelle Form beschränkt ist. Das Ergebnis ist ein Objekt, das in eine nahezu

beliebige Form transformiert werden könnte. Beispielsweise könnte das Ergebnis in XML oder in ein formatiertes PDF-Dokument transformiert werden.

```
Berechnung der Beschädigtenrente gemäß §§ 7 ff. KOVG für 2014-02:

Berechnung der Grundrente gemäß § 11:
  Grundbetrag (Abs. 1)                          €   255,50
  Erhöhung (Abs. 2 und 3)                       €   114,90
                                                ----------
Gesamtbetrag Grundrente                                    €   370,40

Berechnung der Zusatzrente gemäß § 12:
  Grundbetrag (Abs. 2)                          €   266,90
  Ausgleichszulage (Abs. 3)                     €     2,80
                                                ----------
Gesamtbetrag Zusatzrente                                   €   269,70

Berechnung Zulagen zur Zusatzrente:
  Diätzuschuss (§ 14)                           €    63,90
  2  Familienzulage(n) (§ 16)                   €   123,40
                                                ----------
Gesamtbetrag der Zulagen zur Zusatzrente                   €   187,30

Berechnung Zulagen zur Beschädigtenrente:
  Schwerstbeschädigtenzulage (§ 11a)            €   306,60
  Pflegezulage (§ 18)                           € 1.678,90
  2 Kleiderpauschale(n) (§ 20a)                 €    82,90
                                                ----------
Gesamtbetrag der Zulagen zur Beschädigtenrente             € 2.068,40

                                                ----------
Gesamtbetrag der Beschädigtenrente                         € 2.895,80
```

Abbildung 9: Ausgabe des Ergebnisses

Damit die Berechnung besser nachvollzogen werden kann und um die Anforderung der Protokollierung zu erfüllen, gibt das Programm detaillierte Meldungen während der Berechnung aus. Abbildung 10 zeigt einen Ausschnitt der Protokollmeldungen, die bei der Berechnung der Zusatzrente ausgegeben werden.

Um die einzelnen Ausgaben besser zuordnen zu können, wird zu Beginn jeder Meldung (nach der Uhrzeit) der zugehörige Paragraf und Absatz ausgegeben. Relativ gut erkennbar ist beispielsweise die Erhöhung der Einkommensgrenze aufgrund des Anspruchs auf Familienzulage.

```
Berechne Zusatzrente...
15:36:01.465 § 12: Berechne Zusatzrente (samt Erhöhungen)...
     15:36:01.465 Einkommensgrenze gemäß § 12 Abs. 2 ohne Familienzulagen:
          Grundrente EUR 511.00 + Zusatzrente EUR 266.90 = EUR 777.90
15:36:01.465 § 12 Abs. 2: Einkommensgrenze ohne Familienzulagen: EUR 777.90
15:36:01.465 § 12 Abs. 2: EUR 40.60 * 2 = Erhöhung EUR 81.20
15:36:01.465 § 12 Abs. 2: EUR 777.90 + Erhöhung EUR 81.20 = Einkommensgrenze
EUR 859.10
15:36:01.465 § 12 Abs. 2: max(0, Einkommensgrenze EUR 859.10 - Einkommen EUR
550.00) = EUR 309.10
15:36:01.465 § 12 Abs. 2: Grundbetrag Zusatzrente: min(EUR 309.10, EUR 266.90)
= EUR 266.90
15:36:01.465 § 12 Abs. 2: Grundbetrag Zusatzrente: EUR 266.90
15:36:01.465 § 12 Abs. 3: ASVG Richtsatz EUR 552.80 - Einkommen EUR 550.00 =
Ausgleichszulage EUR 2.80
15:36:01.465 § 12 Abs. 3: Ausgleichszulage zur Zusatzrente: EUR 2.80
15:36:01.465 § 12: Zusatzrente: Zusatzrente{grundbetrag=EUR 266.90,
ausgleichszulage=EUR 2.80, familienzulagen=null, diaetzuschuss=null,
betrag=EUR 269.70}
```

Abbildung 10: Protokollauszug der Berechnung der Zusatzrente

9.5.5. Entwicklungsmethode

Für die Implementierung wird ausschließlich der Gesetzestext als Quelle für die Extraktion der Normen herangezogen. Die Normen werden dabei interpretiert und in ein formales Modell übersetzt. Die Normen werden dabei als konditionale Strukturen betrachtet, die schließlich in entsprechende if-Bedingungen im Javacode übersetzt werden. Für weitere Details sei an dieser Stelle auf Abschnitt 9.5.9.3. verwiesen.

Mangels eines etablierten Standards für die Implementierung von E-Government-Software und die Formalisierung von Verwaltungsrecht wird auf bewährte Ansätze der Softwareentwicklung zurückgegriffen. Zu diesen gehören insbesondere Designprinzipien und auf diesen aufbauende Entwurfsmuster, auf die an dieser Stelle nicht weiter eingegangen werden soll.

In der Praxis sind oftmals die in Lastenheften oder Interviews enthaltenen Anforderungen die primäre Grundlage für die Implementierung solcher Systeme. Das Gesetz spielt dabei eine wichtige Rolle und ist die Grundlage für viele Anforderungen, die in Lastenheften zum Ausdruck gebracht werden. Im Lastenheft werden Ungenauigkeiten oder Widersprüche des Gesetzes vom Auftraggeber (der Behörde) geglättet, was insofern einen Vorteil gegenüber der direkten Verwendung des Gesetzes darstellt.

Der hier gewählte Ansatz weicht demnach – mangels eines realen Auftraggebers – vom in der Praxis vorherrschenden Vorgehen ab. Das Gesetz wird jedoch auch dort zur Akquisition von zusätzlichem Wissen herangezogen, wenn die Beschreibung des Systems unzureichend und lückenhaft ist. Das Gesetz ist also in jedem Fall eine wichtige Wissensquelle, womit sich die beiden Ansätze in weiten Bereichen decken.

9.5.6. Entwicklungsumgebung

Für die Formalisierung der Normen und die Implementierung des Programms wird Java Version 7 (Codename „Dolphin“) – genauer das Java 7 Development Kit (JDK) – verwendet. Die plattformunabhängige Entwicklungsumgebung Eclipse Version 4.2 (Juno) wird dabei für die Entwicklung des Projekts eingesetzt.

Für die Kompilierung und Verwaltung des Projekts wird das quelloffene Build-Management-Tool Apache Maven in Version 3.0 eingesetzt. Maven ermöglicht eine weitgehend standardisierte Erstellung und Verwaltung von Javaprojekten. Für Eclipse existieren Plugins, die Maven direkt in die grafische Entwicklungsumgebung integrieren.

Die wichtigsten Bibliotheken, die für die Implementierung verwendet werden, sind das Spring Framework 3.2, insbesondere für die „Dependency Injection“, Google Guava 16.0, Joda-Time 2.3 für Datumsberechnungen sowie JUnit 4.11 zur Durchführung von Modultests.

Der gesamte Sourcecode wird in einem lokalen Subversion-Repository zur Versionskontrolle verwaltet.

9.5.7. Architektur

Modularisierung: Dem bewährten Vorgehen in der Softwareentwicklung folgend wird die Software zur Berechnung der Beschädigtenrente nach funktionalen Kriterien in mehrere Module unterteilt. Da die einzelnen Module voneinander abhängig sind und demnach als aufeinander lagernde Schichten begriffen werden können, spricht man in diesem Zusammenhang auch von „Schichtenarchitektur“. Ein Modul entspricht in diesem Kontext einer Schicht.

Das Gesamtsystem besteht aus mehreren solchen Schichten (englisch „layer“), die unterschiedliche Funktionen beherbergen. Jede dieser Schichten entspricht dabei einem Modul, das zum Beispiel Interfaces, Klassen und Konfigurationsdateien etc. enthält.

Angemerkt sei, dass der Begriff „Modul“ an dieser Stelle nicht im Sinne einer Komponente als Modul mit einer definierten Schnittstelle gebraucht wird. Vielmehr wird „Modul“ hier zur Bezeichnung von größeren Teilen eines Gesamtsystems – den „Schichten“ – verwendet.

Im Detail handelt es sich um die Schichten Domain, Service und Repository. Jede Schicht repräsentiert abstrakt ein Bündel von Methoden, die durch Interfaces bereitgestellt werden.

Aufgrund des vergleichsweise geringen Umfangs des Projekts werden diese aber aus technischer Sicht nicht in separaten Subprojekten entwickelt, sondern bloß durch unterschiedliche Packages getrennt. Alle Subpackages befinden sich unterhalb des Packages „at.js.diss.simple“, das das Basispackage der Anwendung repräsentiert, und heißen „domain“, „service“ und „repository“.

In größeren Projekten wird die Problemdomäne üblicherweise in mehrere Teile gegliedert. Im konkreten Fall bestünde die Problemdomäne aus dem gesamten Versorgungsrecht und könnte demnach in die einzelnen Gesetze gegliedert werden. Jeder Teil der Problemdomäne könnte in einem eigenen Projekt entwickelt werden, das jeweils in einzelne Module unterteilt ist. Der Name des Basispackages jedes Teilprojekts würde dann den Namen des Gesetzes zur Unterscheidung beinhalten. Das Teilprojekt für das KOVG demnach „at.js.simple.kovg“, jenes für das HVG „at.js.simple.hvg“ usw.

Die Schichten sind durch die durchgängige Verwendung von Interfaces nicht starr aneinander gekoppelt. Die konkrete Implementierung eines Interfaces bleibt für den Verwender verborgen. Änderungen, die die Implementierung betreffen, wirken sich somit nicht auf die restlichen Teile des Systems aus.

Überdies erlaubt die lose Koppelung der Systemteile aneinander, dass die jeweilige Implementierung durch eine andere ausgetauscht werden kann. So könnte beispielsweise die direkte Implementierung der Berechnung der Beschädigtenrente in Java durch eine andere ersetzt werden, die zum Beispiel eine Rule Engine verwendet.

Die Domain-Schicht repräsentiert die Begriffe der Problemdomäne – des Gesetzes – und die Beziehungen zwischen diesen. Technisch gesehen werden die Begrifflichkeit des Gesetzes und dessen Struktur durch ein entsprechendes Klassenmodell repräsentiert. Die Klassen des Domänenmodells werden von den anderen Schichten zum Datenaustausch genutzt. Das Domänenmodell wird in Abschnitt 9.5.8. im Detail beschrieben.

Der Code, der die eigentliche automatisierte rechtliche Beurteilung der Fakten durchführt, ist in der Service-Schicht angesiedelt. Bei dieser Schicht handelt es sich sozusagen um den Kern der Anwendung, in der die eigentliche automatisierte Beurteilung der Fakten erfolgt. Die Service-Schicht wird in Abschnitt 9.5.9. näher dargestellt.

Die unterste Ebene der Anwendung bildet die Repository-Schicht. Die Service-Schicht bedient sich der Repository-Schicht, die für die Speicherung und den Zugriff auf die Daten zuständig ist. Als Daten sind in diesem Fall vor allem die Geldbeträge in Euro zu verstehen, die das KOVG normiert. Für weitere Ausführungen bezüglich dieser Schicht sei auf Abschnitt 9.5.10. verwiesen.

Dependency Injection: Um das System flexibler zu machen und Tests zu erleichtern, wird das Spring Framework für die „dependency injection" verwendet. Der IoC-Container von Spring wird mit XML-Dateien konfiguriert und kümmert sich um die Erzeugung und Injektion der Objekte, wo diese benötigt werden (Walls, 2011, S. 30 ff.).

Dadurch wird der Grad der losen Koppelung der Software weiter erhöht und, Tests werden erleichtert, da sich konkrete Implementierungen durch bloße Änderung der Konfiguration austauschen lassen.

Externe Konfiguration: Die Software wird durch externe Konfigurationsdateien im XML-Format konfiguriert. Den Großteil machen dabei die Dateien des Spring Frameworks für die Konfiguration der Dependency Injection aus, die sich innerhalb des Projekts im Verzeichnis „spring" befinden und die Endung „-ctx" im Namen tragen.

Modultests: Die implementierten Klassen, insbesondere die, die für die Berechnung der Beschädigtenrente zuständig sind, werden mit Modultests überprüft, um ihre Funktionalität zu gewährleisten. Diese Tests befinden sich in einem eigenen Verzeichnis innerhalb des Projekts, getrennt vom produktiven Code.

9.5.7.1. Konventionen

Die Semantik der Syntax von Java sowie das Laufzeitverhalten der Ausdrücke sind durch die Sprachspezifikation von Java (Gosling et al., 2013) und die Spezifikation der virtuellen Maschine (Lindholm, Yellin, Bracha, & Buckley, 2013) definiert.

Dies umfasst jedoch nicht die Definition der Bedeutung der Objekte für das Programm selbst. Diese zusätzliche Semantik kann nicht formal im Programmcode repräsentiert

werden, sondern wird oftmals in der Inline-Dokumentation des Codes zum Ausdruck gebracht.

Repräsentation von Geldwerten und Leistungen: Im Zusammenhang mit Geldwerten, die ausschließlich mithilfe der Klasse `Euro` repräsentiert werden, ist auf die Unterscheidung zwischen „null“ und „0,00“ hinzuweisen. Die folgenden Ausführungen gelten für die gesamte API des Programms gleichermaßen. „Null“ in Java meint eine Null-Referenz, die nicht zu verwechseln ist mit dem numerischen Wert 0.

Um zwischen den Rechtspositionen „es besteht kein Anspruch dem Grunde nach“ und „der Anspruch besteht dem Grunde nach“ unterscheiden zu können, bedarf es einer informalen Spezifikation.

Der Wert „null“ bedeutet der Spezifikation zufolge „es besteht kein Anspruch dem Grunde nach“. Jede Variable vom Typ `Euro`, die mit einem Wert ungleich null belegt ist, bedeutet „der Anspruch besteht im Grunde nach“ in der Höhe des jeweiligen Betrags. So kann eine Leistung zwar grundsätzlich zustehen, aufgrund von allfälligen Kürzungen aber trotzdem € 0,00 betragen.

Die Verwendung von Klassen zur Repräsentation von Geldwerten bringt in diesem Fall einen Vorteil gegenüber primitiven Datentypen, die automatisch den Wert 0 (oder 0,0) zugewiesen bekommen und per Spezifikation niemals „null“ sein können.

Dieses Konzept wird analog auch auf sämtliche Variablen der Containerklassen Beschaedigtenrente, Grundrente und Zusatzrente angewendet. Ein Wert von „null“ bedeutet auch hier keinen Anspruch auf die jeweilige (Teil-)Leistung.

Repräsentation von kumulativen Leistungen: Vereinzelt wird eine Zulage etc. nicht bloß einmal gewährt, sondern unter Umständen auch kumulativ, wie im Beispiel der Wäschepauschale gemäß § 20a, wenn mehrere der Voraussetzungen nach Abs. 1 Zf. 1 bis 3 erfüllt sind.

Kann eine Zulage öfters gewährt werden, stellt sich die Frage, wie dies im Programm – auch im Sinne des Isomorphismus – repräsentiert werden soll?

Im Sinne der Transparenz und Nachvollziehbarkeit ist es nicht zielführend, die einzelnen Beträge zu summieren. Aus diesem Grund werden kumulativ zu gewährende Beträge durch ein Objekt vom Typ `List` in Java repräsentiert.

9.5.8. Das Domänenmodell

Das Domänenmodell befindet sich in der Domain-Schicht der Anwendung und repräsentiert die Begriffe der Problemdomäne sowie die Beziehungen zwischen diesen. Es wird verwendet, um die signifikanten Objekte einer Domäne darzustellen. Im konkreten Fall werden die Rechtsvorschriften des Gesetzes durch entsprechende Klassen repräsentiert.

Java bietet zur groben Strukturierung von Klassen das Konzept der „Packages“. Ein Package ist dabei nichts anderes als eine verschachtelte Verzeichnisstruktur innerhalb des Projekts. Die Klassen des Domänenmodells befinden sich im Package „at.js.diss.simple.domain“.

Zentraler Anknüpfungspunkt für die Modellierung der Rechtsvorschriften sind die im Text enthaltenen Nomen, wie beispielsweise Dienstbeschädigung (§ 1), Beschädigter (§ 2) und Beschädigtenrente (§ 7). Diese Begriffe werden durch entsprechende Klassen im Programm formalisiert. Die Klassen enthalten dabei die Attribute, die das Gesetz mit einem Begriff verknüpft, und so dessen Sinngehalt und Zweck ausdrücken. Im Fall der Klasse `Dienstbeschaedigung` ist dies beispielsweise das Attribut „MdE“, das die Minderung der Erwerbsfähigkeit ausdrückt.

Um eine möglichst direkte Übersetzung der Begriffe in ihre formale Repräsentation gemäß dem Prinzip des Isomorphismus zu erhalten, wird die Begrifflichkeit des Gesetzes für die Benennung der Klassen und ihrer Attribute verwendet.

Des Weiteren wird danach gestrebt, den Aufbau und die Struktur des Gesetzes möglichst getreu im Klassenmodell wiederzugeben. Die Klasse `Beschaedigtenrente` besitzt, der Terminologie von § 10 folgend, die Attribute „Grundrente“ und „Zusatzrente“

Welche Klassen und Attribute implementiert werden, hängt ausschließlich von deren Notwendigkeit für die automatisierte Vollziehung ab. Das Domänenmodell repräsentiert damit keinesfalls sämtliche Begriffe des Gesetzes.

Neben gewöhnlichen Klassen kennt Java seit Version 1.5 sogenannte „Enumerations“ (Enums). Mithilfe von Enums lassen sich in Java abgeschlossene, unveränderbare und typisierte Wertevorräte realisieren. Eine Enum repräsentiert damit eine abgeschlossene Menge von Konstanten. Vor deren Einführung wurden für solche Zwecke statische Klassenvariable verwendet, die aber gegenüber Enums eine Reihe von Nachteilen aufweisen (Bloch, 2008, S. 147 ff.). Enums können jedoch wie gewöhnliche Klassen auch Methoden

enthalten, die eine für die enthaltenen Konstanten spezifische Funktionalität implementieren.

Bei der Formalisierung des KOVG kommen Enums zum Einsatz, wenn das Gesetz Aufzählungen enthält, die formal gesprochen eine abgeschlossene Menge repräsentieren. Dies ist insbesondere bei der Pflegezulage gemäß § 18 der Fall, der eine Einordnung in die Pflegestufen I bis IV zugrunde liegt.

Es eignen sich nur statische Mengen zur Repräsentation durch Enums. Die Aufzählungen des Gesetzes, d.h. die Ziffern eines Absatzes etc., sind in diesem Fall relativ konstant und werden daher durch Enums repräsentiert.

Im Gegensatz zu den anderen Schichten besteht das Domänenmodell überwiegend aus konkreten Klassen ohne entsprechendes Interface. Dies liegt in der Tatsache begründet, dass es sich bei diesen Klassen um bloße Datencontainer handelt, die nicht von einer konkreten Implementierung abhängig sind. Es ist schwer vorstellbar, dass es einen sinnvollen Anwendungsfall gibt, in dem solch eine Klasse durch eine andere Implementierung ausgetauscht werden müsste.

Nach dieser Einleitung in das Design des Domänenmodells und die dahinter liegenden Prinzipien sollen die wichtigsten Klassen nun dargestellt werden. Wie bereits erwähnt befinden sich die Klassen des Domänenmodells im Package „at.js.simple.diss.domain“.

Euro: In Geldwerten bestehende Versorgungsleistungen sind ein wesentlicher Teil des KOVG. Java bietet jedoch keinen primitiven Festkommadatentyp an, der sich für die Berechnung von monetären Werten eignen würde. Explizit sei darauf hingewiesen, dass die primitiven Gleitkommadatentypen `float` und `double` für die Berechnung von Geldwerten *ungeeignet* sind.

Java bietet für Festkommaoperationen die Klasse `BigDecimal` mit beliebiger Präzision an. Für korrekte Berechnungen mit Geldbeträgen ist diese Klasse unbedingt zu bevorzugen (Bloch, 2008, S. 218 ff.). Berechnungen sind mit Objekten dieser Klasse langsamer als mit primitiven Datentypen, die Geschwindigkeit ist jedoch in der Regel für den Zweck der automatisierten Vollziehung ausreichend.

Die Klasse `BigDecimal` repräsentiert eine beliebige Festkommazahl und nicht unbedingt einen monetären Betrag. Aus diesem Grund wurde die Klasse Euro implementiert, die intern

mit einem `BigDecimal` mit der Präzision zwei (für zwei Nachkommastellen) arbeitet und zusätzlich diverse Methoden zur Unterstützung der Berechnung anbietet. Damit wird gewährleistet, dass die Semantik des Gesetzes – nämlich Geldbeträge in Euro – direkt im Domänenmodell repräsentiert werden.

Die Klasse führt automatisch eine kaufmännische Rundung von Berechnungen auf zwei Nachkommastellen durch. Falls notwendig kann die Klasse auch mit der Methode `applyTenCentRounding` in den Rundungsmodus auf volle zehn Cent umgeschaltet werden, wie dies in den §§ 63 Abs. 3 und 67 vorgeschrieben wird. Gemäß § 67 sind die Versorgungsleistungen nach dem KOVG auf volle zehn Cent zu runden.

Beschädigtenrente: Die Klasse `Beschaedigtenrente` ist ein Container, der die Beschädigtenrente samt allfälligen Zulagen repräsentiert. Die Klasse repräsentiert entsprechend der in Abschnitt 9.3.2. eingeführten Terminologie die Beschädigtenrente im weiteren Sinn. Die Klasse `Beschaedigtenrente` geht demnach über die Definition des § 10 hinaus, da sie auch die diversen Zulagen zur Beschädigtenrente als Attribute enthält. Diese Designentscheidung wurde aufgrund der Bündelung sämtlicher Teilleistungen der Beschädigtenrente getroffen. Ohne diese hätte eine weitere Klasse mit eigenem Namen implementiert werden müssen, die als Klammer um die Beschädigtenrente und die Zulagen dient. Die Bezeichnung dieser Klasse hätte jedoch ebenfalls keine Deckung in der gesetzlichen Grundlage und damit keinen Mehrwert im Hinblick auf den anzustrebenden Isomorphismus.

Grundrente: Die Klasse `Grundrente` repräsentiert die Grundrente gemäß § 11 als Teilleistung der Beschädigtenrente samt allfälligen Erhöhungen. Zur Grundrente selbst gebühren keine Zulagen.

Zusatzrente: Die Klasse `Zusatzrente` repräsentiert die Zusatzrente gemäß § 12 samt allfälligen Zulagen. Demnach repräsentiert sie die Zusatzrente im weiteren Sinn entsprechend der Terminologie des Abschnitt 9.3.2. Die Konsolidierung der Zusatzrente und der Zulagen erfolgt aus denselben Gründen wie bei der Modellierung der Klasse `Beschaedigtenrente`.

Beschaedigter: Die Klasse `Beschaedigter` kapselt sämtliche Fakten, die für die rechtliche Beurteilung notwendig sind. Wie aus den Attributen der Klasse ersichtlich, gehören zu diesen Attributen unter anderem die Dienstbeschädigungen, das Einkommen, die Anzahl der

Familienangehörigen etc. Für die Repräsentation der Anspruchsgrundlage für diverse Zulagen enthält die Klasse eine Reihe von Enums wie beispielsweise die Pflegestufe.

Pflegestufe: Die Pflegestufe gemäß § 18 wird durch eine gleichnamige Enumeration repräsentiert, die die Konstanten I, II, III, IV und V sowie eine fiktive Stufe namens V_PLUS enthält. Die Stufen I bis V sind dabei explizit im Gesetzestext enthalten. Bei der Konstante V_PLUS handelt es sich um eine fiktive Stufe, die die erhöhte Pflegestufe V gemäß § 18 Abs. 5 repräsentiert. Diese Bezeichnung findet im Gesetz freilich keine Deckung und ist eine rein programmtechnische Schöpfung.

Diätverpflegung: Gemäß § 14 ist Schwerbeschädigten für eine verordnete Diätverpflegung ein Zuschuss zu gewähren, wenn diese wegen einer der aufgezählten Krankheiten erforderlich ist. Das Gesetz spricht hier nicht explizit von Stufen oder dergleichen. Die einzelnen Krankheiten sind lediglich in den Ziffern 1 bis 5 des 1. Absatzes aufgezählt. Ein Sachbearbeiter beurteilt im Vorfeld, ob und welche Diätverpflegung dem Schwerbeschädigten zusteht. Diese Beurteilung wird abstrakt in Form der Enum `Diaetverpflegung` repräsentiert. Die Ziffern 1 bis 5 des Gesetzes entsprechen den Konstanten I bis V. Bei dieser Enum handelt es sich also um eine fiktive Schöpfung, die rein dem Zweck der Repräsentation der Fakten dient.

Kleiderverbrauch: Ähnlich der Enum `Diaetverpflegung` dient die Enum `Kleiderverbrauch` zur Repräsentation des Anspruchs auf Kleiderpauschale gemäß § 20a. Die Ziffern des Gesetzes entsprechen den Konstanten I bis III der Enumeration.

9.5.9. Die Implementierung der automatisierten Vollziehung

In diesem Abschnitt wird auf die Implementierung der Berechnung der Beschädigtenrente in Java eingegangen. Zentrale Schnittstelle ist das Interface `CalculationEngine` und dessen Implementierung `JavaCalculationEngine`. Da es sich bei den Regeln zur automatisierten Vollziehung um „Business Logic“ handelt, befinden sich sämtliche Klassen und Interfaces in der Service-Schicht im Package „at.js.diss.simple.service“.

9.5.9.1. Das Interface CalculationEngine

Das Interface `CalculationEngine` definiert formal die Schnittstelle für die Durchführung der monatlichen Berechnung der Beschädigtenrente. Wie für eine Abstraktion in der Programmierung üblich, ist das Interface nicht an eine konkrete Implementierung gebunden.

Die im nächsten Abschnitt beschriebene `JavaCalculationEngine` ist beispielsweise eine rein in Java geschriebene Implementierung. Es könnte jedoch eine beliebige Technologie – zum Beispiel eine „Rule Engine“ wie Drools Expert von JBoss – zur Implementierung der Normen verwendet werden.

Das Interface CalculationEngine ist recht überschaubar und beinhaltet nur die Methode „calculate“:

```
public Beschaedigtenrente calculate(Beschaedigter beschaedigter,
DateTime now);
```

Die Methode „calculate“ liefert ein Objekt der Klasse `Beschaedigtenrente` zurück, das die für den jeweiligen Monat zustehende Beschädigtenrente repräsentiert.

Der Parameter „beschaedigter“ repräsentiert die Fakten des aktuellen Falles. Zu diesen gehören unter anderem das Einkommen und die Dienstbeschädigungen samt der Bewertung der Minderung der Erwerbsfähigkeit. Das Objekt vom Typ `Beschaedigter` repräsentiert eine abstrakte Beschreibung des Sachverhalts, die auf die für die Berechnung relevanten Fakten reduziert wurde.

Der Parameter „now“ repräsentiert das gewünschte Datum der Berechnung. Für die Berechnung sind bloß das Jahr und das Monat relevant, nicht jedoch der Tag des Monats oder die Uhrzeit.

Die Schnittstellendefinition macht deutlich, dass bestimmte Bewertungen und Berechnungen vorausgesetzt werden. Diese können entweder von einem menschlichen Sachbearbeiter oder einem anderen Programmmodul, das beispielsweise das Einkommen berechnet, zur Verfügung gestellt werden.

Das Design der Schnittstelle ist ein gutes Beispiel für die Anwendung des Separation-of-Concerns-Prinzips. Die `CalculationEngine` soll sich schon ihrem Namen nach ausschließlich um die Berechnung der Beschädigtenrente kümmern. Die rechtliche Beurteilung, ob eine Pflegezulage zusteht oder nicht, und die Berechnung des Einkommens sind Beispiele für Belange, die nichts mit dem eigentlichen Zweck der Schnittstelle zu tun haben. Würde man diese Angelegenheiten ebenfalls in die Zuständigkeit der `CalculationEngine` verlagern, so würden deren Schnittstelle und vor allem deren Implementierung unnötig verkompliziert. Die Berechnung des Einkommens kann

beispielsweise ohne Probleme von einer speziellen vorgelagerten Komponente durchgeführt werden, die unter Umständen die Einkommensdaten aus einem externen Register abfragt.

Schwierige (rechtliche) Beurteilungen können im Vorfeld von einem Fachexperten – gegebenenfalls mit einer entsprechenden Unterstützung durch die Software – durchgeführt werden. Durch die Einbindung des Menschen als Problemlöser können Hürden für die Software überwunden und auch komplexe Problemstellungen der semi-automatisierten Vollziehung zugänglich gemacht werden.

Es gilt stets die Frage zu beantworten, wie weit man formalisieren möchte? Die äußerste Grenze des technisch Machbaren deckt sich nicht immer mit dem, was noch sinnvoll mithilfe der Software formalisiert und automatisiert werden kann. Bei der Beantwortung dieser Frage stellen sich oftmals Effizienzüberlegungen – eine Abwägung der Kosten und des Nutzens.

9.5.9.2. Die JavaCalculationEngine

Wie der Name „JavaCalculationEngine“ andeutet, handelt es sich um eine Implementierung, die ausschließlich in Java geschrieben ist und keine externen Bibliotheken verwendet. Sämtliche Regeln, die die Normen repräsentieren, sind im Code direkt implementiert.

Die JavaCalculationEngine bedient sich der DataRepository-Abstraktion zum Lesen sämtlicher Geldbeträge, die aus dem Gesetz extrahiert wurden.

Für die Implementierung von Algorithmen gibt es in der Softwareentwicklung das sogenannte Template-Method-Entwurfsmuster (Freeman & Freeman, 2004, S. 275 ff.). Die Berechnung der Beschädigtenrente als eine Form der automatisierten Vollziehung kann als Algorithmus betrachtet werden.

Die Implementierung der weiter oben bereits erwähnten Methode „calculate“ erfolgt daher in Form einer solchen Template-Methode wie Abbildung 11 zeigt.

```
82          // 1 Grundrente
83          Grundrente grundrente = calculateGrundrente(beschaedigter, now);
84          rente.setGrundrente(grundrente);
85          logger.info("§ 11: Grundrente: {}", grundrente);
86
87          if (grundrente == null) {
88              logger.info("Kein Anspruch auf Grundrente und somit auf Beschädigtenrente.");
89              return null;
90          }
91
92          // 2 Zusatzrente
93          Zusatzrente zusatzrente = calculateZusatzrente(beschaedigter, now);
94          rente.setZusatzrente(zusatzrente);
95          logger.info("Zusatzrente: {}", zusatzrente);
96
97          // 3 Zulagen zur Beschädigtenrente
98          // 3.1 Schwerstbeschädigtenzulage
99          Euro schwerstbeschaedigtenzulage = calculateSchwerstbeschaedigtenzulage(beschaedigter);
100         rente.setSchwerstbeschaedigtenzulage(schwerstbeschaedigtenzulage);
101         logger.info("§ 11a: Schwerstbeschädigtenzulage: {}",
102                 schwerstbeschaedigtenzulage);
```

Abbildung 11: Ausschnitt der Template-Methode "calculate".

Wie aus Abbildung 11 hervorgeht, definiert eine Template-Methode die einzelnen Schritte des Algorithmus. Die konkrete Implementierung jedes Schritts erfolgt in eigenen Methoden, die möglicherweise selbst wiederum nach diesem Entwurfsmuster gestaltet sind.

Das Codefragment zeigt recht anschaulich, dass die Berechnung der Grund- und Zusatzrente an entsprechende Methoden delegiert wird. Es gilt dabei grundsätzlich das Prinzip, dass die einzelnen Methoden sämtliche Prämissen, die für die Gewährung einer Leistung erfüllt sein müssen, selbstständig prüfen.

Die Methode „calculate“ sammelt die Ergebnisse der Berechnung und führt grundsätzliche Anspruchsprüfungen durch. Besteht beispielsweise kein Anspruch auf Grundrente (`grundrente == null`), wird die weitere Berechnung abgebrochen. Die einzelnen Teilleistungen werden in der Variable „rente“ vom Typ Beschaedigtenrente gekapselt.

Aus logischer Sicht wird die Prüfung der Prämisse zur Gewährung der anderen Teilleistungen, dass dem Beschädigten eine Grundrente zustehen muss, vorweggenommen. Diese Bedingung wird daher bei der Berechnung der anderen Teilleistungen implizit als wahr angenommen und nicht mehr gesondert überprüft.

Diese technisch bedingte Vereinfachung steht in einem gewissen Spannungsfeld zum Prinzip des Isomorphismus. § 12 spricht beispielsweise von der Gewährung der Zusatzrente „zur Grundrente“. Streng genommen ist daher der Anspruch auf Grundrente eine Prämisse der Regel zur Berechnung der Beschädigtenrente.

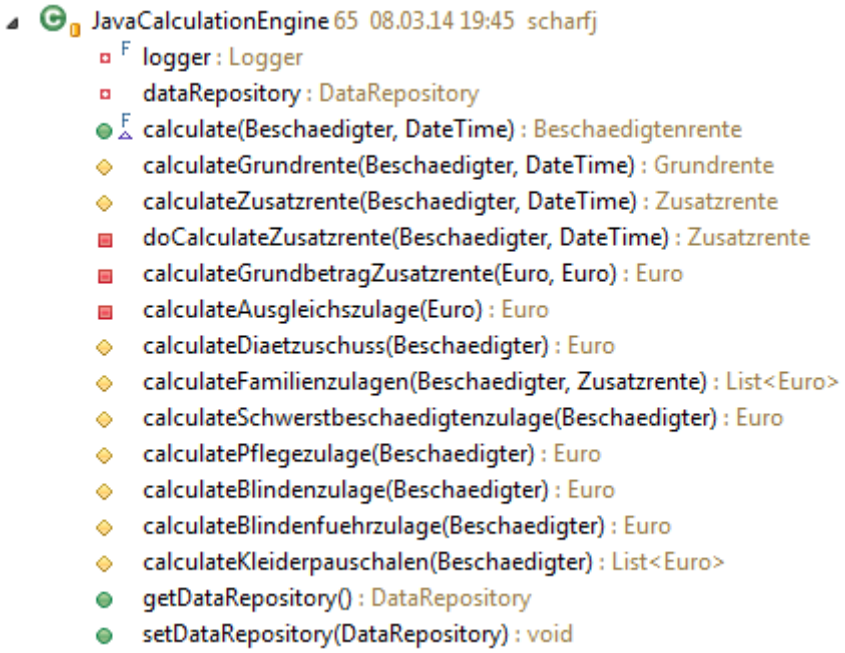

Abbildung 12: Methoden der JavaCalculationEngine

Abbildung 12 zeigt die Methoden der Klasse JavaCalculationEngine im Überblick. Gut erkennbar ist, dass für jede Teilleistung die Berechnung in einer eigenen Methode erfolgt. Daneben existieren noch private „Hilfsmethoden“ wie „calculateGrundbetragZusatzrente“, die Teile der Berechnung durchführen.

Diese Auslagerung von Teilschritten der Berechnung ist oftmals notwendig, da andere Leistungen exakt von diesem Teilbetrag einer Leistung abhängig sind. Der Grundbetrag der Zusatzrente muss beispielsweise, um die Kürzung der Familienzulagen bestimmen zu können, mit einer fiktiven Einkommensgrenze berechnet werden. Aus diesem Grund war eine Methode mit entsprechender Signatur vorzusehen.

9.5.9.3. Die Modellierung von Normen in Java

Obwohl es grundsätzlich verschiedene Arten von Normen gibt, wird für die Zwecke der Formalisierung in weitgehender Übereinstimmung mit der Rechtstheorie angenommen, dass Normen grundsätzlich folgende konditionale Struktur haben: wenn A_1,…,A_n dann B. Die Normen werden in diesem Fall in schriftlicher Form durch den Gesetzestext zum Ausdruck gebracht.

Eine *Regel* ist in diesem Kontext eine logische Formalisierung des Rechtstexts in der Form von Prämisse und Konklusion.

Die Programmiersprache Java selbst ist nicht regelbasiert. Das Konzept der „Regel“, bestehend aus „Body“, der die Bedingungen, und „Head“, der die Konklusion (die Rechtsfolgen) repräsentiert, existiert nicht.

In Java werden logische Prüfungen mithilfe von if-Bedingungen mit folgender Struktur durchgeführt:

```
if(Bedingung == true) {
     // Anweisung
}
else {
     // Optionale Anweisung wenn Bedingung == false
}
```

Zusätzlich gibt es Verzweigungen in Form von „else if“ Bedingungen, die als alternative Verzweigungen gesehen werden können. Es kommt dabei höchstens eine Anweisung zur Ausführung:

```
if(Bedingung1 == true) {
     // Anweisung 1
}
else if(Bedingung 2 == true) {
     // Anweisung 2; kommt nur zur Anwendung wenn nicht bereits
Bedingung1 == true
}
```

Die Bedingung kann in Java ein beliebiger boolescher Ausdruck sein, wobei eine Bedingung durch logische Operatoren verknüpft werden kann. In Java existieren die logischen Operatoren NOT (!), AND (&&), OR (||) und XOR (^). Mithilfe dieser booleschen Operatoren lassen sich auch sehr komplexe Bedingungen formulieren, falls dies erforderlich ist.

Für die Formalisierung der Beschädigtenrente soll grundsätzlich der Tatbestand einer Norm mithilfe einer entsprechenden if-Bedingung formalisiert werden. Wenn im Folgenden daher von einer Regel gesprochen wird, meint dies eine if-Bedingung mit ihrem Bedingungsteil („Body“) und den zugehörigen Anweisungen („Head“).

9.5.9.4. Die Formalisierung der Grundrente gemäß § 11 KOVG

In den folgenden Abschnitten wird die Implementierung von ausgewählten Teilleistungen näher dargestellt. Anhand dieser Ausführungen werden die Möglichkeiten zur Repräsentation von Normen diskutiert.

In diesem Abschnitt wird die Formalisierung der Grundrente gemäß § 11, gegliedert nach den Absätzen eins bis drei, diskutiert. Die Berechnung der Grundrente ist in der Methode „calculateGrundrente“ implementiert.

```
146         int mde = beschaedigter.getMaxMdE();
147
148         if (mde < 20) {
149             // Kein Anspruch
150             logger.debug("§ 11: MdE {} < 20 - kein Anspruch auf Grundrente.",
151                     mde);
152
153             return null;
154         }
155
156         Grundrente rente = new Grundrente();
157
158         // 1 Ermittlung Grundbetrag gemäß § 11 Abs. 1 KOVG
159         Euro grundbetrag = dataRepository.getGrundbetragGrundrente(mde);
160         rente.setGrundbetrag(grundbetrag);
161         logger.debug("§ 11 Abs. 1: Grundbetrag der Grundrente für MdE {}: {}",
162                 mde, grundbetrag);
```

Abbildung 13: Berechnung des Grundbetrags der Grundrente

Abbildung 13 zeigt die Formalisierung von § 11 Abs. 1. Die Zeilen 148 bis 154 können als folgendes logisches Argument formuliert werden: „Wenn MdE < 20, dann kein Anspruch auf Grundrente“. Die Prämisse „MdE < 20“ ist als ausschließende Bedingung formuliert und die logische Umkehrung der Prämisse „MdE >= 20“, die in § 11 Abs. 1 implizit enthalten ist. Unter einer MdE von 20 besteht demnach kein Anspruch auf Grundrente. Die Ausführung der Methode ist mit „return null“ beendet, sofern die Bedingung „MdE < 20“ erfüllt ist.

Die Prüfung, ob grundsätzlich eine Grundrente zusteht, wird daher für die Erhöhungen nach Abs. 2 und 3 nicht mehr explizit geprüft, sondern implizit als wahr angenommen.

Angemerkt sei, dass die Prüfung MdE >= 20 sich mit der in § 7 zum Ausdruck gebrachten Bedingung für die Gewährung einer Beschädigtenrente an sich deckt. Die grundlegende Prüfung gemäß § 7 wird daher implizit im Kontext der formalen Repräsentation des § 11 durchgeführt.

Eine explizite Formalisierung von § 7 könnte in der „calculate“-Methode durch die if-Bedingung „MdE >= 20“ erfolgen. Da die Erfüllung der Bedingung aber auch implizit eine Voraussetzung für die Gewährung einer Grundrente ist und daher ohnehin geprüft werden muss, wurde aus Gründen der Redundanz auf eine explizite Formalisierung des § 7 verzichtet.

Nach der Prüfung der grundlegenden Bedingung „MdE >= 20“ erfolgt die Ermittlung des eigentlichen Betrags in Abhängigkeit von der Minderung der Erwerbsfähigkeit (Zeile 159). Die in 10er-Schritten abgestufte MdE wird dabei auf den nach § 11 Abs. 1 berechneten Hundertsatz von € 511 abgebildet. Diese Funktionalität ist im `DataRepository` implementiert, an das in Zeile 159 die MdE einfach übergeben wird.

Betrachtet man § 11 Abs. 1 vom logischen Standpunkt aus, so erkennt man, dass sich darin mehrere Regeln verbergen: „Wenn MdE >= 90, dann beträgt die Grundrente € 511“, „Wenn MdE >= 20 und < 30, dann beträgt die Grundrente 10 % von € 511“, „Wenn MdE >= 30 und < 40, dann beträgt die Grundrente 20 % von € 511“, usw.

Wir erkennen also, dass die Norm des Abs. 1 als Bedingung mehrere sich gegenseitig ausschließende, halboffene (genauer rechtsoffene) Intervalle $[20{,}30)$, $[30{,}40)$, usw. definiert. Die Berechnung des Grundbetrags der Grundrente lässt sich in Java relativ elegant mithilfe einer `Map` implementieren – siehe Zeile 61 in Abbildung 14. Eine `Map` ist eine Datenstruktur, die Schlüssel-Werte-Paare enthält. Die linke Grenze jedes Intervalls – also 20, 30 etc. – dient dabei als Schlüssel für den jeweiligen Betrag in Euro.

```
59    @Override
60    public Euro getGrundbetragGrundrente(int mde) {
61        Euro betrag = mdeToGrundrente.get(mde);
62
63        if (betrag == null) {
64            throw new IllegalArgumentException("Konnte für MdE [" + mde
65                    + "] keine Grundrente ermitteln.");
66        }
67
68        return betrag;
69    }
```

Abbildung 14: Mapping MdE auf Grundrente in der Klasse InMemoryDataRepository

Diese aus technischer Sicht recht elegante Lösung ist unter dem Aspekt des Isomorphismus differenziert zu beurteilen. Die Regeln des § 11 Abs. 1 können von einem juristischen Fachexperten eher schwer im gezeigten Codefragment gefunden werden.

Kommen wir nun zur Formalisierung der Erhöhung der Grundrente nach den Absätzen eins und zwei des § 11 KOVG.

Systematisch ist in Absatz zwei und drei eine Sonderbestimmung zur Grundregel des Absatzes eins vorgesehen, die unter gewissen Umständen eine Erhöhung der Grundrente vorsieht. Absatz drei ist dabei eine Ausnahmebestimmung zu Absatz zwei und geht

Letzterem vor. Abbildung 15 zeigt die Implementierung der Erhöhung der Grundrente in Java.

```
177         Euro erhoehung = ZERO;
178
179         // 2 Altersabhängige Erhöhung, § 11 Abs. 2 KOVG
180         if ((beschaedigter.getGeschlecht() == MAENNLICH && alter >= 60)
181                 || (beschaedigter.getGeschlecht() == WEIBLICH && alter >= 55)) {
182             erhoehung = of("20.90");
183
184             logger.debug(
185                     "§ 11 Abs. 2: Erhöhung für Geschlecht: {}, Alter: {}, Betrag: {}",
186                     beschaedigter.getGeschlecht(), alter, erhoehung);
187         }
188
189         // 3 Altersabhängige Erhöhung, § 11 Abs. 3 KOVG
190         if (alter >= 65) {
191             erhoehung = dataRepository.getErhoehungGrundrente(65, mde);
192
193             logger.debug("§ 11 Abs. 3: Alter >= 65: {}", erhoehung);
194         }
195         if (alter >= 70) {
196             erhoehung = dataRepository.getErhoehungGrundrente(70, mde);
197
198             logger.debug("§ 11 Abs. 3: Alter >= 70: {}", erhoehung);
199         }
```

Abbildung 15: Erhöhung der Grundrente gemäß § 11 Abs. 2 und 3

Der Betrag für die Erhöhung wird mit € 0,00 initialisiert (Zeile 177). Dies hat den Vorteil, dass der Betrag ergebnisneutral zum Grundbetrag der Grundrente addiert werden kann, sollte keine Erhöhung nach Absatz zwei oder drei gebühren.

Als logisches Argument formuliert, stellt sich § 11 Abs. 2 wie folgt dar (die Syntax ist angelehnt an Bench-Capon und Gordon (2009)):

```
Regel 1a.
     Die Grundrente ist um € 20,90 zu erhöhen
wenn folgende Bedingungen erfüllt sind:
     - Der Beschädigte hat Anspruch auf Grundrente
     - MdE >= 50
     - (Geschlecht ist männlich und Alter zumindest 60) oder
(Geschlecht ist weiblich und Alter zumindest 55)
```

Die formale Repräsentation des § 11 Abs. 2 ist in den Zeilen 180 bis 187 ersichtlich. Die Prämisse, ob der Beschädigte überhaupt Anspruch auf Grundrente hat, ist in diesem Codefragment nicht enthalten. Diese wird in den Absätzen eins und zwei des Gesetzes implizit zum Ausdruck gebracht. Die Prüfung, ob überhaupt eine Grundrente zusteht, wird bereits im Kontext der Formalisierung des Grundbetrags nach Abs. 1 vorweggenommen und muss daher an dieser Stelle nicht noch einmal vorgenommen werden.

Ähnlich verhält es sich mit der Prüfung der zweiten Prämisse, ob es sich um einen Schwerbeschädigten handelt (MdE >= 50). Diese ist im Programm an einer Stelle für Abs. 2 und 3 implementiert und nicht in Abbildung 15 ersichtlich.

Die dritte Prämisse ist im Code hingegen gut erkennbar. Der Beschädigte muss männlich und zumindest 60 Jahre alt sein oder es handelt sich um eine Beschädigte, die zumindest 55 Jahre alt sein muss. Die Konklusion ist die Erhöhung der Grundrente um € 20,90 (Zeile 182).

§ 11 Abs. 3 enthält explizit eine Ausnahmebestimmung zu Abs. 2. Aus rechtstechnischer Sicht ist daher Abs. 2 nicht anzuwenden, falls der Tatbestand des Abs. 3 erfüllt ist. Abs. 2 ist daher eine „anfechtbare“ („defeasible“) Bestimmung. Dies bedeutet, dass auch wenn deren Prämissen wahr sind, nicht unbedingt auch deren Konklusion zur Anwendung gelangt.

Im Gegensatz zu Abs. 2 ist für eine Erhöhung nach Abs. 3 das Geschlecht des Beschädigten unerheblich. Die Norm knüpft lediglich an das Alter und die MdE des Beschädigten an, wobei die Erhöhung abgestuft nach Alter und MdE gewährt wird.

Abs. 3 enthält aus logischer Sicht mehrere Argumente mit folgender Struktur:

```
Regel 1b.
     Die Grundrente ist um € 22,90 zu erhöhen
wenn folgende Bedingungen erfüllt sind:
     - Der Beschädigte hat Anspruch auf Grundrente
     - MdE >= 50
     - Alter >= 65
     - Alter < 70
```

Die Implementierung dieser Argumente ist in den Zeilen 190 ff. ersichtlich. Der Kürze wegen sind nur zwei Erhöhungen nach Abs. 3 – Alter >= 65 und Alter >= 70 – dargestellt. Wie aus dem Code hervorgeht, wird lediglich die Prämisse zur Prüfung des Alters implementiert. Die Prüfung MdE >= 50 ist, wie bereits erwähnt, nicht im Codefragment enthalten und wird implizit an dieser Stelle als wahr angenommen. Die Prämissen bezüglich des Alters sind wie schon bei der Formalisierung des Grundbetrags der Grundrente nach Abs. 1 als einfaches Mapping im `DataRepository` implementiert.

Aus Sicht der Formalisierung ist die für die Implementierung des Abs. 3 als Ausnahmebestimmung zu Abs. 2 angewendete Methode interessant.

Wie aus den Zeilen 190 ff. in Abbildung 15 hervorgeht, wird der scheinbare Konflikt zwischen Abs. 2 und 3 gelöst, indem die Konklusion der formalen Regel(n), die Abs. 3 repräsentieren, die Konklusion des Abs. 2 überschreiben (siehe Zeilen 191 und 196).

Es wird jedoch nicht verhindert, dass die Konklusion des Abs. 2 überhaupt zur Anwendung gelangt. Das Programm simuliert gewissermaßen bloß die dem Recht immanente „Anfechtbarkeit" von Normen, durch entsprechende Ausnahmebestimmungen. Abs. 3 wird als Regel höherer Priorität formalisiert, indem dessen Regeln im Code nach der Formalisierung des Abs. 2 implementiert werden. Dieser Ansatz wird auch für die Formalisierung des Abs. 3 selbst angewendet, der – wie bereits weiter oben ausgeführt – mehrere logische Argumente enthält.

Damit dieser Ansatz funktioniert, muss die Reihenfolge der Regeln, die deren Priorität ausdrückt, aus der Struktur des Gesetzes abgeleitet werden. Im gegenständlichen Fall ist dies einfach möglich, da Abs. 3 explizit als Ausnahme von Abs. 2 formuliert ist.

Im Fall von „lex specialis" und „lex posterior" wäre diese Reihenfolge lediglich implizit vorhanden. Eine solche implizite Rangordnung lässt sich auch zwischen den aus Abs. 3 abgeleiteten formalen Regeln feststellen. Dabei handelt es sich jedoch streng genommen nicht um Fälle der „lex specialis".

Angemerkt sei, dass dieser Ansatz die „Anfechtbarkeit" von Rechtsnormen bloß simuliert. Eine „richtige" Implementierung wäre zurückhaltender und würde die Konklusion einer „anfechtbaren" Regel nicht ausführen, bis der Konflikt mit der „anfechtenden" Regel durch die Anwendung entsprechender Mechanismen („Metaregeln") gelöst ist (Athan et al., 2013b, S. 6). Eine solche Implementierung würde wohl eher dem Vorgehen eines menschlichen Rechtsanwenders entsprechen.

Dieser Ansatz entspricht möglicherweise dem Prinzip des Isomorphismus am besten, da grundsätzlich jeder Paragraf als eigene Regel formalisiert wird. Wie jedoch gezeigt wurde, können aus einem Paragraf und sogar dessen einzelnen Absätzen mehrere logische Regeln abgeleitet werden, wodurch eine klare Zuordnung je nach gewählter Implementierung unter Umständen erschwert wird.

9.5.9.5. Die Formalisierung der Pflegezulage gemäß § 18 KOVG

Im vorherigen Abschnitt wurde gezeigt, wie Konflikte zwischen Regeln mithilfe von Prioritäten gelöst werden können. An dieser Stelle wird ein alternativer Ansatz zur Lösung dieses Problems anhand der Pflegezulage gemäß § 19 präsentiert.

Der Pflegebedarf des Beschädigten wird in die Stufen I bis V eingestuft und dementsprechend eine entsprechende Pflegezulage gewährt. Aus logischer Sicht zerfällt § 18 wiederum in mehrere Regeln, wobei eine Regel für die Formalisierung einer „Pflegestufe“ notwendig ist.

Als logisches Argument formuliert stellt sich die Pflegezulage der Stufe I wie folgt dar:

```
Regel 2a.
     Die Pflegezulage ist in der Höhe von € 671,60 zu gewähren
wenn folgende Bedingungen erfüllt sind:
     - Der Beschädigte erhält eine Beschädigtenrente
     - Der Beschädigte ist nicht blind
     - Der Pflegebedarf entspricht Stufe I
```

Abbildung 16 zeigt die Implementierung der Pflegezulage gemäß § 18. Die grundlegende Prämisse, dass der Beschädigte überhaupt eine Beschädigtenrente erhält, wurde der Vollständigkeit halber in das logische Argument aufgenommen. Die Prüfung, ob eine Grundrente und damit eine Beschädigtenrente zusteht, erfolgt jedoch an zentraler Stelle in der Methode „calculate“ (siehe Abbildung 11).

```
607         if (beschaedigter.isBlind()) {
608             logger.debug("§ 19: Beschädigter ist blind - kein Anspruch auf Pflegezulage.");
609             return null;
610         }
611
612         Pflegestufe stufe = beschaedigter.getPflegestufe();
613
614         if (stufe == null) {
615             logger.debug("§ 18: Kein Pflegebedarf - kein Anspruch auf Pflegezulage.");
616             return null;
617         }
618
619         Euro zulage = dataRepository.getPflegezulage(stufe);
```

Abbildung 16: Implementierung der Pflegezulage gemäß § 18

Aus Sicht der Formalisierung ist die Prüfung der zweiten Prämisse am interessantesten. In Zeile 607 erfolgt die Prüfung, ob der Beschädigte blind ist. Wenn diese Bedingung wahr ist, wird die Berechnung der Pflegezulage abgebrochen. In diesem Fall wurde also die Ausnahmebestimmung des § 19 als Prämisse in die Formalisierung des § 18 übernommen.

Dieser Ansatz stellt eine Alternative zur Auflösung von Konflikten zwischen Normen durch Priorisierung von Regeln dar, wie sie in Abschnitt 9.5.9.4. präsentiert wurde. Der Nachteil ist, dass dem Prinzip des Isomorphismus nicht zur Gänze Genüge getan wird, da verschiedene Normen in einer Regel formalisiert werden. Trotzdem ist diese Repräsentation relativ kompakt und übersichtlich.

9.5.9.6. Die Formalisierung der Familienzulage(n) gemäß § 16 KOVG

Die Formalisierung der Familienzulage gemäß § 16 KOVG soll als Beispiel für die Implementierung einer Leistung dienen, deren Berechnung abhängig ist von der Höhe einer anderen Leistung.

Beginnen wir zunächst wieder mit der Formulierung des grundsätzlichen Anspruchs auf Familienzulagen als logisches Argument:

```
Regel 3a.
     Dem Schwerbeschädigten gebührt eine Familienzulage in der Höhe
von € 81,20 für jeden Familienangehörigen
wenn folgende Bedingungen erfüllt sind:
     - MdE >= 50
     - Anzahl der Angehörigen > 0
```

Abbildung 17 zeigt die Berechnung der Familienzulagen. Auf die Darstellung der Prüfung der Prämissen wird aufgrund deren Trivialität an dieser Stelle verzichtet. Der Gesamtbetrag für die Familienzulagen errechnet sich aus der Anzahl der Angehörigen, multipliziert mit dem Betrag von € 81,20, der im Code durch die Konstante „FAMILIENZULAGE“ repräsentiert wird.

Da jede Familienzulage separat repräsentiert werden soll, wird die Berechnung nicht mit einer einfachen Multiplikation durchgeführt, sondern für jeden Angehörigen wird eine Familienzulage einzeln hinzugefügt (Zeilen 422 bis 424).

```
419         List<Euro> familienzulagen = new ArrayList<>(angehoerige);
420
421         // 1. Berechnung Familienzulagen ohne Kürzung
422         for (int i = 0; i < angehoerige; i++) {
423             familienzulagen.add(FAMILIENZULAGE);
424         }
```

Abbildung 17: Berechnung der Familienzulagen ohne Kürzung

Eine allfällige Kürzung der Familienzulage gemäß § 16 Abs. 1, aufgrund einer Erhöhung der Zusatzrente durch den Anspruch auf Familienzulage, lässt sich als logisches Argument wie folgt formulieren:

Regel 3b.

Die Familienzulage ist um jenen Betrag zu kürzen, um den die Zusatzrente wegen des Anspruchs auf die Familienzulage erhöht wird wenn

die Zusatzrente aufgrund des Anspruchs auf Familienzulage erhöht wird

```
422         // 2. Berechnung der Erhöhung der Zusatzrente
423         // 2.1 Berechnung Zusatzrente ohne Erhöhung der Einkommensgrenze durch
424         // Familienzulagen
425         Euro einkommensgrenze = calculateStandardEinkommensgrenze();
426         Euro einkommen = beschaedigter.getEinkommen();
427         Euro zrOhneErhoehung = calculateGrundbetragZusatzrente(
428                 einkommensgrenze, einkommen);
429
430         logger.debug(
431                 "§ 16: Zusatzrente ohne Erhöhung der Einkommensgrenze: {}",
432                 zrOhneErhoehung);
433
434         // 2.2 Berechnung Erhöhung Zusatzrente wegen Erhöhung der
435         // Einkommensgrenze
436         // aufgrund von Familienzulagen
437         Euro zrMitErhoehung = zusatzrente.getGrundbetrag();
438         Euro erhoehungZusatzrente = zrMitErhoehung.minus(zrOhneErhoehung);
```

Abbildung 18: Berechnung der Erhöhung der Zusatzrente durch Familienzulagen

Abbildung 18 zeigt die Berechnung der Erhöhung der Zusatzrente aufgrund von Familienzulagen. Zunächst wird in Zeile 425 die Berechnung der Einkommensgrenze – Grundrente für Erwerbsunfähige plus maximale Zusatzrente – ohne Erhöhung durch Familienzulagen aufgerufen.

Die Einkommensgrenze ohne Erhöhung durch Familienzulagen wird samt dem Einkommen des Antragstellers schließlich an die Methode „calculateGrundbetragZusatzrente“ (Zeile 427) übergeben, um den Grundbetrag der Zusatzrente ohne Erhöhungen und Zulagen zu berechnen. Das Ergebnis dieser Berechnung ist eine fiktive Zusatzrente, die, abhängig vom Einkommen, eventuell niedriger ist als die tatsächlich zustehende. Die Erhöhung der Zusatzrente ergibt sich sodann aus der Differenz der tatsächlichen Zusatzrente minus der fiktiven Zusatzrente (siehe Zeilen 437 und 438 in Abbildung 18). Eine allfällige Erhöhung wird in weiterer Folge von den Familienzulagen zu gleichen Teilen abgezogen.

Wie in Abbildung 19, Zeile 306, zu sehen ist, wird die Methode „calculateGrundbetragZusatzrente“ auch bei der Berechnung des Grundbetrags der Zusatzrente gemäß § 12 Abs. 2 aufgerufen. Der Wert der Einkommensgrenze wird jedoch in diesem Fall anders berechnet. Die Einkommensgrenze wird zunächst wie oben beschrieben berechnet (nicht in Abbildung 19 ersichtlich). Die sich aus dem Betrag der Grundrente für

Erwerbsunfähige plus der maximalen Zusatzrente ergebende Einkommensgrenze wird sodann um die Anzahl der Familienangehörigen mal € 40,60 erhöht. Die Berechnung der Erhöhung der Einkommensgrenze ist in den Zeilen 293 bis 300 in Abbildung 19 dargestellt.

```
290         // 2. Erhöhung Einkommensgrenze aufgrund von Familienzulagen
291         int angehoerige = beschaedigter.getAngehoerige();
292         if (angehoerige > 0) {
293             Euro erhoehungEinkommensgrenze = ERHOEHUNG_EINKOMMENSGRENZE
294                     .multiply(angehoerige);
295             logger.debug("§ 12 Abs. 2: {} * {} = Erhöhung {}",
296                     ERHOEHUNG_EINKOMMENSGRENZE, angehoerige,
297                     erhoehungEinkommensgrenze);
298 
299             Euro egAlt = einkommensgrenze;
300             einkommensgrenze = einkommensgrenze.plus(erhoehungEinkommensgrenze);
301             logger.debug("§ 12 Abs. 2: {} + Erhöhung {} = Einkommensgrenze {}",
302                     egAlt, erhoehungEinkommensgrenze, einkommensgrenze);
303         }
304 
305         // 3. Berechnung der Zusatzrente
306         Euro zusatzrente = calculateGrundbetragZusatzrente(einkommensgrenze,
307                 beschaedigter.getEinkommen());
308         rente.setGrundbetrag(zusatzrente);
```

Abbildung 19: Berechnung des Grundbetrags der Zusatzrente gemäß § 12 Abs. 2

Wie dieses Beispiel veranschaulicht, ist die Berechnung der Familienzulagen gemäß § 16 von der Zusatzrente, genauer von deren Erhöhung durch den Anspruch auf Familienzulage, abhängig. Aus technischer Sicht ist es demnach notwendig, dass die Berechnung der Familienzulagen auf den Grundbetrag der Zusatzrente gemäß § 12 Abs. 2 zugreifen kann.

Um eine effiziente Programmierung zu gewährleisten und eine Duplizierung des Codes gemäß dem Prinzip „Don't repeat yourself" zu verhindern, war es notwendig, die Berechnung des Grundbetrags der Zusatzrente herauszulösen und in der Methode „calculateGrundbetragZusatzrente" zu kapseln.

Durch eine entsprechende Methodensignatur, die neben dem Einkommen des Beschädigten auch die für die Berechnung relevante Einkommensgrenze enthält, lässt sich die Berechnung parametrisieren. Damit ist es möglich, im Kontext des § 16 die Berechnung einer fiktiven Zusatzrente ohne Erhöhung der Einkommensgrenze durch Familienzulagen durchzuführen.

Zusammengefasst sollte dieses Beispiel eine Wechselwirkung zwischen verschiedenen Leistungen veranschaulichen. Wie gezeigt wurde, lassen sich mit Java auch solche relativ komplexen Berechnungen durchführen.

Allgemein lässt sich feststellen, dass zur Durchführung der Berechnung abhängiger Leistungen der Zugriff auf vorangegangene Berechnungsergebnisse möglich sein muss. Überdies ist es unter Umständen notwendig, dass Teile des Berechnungsalgorithmus mit unterschiedlichen Werten aufgerufen werden müssen, wie im gegenständlichen Fall die Berechnung des Grundbetrags der Zusatzrente gemäß § 12 Abs. 2.

Der Code zur Berechnung ist ziemlich komplex und schwer nachvollziehbar. Die vorliegende Implementierung entspricht daher nicht dem Prinzip des Isomorphismus.

Es bleibt offen, ob andere Sprachen zur Formalisierung ausdrucksstark genug sind, um dieses Beispiel zu implementieren, und ob dem Prinzip des Isomorphismus besser entsprochen werden kann.

9.5.10. Datenzugriff

Die Datenzugriffsschicht, abstrahiert durch das Interface `DataRepository`, übernimmt in diesem Fall vor allem die Bereitstellung der im Gesetz vorhandenen Geldbeträge. Die Implementierung `InMemoryDataRepository` ist dabei bewusst einfach gehalten, auf eine echte Datenbank (wie beispielsweise Oracle) zur Speicherung wird verzichtet. Stattdessen werden sämtliche Datenstrukturen bei der Initialisierung der Klasse mit fixen Werten in den Speicher geladen und dort bereitgehalten.

Die Beträge der Grundrente, die altersabhängigen Erhöhungen etc. werden durch geeignete Datenstrukturen repräsentiert.

Da es sich im Falle der Beschädigtenrente meistens um eine binäre Beziehung zwischen einem Wert (MdE, Pflegestufe etc.) und einem Geldbetrag handelt, wird zu deren Repräsentation eine `HashMap` eingesetzt. Bei einer `HashMap` handelt es sich um eine Klasse des „Java Collections Frameworks“, das Teil des „Standard Development Kits“ ist.

Kompliziertere tabellarische Strukturen wie jene des § 11 Abs. 3 werden durch Instanzen vom Typ `Table` der Google Guava Bibliothek repräsentiert.

Wichtig ist, dass die Abfrage der Werte mit exakten Schlüsseln erfolgen muss. Kann beispielsweise zu einer MdE kein Betrag gefunden werden, wird dies als Fehler interpretiert und eine `IllegalArgumentException` geworfen. Es ist die Aufgabe der `JavaCalculationEngine`, die tatsächlichen Werte gegebenenfalls zu „normalisieren“, damit der Aufruf mit dem richtigen Schlüsselwert durchgeführt werden kann.

9.6. Diskussion

In diesem Kapitel wurde die Formalisierung der Beschädigtenrente, einer monetären Leistung nach dem Kriegsopferversorgungsgesetz, dargestellt.

An die Darstellung der gesetzlichen Grundlage schloss sich eine abstrakte Beschreibung eines Systems an, wie es zur automatisierten Vollziehung der Beschädigtenrente dienen könnte. Dabei wurde insbesondere auf das Subsumtionsmodul als zentraler Teil der Software eingegangen.

Der Hauptteil dieses Kapitels beschäftigte sich mit einer Implementierung der Beschädigtenrente in Java. Die Schnittstelle des Subsumtionsmoduls wurde in Abschnitt 9.5.9.1. und dessen Implementierung im Abschnitt 9.5.9.2. beschrieben.

Anhand von Codebeispielen wurden unterschiedliche Möglichkeiten zur Modellierung von Normen in Java dargestellt. Dabei wurde der Schwerpunkt der Präsentation auf die Auflösung von Konflikten zwischen Normen gelegt.

9.6.1. Die Behandlung von Konflikten

Abschnitt 9.5.9.4. präsentierte die Erhöhungen der Grundrente nach § 11 Abs. 2 und 3 als ein Beispiel für Normen, die miteinander in Konflikt stehen. In diesem Fall enthält Abs. 3 explizit eine Ausnahmebestimmung zu Abs. 2. Der Ansatz folgt dem Prinzip, dass jede Norm als eigene Regel implementiert wird. Den formalen Regeln werden sodann Prioritäten zugewiesen.

In Java existiert jedoch kein abstraktes Konzept der „Regel“ in Form einer abstrakten Datenstruktur. Wie ausgeführt wurde, entspricht am ehesten eine if-Bedingung in Java einer Regel, bestehend aus Prämissen und Konklusion. Die beabsichtigte Priorisierung wird durch eine entsprechende Anordnung der Anweisungen im Programm erreicht. Anweisungen höherer Priorität werden später aufgerufen und können so die Konklusion früherer Regeln widerrufen.

Ein alternativer Ansatz für die Auflösung von Konflikten zwischen Normen wurde in Abschnitt 9.5.9.5. präsentiert. Blinden wird anstelle der Pflegezulage (§ 18) eine Blindenzulage (§ 19) gewährt. Zur Formalisierung der in § 19 enthaltenen Ausnahmebestimmung wurde eine negativ formulierte Prämisse in die Repräsentation des § 18 aufgenommen.

Grundprinzip dieses Ansatzes ist, dass Ausnahmebestimmungen als negativ formulierte Prämisse in die formale Repräsentation der Norm aufgenommen werden. Dadurch lässt sich im Code eine relativ übersichtliche Repräsentation einer Norm samt den Bedingungen für ihre Anwendung erreichen. Trotzdem widerspricht diese Modellierungsform dem Prinzip des Isomorphismus, da unterschiedliche Normen durch eine einzige formale Regel repräsentiert werden.

Die Forschungen zur „defeasible logic" beschäftigen sich mit der Modellierung und haben eine eigene Terminologie hervorgebracht. Die Norm, die von einer Ausnahme betroffen ist und demnach nicht angewendet werden soll, wird „defeasible" genannt, die Ausnahmebestimmung als „anfechtende" Regel „defeater". Diese Terminologie soll im Folgenden für die weiteren Erläuterungen aufgegriffen werden.

Zusammengefasst können in Konflikt stehende Normen wie folgt modelliert werden:

1. Die „angefochtene" Norm („defeater") und die „anfechtende" Norm („defeasible") werden als separate Regeln formalisiert. Der Ausnahmebestimmung, damit ist die „anfechtende" Norm gemeint, wird eine höhere Priorität zugewiesen. Sie wird später ausgeführt und kann damit die Konklusion der „angefochtenen" Norm aufheben oder verändern. Dem Prinzip des Isomorphismus wird weitgehend Rechnung getragen. Die „Anfechtbarkeit" von Normen wird jedoch bloß simuliert, da die Konklusion der angefochtenen Bestimmung zunächst ausgeführt und später überschrieben wird.
2. Die nicht anzuwendende Norm („defeasible") und die Ausnahmebestimmung („defeater") werden in einer formalen Regel modelliert. Im Detail wird die Bedingung für die Ausnahme als negativ formulierte Prämisse(n) in die Regel aufgenommen. Diese Repräsentation ist nicht isomorph.

Eine dritte Alternative zur Lösung von Konflikten zwischen Rechtsvorschriften ist, die anfechtende Norm logisch so zu modellieren, dass die angefochtene Norm nicht zur Anwendung gelangt (Bench-Capon & Gordon, 2009, S. 15 f.). Für die Implementierung dieses Ansatzes ist es erforderlich, dass zunächst ein abstraktes Konzept der formalen Regel, bestehend aus Prämisse und Konklusion, implementiert wird. Instanzen solcher Regeln könnten dann mit Prioritäten – beispielsweise im Fall einer „lex specialis" – versehen werden oder auch explizite Verweise zwischen anfechtender und angefochtener Norm enthalten. Anhand solcher „Metaregeln" könnte der Lösungsalgorithmus die anzuwendenden Normen

ermitteln. Normen, die von einer Ausnahmebestimmung betroffen sind, würden nicht wie bei dem hier präsentierten Lösungsansatz mit Prioritäten zuerst zur Anwendung gelangen.

Zusammengefasst sollte die „Anfechtbarkeit“ von Normen auf höherer Ebene als abstraktes Konzept im Programm implementiert werden. Für weitere Ausführungen zur „Defeasibility“ sei auf Athan et al. (2013b, S. 6 f.) verwiesen.

9.6.2. Gruppierung von Regeln

Das in Abschnitt 9.5.9.6. präsentierte Beispiel der Berechnung der Familienzulage hat gezeigt, dass zwischen Normen auch komplexere Wechselwirkungen bestehen können. Für eine effiziente Implementierung war es notwendig, Teile der Berechnung in eine eigene Methode auszulagern. Dies wurde durch den expliziten Verweis von § 16 auf § 12 Abs. 2 bedingt.

Es wird deutlich, dass solche Verweisungen des Gesetzes unmittelbare Auswirkung auf die Struktur des Programmes haben. Es kommt zur Verteilung der Implementierung einer Norm auf mehrere Methoden. Vom Standpunkt des Isomorphismus wird es schwieriger nachzuvollziehen, welche Rechtsgrundlage die einzelnen Methoden haben.

Die Beobachtungen von Bench-Capon und Gordon (2009, S. 17 f.), dass die Paragrafen eines Gesetzes notwendigerweise in einer bestimmten Reihenfolge ausgeführt und irgendwie gruppiert werden müssen konnten bestätigt werden. Ein weiterer Aspekt, den die soeben erwähnten Autoren in ihrem Paper behandelt haben, ist die Verteilung der Beweislast durch die verschiedenen Möglichkeiten zur Modellierung von Normen. Da in diesem Fall sämtliche Daten bereits als „Fakten“ angesehen und zur Berechnung übergeben werden, wurde dieser Gesichtspunkt hier vernachlässigt.

Für die Implementierung des Programms hat es sich bewährt, die Terminologie und Struktur des Gesetzes so weit wie möglich beizubehalten. Wichtige Begriffe des Gesetzes wie Beschädigtenrente, Grundrente etc. wurden für die Benennung von Klassen verwendet. Der Berechnungsalgorithmus wurde dementsprechend auf mehrere Methoden aufgeteilt, wobei jede Methode für die Konstruktion und Berechnung von Objekten der eben erwähnten Klassen zuständig ist. Diese Objekte repräsentieren Teilleistungen sowie die Beschädigtenrente insgesamt.

Wie schon Sartor (2009b) festgestellt hat, bieten rechtliche Begriffe einen guten Ausganspunkt, um miteinander verwandte Regeln zu bündeln. Dieser Ansatz trägt maßgeblich zur besseren Strukturierung des Programms in entsprechende Packages, Klassen und Methoden bei. Komplexe Berechnungsschritte können bei Bedarf auch weiter unterteilt durch mehrere Methoden repräsentiert werden.

9.6.3. Schlussbetrachtung

Bei der Modellierung des Gesetzes war auffällig, dass die Implementierung einer logischen Regel oftmals in zahlreiche if-Bedingungen zerfällt. Dies war insbesondere bei Regeln mit vielen Prämissen notwendig um zu verhindern, dass die korrespondierende if-Bedingung unübersichtlich wird.

Die Grundlage für Berechnungen von Leistungen wie der Beschädigtenrente ist naturgemäß eine Vielzahl von Geldbeträgen. Diese werden in diversen Normen zum Ausdruck gebracht. Wie schon bei der Berechnung weiter oben erwähnt wurde, verweisen auch hier Normen auf Beträge, die in anderen Rechtsvorschriften normiert sind. Aus diesem Grund wurden sämtliche Beträge in ein eigenes Modul, das „DataRepository“, ausgelagert und stehen dort zentral zur Verfügung. Dies ist aus dem Blickwinkel des Isomorphismus betrachtet problematisch, da im Code, der die Berechnung durchführt, die tatsächlichen Werte nicht ersichtlich sind.

Insgesamt kann gesagt werden, dass die Formalisierung der Beschädigtenrente in Java gute Ergebnisse zeigt. Die Berechnung funktioniert, das Programm ist jedoch mit ca. 1.800 Zeilen ziemlich umfangreich, obwohl nur wenige Normen formalisiert wurden. Zudem wurden nicht alle Normen wie Sonderzahlungen, Einmalzahlungen und das Ruhen der Versorgung bei Gefängnisstrafe implementiert. Dies erweckt eine ungefähre Vorstellung vom Ausmaß der Formalisierung einer umfangreicheren Rechtsmaterie, wie beispielsweise dem Steuerrecht.

Das Prinzip des Isomorphismus ist nur bedingt erfüllt. Durch das Fehlen des Konzepts einer „echten“ Regel in Java verteilt sich die Repräsentation der Normen oft über zahlreiche if-Bedingungen und Methoden. Normen und Werte aus anderen Gesetzen, wie in diesem Fall dem ASVG, werden statisch in das formale Modell „importiert“.

Die Implementierung des abstrakten Konzepts der Regel – bestehend aus Antezedens und Konsequenz – könnte entscheidend zur Verbesserung der Modellierung beitragen. Mithilfe

abstrakter Regeln wäre eine direkte Formalisierung von Normen möglich und damit dem Isomorphismus besser entsprochen.

Betrachtet man das Javaprogramm aus Sicht der Wissensrepräsentation, so handelt es sich streng genommen nicht um ein wissensbasiertes System, da das Wissen (die Regeln) nicht vom Algorithmus getrennt ist. Regeln und Algorithmus sind im Javacode an derselben Stelle implementiert. Eine vom Algorithmus getrennte Repräsentation der Normen würde die Qualität und den Grad der Wiederverwendbarkeit des formalen Modells erheblich steigern.

Überdies muss auch das Ergebnis – die Rechtsfolgen – der automatisierten Vollziehung isomorph sein. Dies bedeutet erstens, dass das formale Ergebnis (in diesem Fall ein Java-Objekt) die Teilergebnisse der Berechnung adäquat repräsentieren muss, und zweitens, dass das Ergebnis mit den entsprechenden Regeln verknüpft sein muss. Nur so kann gewährleistet werden, dass vom Ergebnis auf die formalen Regeln, die dieses produziert haben, und weiter auf deren normative Grundlage geschlossen werden kann.

Diese Erweiterung des Prinzips des Isomorphismus auf die formale Repräsentation des Ergebnisses der automatisierten Rechtsanwendung ist unabdingbar für die juristische Begründung. Nur durch die Verknüpfung von Norm, formaler Regel und Ergebnis (den Rechtsfolgen) kann die automatisierte Rechtsanwendung schlüssig nachvollzogen und gerechtfertigt werden. Bisherige Arbeiten zum Thema Isomorphismus auf dem Gebiet der KI und Recht haben diesen Aspekt nicht explizit erwähnt und scheinen diesen zu vernachlässigen.

Ein weiterer wichtiger Aspekt, der bei der Implementierung vernachlässigt wurde, ist die Änderung von Normen durch Novellen. Solche Änderungen betreffen vor allem die Geldbeträge, aber selbstverständlich auch die grundlegende „Struktur“ des Gesetzes. So können der Tatbestand von Normen geändert, diese aufgehoben oder neue hinzugefügt werden. Solche Änderungen würden im gegenständlichen Fall zur Anpassung des Lösungsalgorithmus führen. Das Programm kann damit nur bedingt auf Änderungen der normativen Grundlage reagieren. Bei Novellen, die über die Anpassung von Werten hinausgehen, wäre eine Änderung des Codes erforderlich.

Auch hier könnte die Implementierung des Konzepts der abstrakten Regel mit einem entsprechenden Framework, um zeitliche Aspekte zu formalisieren, hilfreich sein. Für einen effizienten Umgang mit gesetzlichen Novellen ist neben einer geeigneten technischen

Implementierung auch ein integrierter Prozess unter Beteiligung juristischer Experten erforderlich (van Engers, Gerrits, Boekenoogen, Glassée, & Kordelaar, 2001). Auf diesen zweiten Problembereich kann an dieser Stelle jedoch nicht eingegangen werden.

Abschließend soll ein Ausblick auf mögliche zukünftige Forschungen gegeben werden:

- Die „Anfechtbarkeit“ von Rechtsnormen sollte durch „Metaregeln“ formalisiert und auf einer abstrakten Ebene behandelt werden. Durch einen entsprechenden Algorithmus können so die anzuwendenden Normen ausgewählt werden.
- Implementierung eines Frameworks, um mit zeitlichen Aspekten von Normen umgehen zu können. Dies umfasst die Konzipierung einer Wissensbasis, in die Novellen eingepflegt werden können sowie ein abstraktes Konzept für Verweisungen auf Normen (auch anderer Gesetze) in einer bestimmten Version.
- Verbesserung der isomorphen Repräsentation von Normen durch Implementierung des Konzepts der abstrakten Regel, bestehend aus Prämissen und Konklusion. An Objekte solcher Regeln könnten beliebige Metadaten geknüpft werden.
- Entwicklung einer Sprache zur Modellierung mathematischer Berechnungen. Es bedarf eines Formalismus, der einen höheren Abstraktionsgrad zur Modellierung mathematischer Berechnungen aufweist.

10. Die Ontologie des Kriegsopferversorgungsgesetzes

10.1. Einleitung

Als Ontologien zu Beginn Einzug in den Bereich KI und Recht hielten, wurden diese verwendet, um das Vokabular, das für die Erstellung von Regeln verwendet werden kann, zu spezifizieren und vorzugeben (vgl. Bench-Capon & Visser, 1997). Zu dieser Zeit steckte die Beschreibungslogik noch in den Kinderschuhen, obwohl einige Autoren schon eine breitere Konzeption der Rolle von Ontologien andachten (Valente & Breuker, 1995).

Die Ankunft des Semantic Web steigerte die Popularität von Ontologien erheblich. Die LKIF-Core Ontologie stellt die derzeit wohl wichtigste Entwicklung in diesem Bereich dar. Diese Ontologie wurde als Teil des Legal Knowlege Interchange Format (LKIF) im Rahmen des ESTRELLA Projekts entwickelt. Sie stellt das grundlegende Vokabular für die Entwicklung spezifischer juristischer Ontologien bereit (Breuker et al., 2007; Breuker, Boer, Hoekstra, & van den Berg, 2006; Hoekstra et al., 2007, 2009).

Aufgrund des hohen Abstraktionsniveaus der enthaltenen Begriffe kann die LKIF-Core Ontologie für verschiedenste Zwecke verwendet werden. Zu diesen gehören unter anderem die Modellierung von Rechtsnormen (Hoekstra et al., 2009, S. 44 ff.), die rechtliche Beurteilung von Fällen (van de Ven, Breuker, Hoekstra, & Wortel, 2008; van de Ven, Hoekstra, et al., 2008), der Vergleich von Rechtsnormen und Konsistenzprüfung (Palmirani, Contissa, & Rubino, 2009) sowie die Repräsentation komplexer Strukturen, beispielsweise von Rechtsgeschäften (Conetta & Schafer, 2013).

In diesem Kapitel wird die LKIF-Core Ontologie als Grundlage für die Erstellung einer juristischen Ontologie für die Durchführung der automatisierten rechtlichen Beurteilung implementiert. Gegenstand der Untersuchung ist die Formalisierung der Beschädigtenrente (§§ 7 ff.). Der Ansatz basiert ausschließlich auf Beschreibungslogik, für die Modellierung der Ontologie wird OWL 2 (fortan nur OWL) verwendet.

Ziel der Forschungen ist die Untersuchung, ob OWL für die Formalisierung von Verwaltungsrecht ausdrucksstark genug ist. Verwaltungsgesetze zeichnen sich in der Regel durch eine hohe Normendichte aus und erfordern oftmals mathematische Berechnungen zur Vollziehung. Dem Kapitel liegt das in Abschnitt 1.1. entwickelte Verständnis des Begriffs

„Verwaltungsrecht“ zugrunde, sodass sich die Untersuchung auf die Modellierung des Kriegsopferversorgungsgesetzes beschränkt.

Auf dem Gebiet der Verwaltungsautomatisierung handelt es sich um einen neuartigen Ansatz der Formalisierung, der jedenfalls als experimentell angesehen werden muss. Bisher erfolgte die Implementierung von Software zur automatisierten Vollziehung durch einfache Programme, geschrieben in Java oder COBOL etc. Fiedler (2004) spricht in dieser Hinsicht von „stiller Formalisierung“.

Da die Syntax von OWL an die natürliche Sprache angelehnt ist, wird erwartet, dass das KOVG auf eine übersichtliche Art und Weise dargestellt werden kann. Diese – auch für Juristen – gut lesbare Darstellung des KOVG sollte die einfache Änderbarkeit des Modells ermöglichen, was wiederum die Häufigkeit von Fehlern verringern kann. Zudem ist vorstellbar, dass die formale Repräsentation des Gesetzes in Zukunft auch für dessen Vereinfachung als Grundlage dienen könnte (Philipps, 2011). Änderungen des Programmcodes sollen vermieden werden und ausschließlich in der Ontologie stattfinden. Dies kann erheblich zu einer Senkung der Wartungskosten beitragen, da es nicht mehr erforderlich ist, bei Novellen einen IT-Dienstleister mit der Durchführung von Anpassungen der Software zu beauftragen.

Das Kapitel beginnt mit der Vorstellung der Formalisierung mit OWL sowie einem Überblick über die Ontologie des Kriegsopferversorgungsgesetzes und die verwendete Methodologie. Im Anschluss werden die Module der Ontologie beginnend mit der Domänenontologie präsentiert.

Für die Formalisierung der Anspruchsgrundlagen werden zwei Ansätze repräsentiert. Die „einfache“ Ontologie kommt ohne die Verwendung der LKIF-Core Ontologie aus, weist jedoch im Hinblick auf die Repräsentation von Normen Schwächen auf. Dieser Schwächen nimmt sich das darauffolgende Kapitel an, in dem ein auf der LKIF-Core Ontologie basierender Ansatz präsentiert wird.

Im Anschluss an die Ausführungen über das ontologische Modell wird eine hybride Architektur präsentiert, um terminologisches Wissen in Ontologien mit regelbasierten Formalismen zu kombinieren. Eine solche Kombination ist erforderlich, um eine Lösung für die mangelnde Ausdrucksstärke von OWL zu finden.

Ein Überblick über die verwandten Ansätze mit anschließender Diskussion, die das Ergebnis der Untersuchung präsentiert und einen Ausblick auf zukünftige Forschungen gibt, beendet das Kapitel.

10.1.1. Die LKIF-Core Ontology

Die LKIF-Core Ontologie ist eine sogenannte „Upper“ oder „Core Ontology“, die im Wesentlichen die „top-down“ Modellierung und Repräsentation von Wissen in Domänenontologien (engl. domain ontology) ermöglicht. Die LKIF-Core Ontologie enthält eine Vielzahl abstrakter Begriffe (in Form von Superklassen). Diese Begriffe sind unabhängig von einem bestimmten Rechtsbereich und stellen die Struktur für Domänenontologien, wie jene, die das KOVG abbilden soll, bereit.

Der hohe Abstraktionsgrad der erwähnten Begriffe ermöglicht die universelle Verwendung der LKIF-Core Ontologie für die Modellierung von beliebigen Domänenontologien, unabhängig von deren normativer Grundlage. Die Verwendung der LKIF-Core Ontologie, die zu einer ähnlichen Struktur der Domänenontologien führt, erlaubt es außerdem, die Gemeinsamkeiten und Unterschiede verschiedener Gesetze zu erkennen, und trägt so zum besseren Verständnis der Normen bei.

Im gegenständlichen Kontext – der Modellierung der Beschädigtenrente des KOVG – soll die LKIF-Core Ontologie vor allem für die Strukturierung von Informationen sowie für die Entscheidungsfindung und Problemlösung eingesetzt werden. Es sei erwähnt, dass dies nur zwei der verschiedenen Rollen sind, die eine Kernontologie (engl. core ontology) für die Unterstützung der Wissensmodellierung spielen kann (Breuker et al., 2007, S. 18 ff.).

Die erste Rolle betrifft im Wesentlichen die Vorgabe einer Struktur für die Entwicklung von Domänenontologien, die eine „top-down“ Modellierung von Wissen ermöglicht. So können grundsätzlich alle Begriffe einer Domäne als Subklassen der in der Kernontologie vorhandenen abstrakten Konzepte modelliert werden. Als Beispiel sei hier die Klasse „Norm“ der LKIF-Core Ontologie erwähnt, unter die Normen eines beliebigen Gesetzes subsumiert werden können.

Die zweite Rolle betrifft die Repräsentation des Wissens einer Domäne in einer Form, die eine automatisierte Problemlösung und Entscheidungsfindung ermöglicht. Probleme sollen auf eine Art und Weise modelliert werden, die eine automatisierte Lösungsfindung erlaubt.

Die Ontologie enthält dabei alle Begriffe sowie die Beziehungen zwischen diesen und erlaubt es so, den Sachverhalt zu modellieren. Der Sachverhalt wird sodann unter die in der Ontologie modellierten Tatbestände der Normen klassifiziert, die – ähnlich dem Sachverhalt – eine Situation beschreiben. Mithilfe der Inferenz ist es damit möglich, den Sachverhalt einer deontischen Qualifikation zu unterziehen, die ausdrückt, ob dieser geboten, verboten oder erlaubt ist.

Ausgehend von den in der LKIF-Core Ontologie definierten abstrakten Konzepten kann die Erstellung einer Domänenontologie „top-down“, beginnend mit den abstraktesten zu den konkretesten Begriffen, erfolgen.

10.1.2. Das KOVG

In diesem Beitrag soll näher auf die Modellierung der Abschnitte eins bis drei des ersten Hauptstücks des Kriegsopferversorgungsgesetzes eingegangen werden. Dabei geht es in erster Linie nicht um die vollständige Abbildung der in den erwähnten Abschnitten normierten Paragrafen, sondern es sollen die wichtigsten Begriffe des Gesetzes mithilfe der LKIF-Core Ontologie modelliert werden.

Insbesondere sind damit die erfassten Personengruppen, die zustehenden Leistungen und deren Anspruchsvoraussetzungen gemeint. Keinesfalls geht es um eine vollständige Modellierung des KOVG.

Die Schwierigkeit bei der Formalisierung des KOVG besteht vor allem in der Abbildung der zahlreichen Varianten, die im KOVG normiert sind. Da das KOVG eine erlittene Gesundheitsschädigung auszugleichen versucht, war es wohl ein Anliegen des Gesetzgebers, hier möglichst gerecht auf individuelle Situationen zu reagieren. So gibt es zur Grundrente eine Vielzahl an Zulagen, die wiederum vom Gesundheitszustand und Einkommen des Antragstellers abhängig sind. Im Hinblick auf die Anrechnung des Einkommens sei erwähnt, dass es „einkommensabhängige“ und „einkommensunabhängige“ Leistungen im KOVG gibt. Es reicht also nicht aus, das Einkommen einfach auf den berechneten Gesamtbetrag der Leistungen anzurechnen. Vielmehr bemisst sich die Höhe der „einkommensabhängigen“

Leistungen nach dem anrechenbaren Einkommen[146], welches direkt bei den einzelnen Leistungen in Anschlag zu bringen ist.

Diese Umstände machen es aus technischer Sicht schwierig, Programme zu entwickeln, die das KOVG vollziehen können. Die Struktur des Gesetzes führt zu einem sehr komplexen Programm, das viele Verzweigungen enthält.

Wie eingangs bereits erwähnt wurde, soll in diesem Beitrag ein Modellierungsversuch dargestellt werden, der die angesprochene Komplexität zu mindern versucht. Es sollen die Abschnitte eins bis drei in einem ontologischen Modell, das auf der LKIF-Core Ontologie aufbaut, dargestellt werden.

10.2. Die Verwendung von OWL zur automatisierten Ermittlung von Ansprüchen

Für die Feststellung, ob dem Antragsteller Leistungen rechtlich zustehen, ist es erforderlich, dass ein Sachverhalt anhand der anwendbaren Normen überprüft wird. Dies erfordert die Beschreibung des Sachverhalts, dessen Interpretation und die Anwendung von Rechtsnormen, um entsprechende Ansprüche aus der gesetzlichen Grundlage ableiten zu können.

Wie im vorherigen Kapitel gezeigt wurde, werden Normen oftmals durch einfache Bedingungen im Programmcode oder durch Regeln repräsentiert. In diesem Kapitel wird der Versuch dargestellt, ausgewählte Normen des KOVG mit OWL zu formalisieren und in einer Ontologie zu repräsentieren. Die Wissensbasis ist modular aufgebaut und besteht aus mehreren Ontologien: Einer Domänenontologie, einer Ontologie, die ausgewählte Normen des KOVG repräsentiert, und einer Ontologie, die die Beschreibung des Sachverhalts enthält. Die Beschreibung des Sachverhalts besteht aus den rechtlich relevanten Fakten, die eine bestimmte Situation kennzeichnen.

Ein Vorteil dieses Ansatzes ist, dass das in der Ontologie modellierte Wissen vollständig genutzt werden kann. Neues Wissen wird automatisch abgeleitet, sobald Aussagen zur Ontologie hinzugefügt werden. Beispielsweise werden von den Instanzen neue Properties abgeleitet, die so die Sachverhaltsbeschreibung ergänzen. So könnte beispielsweise aus der

[146] Zur Berechnung siehe § 13 KOVG.

Aussage, dass das Auto A links von B ist, abgeleitet werden, dass B sich auf der rechten Seite von A befindet.

Zweitens geht durch den Einsatz einer Ontologie kein Wissen verloren oder wird vernachlässigt. Die Inferenzmaschine berücksichtigt sofort neue Aussagen, die zur Ontologie hinzugefügt werden und so möglicherweise das Ergebnis der rechtlichen Beurteilung beeinflussen.

Ein weiterer wichtiger Vorteil ist, dass die Inferenzmaschine automatisch ableiten kann, welche Normen Ausnahmebestimmungen sind. Die explizite Modellierung von Ausnahmen ist damit im Fall einer Lex specialis nicht mehr notwendig. Durch die hinter OWL stehende Beschreibungslogik wird dieser Vorteil ohne zusätzlichen Aufwand erlangt. Bei (anfechtbaren) regelbasierten Formalismen müssen solche Ausnahmebedingungen explizit modelliert werden. Die Modellierung solcher Ausnahmetatbestände ist für den Menschen oft schwierig und mit Fehlern verbunden.

OWL besitzt bestimmte logische Eigenschaften, die eine korrekte, vollständige und entscheidbare Schlussfolgerung gewährleisten, die Richtigkeit der Entscheidung ist daher ausschließlich von der Qualität der Modellierung abhängig.

Letztendlich garantiert OWL eine korrekte, vollständige und entscheidbare Schlussfolgerung – logische Eigenschaften, die dazu führen, dass die Richtigkeit der Entscheidungsfindung einzig und allein von der Qualität der Modellierung abhängig ist.

Die Verwendung von Beschreibungslogik hat jedoch auch Nachteile und ist nicht so selbsterklärend, wie der Einsatz von Regeln zur Modellierung von Normen.

10.3. Methodologie

In diesem Abschnitt werden die wichtigsten Aspekte der Modellierung der Ontologie des KOVG in OWL präsentiert. Die (neuen) Features von OWL 2 werden genutzt und die Ausdrucksstärke der Sprache voll ausgereizt.

Für das Design einer Ontologie existieren, ähnlich wie bei der Softwareentwicklung, keine allgemein gültigen Regeln für die Modellierung. Weder die formale Beschreibungslogik, auf der OWL 2 basiert, noch die Entwicklungswerkzeuge geben die Struktur für eine Ontologie vor. Da die Struktur und der Inhalt einer Ontologie maßgeblich vom angestrebten

Verwendungszweck abhängig sind, ist es bloß möglich, Empfehlungen hinsichtlich der Struktur abzugeben.

Aus diesem Grund haben diverse Autoren Entwurfsmuster für Ontologien entwickelt (Gangemi, 2007; Hoekstra, 2009). Entwurfsmuster sollen Lösungen für bekannte Probleme der Modellierung bereitstellen. Sie repräsentieren gewissermaßen Best Practices. Ähnlich wie in der Softwareentwicklung fanden diese auch Einzug in das Knowledge Engineering und die Entwicklung von wissensbasierten Systemen.

Dieser Trend wurde bereits von Mylopoulos (1980) vorhergesehen, der mit einer Verzögerung von acht bis zehn Jahren rechnete. Auch wenn diese Einschätzung etwas zu optimistisch war, haben sich mit einiger Verspätung nicht zuletzt durch die Entwicklung der LKIF-Core Ontologie Entwurfsmuster für die Modellierung von Ontologien herausgebildet.

Die vorhandenen Entwurfsmuster werden für die Modellierung der Ontologie des KOVG soweit zweckdienlich herangezogen. Vorhandene Best Practices für die Modellierung mit OWL sollen berücksichtigt werden, um übliche Fehler bei der Erstellung der Ontologie zu verhindern und deren Qualität zu steigern (Rector et al., 2004). Dazu gehört unter anderem die Definition von Klassen als „disjoint“ miteinander, sofern nicht explizit anders gewünscht. In der Beschreibungslogik „überlappen“ sich Klassen ansonsten standardmäßig.

Das Design der Ontologie basiert auf einem middle-out Ansatz: Bottom-up, um die Domänenontologie zu erstellen, top-down, um abstrakte Begriffe zu identifizieren und die Strukturen für die rechtliche Entscheidungsfindung zu modellieren.

Für die Erstellung der Ontologie ist es notwendig, die wichtigsten Begriffe des KOVG zu identifizieren und zu modellieren. Die identifizierten Begriffe werden als Subklassen der abstrakten Begriffe der LKIF-Core Ontologie modelliert. Insofern kann man von einem „top-down“ Ansatz sprechen. Andererseits wird vom Gesetzestext ausgegangen, um diese Konzepte zu identifizieren.

Aus dem Rechtstext werden abstrakte generellere Begriffe abgeleitet, die als Superklassen modelliert werden. Aus dieser Sicht könnte man also von einem „bottom-up“ Ansatz sprechen. Da jedoch der Blick bei der Modellierung zwischen den abstrakten Begriffen der LKIF-Core Ontologie und der gesetzlichen Grundlage – dem KOVG – „hin und her“ schweift, sollte besser von einem „middle-out“ Ansatz gesprochen werden.

Ausgehend von der gesetzlichen Grundlage werden zuerst die grundlegenden Begriffe identifiziert und modelliert. Diese Tätigkeit besteht im Wesentlichen aus einem Abstraktionsvorgang, der Gemeinsamkeiten und Ähnlichkeiten der Normen des KOVG identifiziert.

Sämtliche Begriffe werden als Klassen modelliert und durch „notwendige und hinreichende Bedingungen“ beschrieben. Dies wird durch die Verwendung von `owl:equivalentClass` Axiomen erreicht. Dieser Ansatz wurde auch bei der Erstellung der LKIF-Core Ontologie verfolgt (Hoekstra et al., 2007, 2009). Die vollständige Beschreibung einer Klasse ist Voraussetzung für die Ableitung des Typs einer Instanz aus der Ontologie. Die Inferenzmaschine kann überdies alle impliziten Beziehungen zwischen Begriffen folgern. Diese Vorteile wären bei einer bloß teilweisen Klassenbeschreibung nicht möglich.

Die Bedeutung der Begriffe muss immer im Kontext ihrer Verwendung gesehen werden. Prinzipiell könnte jede besondere Ausprägung eines Begriffs als Subklasse eines generelleren Begriffs modelliert werden. Dies würde jedoch vorhersehbar rasch zu einer Ausuferung der Subklassen führen, da für jede mögliche Situation eine Subklasse definiert werden müsste (Breuker et al., 2007, S. 7 ff.).

Zielführend ist vielmehr, einen Begriff durch seine Beziehungen zu anderen Begriffen zu definieren. Eine Klasse muss daher hinreichend durch Restriktionen beschrieben werden, die letztendlich den Kontext der Verwendung der Klasse repräsentieren. Ein Begriff samt seinen Beziehungen repräsentiert das gesamte Wissen, das in der Ontologie über diesen Begriff vorhanden ist. Der modellierte Kontext wird durch den beabsichtigten Verwendungszweck begrenzt. Eine Ontologie, die dem reinen Information Retrieval dient, benötigt in der Regel nicht so eine hohe Begriffsdichte wie eine solche, die für die automatisierte Entscheidungsfindung und Problemlösung verwendet werden soll.

Die Terminologie des Gesetzes wird, um dem Prinzip des Isomorphismus zu entsprechen, in die Ontologie übernommen. Dies erleichtert das Auffinden der Begriffe in der Ontologie erheblich und trägt zum besseren Verständnis bei.

Aufgrund der Tatsache, dass die Klassen und deren Eigenschaften in der LKIF-Core Ontologie englisch benannt sind, entsteht so eine gewisse Vermengung deutscher und englischer Begriffe. Die Übersetzung der amtsdeutschen Begriffe des KOVG ins Englische ist aber leider

nicht immer exakt durchführbar. Aus diesem Grund soll die Begrifflichkeit des KOVG direkt übernommen werden.

Um die Wiederverwendbarkeit der Ontologie zu steigern, werden möglichst aufschlussreiche Namen für Klassen, Properties und Instanzen verwendet. Dabei werden folgende Namenskonventionen berücksichtigt (Allemang & Hendler, 2011, S. 310 f.):

- Ressourcen werden in CamelCase benannt: CamelCase ist ein Stil für Bezeichnungen, die aus mehreren Wörtern bestehen. Es kommen dabei Binnenversalien innerhalb eines Wortes zum Einsatz, z.B. MonetäreVersorgungsleistung für die Klasse aller monetären Versorgungsleistungen.
- Klassennamen beginnen mit einem Großbuchstaben.
- Namen von Instanzen beginnen mit einem Großbuchstaben.
- Propertynamen beginnen mit einem Kleinbuchstaben.
- Klassen und Instanzen werden mit einem Nomen im Singular bezeichnet.

Die Bezeichnung von Klassen- und Methodennamen im CamelCase ist auch in der objektorientierten Programmierung gebräuchlich. In der Regel wird davon abgeraten Umlaute zu verwenden, da dies oftmals zu Problemen mit dem Quellcode führt, wenn auf Systemen unterschiedliche Zeichensätze verwendet werden. Im Hinblick auf Ontologien können ohne Probleme Umlaute in Bezeichnungen verwendet werden, da das Namenssystem von OWL auf sogenannten IRIs (Internationalized Resource Identifier) basiert, einer Weiterentwicklung des URI, die auch Unicode zulässt.

Mit Hilfe der Inferenzmaschine (Reasoner) kann die Klassenhierarche von Begriffen und der Typ von Instanzen gefolgert werden. Dadurch wird schnell ersichtlich, ob die Ontologie noch konsistent ist. Dieser Ansatz beruht demnach auf der Modellierung der Klassen und der anschließenden Klassifizierung der Ontologie.

Sofort nachdem ein neuer Begriff samt seinen Restriktionen als Klasse hinzugefügt wurde, wird der Reasoner ausgeführt um zu überprüfen, ob diese korrekt klassifiziert, d.h. an der richtigen Stelle in der Ontologie positioniert wird. Danach wird eine Instanz (engl. „Individual“ genannt) dieses Begriffs erstellt und der Reasoner erneut ausgeführt um zu überprüfen, ob die Instanz unter den richtigen Begriff klassifiziert wird.

Dieser Ansatz baut stark auf der Verwendung der Inferenzmaschine auf, deren Ausführung unter Umständen auch länger dauern kann, da die LKIF-Core Ontologie an sich schon sehr umfangreich ist.

Die Modellierung der Ontologie wird durch den OWL Editor Protégé 4.3 unterstützt und für die Schlussfolgerung die Inferenzmaschine HermiT 1.3.8 verwendet.

10.4. Die Ontologie des Kriegsopferversorgungsgesetzes im Überblick

In diesem Abschnitt soll näher auf die Ontologie des KOVG eingegangen werden. Die Ontologie muss geeignet sein, die automatisierte Subsumtion zu unterstützen. Zu diesem Zweck muss die Ontologie ausdrucksstark genug sein und die individuellen Sachverhalte und Normen adäquat repräsentieren.

Wie bereits im vorherigen Abschnitt erwähnt wurde, besteht die Ontologie des KOVG aus mehreren kleineren Ontologien. Diese Module unterscheiden sich nach der Abstraktionsebene der enthaltenen Begriffe.

Unter Einbeziehung der LKIF-Core Ontologie ergibt sich folgender Aufbau der Ontologie:

(1) Die LKIF-Core Ontologie stellt die Grundlage des ontologischen Modells dar. Sie hat ebenso einen modularen Aufbau. Für die Modellierung von Rechtsnormen wird insbesondere das Normmodul benötigt.

(2) Die Domänenontologie enthält die wichtigsten Begriffe, die aus dem Gesetz abgeleitet werden, wie zum Beispiel „Beschädigter“, „Schwerbeschädigter“ etc. Dieses Modul beinhaltet das grundlegende Vokabular zur Modellierung von Beziehungen zwischen Begriffen und für die Repräsentation von Rechtsnormen.

(3) Das dritte Ontologiemodul – genannt Normenontologie – enthält eine Repräsentation der Rechtsnormen des KOVG. Für die Darstellung letzterer wird das Normenmodul der LKIF-Core Ontologie benötigt und importiert. Ebenso wird die Domänenontologie mit der Begrifflichkeit des KOVG importiert. Die Normenontologie enthält Begriffe und Beziehungen zur Erfüllung einer bestimmten Aufgabe, nämlich zur Repräsentation von Normen. Aus diesem Grund ist diese Ontologie am ehesten als Taskontologie zu klassifizieren.

(4) Das vierte Ontologiemodul importiert die Normenontologie und damit auch indirekt die Domänenontologie des KOVG. Die in der Ontologie modellierten Sachverhalte benötigen für

die Darstellung die Begriffe aus der Domänenontologie und die Normen aus der Normenontologie. Diese Ontologie enthält Begriffe und Beziehungen, die den Sachverhalt repräsentieren, und weist ein niedriges Abstraktionsniveau auf. Sie kann als Anwendungsontologie (application ontology) klassifiziert werden.

Der Zweck der gesamten Ontologie als Summe der einzelnen Module ist die Unterstützung der automatisierten Vollziehung. Um dieses Ziel zu erreichen und im Nachhinein die Qualität der Ontologie beurteilen zu können, empfiehlt es sich, für die Modellierung eine Reihe von Fragen aufzustellen, die das ontologische Modell beantworten soll (Noy & McGuinness, 2001). Der Ansatz folgt damit der von Fox, King, Gruninger und Uschold entwickelten Methode zur Entwicklung von Ontologien, die die Formulierung sogenannter „Competency Questions“ vorsieht (Gruninger & Fox, 1994; Uschold & Gruninger, 1996; Uschold & King, 1995; Uschold, 1996).

Im Wesentlichen ist für die Feststellung der zustehenden Leistungen die Beantwortung folgender Fragen (competency questions) notwendig:

1. Welcher Personengruppe gehört der Antragsteller aufgrund der festgestellten Minderung der Erwerbsfähigkeit an?
2. Welche Versorgungsleistungen gebühren dem Antragsteller grundsätzlich?
3. Wie hoch ist der Betrag der zustehenden Leistung?

Die Beantwortung dieser einfachen Fragen, insbesondere der letzten, ist im Detail relativ komplex. Fraglich ist, ob die Inferenzmaschine direkt die Antworten auf diese Fragen aus der Ontologie ableiten kann.

Der erste Problemkreis betrifft die Ermittlung und Ableitung der Rechtsfolgen aus der Ontologie durch die Inferenzmaschine. Eine besondere Herausforderung stellt die Modellierung und Durchführung der notwendigen Berechnungen für monetäre Versorgungsleistungen, insbesondere für die Beschädigtenrente, dar.

Die folgenden Untersuchungen werden zeigen, wie ausdrucksstark die Ontologie ist, und ob nicht ein hybrider Ansatz zielführender wäre. Komplexe Regeln könnten beispielsweise als Tripel in der Ontologie modelliert und dann von einem Javaprogramm ausgeführt werden. Mithilfe solcher Tripel ließen sich beispielsweise auch Formeln und ihre Terme ausdrücken,

ohne dass die Inferenzmaschine diese jedoch berechnen kann. Da das Gesetz jedoch viele Berechnungsvarianten enthält und die Beschreibungslogik keine Variablen zulässt, würde dies rasch zu einer Ausuferung von Formeln führen, was die Lesbarkeit des Modells beeinträchtigt.

Eine „einfache“ Modellierung der Formel ist nicht immer möglich, da für einzelne Teile der Berechnung auf Ergebnisse zurückgegriffen werden muss, die erst zur Laufzeit vorher berechnet wurden. Ebenso müssen Faktoren wie insbesondere das Einkommen des Antragstellers, die Anzahl der Familienangehörigen etc. berücksichtigt werden, um zum korrekten Ergebnis zu gelangen.

Eine wichtige Frage, die diese Dissertation beantworten soll, ist, inwiefern die Ontologie die automatisierte Subsumtion durch die Software unterstützen kann, und wo die Grenzen des ontologischen Modells in dieser Hinsicht liegen.

Soweit möglich soll bereits die Inferenzmaschine die grundlegende Klassifizierung durchführen und Informationen über die zu gewährenden Versorgungsleistungen liefern. Aufgrund der Tatsache, dass OWL eine rein logische Sprache ist, kann die Inferenzmaschine selbst ohne spezielle Erweiterungen keine Berechnungen durchführen. Die Auswirkungen dieser Feststellung werden im Folgenden thematisiert.

Zum einen müssen Berechnungen, die zur Feststellung monetärer Versorgungsleistungen wie z.B. der Beschädigtenrente notwendig sind, von der Software selbst durchgeführt werden. Zum anderen tritt diese Problematik bei der Subsumtion und der Rechtsfolgenermittlung in Erscheinung. Oftmals ist es erforderlich, dass Berechnungen durchgeführt werden um feststellen zu können, ob die jeweilige Versorgungsleistung gewährt werden soll[147]. In solchen Fällen ist es also eher unwahrscheinlich, dass bereits die Ontologie im weitesten Sinne[148] ein fertiges Ergebnis der Subsumtion liefern kann.

Bei derartigen Problemen wird es sich eher anbieten, dass die Software die Regeln für die rechtliche Beurteilung aus der Ontologie ableitet und diese schließlich auf die vorhandenen Fakten anwendet. Dies schließt nicht aus, dass die Software nach einer derartigen Operation

[147] Vgl. dazu insbes. § 12 Abs. 2 KOVG, der den Anspruch auf eine Zusatzrente von der Höhe des Einkommens des Antragstellers abhängig macht.

[148] Darunter sind an dieser Stelle die eigentliche Ontologie samt der verwendeten Programmbibliothek für den Zugriff und die Inferenzmaschine zu verstehen.

das Ergebnis als eine neue Aussage in die Ontologie einfügt, aufgrund derer neue Schlüsse von der Inferenzmaschine gezogen werden können, und so die Subsumtion schrittweise voran schreitet.

Etwas einfacher stellt sich das Problem auf Seite der Rechtsfolgenermittlung dar. Sobald festgestellt wurde, dass die jeweilige Versorgungsleistung zu gewähren ist, muss die Software die notwendigen Berechnungen durchführen, um den monatlich zu gewährenden Betrag festzustellen[149].

Das ontologische Modell präzise zu gestalten und dabei möglichst einfach zu halten, stellt sich als eine durchaus anspruchsvolle Aufgabe dar, die es zu lösen gilt. Es scheint darauf hinauszulaufen, dass einfache Subsumtionsvorgänge durchaus vom ontologischen Modell direkt durchgeführt werden können, während kompliziertere Fragestellungen, die zumeist mathematische Berechnungen voraussetzen, von der Software – aufgrund von in der Ontologie modellierten Regeln – beantwortet werden müssen.

Der zweite Problemkreis betrifft die Formalisierung von zeitlichen Aspekten. In dieser Hinsicht weisen die bisherigen Ansätze erhebliche Nachteile auf, da zeitliche Bedingungen nicht adäquat als Metaregeln repräsentiert werden. Die Ontologie muss mehrere Versionen einer Norm abbilden, damit Novellen in das Modell eingepflegt werden können. Dafür könnten einzelne Regeln mit einem Gültigkeitszeitraum versehen werden (Klarman et al., 2008).

10.5. Die Domänenontologie

Die Domänen-Ontologie des KOVG stellt die Begrifflichkeit zur Verfügung, die als Terminologie für die Modellierung der Normen und Sachverhaltsbeschreibungen verwendet wird.

Für die Modellierung eines Bereichs (der Domäne) ist es erforderlich, zuerst die wichtigsten Begriffe zu identifizieren. Welche Begriffe ausgewählt und wie die Ontologie strukturiert wird, ist maßgeblich vom Zweck abhängig, den das Modell erfüllen muss. Die erforderliche Anzahl von Begriffen kann im Vorhinein schwer abgeschätzt werden. Eine Proliferation der

[149] Vgl. dazu insbes. § 16 Abs. 1 KOVG, der dem Antragsteller für jeden Familienangehörigen monatlich eine Familienzulage in doppelter Höhe der Zusatzrente gewährt.

Begrifflichkeit sollte jedoch tunlichst vermieden werden, um das Modell nicht unnötig zu verkomplizieren.

Die Ontologie enthält unter anderem die Begrifflichkeit, die zur Beschreibung eines Antragstellers und dessen Beziehungen zu Angehörigen dient. Ebenso ist die Hierarchie zwischen dem Antragsteller und den Subklassen Beschädigter, Schwerbeschädigter und Erwerbsunfähiger abgebildet.

Das KOVG definiert grundsätzlich unterschiedliche Arten von Versorgungsleistungen. Zu diesen gehören unter anderem monetäre Versorgungsleistungen und berufliche sowie soziale Maßnahmen. Derzeit ist in der Ontologie nur die Klasse der monetären Versorgungsleistungen repräsentiert.

Abbildung 20 zeigt die wichtigsten Begriffe des KOVG im Überblick. Angemerkt sei, dass nur jene Begrifflichkeit repräsentiert wird, die für die Prüfung des Sachverhalts zur Ermittlung des Anspruchs auf Beschädigtenrente notwendig ist. Die Begrifflichkeit des gesamten KOVG wird jedoch bewusst aus Gründen der Vereinfachung nicht abgebildet.

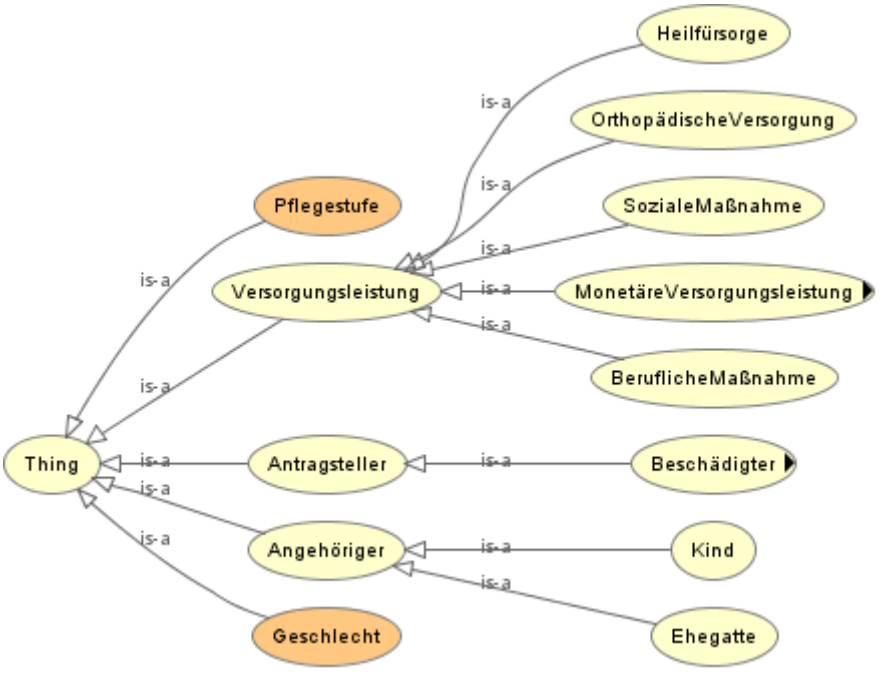

Abbildung 20: Domänenontologie des KOVG

10.5.1. Taxonomie

Die wichtigsten Begriffe, die identifiziert und aus dem Gesetz abgeleitet wurden, werden nun näher dargestellt.

Antragsteller: Jede natürliche Person, die einen Antrag auf Zuerkennung bestimmter Leistungen einbringt, wird als „Antragsteller“ klassifiziert. Die Klasse `Antragsteller` beinhaltet damit alle natürlichen Personen, die einen Antrag bei der Behörde einbringen.

Beschädigter (Subklasse von Antragsteller): Darunter sind vor allem Soldaten (im Sinne des KOVG) zu verstehen, die militärische Dienste geleistet und dadurch eine Gesundheitsschädigung („Dienstbeschädigung“) erlitten haben. Diese Personen haben Anspruch auf berufliche und soziale Maßnahmen (§§ 21 ff.), Heilfürsorge (§§ 23 ff.) und, sofern die Minderung der Erwerbsfähigkeit mindestens 20 % beträgt, auf Beschädigtenrente (§§ 7 ff.).

Schwerbeschädigter (Subklasse von Beschädigter): Beschädigte, deren Grad der Minderung der Erwerbsfähigkeit mindestens 50 % beträgt, werden vom Gesetz als „Schwerbeschädigte“ bezeichnet (§ 9 Abs. 2) und haben Anspruch auf diverse Erhöhungen der Beschädigtenrente.

Erwerbsunfähiger (Subklasse von Schwerbeschädigter): Schwerbeschädigte mit einer Minderung der Erwerbsfähigkeit ab 90 % gelten als erwerbsunfähig. Diese Personengruppe hat Anspruch auf sämtliche Versorgungsleistungen des KOVG.

Dienstbeschädigung: Eine Gesundheitsschädigung wird als Dienstbeschädigung anerkannt, wenn diese durch das schädigende Ereignis selbst oder zumindest durch die der Dienstleistung (als Soldat) eigentümlichen Verhältnisse verursacht wurde. Dies gilt grundsätzlich nicht, wenn das schädigende Ereignis selbst herbeigeführt wurde.

Versorgungleistung: Diese Klasse repräsentiert den abstrakten Oberbegriff für alle Versorgungsleistungen, die das KOVG normiert. Das Gesetz unterscheidet monetäre, berufliche und soziale sowie medizinische Versorgungsleistungen. Hinterbliebenen stehen nur monetäre Versorgungsleistungen zu.

MonetäreVersorgungsleistung: Die Klasse `MonetäreVersorgungsleistung` repräsentiert eine Kategorie von Versorgungsleistungen, die in Form von wiederkehrenden monatlichen Gelbeträgen gewährt werden. Diese Klasse wird weiter in die Kategorien `Rente` und `Zulage` unterteilt.

Rente: Klasse, die selbstständige monetäre Versorgungsleistungen, die sogenannten „Renten“, repräsentiert. Diese werden unabhängig von anderen Versorgungsleistungen – im

Unterschied zu Zulagen – dem Antragsteller zuerkannt, sofern die Voraussetzungen gegeben sind.

Beschädigtenrente: Die Klasse `Beschädigtenrente` (Subklasse von `Rente`) ist die wichtigste monetäre Versorgungsleistung.

Der Beschädigte hat Anspruch auf die „Beschädigtenrente“, wenn seine Erwerbsfähigkeit um mindestens 20 % v.H. vermindert ist. Diese Einschätzung ist nach Richtsätzen vorzunehmen, weitere Details sind aber für diesen Beitrag irrelevant.

Die Beschädigtenrente selbst besteht aus der Grund- und der Zusatzrente. Die Grundrente ist nach der Höhe der Erwerbsminderung abgestuft, wobei es diverse Zulagen gibt. Die Zusatzrente erhalten nur Schwerbeschädigte, die das achtzehnte Lebensjahr vollendet haben, und deren Einkommen nicht die Gewährung einer Zusatzrente ausschließt.

An dieser Stelle sei angemerkt, dass das KOVG zwei Arten monetärer Leistungen kennt. Einerseits gibt es Leistungen, die abhängig vom Einkommen des Beschädigten sind, und wiederum solche, die unabhängig von diesem gewährt werden. Diese Differenzierung sollte ebenfalls im Modell zum Ausdruck gebracht werden.

Derzeit wird die Beschädigtenrente in der Ontologie noch nicht als eine Zusammensetzung aus Grund- und Zusatzrente repräsentiert. Für diesen Zweck bietet sich das Mereologiemodul der LKIF-Core Ontologie an, das für die Repräsentation des Verhältnisses von Teilen zu einem Ganzen geeignet ist.

Zulage: Das KOVG normiert unselbstständige monetäre Versorgungsleistungen, die zusätzlich zu einer Rente zugesprochen werden. Voraussetzung für die Zuerkennung einer Zulage ist stets ein Anspruch auf eine Rente.

In der Ontologie werden diese unselbstständigen Versorgungsleistungen von der Klasse `Zulage` erfasst. Die einzelnen Zulagen selbst, wie zum Beispiel die `Schwerstbeschädigtenzulage`, werden als Instanzen der Klasse `Zulage` modelliert.

An dieser Stelle sei erwähnt, dass es neben den, auch im Gesetz so genannten „Zulagen“ auch „Zuschüsse“[150] und „Pauschbeträge“ gibt. Diese beiden Kategorien sind aber problemlos unter den Begriff „Zulage“ zu subsumieren, da sie ebenfalls von einer Rente

[150] Siehe § 14 Diätverpflegung und §20a Pauschbeträge für außergewöhnlichen Kleider- und Wäscheverbrauch.

abhängig sind und nicht selbstständig zugesprochen werden können. Die Begriffe `Blindenzulage`, `Familienzulage`, `Pflegezulage` und `Schwerstbeschädigtenzulage` sind Subklassen dieser Klasse.

Derzeit ist die Abhängigkeit einer Zulage von einer Rente in der Ontologie noch nicht modelliert und bietet damit Potential für zukünftige Verbesserungen.

10.6. Eine simple Ontologie zur Repräsentation von Normen

Dieser Abschnitt präsentiert ein ontologisches Modell zur Formalisierung des KOVG ohne die Verwendung der LKIF-Core Ontologie.

10.6.1. Die Klassifizierung des Antragstellers

Der erste Schritt bei der Ermittlung der Ansprüche ist die Klassifizierung des Antragstellers gemäß § 9 KOVG. Wie im Rahmen der Darstellung der Domänenontologie bereits erwähnt wurde, ordnet das KOVG den Antragsteller in die Kategorien Beschädigte, Schwerbeschädigte und erwerbsunfähige Schwerbeschädigte ein. Die einzelnen Kategorien unterscheiden sich durch die Höhe der Minderung der Erwerbsfähigkeit – Beschädigter >= 20, Schwerbeschädigter >= 50 und Erwerbsunfähiger >= 90.

Da beispielsweise ein Antragsteller der Kategorie „Schwerbeschädigte“ aufgrund der Minderung der Erwerbsfähigkeit gleichzeitig auch der Kategorie „Beschädigte“ zuzuordnen ist, werden die Kategorien durch eine Subklassenhierarchie repräsentiert. Ein Mitglied einer Subklasse hat immer auch Anspruch auf die Leistungen, die Mitgliedern der Superklasse gewährt werden. Diese durch die Normen zum Ausdruck gebrachte Gesetzmäßigkeit wird damit adäquat repräsentiert.

Um eine Kategorisierung des Antragstellers durch die Inferenzmaschine zu ermöglichen, ist es erforderlich, entsprechende Axiome zur Ontologie hinzuzufügen. Zu diesem Zweck wird für jede Kategorie eine Datatype Restriction für das Property `hatMdE` eingeführt und diese Klasse sodann mit einer Equivalent Class Restriction dem jeweiligen Begriff gleichgestellt.

Für die Darstellung der Axiome wird die relativ gut lesbare „Manchester OWL Syntax“ verwendet. Diese kommt ohne die in solchen Definitionen üblichen beschreibungslogischen Symbole aus, die Definitionen sind dadurch aber etwas länger. Für weitere Details zur verwendeten Syntax sei auf das Paper von Horridge et al. (2006) verwiesen.

Die vollständige Definition der Klasse `Beschädigter` lautet wie folgt:

```
Class: <http://js.at/diss/kovg#Beschädigter>
    EquivalentTo:
        <http://js.at/diss/kovg#hatMdE> some xsd:short[>= 20]
    SubClassOf:
        <http://js.at/diss/kovg#Antragsteller>
```

Gut erkennbar ist die Bedingung `hatMdE some short[>=20]` für die Kategorisierung des Antragstellers in die Klasse „Beschädigter“. Angemerkt sei, dass eine Datatype Restriction ein neues Sprachkonstrukt ist, das in OWL 2 eingeführt wurde. Das Object Property `hatMdE` repräsentiert den Grad der Minderung der Erwerbsfähigkeit (MdE).

Mithilfe solch einfacher Axiome lässt sich die grundlegende Klassifizierung des Antragstellers mithilfe der Beschreibungslogik bewerkstelligen. Der Typ einer Instanz kann von der Inferenzmaschine automatisch den beschriebenen Klassen zugeordnet werden und steht sodann für weitere Schlussfolgerungen in der Ontologie zur Verfügung.

Hat eine Person beispielweise eine Minderung der Erwerbsfähigkeit von 30 %, dann greift die definierte Restriktion „MdE >= 20“ und der Antragsteller wird der Klasse `Beschädigter` zugeordnet.

10.6.2. Die automatisierte Feststellung der Ansprüche

10.6.2.1. Die Repräsentation der Anspruchsgrundlage

Die Repräsentation der Normen, die die Anspruchsgrundlage für eine Leistung darstellen, erfolgt als eine Reihe von Bedingungen in konjunktiver Normalform, die gemeinsam eine Beschreibung der erfassten Situation bilden. Die Bedingungen werden durch Klassenaxiome (anonyme Klassen, definiert durch Restriktionen) repräsentiert, die Klassen und Properties verwenden, die in der Domänenontologie definiert sind. Dabei kann direkt an die im vorherigen Schritt durchgeführte Klassifizierung des Antragstellers angeknüpft werden, indem der erforderliche Typ des Antragstellers in die Restriktion aufgenommen wird. Das Axiom, das die Restriktion definiert, ist eine anonyme Klasse in OWL, die sodann einer benannten Klasse gleichgestellt wird, die die zugrundeliegende Norm darstellt.

Ein Sachverhalt wird als eine Menge statischer Aussagen modelliert, die den Sachverhalt repräsentieren. Die Subsumtion des Sachverhalts unter den Tatbestand der Norm erfolgt durch die beschreibungslogische Realisation, ausgeführt von der Inferenzmaschine. Ein

Sachverhalt wird von der Inferenzmaschine einer Norm zugeordnet, wenn dieser die Bedingungen der OWL Restriktion erfüllt.

Da der Tatbestand als gewöhnliche Klassenbeschreibung modelliert wird, kann dieser selbst mit anderen Tatbeständen Hierarchien bilden. Diese Hierarchien werden automatisch von der Inferenzmaschine generiert und spiegeln das Spezialitätsverhältnis zwischen Subklasse – der Lex specialis – und der Superklasse – der Lex generalis – wider. Diese Hierarchiebildung zwischen Tatbestandsbeschreibungen ist möglich, da jeder Tatbestand durch eine Restriktion in OWL repräsentiert wird. Eine Restriktion stellt selbst eine Klasse in OWL dar, die ihre Mitglieder anhand bestimmter Eigenschaften definiert. Die speziellere Restriktion hat definitionsgemäß zumindest um eine Bedingung mehr als die generelle und ist ansonsten ident. Die Instanzen, die die speziellere Restriktion erfüllen, sind damit zwangsweise auch Instanzen der generelleren Restriktion und repräsentieren damit eine Teilmenge. Diese Mengenbeziehung wird in OWL durch die Klassenhierarchie abgebildet.

Angemerkt sei jedoch, dass die Klassenhierarchie von Tatbestandsbeschreibungen zwar von einem Programm verwendet werden kann, um normative Konflikte zu lösen, jedoch die Ableitung der Rechtsfolgen der Lex generalis in der Ontologie nicht rückgängig gemacht werden kann. OWL basiert auf Beschreibungslogik, einer monotonen Logik, die es nicht erlaubt, bereits gezogene Schlüsse zu revidieren, sobald neue Informationen verfügbar sind.

Der erste Schritt der Anspruchsprüfung, nämlich die Subsumtion des Sachverhalts unter die Elemente des Tatbestands, ist damit abgeschlossen. Was wurde dadurch gewonnen? Anhand der Zuordnung zu einer Klasse wäre es grundsätzlich bereits möglich, dem Antragsteller eine bestimmte Leistung zuzusprechen. Die Attribute der zu gewährenden Leistung sind damit jedoch noch nicht ausreichend beschrieben, insbesondere wurde deren Höhe noch nicht festgelegt. Es ist jedoch nicht möglich, einer Klasse ein Data Property zuzuweisen, sodass dieser Ansatz noch erweitert werden muss. Der folgende Abschnitt widmet sich dieser Aufgabe.

10.6.2.2. Die Verbindung von Anspruchsgrundlage und Leistung

Das Modell muss also um Axiome zur Beschreibung der Leistung erweitert werden. Damit Aussagen über eine bestimmte Leistung getroffen werden können, muss diese als Instanz in der Ontologie repräsentiert werden. Dieser Instanz kann sodann beispielsweise ein

bestimmtes Data Property zugewiesen werden, das die Höhe der Geldleistung repräsentiert. Die Definition von Restriktionen hilft hier nicht weiter, da diese informell bloß zur Beschreibung der Mitglieder einer Klasse dienen.

Weiters ist erforderlich, dass aus der Typisierung einer Instanz ein Object Property abgeleitet werden kann, das auf eine Instanz verweist, die die jeweilige Leistung repräsentiert. Aus der Typisierung der Instanz I als Mitglied der Klasse K soll gefolgert werden: I hat Anspruch auf die Leistung L. Dazu ist es notwendig, eine Beziehung zwischen der Klasse K und der Leistung L herzustellen, die an sämtliche Instanzen von K weitergegeben wird.

Um die Verbindung zwischen einer Klasse und einer Instanz durch ein Object Property herzustellen, gibt es in OWL 2 prinzipiell zwei Möglichkeiten:

1. Punning[151]: Mit OWL 2 wurde die strikte Trennung zwischen Namen von Klassen und Instanzen gelockert, um damit die Möglichkeiten der Metamodellierung zu erweitern. In einer Ontologie darf damit ein Begriff als Bezeichnung einer Klasse und Instanz verwendet werden. Damit ist es möglich, ein Object Property Axiom einer Klasse zuzuweisen und diese wie eine Instanz zu behandeln.
2. Class-Individual Mirror Pattern (Allemang & Hendler, 2011, S. 243 ff.): Mithilfe einer `owl:hasValue` Restriktion wird eine Beziehung zwischen einer Instanz und einer Klasse hergestellt. Die Klasse beschreibt eine Menge von Dingen, die zu dieser Instanz auf bestimmte Weise in Beziehung stehen.

Obwohl es in OWL 2 grundsätzlich möglich ist, durch „Punning“, was so viel wie „Wortspiel“ bedeutet, einer Klasse den gleichen Namen zu geben wie einer Instanz, raten Allemang und Hendler ausdrücklich davon ab. Ein Modell sollte strikt zwischen einer Instanz und einer Menge dieser Instanzen unterscheiden. Die Wiederverwendbarkeit und Lesbarkeit eines Modells wird durch die Verwendung von Punning beeinträchtigt (Allemang & Hendler, 2011, S. 329).

Für den gegenständlichen Zweck ist jedoch ein anderer Aspekt wichtiger. Aus dem Einsatz von Punning lassen sich nämlich keine Inferenzen ableiten. Obwohl beispielsweise eine Klasse und eine Instanz syntaktisch gleich benannt werden können, sind sie semantisch getrennt zu betrachten (Grau, Horrocks, Parsia, Patel-Schneider, & Sattler, 2006, S. 6). Durch

[151] *W3C*, Punning. http://www.w3.org/TR/owl-new-features/#F12:_Punning aufgerufen: 29.4.2014.

den Einsatz von Punning lässt sich also aus der Mitgliedschaft einer Instanz in einer Klasse keine Aussage über die zustehende Leistung treffen. Es kann kein Object Property, das von der Instanz auf die Leistung verweist, abgeleitet werden.

Das Class-Individual Mirror Pattern lässt sich recht einfach implementieren. Es besteht aus einer einfachen „Has Value Restriction“, die durch ein `owl:subClassOf` oder `owl:equivalentClass` mit einer benannten Klasse in Beziehung gesetzt wird. Abbildung 21 soll dies veranschaulichen.

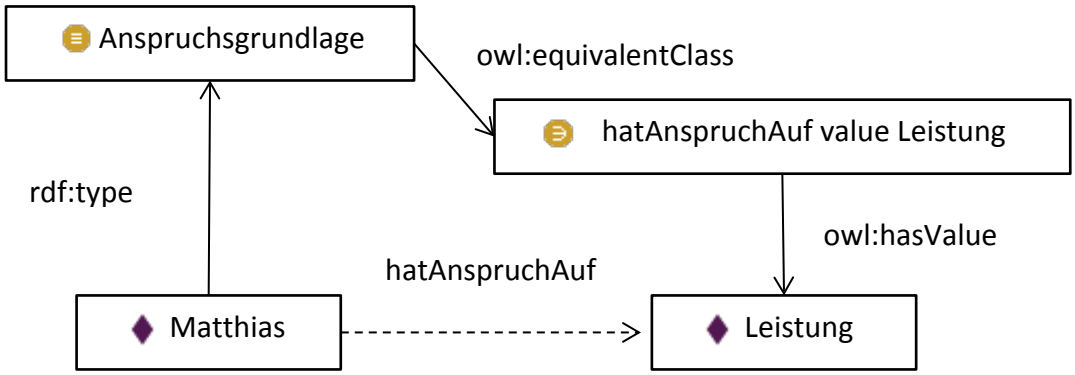

Abbildung 21: Class-Individual Mirror Pattern

Im Beispiel wird die Instanz „Matthias“ der Klasse „Anspruchsgrundlage“ zugeordnet. Informal ist die Klasse „Anspruchsgrundlage“ durch die Restriktion der Menge der Instanzen gleichgestellt, die das Object Property „hatAnspruchAuf“ mit dem Wert „Leistung“ besitzen.

Da die Instanz „Matthias“ vom Typ „Anspruchsgrundlage“ ist, wird von der Inferenzmaschine gefolgert, dass „Matthias“ das Property „hatAnspruchAuf“ mit dem Wert „Leistung“ besitzt. Der strichlierte Pfeil symbolisiert diese Ableitungsrelation.

Übertragen auf den gegenständlichen Anwendungsfall dieses Musters repräsentiert die Instanz „Matthias“ die Sachverhaltsbeschreibung – also den Antragsteller samt den relevanten Daten – und die Instanz „Leistung“ eine konkret zustehende Versorgungsleistung.

Angemerkt sei, dass in der Ontologie die Typisierung der Klasse „Matthias“ nicht statisch durch ein Axiom erfolgt, sondern durch eine Restriktion geschieht, die die Anspruchsgrundlage darstellt.

10.6.3. Beispiele

In diesem Abschnitt soll der vorgestellte Ansatz zur Formalisierung anhand einiger ausgewählter Beispiele erläutert und gleichzeitig evaluiert werden. Es sei darauf hingewiesen, dass die Paragrafen des KOVG nicht mit allen Details formalisiert werden. Dazu wären insbesondere Berechnungen erforderlich, die in einer rein logischen Sprache wie OWL nicht möglich sind.

Den Beispielen liegt folgender Sachverhalt zugrunde: Der Antragsteller Johannes ist männlich, 62 Jahre alt, hat eine Minderung der Erwerbsfähigkeit von 60 %, hat einen Pflegebedarf der Stufe II und ist blind. Der Antragsteller hat außerdem zwei Angehörige, nämlich seine Frau Christina und sein Kind Johanna.

Der Sachverhalt in OWL formalisiert stellt sich wie folgt dar (die Angabe der IRI entfällt zwecks Übersichtlichkeit):

```
Individual: <Johannes>
    Facts:
     <hatPflegestufe>  <Pflegestufe_III>,
     <hatEhegatten>  <Christina>,
     <hatKind>  <Johanna>,
     <hatGeschlecht>  <Maennlich>,
     <istBlind>  true,
     <hatAlter>  "62"^^xsd:short,
     <hatMdE>  "60"^^xsd:short
```

10.6.3.1. Die Grundrente gemäß § 11

Folgende Axiome zeigen die Formalisierung der Absätze eins und zwei des § 11 KOVG, der die Anspruchsvoraussetzungen der Grundrente normiert.

Zum besseren Verständnis seien die Axiome zur Formalisierung der Anspruchsgrundlage des § 11 Abs. 1 und 2 hier dargestellt:

```
§ 11 Abs. 1:
Class: <Art11_1_AnspruchGrundrente>
    EquivalentTo:
        <Beschaedigter>,
        <hatAnspruchAuf> value <Art11_1_Grundrente>
    SubClassOf:
        <Anspruch>
```

```
§ 11 Abs. 2:
Class: <Art11_2_ErhöhungGrundrente>
    EquivalentTo:
```

```
        <Schwerbeschädigter>
         and (((<hatGeschlecht> value <Maennlich>)
         and (<hatAlter> some xsd:short[>= 60]))
         or ((<hatGeschlecht> value <Weiblich>)
         and (<hatAlter> some xsd:short[>= 55]))),
        <hatAnspruchAuf> value <Art11_2_ErhöhungGrundrente>
    SubClassOf:
        <Anspruch>
```

Jeder Beschädigte hat Anspruch auf den Grundbetrag der Grundrente. Aus diesem Grund enthält die Restriktion der Klasse Art11_1_Grundrente bloß „Beschädigter“ als einzige Bedingung.

Für die Formalisierung der Anspruchsgrundlage für die Erhöhung der Grundrente gemäß § 11 Abs. 2 ist es nicht ausreichend, den Typ „Schwerbeschädigter“ vorauszusetzen, sondern darüber hinaus müssen das Alter und das Geschlecht in die Restriktion aufgenommen werden. Die etwas kompliziert anmutende Definition der Restriktion kommt daher, dass für Männer und Frauen ein unterschiedliches Alter zu berücksichtigen ist.

Wenn eine Instanz mit ihren Properties eine der Restriktionen erfüllt, wird diese automatisch gemäß der Definition des Axioms der äquivalenten Klasse zugeordnet. Im Fall der oben stehenden Formalisierung des § 11 Abs. 1 ist das einfach. Sobald eine Instanz der Klasse „Beschädigter“ angehört, wird diese auch der Klasse „Art11_1_AnspruchGrundrente“ zugeordnet. Für die Zuordnung zur Klasse „Art11_2_ErhöhungGrundrente“ muss neben dem Typ „Schwerbeschädigter“ auch der restliche Teil der Restriktion erfüllt sein.

10.6.3.2. Die Familienzulage gemäß § 16

Für die Ermittlung des Anspruchs auf Familienzulage ist es notwendig, dass der verwendete Formalismus numerische Vergleiche beherrscht. Wie am Anfang dieses Kapitels in der axiomatischen Definition des Sachverhalts ersichtlich, beherrscht die konstruierte Domänenontologie eine primitive Repräsentation von familiären Beziehungen.

Die Instanz, die den Antragsteller symbolisiert, ist mit den entsprechenden Properties `hatEhegatten` und `hatKind` mit seinen Angehörigen verbunden. Hierbei handelt es sich um eine vereinfachte Repräsentation, da grundsätzlich nur jene Kinder bei der Berechnung der Familienzulagen zu berücksichtigen sind, die noch nicht volljährig sind. Das Alter wird aber bei der gegenständlichen Repräsentation vernachlässigt.

§ 16 KOVG wurde wie folgt in OWL formalisiert:

```
Class: <Art16_Familienzulage>
    EquivalentTo:
        <Schwerbeschädigter>
         and (<hatAngehörigen> min 1 <Angehöriger>),
         hatAnspruchAuf> value <Art16_Familienzulage>
    SubClassOf:
        <Anspruch>
```

Das wichtigste Axiom in dieser Klassendefinition ist die Datatype Restriction, die erfüllt ist, wenn eine Instanz zumindest ein Property `hatAngehörigen` aufweist. Die bei der Definition des Sachverhalts verwendeten Properties `hatEhegatten` und `hatKind` sind Subproperties von `hatAngehörigen`.

An diesem Beispiel zeigt sich, dass OWL zwar grundsätzlich „zählen“ kann, es ist jedoch nicht möglich, pro Angehörigem einen Anspruch auf Familienzulage abzuleiten. Nach § 16 KOVG gebührt nämlich für jeden Angehörigen ein bestimmter Betrag als Familienzulage. Spricht man von der „Familienzulage“ im Sinne des Gesetzes, bezeichnet dies in Wahrheit die Summe der einzelnen Familienzulagen.

10.6.3.3. Die Pflege- und Blindenzulage gemäß der §§ 18 und 19

Die Formalisierung der Pflege- und Blindenzulage soll als Beispiel der Abbildung einer Lex-Specialis-Relation zwischen zwei Normen dienen.

Zur Beschädigtenrente wird nach § 18 eine Pflegezulage gewährt, wenn der Beschädigte für lebenswichtige Verrichtungen der Hilfe einer anderen Person bedarf. Gemäß § 19 ist Blinden an Stelle der Pflegezulage eine Blindenzulage zu leisten. Die Leidenszustände bei Blindheit sind dabei bestimmten Pflegestufen gleichgestellt.

Auch wenn hier § 19 eine explizite Ausnahme zu § 18 enthält und demnach rechtsdogmatisch kein Fall der lex specialis gegeben ist, soll zu Demonstrationszwecken § 19 als lex specialis formalisiert werden. Selbst wenn dies nicht ganz der gesetzlichen Formulierung entspricht, wird angenommen, dass der Tatbestand des § 19 dieselben Tatbestandsmerkmale wie § 18, nämlich den grundsätzlichen Anspruch auf Beschädigtenrente und den „Pflegebedarf“, ausgedrückt durch eine Pflegestufe, aufweist. Darüber hinaus verlangt § 19 zusätzlich, dass der Beschädigte blind ist.

Ausgehend von dieser Darstellung stellt sich die Formalisierung der §§ 18 und 19 wie folgt dar:

```
Class: <Art18_Pflegezulage>
    EquivalentTo:
        (<hatPflegestufe> some <Pflegestufe>)
         and (<hatAnspruchAuf> value <Art11_1_Grundrente>),
        <hatAnspruchAuf> value <Art18_Pflegezulage>
    SubClassOf:
        <Anspruch>

Class: <Art19_Blindenzulage>
    EquivalentTo:
        <hatAnspruchAuf> value <Art19_Blindenzulage>,
        (<hatPflegestufe> some <Pflegestufe>)
         and (<hatAnspruchAuf> value <Art11_1_Grundrente>)
         and (<istBlind> value true)
    SubClassOf:
        <Anspruch>
```

Gut erkennbar ist die grundlegende Bedingung `<hatAnspruchAuf> value <Art11_1_Grundrente>`. Hier zeigt sich der eingangs erwähnte Vorteil der ontologischen Formalisierung, dass neue Informationen, die von der Inferenzmaschine abgeleitet werden, verwendet werden können. In diesem Fall wird die formulierte Restriktion erst erfüllt, sobald aus der Ontologie „Anspruch auf Grundrente“ abgeleitet wurde.

Weiter zeigt das Beispiel, dass die Restriktion von § 18 exakt ident ist mit jener des § 19, mit der Ausnahme, dass die zusätzliche Bedingung `<istBlind> value true` hinzugefügt wurde. Wie in Abschnitt 10.6.2.2. ausgeführt wurde, bildet die Inferenzmaschine aus diesem Grund eine Hierarchie zwischen den §§ 18 und 19. Im Detail leitet die Inferenzmaschine aus der Ontologie ab, dass `Art19_Blindenzulage` eine Subklasse von `Art18_Pflegezulage` ist. Ein Programm, das sich der Ontologie bedient, könnte diese Klassenhierarchie zur Lösung des scheinbaren Konflikts zwischen § 18 und § 19 verwenden, indem dieses die Lex-Specialis-Regel anwendet.

Explizite Ausnahmen zwischen Normen, die das Gesetz normiert, können auf diese Art und Weise nicht aufgelöst werden, da die Inferenzmaschine keine Hierarchie ableiten kann. Selbstverständlich könnte man sich in diesen Fällen mit Properties wie „hatAusnahme“ oder „istAusnahmeVon“ behelfen, die wiederum vom Programm entsprechend verarbeitet werden können.

10.7. Die Verwendung der LKIF-Core Ontologie

Der in vorherigen Abschnitten entwickelte „einfache“ Ansatz funktioniert zwar grundsätzlich recht gut, ist aber doch mit einigen Unzulänglichkeiten behaftet.

Zentrales Element des Ansatzes ist die Formalisierung der Anspruchsgrundlage, aus der schließlich die zustehende Leistung abgeleitet wird. Die Anspruchsgrundlage wird durch eine Klasse mit einer zugehörigen Restriktion repräsentiert. Die Restriktion beschreibt alle Mitglieder der Klasse abschließend im Sinne von „necessary and sufficient conditions“. Eine Inferenzmaschine kann dadurch die impliziten Instanzen dieser Klasse aus der Ontologie ableiten.

Bei genauerer Betrachtung fällt auf, dass die Anspruchsgrundlage nicht zwischen der eigentlichen Norm und der normativen Aussage (der Bedeutung der Norm) unterscheidet. Dies erlaubt es nicht, über die eigentliche Norm als Objekt getrennt von deren normativer Bedeutung Aussagen zu machen. Beispielsweise könnte es nützlich sein, Axiome über den zeitlichen Anwendungsbereich einer Norm zur Ontologie hinzuzufügen, damit dieser von einer Inferenzmaschine berücksichtigt werden kann.

Eine Norm trägt grundsätzlich zwei Bedeutungen. Sie ist *deontisch*, da sie ein Verhalten als rechtlich akzeptabel oder inakzeptabel qualifiziert. Darüber hinaus ist die Norm *befehlend*, da sie den Adressaten zu einem rechtlich gebotenen Verhalten unter Androhung einer Sanktion verpflichtet. Diese Bedeutungen müssen nicht zwingend gemeinsam auftreten. Es ist möglich, ein Verhalten als rechtlich inakzeptabel zu qualifizieren, ohne eine Sanktion zu verhängen. Ebenso ist es umgekehrt zulässig, eine Anordnung aufgrund einer bloßen Warnung zu erlassen, die nicht die Qualität einer Rechtsnorm aufweist. Ein Formalismus sollte demnach zwischen der deontischen Qualifikation eines Verhaltens und der Anordnung von Maßnahmen zur Beseitigung eines rechtswidrigen Zustandes unterscheiden (Breuker et al., 2006, S. 173).

Im Hinblick auf die automatisierte Beurteilung von Leistungsansprüchen bedeutet dies, dass zwischen der rechtlichen Qualifikation des Sachverhalts und den daraus resultierenden Rechtsfolgen unterschieden werden muss. Beide Aspekte müssen im Modell adäquat repräsentiert werden.

Im Kontext dieser Ausführungen erscheint die separate Repräsentation der deontischen Qualifikation des Sachverhalts und der Leistung zielführend. Diese Modellierungsform wurde auch im „einfachen“ Ansatz (siehe Abschnitt 10.6.) gewählt, und unterscheidet theoretisch korrekt zwischen der deontischen Qualifikation des Sachverhalts und der daraus resultierenden Sanktion.

Wie bereits an anderer Stelle in dieser Arbeit ausgeführt wurde, lassen sich grob konstitutive (insbesondere Legaldefinitionen) und präskriptive (Verbote, Gebote, Erlaubnisse) Normen unterscheiden. Die folgenden Kapitel befassen sich mit der Formalisierung dieser beiden Normtypen unter Verwendung der LKIF-Core Ontologie. Die erwähnten Unzulänglichkeiten hinsichtlich der Formalisierung von Norm und Sachverhalt sollen durch eine adäquate Repräsentation ausgeräumt werden.

10.7.1. Die Formalisierung von Legaldefinitionen

Bislang wurde die in § 9 enthaltene Legaldefinition von „Schwerbeschädigten“ und „erwerbsunfähigen Schwerbeschädigten“ relativ einfach durch eine Klasse und eine äquivalente Restriktion repräsentiert. Hier mangelt es an der rechtstheoretisch korrekten Trennung zwischen der Norm als Objekt und den daraus resultierenden Rechtsfolgen.

Um diese Limitierung zu überwinden, wurde eine Erweiterung der LKIF-Core Ontologie zur Formalisierung von Legaldefinitionen entwickelt. Diese Erweiterung besteht aus zwei Klassen und zusätzlichen Object Properties.

Zur Repräsentation von konstitutiven Rechtsvorschriften wurde die Klasse `Definition` als Subklasse von `Norm` wie folgt definiert:

```
Class: <http://js.at/diss/kovg-norms#Definition>
    EquivalentTo:
        (<http://js.at/diss/kovg-norms#defines> some
<http://js.at/diss/kovg-norms#Defined>)
         and (<http://js.at/diss/kovg-norms#defines> only
<http://js.at/diss/kovg-norms#Defined>)
    SubClassOf:
        <http://www.estrellaproject.org/lkif-core/norm.owl#Norm>
```

Das Spiegelbild der Definition manifestiert sich in der Klasse `Defined` durch folgende Axiome:

```
Class: <http://js.at/diss/kovg-norms#Defined>
    EquivalentTo:
```

```
        <http://js.at/diss/kovg-norms#defined_by> some
<http://js.at/diss/kovg-norms#Definition>
    SubClassOf:
        <http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified>
```

Mithilfe dieser Axiome wird beispielsweise die Klasse `Schwerbeschädiger` wie folgt definiert:

```
Class: <http://js.at/diss/kovg#Schwerbeschädigter>
    EquivalentTo:
        <http://js.at/diss/kovg#hatMdE> some xsd:short[>= 50]
    SubClassOf:
        <http://js.at/diss/kovg-norms#defined_by> value
<http://js.at/diss/kovg-norms#Art9>
```

Aus dem SubClassOf Axiom folgert die Inferenzmaschine die gewünschte Ableitung, dass die Klasse `Schwerbeschädigter` durch die Norm `Art9` definiert wird. Durch die Verwendung des Properties `defined_by` wird die Verbindung zwischen der Norm `Art9` und deren Rechtsfolge – das ist die Legaldefiniton – hergestellt.

Da das Property `defines` die Umkehrung von `defined_by` repräsentiert und die Klasse `Definition` äquivalent zu einer entsprechenden Restriktion ist, wird für die Instanz `Art9` der Typ `Definition` abgeleitet.

Schließlich ergibt sich aus den angeführten Axiomen, dass `Schwerbeschädigter` eine Subklasse von `Defined` ist. Diese Relation drückt aus, dass die Klasse `Schwerbeschädigter` von einer Legaldefinition definiert wurde.

Angemerkt sei, dass die Klassenaxiome `Definition` und `Defined` von allgemeiner Nützlichkeit sind und daher grundsätzlich in einer Kernontologie modelliert werden sollten. Aus Gründen der Zweckmäßigkeit enthält jedoch die anwendungsspezifische Normenontologie diese beiden Axiome direkt.

10.7.2. Die Formalisierung von präskriptiven Normen und Sachverhalten

Dieser Abschnitt befasst sich mit der Formalisierung von präskriptiven Normen. Die Norm als Objekt, die deontische Qualifikation des Sachverhalts und die daraus abgeleitete Sanktion werden in Einklang mit der theoretischen Grundlage als separate Einheiten repräsentiert.

Eine Norm wird als deontische Qualifikation eines „typischen Falls“ (Generic Case, abgekürzt GC) (Valente, 1995, S. 90 ff.) repräsentiert. Ein GC ist eine Menge von Bedingungen

$\sigma_1 \sqcap \ldots \sqcap \sigma_n$ in konjunktiver Normalform, die gemeinsam eine Beschreibung des Sachverhalts bilden, der von der Norm erfasst wird. Die Bedingungen sind Klassenaxiome, die aus Klassen und Properties bestehen, die in der Domänenontologie definiert wurden.

Ein „individueller Fall“ (Case, abgekürzt C) ist eine Menge von Aussagen $\{c_1, \ldots, c_n\}$, genannt Umstände, die einen bestimmten Sachverhalt beschreiben. C repräsentiert demnach den Sachverhalt.

Die Übereinstimmung zwischen dem GC und C wird von der Inferenzmaschine durch den beschreibungslogischen Vorgang der Realisation überprüft. Eine Übereinstimmung wird erzielt, wenn ein Teil der Sachverhaltsbeschreibung als eine der Bedingungen des GC einer Norm klassifiziert werden kann. Diese Klassifizierung beinhaltet auch alle „Erweiterungen“, die auf die Sachverhaltsbeschreibung angewendet werden können.

Um die rechtliche Beurteilung durchzuführen, muss ein Fall normativ qualifiziert werden. Für diesen Zweck wird zwischen der deontischen Qualifikation einer Norm (Erlaubnis, Verbot, Gebot) und der Qualifikation eines konkreten Sachverhalts unterschieden.

Für eine kürzere Darstellung der Axiome wird die beschreibungslogische Syntax verwendet (Baader, Horrocks, & Sattler, 2008, S. 167).

Die Klasse `Norm` und deren Subklassen `Permission`, `Obligation` und `Prohibition` sind in der LKIF-Core Ontologie wie folgt definiert:

```
Norm ⊑ Qualification ⊓ ∃qualifies.Normatively_Qualified
Permission ⊑ Norm ≡ ∃allows.Allowed ⊓ ∀allows.Allowed
Obligation ⊑ Permission
≡ ∃allows.Obliged ⊓ ∃disallows.Disallowed ⊓ ∀allows.Obliged
⊓ ∀disallows.Disallowed
Prohibition ≡ Obligation
```

Die Klassen `Prohibition` und `Obligation` sind äquivalent, da sie zwei verschiedene Arten bilden um dasselbe auszudrücken. Zum Beispiel ist ein Verbot zu rauchen eine Verpflichtung nicht zu rauchen. Eine `Permission` hingegen erlaubt etwas, verbietet jedoch nichts.

Die in der LKIF-Core Ontologie definierten Axiome eignen sich dazu, einen Sachverhalt dahingehend zu beurteilen, ob dieser erlaubt, verboten oder geboten ist. Je nachdem, ob das jeweilige Gesetz standardmäßig eine bestimmte Situation erlaubt oder verbietet, sind

bei der Modellierung Besonderheiten zu beachten (van de Ven, Breuker, et al., 2008; van de Ven, Hoekstra, et al., 2008). Da in diesem Fall der Schwerpunkt auf der Feststellung der gebührenden Leistungen liegt, soll an dieser Stelle nicht weiter darauf eingegangen werden.

Die Normen, die Anspruchsgrundlagen für Leistungen darstellen, werden in diesem Fall als „Erlaubnisse“ begriffen. Sie gewähren dem Antragsteller das Recht, eine bestimmte Leistung in Anspruch zu nehmen.

Der oben erwähnte „typische Fall“ (GC) repräsentiert in diesem Fall die Beschreibung der Anspruchsgrundlage. Wird der GC durch die beschriebenen Axiome von einer Norm als „erlaubt“ qualifiziert, bedeutet dies, dass grundsätzlich Anspruch auf die jeweilige Leistung besteht. Aus der Ontologie soll jedoch nicht bloß abgeleitet werden, dass eine bestimmte Leistung zusteht, sondern auch weitere Details über diese. Solch ein Detail ist beispielsweise die Höhe der Leistung in Euro.

Um aus der Übereinstimmung einer Norm mit einem GC die zustehende Leistung aus der Ontologie abzuleiten, gibt es prinzipiell zwei Möglichkeiten. Erstens könnte die Instanz, die die jeweilige Norm als Objekt repräsentiert, durch ein entsprechendes Property mit der jeweiligen Leistung verbunden werden. Zweitens könnte die zustehende Leistung direkt aus der deontischen Qualifikation des Sachverhalts abgeleitet werden.

Der zweite Ansatz erscheint aus theoretischer Sicht richtiger, da die Voraussetzungen für den Anspruch auf eine Leistung durch die normative Aussage (den Sinngehalt der Norm) und nicht durch die Norm als Objekt selbst zum Ausdruck gebracht werden.

Folgendes Muster wird für die Formalisierung einer Norm als Anspruchsgrundlage für eine bestimmte Leistung verwendet:

$\mathrm{Norm}_1 \sqsubseteq \mathrm{Permission} \equiv \exists \mathrm{rechtsgrundlageVon}.\{\mathrm{Leistung}_1\}$

Um direkt vom Einzelfall auf die zustehenden Leistungen schließen zu können, ist die Definition eines weiteren sogenannten Property Chain Axioms erforderlich:

$\mathrm{allowed_by} \circ \mathrm{rechtsgrundlageVon} \sqsubseteq \mathrm{hatAnspruchAuf}$

Durch ein Property Chain Axiom lässt sich ein Property als Zusammensetzung anderer Properties definieren. Informal ausgedrückt wird aus der Beziehung von A zu B und B zu C das Verhältnis von A zu C abgeleitet[152].

Die Norm als Objekt mit Eigenschaften wird wie bei der Formalisierung einer Legaldefinition als einfache Instanz dargestellt, die vorzugsweise denselben Namen wie der repräsentierte Paragraf trägt. Der Instanz können beliebige weitere Properties zugeordnet werden. Es wäre beispielsweise möglich, auf eine elektronische Version des Rechtstextes durch einen Uniform Resource Identifier (URI) zu verweisen.

Bevor näher auf das Zusammenspiel der Axiome zur Beurteilung eines Falls eingegangen wird, seien die wichtigsten Komponenten dieses Ansatzes noch einmal zusammengefasst. Der präsentierte Ansatz stützt sich auf folgende Elemente:

1. Die Norm wird als deontische Qualifikation eines Sachverhalts in Form einer Klasse mit äquivalenter Restriktion repräsentiert. An dieser Klasse knüpft auch das Axiom zur Ableitung der zustehenden Leistung an.
2. Der Tatbestand der Norm wird als eine Beschreibung des Sachverhalts als „Generic Case“ in Form einer Klasse mit einer äquivalenten Restriktion modelliert.
3. Die Norm als Objekt wird durch eine Instanz in der Ontologie mit gleichem Namen repräsentiert.

Die Funktionsweise dieses Ansatzes wird nun anhand von § 11 Abs. 1 demonstriert, der den Grundbetrag der Grundrente normiert. Die deontische Qualifikation des Sachverhalts Art11_1_GC wird wie folgt modelliert:

```
Class: <http://js.at/diss/kovg-norms#Art11_1_Permission>
    EquivalentTo:
        <http://js.at/diss/kovg-norms#rechtsgrundlageVon> value
<http://js.at/diss/kovg#Art11_1_Grundrente>,
        {<http://js.at/diss/kovg-norms#Art11_1>}
    SubClassOf:
        <http://www.estrellaproject.org/lkif-
core/norm.owl#Permission>,
        <http://www.estrellaproject.org/lkif-core/norm.owl#allows>
only <http://js.at/diss/kovg-norms#Art11_1_GC>
```

[152] *W3C*, Property Chain Inclusion. http://www.w3.org/TR/owl-new-features/#F8:_Property_Chain_Inclusion aufgerufen: 9.5.2014.

Wie ersichtlich, ist die Klasse Art11_1_Permission eine Subklasse von Permission und repräsentiert damit eine „Erlaubnis“. Durch eine Equivalent Class Restriction wird die Klasse Art11_1_Permission der Instanz Art11_1, die die eigentliche Norm als Objekt repräsentiert, gleichgestellt.

Mithilfe des im Beispiel verwendeten Axioms wird im Ergebnis die Verbindung zwischen einem Sachverhalt und der gebührenden Leistung hergestellt, sobald ersterer vom GC der Norm erfasst wird:

Art11_1_Permission ≡ ∃rechtsgrundlageVon.{Art11_1_Grundrente}

Der GC für § 11 Abs. 1 wird durch folgende Axiome definiert:

```
Class: <http://js.at/diss/kovg-norms#Art11_1_GC>
    EquivalentTo:
        <http://js.at/diss/kovg#Beschädigter>
    SubClassOf:
        <http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by> value <http://js.at/diss/kovg-
norms#Art11_1>,
        <http://js.at/diss/kovg-norms#Sachverhaltsbeschreibung>
```

Die Tatbestandsmerkmale, die gleichzeitig die Anspruchsvoraussetzungen darstellen, werden mithilfe einer Equivalent Class Restriction beschrieben. In diesem Fall ist die einzige Anspruchsvoraussetzung, dass der Antragsteller der Klasse Beschädigter zuordenbar ist. Diese Zuordnung wird durch ein anderes Axiom erreicht, das hier nicht dargestellt wird.

Durch ein Sub Class Of Axiom wird die Klasse Art11_1_GC als eine Subklasse der durch die Restriktion ∃allowed_by.{Art11_1} beschriebenen Klasse aller Instanzen, die von {Art11_1} erlaubt werden, deklariert. Das Property allowed_by ist dabei als die Umkehrung von allowed definiert.

Wenn nun ein Fall C vom GC erfasst wird, sorgt das Subklassenaxiom dafür, dass die Aussage C ⊑ ∃allowed_by.{Art11_1} abgeleitet werden kann. Durch das zur deontischen Qualifikation gehörende Axiom Art11_1_Permission ≡ {Art11_1} ⊓ ∃rechtsgrundlageVon.{Art11_1_Grundrente} kann schließlich die Aussage {Art11_1} ≡ ∃rechtsgrundlageVon.{Art11_1_Grundrente} abgeleitet werden.

Mithilfe der beschriebenen Axiome lassen sich schließlich folgende Relationen ableiten:

{C} allowed_by {Art11_1} rechtsgrundlageVon {Art11_1_Grundrente}

Durch das weiter oben beschriebene Property Chain Axiom `hatAnspruchAuf` lässt sich sodann die finale Konklusion `{C} hatAnspruchAuf {Art11_1_Grundrente}` ableiten.

Abbildung 22 zeigt, wie die Inferenzmaschine anhand der modellierten Axiome die dem Antragsteller zustehenden Leistungen ableitet. Der Antragsteller im Beispiel heißt Martin, ist 65 Jahre alt und hat eine Minderung der Erwerbsfähigkeit von 60 %. Diese Informationen repräsentieren den Sachverhalt und werden in der Ontolgie modelliert. Mithilfe der modellierten Normen folgert die Inferenzmaschine, dass dem Antragsteller Leistungen gemäß § 11 Abs. 1, 2 und 3 KOVG gebühren. Die strichlierten Rechtecke in Abbildung 22 beinhalten die von der Inferenzmaschine abgeleiteten Aussagen. Angemerkt sei, dass § 11 Abs. 3 dem Abs. 2 vorgeht. § 11 Abs. 3 ordnet explizit die Nichtanwendbarkeit von Abs. 2 an. Explizite Ausnahmen zwischen Normen können zwar modelliert werden, jedoch nicht direkt von der Inferenzmaschine interpretiert werden. Zur Verarbeitung expliziter Ausnahmebeziehungen zwischen Normen müsste die Software die in der Ontologie modellierten Beziehungen entsprechend interpretieren.

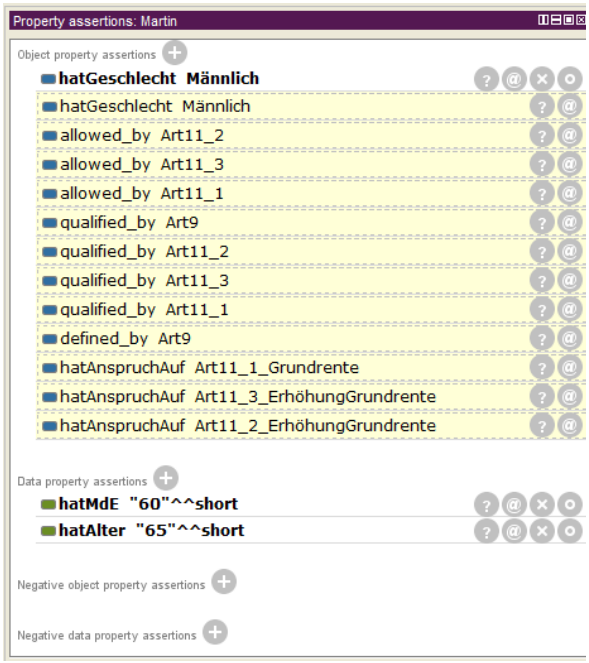

Abbildung 22: Die Ontologie in Aktion

10.7.3. LKIF-Core Light

Durch die Verwendung der LKIF-Core Ontologie für die Modellierung der Klassen und Instanzen erhöhte sich die Ausführungszeit für die Durchführung der Inferenz erheblich. Obwohl die eigene Ontologie relativ wenige Klassen und Instanzen enthält, stieg die Ausführungszeit auf etwa 80 Sekunden an (Intel i5 Quad-Core-CPU @ 2,40 GHz unter Windows 7 32 Bit). Die „einfache“ Ontologie ohne Verwendung von LKIF konnte immerhin in ca. 150 ms klassifiziert werden.

Als Reasoner wurde HermiT[153] 1.3.8 und Protégé 4.3 als Entwicklungsumgebung verwendet. Die anderen Reasoner, darunter Pellet 2.3.0 und Fact++ 1.6.2, konnten die Ontologie offenbar überhaupt nicht mehr klassifizieren oder die Ausführung hätte unbestimmte Zeit gedauert. Die Versuche wurden jedenfalls nach ca. zwei Minuten ohne Ergebnis abgebrochen.

Dieses Ergebnis wurde nicht erwartet, da frühere Versionen von Pellet und Fact++ hinsichtlich der Ausführungszeit HermiT um nichts nachstanden und sogar teilweise schneller waren (Dentler, Cornet, ten Teije, & de Keizer, 2011).

Die lange Ausführungszeit stellte ein erhebliches Problem während der Entwicklung dar. De facto war es unmöglich, Änderungen der Ontologie durch Ausführung der Inferenzmaschine sofort zu verifizieren.

Als Lösung wird gelegentlich die Deklaration der Topkategorien der LKIF-Core Ontologie als „disjoint“ vorgeschlagen, sofern diese nicht benötigt werden. Dadurch verringert sich die Anzahl der Äste der Ontologie, die nach Klassen durchsucht werden müssen (Bron, El Ali, Ji, & Klarman, 2007, S. 6; Hoekstra et al., 2007, S. 60).

Dieser Lösungsansatz wurde ausgewählt und soweit wie möglich die Topkategorien der LKIF-Core Ontologie wie `Change`, `Agent`, `Medium` etc. als disjoint deklariert. Zusätzlich wurden die Klassen und Instanzen der KOVG Ontologie untereinander und mit den Begriffen der LKIF-Core Ontologie als unvereinbar definiert.

Durch diese Maßnahmen konnte die Ausführungszeit auf etwa 20 Sekunden reduziert werden. Auffällig war jedoch, dass die Dauer der Klassifizierung starken Schwankungen unterlag. Gelegentlich dauerte die Ausführung bis zu 60 oder sogar 70 Sekunden. Die Gründe dafür wurden nicht weiter erforscht. Unter Umständen waren diese Ausreißer durch eine gesteigerte Auslastung des Systems bedingt und sind nicht unmittelbar auf die Ontologie oder den Reasoner zurückzuführen.

Aufgrund der immensen Größe der LKIF-Core Ontologie erfolgte die Optimierung der Ontologie nach der Trial-And-Error-Methode. Gelegentlich kam es vor, dass durch die Unvereinbarkeitsaxiome die Ontologie inkonsistent wurde. Insgesamt stellte sich der Ansatz

[153] HermiT OWL Reasoner. http://hermit-reasoner.com/ aufgerufen: 9.5.2014.

als ziemlich zeitintensiv und komplex dar, ohne dass eine großer Verbesserung erzielt werden konnte.

Es scheint, als wäre das volle Potential dieses Ansatzes noch nicht ausgeschöpft und die Ontologie könnte weiter optimiert werden. Aus den genannten Gründen wurde jedoch ein pragmatischer Ansatz gewählt und eine vereinfachte Version der LKIF-Core Ontologie, genannt „LKIF-Core Light“ erstellt.

Die LKIF-Core Light Ontologie enthält nur jene Klassen und Properties aus der LKIF-Core Ontologie, die im konkreten Anwendungsfall aus der KOVG Ontologie benötigt werden. Im Wesentlichen handelt es sich um die Klassen `Norm` und `Qualified` samt Subklassen und zugehörigen Properties. Die Axiome wurden mithilfe des Protégé Editors aus der LKIF-Core Ontologie extrahiert und in die neue Ontologie kopiert.

Durch den Einsatz der LKIF-Core Light Ontologie konnte die Ausführungszeit für die Klassifizierung der Ontologie auf annehmbare ein bis zwei Sekunden gesenkt werden.

10.7.4. Abschließende Bemerkungen

Durch die Verwendung der LKIF-Core Ontologie zur Repräsentation von Rechtsnormen konnte die Qualität der Repräsentation gegenüber dem einfachen Ansatz gesteigert werden. Der Ansatz ist ausdrucksstark genug, um den Sachverhalt, dessen deontische Qualifikation, die Norm und die zustehenden Leistung zu repräsentieren und Aussagen über diese zu treffen.

Die LKIF-Core Ontologie gibt durch abstrakte Begriffe eine grundlegende Struktur für die Modellierung von Rechtswissen vor. Dadurch werden die Lesbarkeit und die Wiederverwendbarkeit des ontologischen Modells gesteigert. Überdies kann der entwickelte Ansatz so besser mit vorhandenen Ontologien, die auf der LKIF-Core Ontologie basieren, verglichen werden.

Gegenüber dem einfachen Ansatz, der ohne die Verwendung der LKIF-Core Ontologie auskommt, gibt es aber auch Nachteile, die im Folgenden diskutiert werden.

Die LKIF-Core Ontologie ist relativ umfangreich, sodass am Anfang der Modellierung eine relativ steile Lernkurve steht. Es ist schwierig, sich einen Überblick über die vorhandenen Klassen und die Beziehungen zwischen diesen zu verschaffen. Dies gilt umso mehr, wenn

komplexe Strukturen wie beispielsweise Rechtsgeschäfte mit mehreren Parteien modelliert werden sollen.

Der modulare Aufbau der LKIF-Core Ontologie vermag hier nur wenig Abhilfe zu verschaffen. In der Regel wird das Normmodul für die Erstellung juristischer Ontologien benötigt. Dieses Modul importiert aber wiederum das „Legal Action“ Modul und damit indirekt einen Großteil der LKIF-Module, wodurch der Vorteil der Modularisierung schwindet.

Ein weiterer aus praktischer Sicht nicht unbedeutender Nachteil ist die (sehr) lange Ausführungsdauer, die die Inferenzmaschine für die Klassifizierung benötigt. Dieses Problem stellt sich bald ein, wenn eigene Klassen als Subklassen der LKIF-Core Ontologie modelliert und eigene Instanzen der Begriffe verwendet werden. Durch zeitintensive Optimierungen lässt sich das Problem teilweise lösen. Für eine automatisierte rechtliche Beurteilung, die möglichst in Echtzeit erfolgen soll, sind aber bereits zehn bis fünfzehn Sekunden Ausführungszeit inakzeptabel.

Wie in Abschnitt 10.7.3. bereits angemerkt wurde, scheint es durchaus noch Potential für Optimierungen des ontologischen Modells zu geben. Es ist jedoch schwierig herauszufinden, durch welche Aspekte des Modells konkret die Ausführungszeit erhöht wird. Hier wäre analog zur Softwareentwicklung ein Debugger sinnvoll, ein Werkzeug, mit dessen Hilfe sich die Arbeit der Inferenzmaschine beobachten und gegebenenfalls messen lässt. Ohne solch eine Unterstützung kann die Optimierung nur durch Ausprobieren diverser Axiome durchgeführt werden.

Es scheint oftmals zufällig, welche Axiome zu einer Geschwindigkeitssteigerung führen. Jedenfalls lässt sich nicht immer nachvollziehen, warum ein Axiom zu einer Verbesserung führt und ein anderes nicht. Zu umfangreich ist die LKIF-Core Ontologie, um die anwendbaren Restriktionen erahnen zu können.

Durch den Einsatz einer leichtgewichtigen Variante der LKIF-Core Ontologie konnte ein Workaround für dieses Problem entwickelt werden. Benötigte Axiome wurden mithilfe der Entwicklungsumgebung aus der LKIF-Core Ontologie in die LKIF-Core Light Ontologie kopiert. Auch dieser Prozess ist relativ fehleranfällig, lässt sich aber im Gegensatz zu den Optimierungen mithilfe von Unvereinbarkeitsaxiomen in angemessener Zeit durchführen. Das Ergebnis ist überdies annehmbar. Die Ausführungszeit des ontologischen Modells wurde auf ca. zwei Sekunden gesenkt.

Der Einsatz der vereinfachten, leichtgewichtigen LKIF-Core Ontologie, genannt „LKIF-Core Light“, ist nur so lange möglich, als nicht ein Großteil der anderen Begriffe benötigt wird. Für komplexere Ontologien erscheint dieser Ansatz eher ungeeignet.

Es empfiehlt sich jedenfalls, die Begriffe und Relationen zwischen diesen in Grundzügen zu definieren und erst in einem zweiten Schritt diese dann den Strukturen der LKIF-Core Ontologie zuzuordnen. Die Grundstruktur des ontologischen Modells lässt sich leichter überblicken und die LKIF-Core Ontologie kann dann relativ einfach integriert werden, indem die vorhandenen Strukturen als Subklassen der Begriffe der Kernontologie definiert werden.

Um die abschließende Diskussion am Ende des Kapitels teilweise vorwegzunehmen, lässt sich insgesamt sagen, dass sich durch den Einsatz von Ontologien und Beschreibungslogik nur sehr einfache Subsumtionsvorgänge durchführen lassen. Selbst hier stößt die Beschreibungslogik jedoch relativ schnell an ihre Grenzen. OWL ist ein rein logischer Formalismus, mathematische Berechnungen liegen daher außerhalb der Möglichkeiten.

10.8. Eine hybride Architektur zur Unterstützung der rechtlichen Entscheidungsfindung

Wie im vorherigen Abschnitt ausgeführt wurde, besitzen Ontologien aufgrund ihrer beschreibungslogischen Grundlage nur eine begrenzte Ausdrucksstärke. Die Inferenzmaschine kann daher nur sehr einfache Fälle der rechtlichen Beurteilung durchführen.

Ontologien eignen sich eher für die strukturierte Darstellung von Sachverhalten, Regeln und Argumenten. Auf dieser Grundlage kann eine „oberflächliche“ Schlussfolgerung innerhalb der Grenzen von OWL 2 durchgeführt werden. Dies umfasst die Ableitung von Aussagen über Begriffe und deren Beziehungen.

Es erscheint vielversprechender, eine hybride Architektur anzustreben, die Regeln, Argumente etc. aus einem ontologischen Modell importieren kann und eine spezielle Inferenzmaschine für den jeweiligen Zweck verwendet.

Die Architektur eines solchen Systems muss dem Stand der Technik entsprechend einem modularen Aufbau folgen. Die Softwarekomponente könnte so in andere Systeme eingebettet werden, die über eine definierte Schnittstelle mit dem Modul kommunizieren.

Das Gesamtsystem folgt einem modularen Aufbau und setzt sich aus mehreren Modulen zusammen, die unterschiedliche Aufgaben erledigen. Die Module werden nach funktionalen Kriterien gegliedert und erfüllen einen bestimmten Zweck.

An dieser Stelle soll auf die technische Architektur dieses Moduls näher eingegangen werden. Das Modul selbst setzt sich aus mehreren Schichten zusammen, die unterschiedliche Abstraktionsniveaus darstellen. Die höhere Schicht kommuniziert mit der niedrigeren Schicht über Interfaces und ruft dabei Methoden derselben auf. Um eine möglichst lose Koppelung der einzelnen Schichten aneinander zu erreichen, hat die niedrigere keine Abhängigkeiten zur höheren Schicht. Eine Kommunikation von unten nach oben ist daher ausgeschlossen.

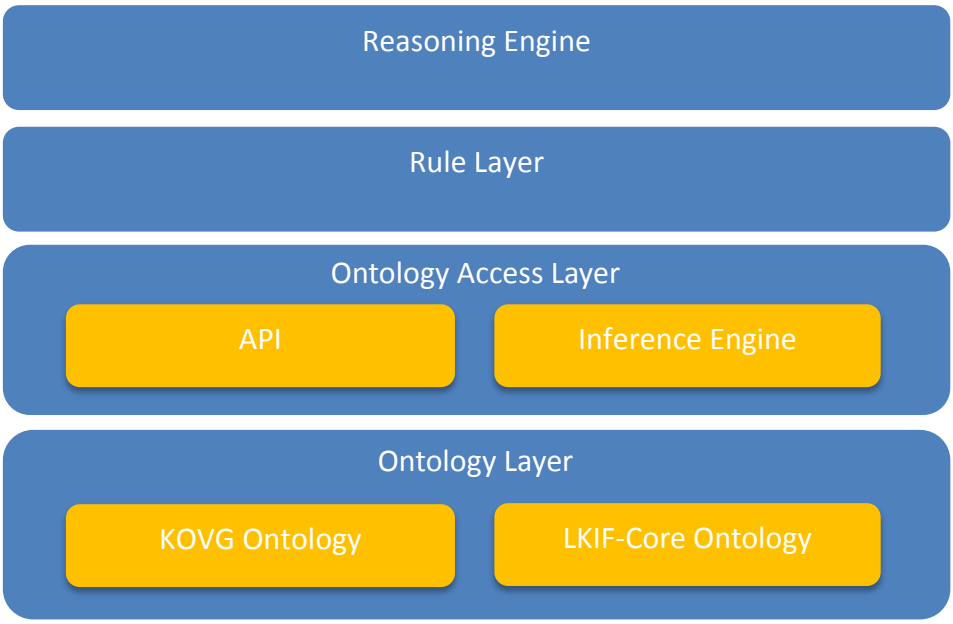

Abbildung 23: Architektur Reasoning Module

Abbildung 23 veranschaulicht die Architektur des Moduls zur Entscheidungsfindung, genannt „Reasoning Module". Im Folgenden werden die einzelnen Schichten (Layer) beginnend mit der untersten näher beschrieben.

Ontology Layer: Dieser Layer beinhaltet die Ontologie des KOVG, die in vorherigen Abschnitten dargestellt wurde, und die LKIF-Core Ontologie (Hoekstra et al., 2007). Die zuletzt genannte Ontologie definiert grundlegende rechtliche Begriffe, die für die Erstellung von fachspezifischen Domänenontologien dienlich sein können. Die „KOVG Ontology" selbst enthält die formalisierten Normen des KOVG und definiert somit die bereits skizzierten

Regeln für die Entscheidungsfindung. Dieser Layer ist demnach nicht ein Teil der Software selbst, sondern stellt eher eine externe Datenquelle dar, derer sich das Programm bedient.

Ontology Access Layer: Diese Schicht repräsentiert eine abstrakte Zugriffsschicht auf das darunterliegende ontologische Modell. Er stellt Methoden für den „Rules Layer“ bereit, um Regeln aus der Ontologie importieren zu können. Die konkrete Struktur der Ontologie und die modellierten Klassen werden dadurch abstrahiert.

Der Ontology Access Layer entkoppelt zusätzlich den Rules Layer von einer bestimmten Programmbibliothek für den Zugriff auf die Ontologie. Eine Programmbibliothek für die Arbeit mit Ontologien ist beispielsweise die OWL API für Java (Horridge & Bechhofer, 2011). Mithilfe dieser Bibliothek können Ontologien bearbeitet und mithilfe einer Inferenzmaschine Schlussfolgerungen durchgeführt werden. Es existieren mehrere Inferenzmaschinen wie Pellet, die zur OWL API kompatibel sind und die entsprechenden Schnittstellen implementieren. Die verwendete Inferenzmaschine lässt sich daher problemlos austauschen.

Die Kapselung der verwendeten Programmbibliothek ist sehr wichtig, da Bibliotheken für den Zugriff auf die Ontologie relativ komplex sind. Diese Komplexität soll verborgen werden und eine stärker funktionsbezogene Schnittstelle dem Rules Layer zur Verfügung gestellt werden.

Angemerkt sei, dass sich die vorhandenen Programmbibliotheken im Funktionsumfang unterscheiden. Die OWL API bietet beispielsweise keine Möglichkeit, um Abfragen in SPARQL abzusetzen. Es gibt jedoch Erweiterungen für die OWL API, die diese fehlende Funktionalität implementieren[154].

Rule Layer: Dieser Layer repräsentiert die Schnittstelle zwischen den eben dargestellten „ontologischen“ Schichten und der Reasoning Engine, die die rechtliche Subsumtion durchführt. Er beheimatet im Wesentlichen Regeln für die rechtliche Beurteilung samt eventuellen Metadaten wie Anwendungszeitraum, Priorität etc. Diese Abstraktion erfüllt mehrere Aufgaben.

[154] Bspw. die SPARQL-DL API. http://www.derivo.de/en/resources/sparql-dl-api.html aufgerufen: 13.5.2014.

Erstens ist der Rule Layer für die Repräsentation der Regeln und das Laden derselben aus dem ontologischen Modell zuständig. Der Reasoning Engine stehen damit fertige Regeln für die Entscheidungsfindung zur Verfügung.

Zweitens ist es möglich, dass einzelne Schritte der rechtlichen Problemlösung auf einer höheren, abstrakteren Ebene abgebildet werden als dies in der darunterliegenden Schicht möglich wäre. Damit wird auf Ebene des Programmcodes eine abstrakte Schnittstelle zur Lösung von Problemen der gegenständlichen Domäne bereitgestellt.

Ein derartiger Ansatz ermöglicht es, dass die Reasoning Engine mit den beschriebenen Regelobjekten gespeist wird und so unabhängig von den darunterliegenden Schichten operieren kann. Dies erleichtert das Testen der Reasoning Engine erheblich. Außerdem wäre es theoretisch möglich, dass die Regeln nicht (ausschließlich) aus einer Ontologie importiert werden, sondern zum Beispiel aus einer relationalen Datenbank. Es müsste dafür lediglich der Rule Layer entsprechend adaptiert und erweitert werden.

Die Schnittstelle, die der Rule Layer zur Verfügung stellt, ist vollkommen unabhängig von der darunterliegenden Technologie.

Reasoning Engine: In diesem Layer ist die eigentliche künstliche Intelligenz angesiedelt, die die rechtliche Subsumtion durchführt. Aufgabe der Reasoning Engine ist es, einen konkreten Sachverhalt mit den Rechtsnormen in Verbindung zu bringen und ein Ergebnis festzustellen.

Die Daten des Sachverhalts, die beispielsweise ein Dialogsystem gesammelt hat, werden der Reasoning Engine für diesen Zweck zur Verfügung gestellt. Erwähnt sei, dass diese Daten wiederum durch Objekte gekapselt und in dieser Form vom Programm verarbeitet werden. Die Daten, wie z.B. der Vorname des Antragstellers oder dessen Geburtsdatum, werden durch Attribute dieser Objekte repräsentiert.

Die Reasoning Engine muss nun anhand dieser Daten mit Hilfe der vom Rule Layer bereitgestellten Regeln die rechtliche Subsumtion durchführen. Im Kontext des KOVG müssen dabei insbesondere auch die notwendigen Berechnungen durchgeführt werden.

Das Ergebnis der rechtlichen Beurteilung wird wiederum durch entsprechende Objekte repräsentiert. Im Fall des KOVG beinhaltet das Ergebnis die Feststellung der zustehenden Leistungen und den jeweiligen Betrag. Diese Objekte können wiederum verwendet werden, um einen Bescheid zu generieren oder die Auszahlung zu veranlassen.

Änderungen der Gesetzeslage werden in die KOVG Ontology eingepflegt, ohne dass die höheren Schichten davon betroffen sind.

10.9. Verwandte Arbeiten

In diesem Abschnitt werden verwandte Arbeiten diskutiert, um die Einordnung des in diesem Kapitel entwickelten Modells zu erleichtern.

10.9.1. Die Formalisierung von Verboten, Geboten und Erlaubnissen mit OWL 2

Angetrieben von der Entwicklung der LKIF-Core Ontologie gibt es auf dem Gebiet der KI und Recht diverse Ansätze, um deren Verwendbarkeit im Hinblick auf die automatisierte rechtliche Beurteilung von Sachverhalten zu testen (van de Ven, Breuker, et al., 2008; van de Ven, Hoekstra, et al., 2008).

Der Tatbestand einer Norm wird bei diesen Ansätzen durch eine Menge von Bedingungen beschrieben. Diese Beschreibung wird Generic Case (GC) genannt. Die Norm selbst wird als eine deontische Qualifikation des GC repräsentiert. Dieser Ansatz wurde bereits von Valente (1995), und Valente und Breuker (1995) präsentiert.

Der GC beschreibt eine Situation, die geboten, verboten oder erlaubt ist, vollständig. Durch Ausführung der Inferenzmaschine wird sodann ein individueller Fall einem GC untergeordnet. Dazu reicht es, dass der Einzelfall und der GC partiell übereinstimmen. Damit der Sachverhalt jedoch einem GC untergeordnet wird, müssen sämtliche Bedingungen des GC erfüllt sein.

Je nachdem, ob die Standardsituation vom jeweiligen Gesetz erlaubt oder verboten wird, muss der GC bei Verboten bzw. Geboten in eine erlaubte und eine verbotene Situation geteilt werden, sodass für den GC zwei Klassendefinitionen erforderlich sind.

Im Hinblick auf die Bedingungen des GC muss zwischen dem Kontext einer Norm und dem Teil unterschieden werden, auf den die Norm gerichtet ist. Dies sei kurz am Beispiel einer Bibliotheksordnung erläutert. Diese enthält folgende Obligation: „Studenten sind für die Leihe von Büchern verpflichtet ihren Ausweis vorzuzeigen“. Der Kontext des GC wäre in diesem Fall „Student X leiht ein Buch“, und „zeigt seinen Ausweis vor“ der Teil, der von der Norm geboten ist.

Da GCs bloße Klassenbeschreibungen sind, werden diese von der Inferenzmaschine automatisch in einer Hierarchie in der Form angeordnet, dass der speziellere eine Subklasse des generelleren GC ist. Eine Lex-Specialis-Beziehung wird damit automatisch von der Inferenzmaschine abgeleitet. Für die Interpretation dieser Klassenhierarchie wird jedoch ein eigenes Programm benötigt, das daraus ableiten kann, welche Norm nun im Endeffekt anwendbar ist.

Im Gegensatz zu den vorhandenen Ansätzen geht es im Kontext dieser Dissertation nicht um die Beurteilung einer Situation, ob diese erlaubt ist etc., sondern um die Ableitung von Ansprüchen aus der Ontologie. Außerdem werden in diesem Kapitel sämtliche Anspruchsgrundlagen als „Erlaubnisse“ repräsentiert. Die Modellierung von Geboten und Verboten war für die gegenständliche Zielsetzung nicht erforderlich.

Für die Anspruchsprüfung liegt kein Sachverhalt vor, der dahingehend geprüft werden könnte, ob dieser erlaubt oder verboten ist. Folgendes Beispiel soll dies verdeutlichen:

Aus dem Axiom „Johannes hatMdE 50“ soll das Axiom „Johannes hatAnspruchAuf Grundrente“ abgeleitet werden. In der Ontologie ist das zweite Axiom selbstverständlich nicht enthalten, sondern soll von der Inferenzmaschinen aus den vorhandenen Axiomen abgeleitet werden.

Im Kontext der bisherigen Ansätze wäre eine vollständige Beschreibung des Sachverhalts, bestehend aus den Axiomen „Johannes hatMdE 50“ und „Johannes hatAnspruchAuf Grundrente“, bereits in der Ontologie vorhanden. Die Inferenzmaschine könnte dann im Beispiel den individuellen Fall als eine Subklasse von `Allowed` klassifizieren. Umgekehrt würde der Sachverhalt „Johannes hatMdE 10“ und „Johannes hatAnspruchAuf Grundrente“ als `Disallowed` klassifiziert werden, da mit einer Minderung der Erwerbsfähigkeit von 10 % kein Anspruch auf Grundrente besteht.

Ein Programm könnte zur Prüfung, ob ein bestimmter Anspruch besteht, entsprechende Axiome in die Ontologie einfügen und dann die Inferenzmaschine ausführen um zu prüfen, ob der Fall erlaubt oder verboten ist. Erlaubt bedeutet in diesem Fall „Leistung steht zu“, und verboten „Leistung steht nicht zu“.

Es liegt auf der Hand, dass dies ein eher ungewöhnlicher Ansatz für die Prüfung von Ansprüchen ist. Sämtliche Leistungsarten (und auch Kombinationen von Leistungen) müssen

dem Programm im Vorhinein bekannt sein, damit dieses entsprechende Axiome generieren kann. Damit wird der Zweck der Ontologie als zentrale Wissensbasis, in die neue Leistungen eingefügt werden können, verfehlt.

Der in diesem Kapitel entwickelte Ansatz ist nicht von diesem Nachteil behaftet. Neue Sachverhaltsbeschreibungen können jederzeit in die Ontologie eingefügt und neue Ansprüche daraus abgeleitet werden.

Die Versorgungsleistungen werden als Instanzen repräsentiert, die grundsätzlich beliebige Properties haben können und beispielsweise auch auf externe Wissensquellen verweisen könnten. Die Instanz selbst kann mit anderen in Beziehung gesetzt werden, um auch komplexere Strukturen zu repräsentieren.

Aus der Ontologie können bei Bedarf auch mehrere Rechtsfolgen aus einem Sachverhalt abgeleitet werden. Dazu muss lediglich eine zusätzliche Restriktion mit der deontischen Qualifikation des Sachverhalts (modelliert als Klasse) in Beziehung gebracht werden.

Aufgrund dieser Flexibilität scheint der Ansatz auch geeignet, um die Implementierung der Wissensbasis eines Entscheidungsunterstützungssystems zu unterstützen. Die Aussagen, die aus einem Sachverhalt abgeleitet werden, können eine beliebige Bedeutung haben und sind nicht auf die Repräsentation von Rechtsfolgen beschränkt.

10.9.2. Die Formalisierung in Java

Die Formalisierung von Rechtsnormen mit Java wurde in Kapitel 9 ausführlich dargestellt. Vorweg sei erwähnt, dass der ontologische Ansatz bei Weitem nicht alle Aspekte der Beschädigtenrente formalisieren konnte. Dies ist hauptsächlich durch die Tatsache bedingt, dass die Beschreibungslogik nicht geeignet ist, mathematische Berechnungen durchzuführen.

Aus theoretischer Sicht werden Rechtsnormen durch das ontologische Modell adäquat repräsentiert. Bei der Formalisierung in Java wird nicht zwischen der Rechtsnorm als Objekt und der normativen Aussage unterschieden.

Die Ontologie stellt die Struktur und Begrifflichkeit des Gesetzes direkt dar. Dem Prinzip des Isomorphismus wird dadurch besser entsprochen. Dies ist ein wesentlicher Unterschied zur Formalisierung der Rechtsnormen in Java.

Ein Javaprogramm kann als abgeschlossene Menge von Regeln begriffen werden. Es ist ohne eine Codeänderung nicht möglich, neue Bedingungen in das Programm einzufügen. Zu einer Ontologie können grundsätzlich jederzeit neue Axiome hinzugefügt werden. Neues Wissen steht damit sofort für neue Schlussfolgerungen zur Verfügung. Ontologien bieten damit eine gute Grundlage, um mit Novellen von Gesetzen umgehen zu können.

Im Gegensatz zu Java ist beim ontologischen Ansatz der Lösungsalgorithmus von der Wissensrepräsentation – der Ontologie – getrennt. Dadurch können die in OWL formalisierten Normen grundsätzlich wiederverwendet werden.

Zusammenfassend lässt sich feststellen, dass Ontologien ausdrucksstärker im Hinblick auf die Repräsentation der Strukturen und der normativen Aussage von Rechtsnormen sind. In diesem Bereich weist der präsentierte Ansatz mit Java Schwächen auf.

Andererseits erreicht OWL im Hinblick auf die Formalisierung (komplexer) Rechtsfolgen bei Weitem nicht die Möglichkeiten von Java. Mathematische Berechnungen, die Wechselwirkungen zwischen Leistungen berücksichtigen müssen, liegen außerhalb der Möglichkeiten des ontologischen Ansatzes.

Die Anweisungen eines Javaprogramms werden in einer fest vorgegebenen Reihenfolge abgearbeitet. Beim Vorgang der Klassifizierung durch die Inferenzmaschine ist nicht nachvollziehbar, in welcher Reihenfolge Schlüsse gezogen werden. Im Gesetz werden Normen jedoch in Abschnitten etc. organisiert. Aus der Reihenfolge der Anwendung von Normen können subtile Unterschiede insbesondere im Hinblick auf die Verteilung der Beweislast folgen (Bench-Capon & Gordon, 2009). Bei der ontologischen Modellierung wurde die Reihenfolge insofern indirekt berücksichtigt, indem falls notwendig zusätzliche Bedingungen in die Restriktion einer Sachverhaltsbeschreibung aufgenommen wurden.

Aus technischer Sicht ist hervorzuheben, dass das objektorientierte Modell eines Javaprogramms im Gegensatz zu einer Ontologie statisch ist. Die Klassenhierarchie formt das Rückgrat des objektorientierten Modells. Sämtliche Instanzen werden als Mitglieder einer bestimmten Klasse erzeugt und deren Verhalten wird durch die Klassenstruktur bestimmt. Änderungen der Klassenstruktur haben weitreichende Auswirkungen auf das Verhalten des Systems. OWL erlaubt, dass Instanzen neuen Klassen zugeordnet werden, sobald neue Informationen vorhanden sind. Diese Informationen müssen nicht zwingend manuell hinzugefügt werden, sondern können auch aus den vorhandenen Axiomen einer Ontologie

durch die Inferenz abgeleitet werden. Durch die Klassifizierung einer Ontologie wird damit gewissermaßen die Klassenstruktur einer Ontologie (teilweise) dynamisch erstellt (Allemang & Hendler, 2011, S. 274).

Im Unterschied zu OWL erlaubt Java Variable, wodurch der Vergleich des Tatbestands einer Norm mit dem Sachverhalt intuitiver modelliert werden kann. Dieser Nachteil von OWL könnte jedoch beseitigt werden, indem ein grafisches Werkzeug für die Modellierung von Sachverhalten und Normen eingesetzt wird, das die Syntax von OWL verbirgt. Solch eine Entwicklungsumgebung für die Modellierung juristischer Probleme in OWL existiert derzeit nicht.

10.10. Diskussion

In diesem Kapitel wurde gezeigt, inwieweit eine in OWL 2 erstellte Ontologie direkt für die rechtliche Beurteilung verwendet werden kann. Das Ziel war, aus der Ontologie anhand statischer Fakten, die den Sachverhalt repräsentieren, durch Anwendung der Normen die zustehenden Leistungen zu ermitteln.

Die Normen wurden zunächst in einem einfachen Modell ohne die Verwendung der LKIF-Core Ontologie formalisiert. Da dieses Modell theoretische Unzulänglichkeiten aufwies, wurde es erweitert und die LKIF-Core Ontologie als Grundlage verwendet.

Hinsichtlich des Protégé OWL Editors sei erwähnt, dass dieser ein unverzichtbares Werkzeug für die Modellierung der Ontologie war. Allerdings scheint die aktuelle Version 4.3.0 noch mit einigen Bugs behaftet zu sein. So stürzte der Editor hin und wieder mit einer Fehlermeldung ab oder das GUI fror ein, sodass der Editor nicht mehr bedient werden konnte.

Das ontologische Modell ist derzeit in der Lage grundsätzlich zu beantworten, ob ein bestimmter Anspruch zusteht. Ausnahmebestimmungen können bis zu einem gewissen Grad von der Ontologie dargestellt werden. Zu den Einschränkungen in dieser Hinsicht sei auf die Ausführungen im nachfolgenden Abschnitt zu den Vor- und Nachteilen des ontologischen Modells verwiesen.

Die Repräsentation der Leistungen erfolgt durch Instanzen, die mithilfe eines Properties mit einem bestimmten Geldbetrag verknüpft werden können. Mathematische Berechnungen liegen jedoch außerhalb der Leistungsfähigkeit der Ontologie. Eine weitere Grenze ist die

Zuerkennung von Leistungen, die dem Antragsteller unter Umständen mehrmalig zustehen, wie die Familienbeihilfe für jeden Angehörigen. OWL 2 kann mithilfe von Cardinality Restrictions „zählen“ (Allemang & Hendler, 2011, S. 260 f.). Damit kann die Inferenzmaschine zwar folgern, dass Familienbeihilfe grundsätzlich zusteht, sofern zumindest ein Angehöriger vorhanden ist, nicht jedoch dass diese ab einer Anzahl von zwei Angehörigen mehrmals gebührt.

Einige Details, die für die Berechnung der Beschädigtenrente erforderlich sind, wie die Feststellung des Alters des Antragstellers oder die relativ komplizierte Anrechnung des Einkommens auf die Zusatzrente, wurden bewusst nicht formalisiert.

Im Hinblick auf die in Abschnitt 10.3. aufgeworfenen Fragen lässt sich daher feststellen, dass Frage eins vollständig beantwortet werden kann. Sieht man von den diversen Vereinfachungen ab, so kann auch Frage zwei nach den grundsätzlich gebührenden Versorgungsleistungen beantwortet werden. Anders verhält es sich mit Frage drei, zu deren Beantwortung mathematische Berechnungen notwendig sind. Es ist nicht ausreichend, eine Leistungsart mit einem bestimmten Betrag zu verknüpfen, was mit dem ontologischen Modell noch bewerkstelligt werden könnte.

Zusammengefasst bleibt die Effektivität der Ontologie insgesamt hinter der Leistungsfähigkeit des in Abschnitt 9.5. präsentierten auf Java basierenden Programms zurück. Aus diesem Grund wurde nicht weiter untersucht, ob die Ontologie besser geeignet ist als ein objektorientiertes Programm, mit Änderungen der Normen im Lauf der Zeit umzugehen.

10.10.1. Vor- und Nachteile des ontologischen Ansatzes

Im Folgenden werden zunächst die Vor- und anschließend die Nachteile des ontologischen Ansatzes im Allgemeinen und insbesondere gegenüber einer Formalisierung in Java diskutiert, die im Zuge der Modellierung erhoben wurden.

In der Literatur wird häufig als Kritik von regelbasierten Ansätzen vorgebracht, dass diese nicht zwischen der Norm und deren normativer Aussage differenzieren (Breuker et al., 2006, S. 173). Die Norm und deren eigentliche Bedeutung werden gewissermaßen vermischt. Produktionsregeln sind zu ausdrucksschwach, um die vollständige Bedeutung von

normativen Aussagen zu formalisieren (Boer et al., 2007, S. 32 ff.). Regelbasierte Formalismen eignen sich nur zur Formalisierung bestimmter Aspekte von Normen.

Mithilfe der Beschreibungslogik lassen sich die Norm und deren normative Aussage getrennt repräsentieren. Der ontologische Ansatz ist daher regelbasierten Formalismen und auch Java überlegen.

Ein wesentlicher Vorteil gegenüber Java und allgemein der Formalisierung von Normen durch Programmcode ist die Möglichkeit, neue Axiome zur Ontologie hinzufügen zu können, die unmittelbar für neue Schlussfolgerungen zur Verfügung stehen. Ein Programm kann grundsätzlich nur durch externe Daten, die aus einer Datenbank oder einer anderen Quelle stammen, parametrisiert werden. Damit lassen sich geringfügige Änderungen des Gesetzes wie Betragsanpassungen bis zu einem gewissen Maß bewältigen. Die Möglichkeiten der Parametrisierung enden aber bald, wenn neue Normen durch eine Novelle eingefügt oder der Tatbestand bzw. die Rechtsfolgen bestehender Normen tiefgreifend verändert werden. In den zuletzt genannten Fällen ist eine Änderung des Codes unvermeidbar.

Wie weiter oben in Abschnitt 10.6.3.3. ausführlich dargestellt wurde, können normative Konflikte durch den beschreibungslogischen Ansatz ohne zusätzlichen Aufwand aufgelöst werden. Diese Aussage muss für ihre Richtigkeit jedoch in mehrerer Hinsicht relativiert werden.

Erstens kann eine Lex-Specialis-Relation nur dann automatisch aus der Ontologie abgeleitet werden, wenn die Lex generalis durch die Bedingungen $\sigma_1 \sqcap \ldots \sqcap \sigma_n$ und die Lex specialis durch die Bedingungen $\sigma_1 \sqcap \ldots \sqcap \sigma_{n+m}$ gekennzeichnet ist. Der Tatbestand der Lex specialis muss daher, bei Übereinstimmung der anderen Tatbestandsmerkmale mit der Lex generalis, mindestens ein Element mehr aufweisen.

Andere Formen der Lex specialis, sofern man diese als solche bezeichnen möchte, wie die tatbestandsmäßige Übereinstimmung einer Norm in einem Sondergesetz mit einer allgemeingültigen Norm, können durch die Inferenzmaschine nicht aufgelöst werden.

Daraus ergibt sich zweitens, dass andere implizite Konflikte zwischen Normen, wie Lex posterior und Lex superior, ebenso nicht durch die Beschreibungslogik alleine formalisiert werden können. Es fehlt in diesen Fällen an der für die Lex Specialis typischen Hierarchie in der Ontologie. Dies gilt gleichermaßen für im Gesetz explizit enthaltene

Ausnahmebestimmungen. In diesen Fällen muss die Beziehung zwischen den Normen explizit in der Ontologie durch ein entsprechendes Property zum Ausdruck gebracht werden.

Drittens sei hinsichtlich der „Auflösung" von Lex-Specialis-Relationen angemerkt, dass diese nicht so verstanden werden dürfen, dass die Rechtswirkungen der Lex generalis nicht angewendet werden. Wie bereits erwähnt, ist die Beschreibungslogik $\mathcal{SROIQ}$, auf der OWL 2 basiert, monoton. Monotone Logiken kennzeichnen sich dadurch aus, dass eine einmal gezoge Konklusion nicht widerrufen werden kann, selbst dann nicht, wenn neue (gegenteilige) Informationen vorliegen. Aus diesem Grund sind die Inferenzen, die aus der Lex generalis und der Lex specialis abgeleitet werden, nebeneinander im ontologischen Modell vorhanden.

Ein Programm kann sodann die Hierarchie des GC der Lex Specialis und der Lex Generalis ausnutzen um zu entscheiden, welche Konklusion die endgültige ist. HARNESS[155] wurde beispielsweise als Plugin für den Protégé-Editor entwickelt und kann die angesprochene Hierarchie verwenden um zu beurteilen, ob ein bestimmter Sachverhalt verboten, geboten oder erlaubt ist (van de Ven, Breuker, et al., 2008). Ein Programm muss also die TBox- und ABox-Axiome der Ontologie selbstständig abfragen und entsprechend interpretieren. Die nicht-monotone Schlussfolgerung wird daher – ähnlich dem Ansatz mit Java – nur simuliert.

Die Beschreibungslogik $\mathcal{SROIQ}$ ist entscheidbar (Horrocks, Kutz, & Sattler, 2006), es existiert damit ein effektiver Algorithmus für die Durchführung der Inferenz. Entscheidbarkeit bedeutet informell, dass jedes Problem garantiert entschieden werden kann und der Algorithmus nicht in einer Endlosschleife hängen bleibt. Zudem ist das Ergebnis aus logischer Sicht beweisbar richtig.

Allerdings relativiert sich der Wert eines deduktiven logischen Beweises einer Aussage, sobald man nach der rechtlichen Qualität eines solchen fragt. Ein deduktiver Beweis sagt nämlich nichts über die rechtliche Qualität der Entscheidung aus, wie in Abschnitt 4.4.2.5. ausführlich diskutiert wurde.

Java und auch alle anderen Programmiersprachen sind nicht entscheidbar. Die fehlende Entscheidbarkeit bedeutet, dass keine formale Semantik existiert, die Voraussetzung ist um garantieren zu können, dass jedes beliebige Javaprogramm zu einem Ergebnis gelangt

[155] Hybrid Architecture for Reasoning with Norms Exploiting Semantic web Standards.

(Allemang & Hendler, 2011, S. 328). Trotz dieser Tatsache werden in der Praxis hinsichtlich der Formalisierung von Normen mit Java relativ gute Ergebnisse erzielt.

Auch wenn es nicht möglich ist, im Allgemeinen zu bestimmen, ob ein beliebiges Programm zu einem Ergebnis gelangt, so ist es grundsätzlich möglich zu prüfen, ob ein bestimmtes Programm ordnungsgemäß durchlaufen wird, und sogar mit welchem Ergebnis. Entscheidbarkeit ist von diesem Standpunkt aus gesehen nicht das Problem. In der Praxis ist es oftmals nicht erforderlich, dass garantiert werden kann, dass sämtliche Inferenzen von einem beliebigen Algorithmus gefunden werden können. Es ist ausreichend, wenn das Programm mit bestimmten Eingabedaten zum erwarteten (richtigen) Ergebnis gelangt. Ein wichtiger Teil des Softwareengineerings befasst sich exakt mit dem Problem der Entwicklung und dem Test von Programmen, um deren Funktionstüchtigkeit zu gewährleisten.

Aufgrund der formalen Semantik von OWL ist es möglich, zumindest eine Erklärung über das Zustandekommen einer Aussage abzuleiten[156]. Bei dieser Erklärung handelt es sich um einen mathematischen Beweis für eine Aussage, der aufgrund der Entscheidbarkeit der Beschreibungslogik möglich ist. Aus technischer Sicht bietet beispielsweise die OWL API die Schnittstelle `ExplanationGenerator` an, mit deren Hilfe sich eine Erklärung für ein Axiom generieren lässt. Diese Funktionalität stellt eine Grundlage für die Erzeugung einer rechtlichen Begründung. Dafür muss die technische Erklärung erstens mit den zugrundeliegenden Rechtsnormen in Verbindung gebracht, und zweitens die Begründung in natürlicher Sprache formuliert werden. Der erste Schritt stellt keine große Hürde dar, da die Klassen und Instanzen einer Ontologie mithilfe einer URI auf die jeweiligen Rechtsnormen verweisen können, die beispielsweise in XML vorhanden sind.

Die in der Ontologie modellierten Klassen und Instanzen repräsentieren Begriffe und Entitäten, die aus dem Gesetz und der realen Welt abgeleitet werden. Die Namen für Klassen, Instanzen und Properties in der Ontologie können aus der Wirklichkeit übernommen werden. Die Beschreibungslogik basiert auf der Syntax und Semantik der englischen Sprache. Dies erleichtert das Verständnis des ontologischen Modells, da angenommen wird, dass die Sprache einer Person eine wichtige Rolle bei der Strukturierung von Gedanken spielt (Mylopoulos, 1980, S. 8).

[156] Da eine Inferenz auf mehrere Arten zustande kommen kann, können auch mehrere Erklärungen vorhanden sein.

Ontologien eignen sich damit gut, um die Begriffe des Gesetzes und deren Beziehungen zu formalisieren. Domänenontologien können als externes Vokabular, das die wichtigsten Begriffe des Gesetzes enthält, begriffen werden. Überdies lassen sich Sachverhalte mit den beteiligten Rechtsubjekten, deren Beziehungen zueinander und sonstigen Objekten wie Ereignisse oder Beweismittel, recht übersichtlich abbilden.

Insbesondere deskriptive (konstitutive) Normen wie Legaldefinitionen können mithilfe der Beschreibungslogik adäquat formalisiert werden. Ein Begriff wird in der Ontologie durch die identifizierenden Merkmale repräsentiert, die diesen kennzeichnen. Ebenso können präskriptive Normen – Verbote, Gebote und Erlaubnisse – formalisiert werden (Hoekstra et al., 2007, S. 57 ff., 2009, S. 44 ff.). Die LKIF-Core Ontologie bietet Unterstützung für die Qualifikation eines Sachverhalts als verboten, geboten oder erlaubt. Für die Repräsentation von komplexen Rechtsfolgen – wie Ansprüche oder Sanktionen im Fall einer Normverletzung – und im allgemeinen Verhalten bietet die LKIF-Core Ontologie nur sehr rudimentär Unterstützung.

Die Begriffe und Begriffscluster der LKIF-Core Ontologie geben eine gute Orientierung für die Modellierung einer Ontologie und erleichtern so die Strukturierung der Begrifflichkeit des Gesetzes.

Mithilfe von SPARQL für RDF und insbesondere SPARQL-DL (Sirin & Parsia, 2007) für OWL Ontologien können Abfragen ausgeführt werden. Für die Abfrage ist es nicht notwendig, Details der Struktur der Ontologie zu kennen, wie dies bei Datenbankabfragen mit SQL erforderlich ist. Vielmehr können Fragen wie „Welche Partei hat Beweismittel X vorgebracht?“ formuliert und an die Ontologie gerichtet werden. Die Formulierung von Fragen findet auf einer semantischen Ebene statt, die von Details des ontologischen Modells abstrahiert.

Der Rest dieses Abschnitts befasst sich mit der Diskussion der Nachteile des ontologischen Modells.

Die Formalisierung von Normen mit OWL ist nicht so intuitiv wie mit einem regelbasierten Formalismus. Für die Modellierung ist genaues Wissen über das Zusammenwirken von Norm, Generic Case, der deontischen Qualifikation und dem Sachverhalt erforderlich. Überdies muss der Verwender des Modells die in der Ontologie vorhandenen Begriffe und

deren Bedeutung exakt kennen. Für den praktischen Einsatz wäre jedenfalls ein grafisches Werkzeug hilfreich, das die Erstellung des Modells unterstützt.

Es hat sich erwiesen, dass die Beschreibungslogik alleine zu ausdrucksschwach ist, um alle Aspekte der Berechnung der Beschädigtenrente zu formalisieren. Die Ontologie konnte nicht alle Fragen beantworten, die für die Feststellung der Ansprüche erforderlich sind. Dies betrifft insbesondere mathematische Berechnungen und Datumsarithmetik.

Aus diesem Grund ist es beispielsweise nicht möglich, das Alter des Antragstellers an einem bestimmten Datum zu ermitteln und für weitere Schlussfolgerungen zu verwenden. Das Alter müsste von einem Programm vorberechnet und in die Ontologie als Axiom eingefügt werden, damit die Inferenzmaschine die erforderlichen Schlüsse ableiten kann. Im Hinblick auf die Repräsentation der Rechtsfolgen ist Java der Beschreibungslogik überlegen.

Die Repräsentation von Situationen und Sachverhalten erfordert oftmals, komplexe Beziehungen zwischen Personen darzustellen. OWL 2 kann beispielweise die Struktur von Rechtsgeschäften nur annähern (Hoekstra & Breuker, 2008). Es ist beispielsweise nicht möglich, „einen Käufer, der den Gegenstand eines Rechtsgeschäfts übergeben erhält, an dem *er* beteiligt ist“, aufgrund des „Tree Model Properties“ zu repräsentieren (Baader et al., 2008, S. 157 f.). Die Beschreibungslogik erlaubt keine Variable. Damit ist es schwierig, die Identität von zwei Instanzen zu repräsentieren. Regelbasierte Formalismen könnten das obige Beispiel einfach darstellen, indem eine Variable verwendet wird, die auf einen bestimmten Käufer oder ein bestimmtes Rechtsgeschäft verweist.

Alternative Ansätze wie „strukturierte Objekte“ (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003, S. 215; Motik, Cuenca Grau, & Sattler, 2008) sind vielversprechend, gehören aber nicht zum Standard und werden daher von aktuellen Inferenzmaschinen nicht unterstützt.

Die Ausdrucksstärke der Ontologie könnte durch den Einsatz von SWRL (Hitzler & Parsia, 2009) oder SPARQL bzw. SPARQL-DL (Sirin & Parsia, 2007) gesteigert werden, wodurch jedoch die Komplexität des Modells erhöht wird. Die Idee, SPARQL zu verwenden, wurde bereits von Breuker et al. (2008) angedacht. SPARQL eignet sich jedoch nicht dazu, Axiome in

einer Ontologie zu definieren. Aus diesem Grund wurde die SPARQL Inferencing Notation[157] (SPIN) entwickelt, die derzeit als W3C-Entwurf vorliegt (Allemang & Hendler, 2011, S. 116). Mithilfe von SPIN können regelartige Axiome in Ontologien eingefügt werden, und wird daher auch SPARQL Rules genannt.

Das Problem von SPARQL im Zusammenhang mit OWL Ontologien ist, dass SPARQL nicht die Semantik von OWL versteht. SPARQL operiert auf dem serialisierten RDF-Graphen einer OWL Ontologie. Aus diesem Grund erscheint die Verwendung von SWRL zur Steigerung der Ausdrucksstärke von OWL Ontologien aussichtsreicher. Es bleibt jedoch abzuwarten, wie sich SPARQL-DL (Sirin & Parsia, 2007) als Abfragesprache für OWL in dieser Hinsicht entwickelt. Eventuell wird SPIN erweitert, um SPARQL-DL in Zukunft als Abfragesprache zu verwenden.

10.10.2. Konklusion und Empfehlungen

Der in diesem Kapitel präsentierte, ausschließlich auf Beschreibungslogik basierende, ontologische Ansatz für die rechtliche Beurteilung von Sachverhalten, hat durchaus seine Vorteile. Aufgrund der Entscheidbarkeit der Beschreibungslogik ist die Richtigkeit der Entscheidung alleine von der Qualität des Modells und nicht vom Algorithmus abhängig.

Boer vertritt in seiner Dissertation hinsichtlich der Verwendbarkeit von OWL DL im rechtlichen Bereich die These (Boer, 2009, S. 9):

„If the Semantic Web does not accommodate defeasible reasoning, one shouldn't have to introduce it to accurately represent knowledge of the law. Falsifiability is not unique to law. If OWL DL is good for anything else, it is also good enough for law."

Einer Gesamtbetrachtung unterzogen ist OWL jedoch zu ausdrucksschwach, um die rechtliche Beurteilung vollständig durchführen zu können. Auch wenn OWL eine rudimentäre Behandlung von Ausnahmebestimmungen ermöglicht, vermag dies nicht die Schwächen der monotonen Logik zu überwinden. Der Vorteil der beweisbaren Schlussfolgerung wird zu einem Großteil zunichtegemacht, da für die Behandlung von Ausnahmebedingungen erst recht ein Programm benötigt wird, das die Axiome der Ontologie entsprechend interpretiert.

---

[157] *W3C*, SPIN - Overview and Motivation. http://www.w3.org/Submission/2011/SUBM-spin-overview-20110222/ aufgerufen: 18.5.2014 (2011).

Die Stärken von OWL zeigen sich bei der Repräsentation von terminologischem Wissen. Grundbegriffe des Gesetzes und deren Beziehungen lassen sich durch Klassen mit entsprechenden Restriktionen in OWL formalisieren.

Diese Begriffe werden üblicherweise von einer Domänenontologie zur Verfügung gestellt, mit deren Unterstützung Sachverhalte, aber auch komplexere Strukturen wie richterliche Urteile, Argumente (Ceci, 2013) und Regeln repräsentiert werden können. Angemerkt sei, dass die Ontologie ausschließlich das begriffliche Substrat, aus dem sich die Regeln und Argumente zusammensetzen, enthält. Die automatisierte Verarbeitung der Regeln und Argumente ist hingegen nicht Aufgabe der Ontologie.

Eine spezielle Software kann dann das in der Ontologie vorhandene Wissen nutzen, um beispielsweise Argumente oder Regeln zu erzeugen, um die rechtliche Beurteilung – beispielsweise durch Anwendung nicht-monotoner Logik – durchzuführen.

Meines Erachtens sollte in der Ontologie nur statisches Wissen, sogenannte „Fakten“, formalisiert werden, deren Bedeutung unzweifelhaft ist. Anfechtbare Elemente der rechtlichen Entscheidungsfindung, wie insbesondere Regeln und Argumente, sollten in einer höheren Ebene verarbeitet werden, die vom darunterliegenden Datenmodell abstrahiert. Die Inferenz sollte nur für oberflächliche Schlussfolgerungen verwendet werden, um die Wissensbasis für Tools vorzubereiten, die unter Anwendung von deontischer anfechtbarer Logik die weitere (rechtliche) Entscheidungsfindung durchführen.

Es sei darauf hingewiesen, dass die Bedeutung von Begriffen nicht immer von einem rein terminologischen oder ontologischen Ansatz erfasst werden kann, sondern sich aus den zugehörigen inferentiellen Verbindungen ergibt. Unter einer inferentiellen Verbindung werden im Wesentlichen normative Regeln (Rechtsnormen) verstanden, der Begriff ist jedoch weiter und umfasst auch moralische Normen (Sartor, 2009b).

Da die Beschreibungslogik für die rechtliche Beurteilung zu ausdrucksschwach ist und die Bedeutung eines Begriffes im Zusammenhang mit den dazugehörigen Rechtsfolgen gesehen werden muss, erscheint ein kombinierter Ansatz aus einem ontologischen Modell und einem regelbasierten Formalismus am aussichtsreichsten.

10.10.3. Zukünftige Forschungen

Um dieses Kapitel abzuschließen sei ein Ausblick auf mögliche zukünftige Forschungen gegeben.

Mathematische Einheiten und Dimensionen sind Bestandteil vieler Gesetze. Man denke beispielsweise an das Kraftfahrzeuggesetz, das die Beschaffenheit von Kraftfahrzeugen bestimmt, oder die Straßenverkehrsordnung, die unter anderem die zulässige Höchstgeschwindigkeit vorschreibt.

Wie in diesem Kapitel schon öfters ausgeführt wurde, beherrscht die Beschreibungslogik keine mathematischen Berechnungen. Dies bedeutet jedoch nicht, dass eine Ontologie keine Einheiten und Dimensionen repräsentieren kann. In diesem Zusammenhang sei die von der NASA entwickelte QUDT[158] (Quantities, Units, Dimensions and Data Types) Ontologie erwähnt. Diese Ontologie könnte als Grundlage für die Modellierung von Rechtsvorschriften im Hinblick auf die Definition von Geschwindigkeit, Größe etc. herangezogen werden, um beispielsweise eine juristische Kernontologie zu ergänzen. Mithilfe entsprechender SPARQL-Abfragen ist sogar eine Umrechnung von Einheiten möglich (Allemang & Hendler, 2011, S. 289 ff.).

Ein gewichtiger Nachteil der rein terminologischen Beschreibungslogik ist die fehlende Möglichkeit, mathematische Berechnungen durchführen zu können. Die Leistungsfähigkeit des ontologischen Modells stieß hinsichtlich der rechtlichen Beurteilung von Sachverhalten rasch an ihre Grenzen. Durch den Einsatz von SWRL-Regeln[159] könnte dieses Problem gelöst und auch mathematische Berechnungen durchgeführt werden[160].

Es bleibt jedoch fraglich, ob auch komplexe Wechselwirkungen zwischen Leistungen mithilfe von SWRL bei der Berechnung berücksichtigt werden können. Dazu ist es oftmals erforderlich, die Berechnung mit hypothetischen Werten zu parametrisieren und das Ergebnis mit dem tatsächlichen Wert zu vergleichen. Für weitere Details sei auf Abschnitt 9.5.9.6. verwiesen, in dem die Berechnung der Familienzulage mit Java dargestellt wird.

---

[158] *Hodgson, R., Keller, P., Hodges, J., Spivak, J.*, QUDT Ontology. http://qudt.org/ aufgerufen: 19.5.2014 (2014).

[159] Semantic Web Rule Language.

[160] *W3C*, SWRL Math Built-Ins. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/#8.2 aufgerufen: 19.5.2014 (2004).

Eine weitere interessante Einsatzmöglichkeit von Ontologien ist die Modellierung von Fällen in einer Form, die verwendet werden kann, um diese für Vergleiche heranzuziehen. Derartige Ansätze reichen von einfachen Vergleichen des Sachverhalts bis hin zu fallbasierter Entscheidungsfindung und zur Bildung von Analogien (Ashley, 2009; Wyner & Hoekstra, 2012; Wyner, 2008). Die zitierten Ansätze sind nicht über die Konzeption einer Ontologie hinausgewachsen. Trotzdem geben sie einen Ausblick auf die Möglichkeiten der ontologischen Modellierung zur Unterstützung von Werkzeugen für die (automatisierte) rechtliche Beurteilung.

In dieser Hinsicht könnte die in diesem Kapitel präsentierte Ontologie erweitert werden, um einfache Vergleiche von Fällen anhand des Sachverhalts durchführen zu können. Ein solches Modell könnte den Sachbearbeiter bei der Entscheidungsfindung, aber auch den Bürger im Hinblick auf die Prüfung potentiell zustehender Leistungen, unterstützen. Die Möglichkeit von Vergleichen ist auch im Hinblick auf eine einheitliche Spruchpraxis wünschenswert. Falls eine Entscheidung von dieser abweicht, kann eine höhere Instanz im Rechtsmittelverfahren angerufen werden, um diese zu korrigieren.

Eine Ontologie hat den immensen Vorteil gegenüber einem Programm, dass diese jederzeit durch neue Axiome erweitert werden kann. Das ontologische Modell kann damit flexibel auf Änderungen reagieren, die durch gesetzliche Novellen verursacht werden. Es wäre denkbar, die vorhandenen Ansätze, die sich mit der Versionierung von Begriffen befassen, auf ihre praktische Verwendbarkeit zu prüfen und gegebenenfalls zu erweitern (Bron et al., 2007; Klarman et al., 2008; Priddle-Higson, 2010).

Hinsichtlich der Versionierung einer Ontologie im Laufe der Zeit wäre die Entwicklung einer Theorie erforderlich, die dieses Problem auf einer abstrakteren Ebene als bisherige Ansätze behandelt. Wie in diesem Kapitel gezeigt wurde, ist ein ontologisches Modell alleine nicht ausreichend, um Rechtsnormen zu repräsentieren. Für die Modellierung von Rechtsnormen ist ein mehrschichtiger Ansatz, der den Rechtstext, die Ontologie und die Regeln umfasst, erforderlich. Ein Ansatz zur Formalisierung zeitlicher Aspekte von Normen kann sich daher nicht bloß auf das ontologische Modell beschränken.

Die derzeitige Version der Ontologie beschränkt sich auf jene Aspekte, die für die automatisierte rechtliche Entscheidungsfindung benötigt werden. Sachbearbeiter und Juristen neigen dazu, sich prozedural im Hinblick auf ein bestimmtes Verfahren

auszudrücken. Eine ähnliche Beobachtung wurde von van Engers et al. (2001) im Hinblick auf Legisten gemacht. Für die vollständige Repräsentation eines Verwaltungsverfahrens inklusive der beteiligten Parteien, der vorgebrachten Beweismittel etc. müsste das ontologische Modell um prozedurale Aspekte erweitert werden. Die LKIF-Core Ontologie bietet unter anderem die Klassen `Agent` und `Role` an, die als Grundlage dienen könnten. Dies könnte das Verständnis und die Akzeptanz des Modells unter Angehörigen der angesprochenen Personenkreise verbessern.

In diesem Kontext könnte auch die Repräsentation der Anspruchsgrundlage (der Normen) überdacht werden, die derzeit als Erlaubnisse formalisiert sind. Möglicherweise könnte der Anspruch als *erlaubendes Recht* (permissive right) im Sinne von Hohfeld angemessener repräsentiert werden (Sartor, 2006a). Demnach ist das Recht des Antragstellers, eine Leistung zu beziehen, gleichzeitig die Verpflichtung der Behörde, diese auszubezahlen.

Ähnlich der Anspruchsgrundlage könnte auch eine Leistung selbst, die durch eine Instanz repräsentiert wird, weiter ausgebaut werden. Neben einem Property, das auf den zugehörigen Betrag verweist, könnte eine komplexere Beschreibung der Leistung in Form einer mathematischen Formel definiert werden. Die hinterlegte Formel könnte dann von einem Programm entsprechend eingelesen und berechnet werden.

11. Erkenntnisse der Formalisierung

11.1. Einleitung

In den vorherigen Kapiteln wurden verschiedene Ansätze zur Formalisierung von Rechtsnormen behandelt. Java scheint auf den ersten Blick recht gut für diesen Zweck geeignet zu sein. Dies gilt zumindest dann, wenn ein wesentlicher Teil der Vollziehung eines Gesetzes aus mathematischen Berechnungen besteht. Normen, die stark auf mathematischen Berechnungen beruhen, sind für eine Formalisierung besonders geeignet.

Dieses Ergebnis deckt sich mit den Beobachtungen auf dem Gebiet der (österreichischen) Verwaltungsautomation, bei der überwiegend für die Implementierung von Systemen zur Unterstützung oder Durchführung der Vollziehung Mehrzwecksprachen wie COBOL und PL/1 zum Einsatz kommen.

Der Einsatz von Java ist aber auch mit schwerwiegenden Nachteilen verbunden. Aus rechtstheoretischer Sicht ist die mangelnde Unterscheidung zwischen der Norm als Objekt und der normativen Aussage selbst zu nennen. Überdies sind die Repräsentation der Normen im Programmcode und der Rechtstext nicht isomorph. Zwischen dem verbindlichen Rechtstext und dessen formaler Repräsentation im Code kann oftmals keine Verbindung hergestellt werden.

Der Programmcode ist starr und kann, nachdem das Programm einmal kompiliert wurde, nicht mehr verändert und erweitert werden. Dies führt im Fall von gesetzlichen Änderungen zur Notwendigkeit der Anpassung des Programmcodes. Das ohnehin schon komplexe Programm wächst damit im Laufe der Jahre unaufhaltsam, wodurch dessen Lesbarkeit erheblich erschwert wird. Dem Prinzip des Isomorphismus ist nur unzureichend Genüge getan. Im Programm ist oftmals nicht mehr erkennbar, auf welcher gesetzlichen Grundlage der Code beruht. Dabei darf nicht vergessen werden, dass der authentische Rechtstext die einzige rechtsverbindliche Fassung einer Norm enthält.

In Kapitel 10 wurde der ontologische Ansatz detailliert dargestellt. Zu dessen Vorteilen zählt vor allem die rechtstheoretisch adäquate Trennung zwischen Norm und deren normativer Aussage. Neue Regeln können grundsätzlich beliebig zur Ontologie hinzugefügt werden und stehen sofort zur Verfügung. Damit ist das Modell flexibel genug, um auf Änderungen der Gesetzeslage reagieren zu können.

Das in einer Ontologie modellierte Wissen kann großteils wiederverwendet werden. Zumindest dann, wenn die Ontologie mit einer Sprache wie OWL 2 modelliert wurde, die eine formale Semantik aufweist. Hier sei eine Einschränkung angemerkt, dass die Wiederverwendbarkeit nicht zuletzt auch von der Struktur des Modells und nicht bloß von der verwendeten Sprache abhängig ist.

Wie die Untersuchungen gezeigt haben, ist die rein terminologische Formalisierung von Normen mit OWL 2 nicht ausreichend, um sämtliche Aspekte der Vollziehung abzubilden. Wünschenswert wäre es, einen Ansatz zu finden, der die Ausdruckstärke von OWL hinsichtlich der Struktur von Rechtsnormen mit den Fähigkeiten von Java zur Bestimmung von Rechtsfolgen vereint.

Im Hinblick auf die Formalisierung von zeitlichen Aspekten der Normen kann mit den untersuchten Ansätzen kein zufriedenstellendes Ergebnis erreicht werden. Für die automatisierte Behandlung des zeitlichen Anwendungsbereichs ist es zumindest erforderlich, dass Datumswerte miteinander verglichen werden können. Eine Ontologie müsste aus diesem Grund beispielsweise mit SWRL-Regeln angereichert werden, um zeitliche Aspekte zu formalisieren.

Mit Java ist die Behandlung von Datumswerten wie mit jeder anderen Programmiersprache möglich. Es fehlt jedoch an einem abstrakten Konzept, das die zeitlichen Aspekte von Rechtsnormen adäquat formalisiert. In Javaprogrammen kommen üblicherweise Bedingungen zum Einsatz, die steuern, ob ein bestimmter Programmzweig zum jeweiligen Datum ausgeführt werden soll. Der zeitliche Anwendungsbereich einer Norm ist jedoch oftmals nicht Teil des Tatbestands, womit es zu einer rechtstheoretisch unzulässigen Vermengung von Elementen verschiedener Ebenen kommt.

11.2. Zeitlicher Geltungsbereich und substantielle rechtliche Beurteilung

Die Anforderungen an Sprachen zur Formalisierung von Rechtsnormen, insbesondere des Verwaltungsrechts, wurden ausführlich in Kapitel 7 behandelt. Im Zuge der Modellierung von Normen mit Java und OWL 2 wurden wichtige Erkenntnisse gewonnen, die eine eingehendere Betrachtung und Bewertung der Anforderungen erlauben.

Ansätze zur Formalisierung von Normen, die auf einem monotonen Schlussfolgerungsmodell basieren, reichen nicht aus, um die Modellierung von Rechtsnormen auf verifizierbare Weise

durchzuführen und gleichzeitig die Struktur des Gesetzes sowie die Besonderheiten der juristischen Entscheidungsfindung zu berücksichtigen.

Gesetze sind grundsätzlich in Form von allgemeinen Normen und Ausnahmebestimmungen organisiert. Rechtsfolgen, die aus Normen abgeleitet werden können, sind anfechtbar. Obwohl der Tatbestand der (generelleren) Norm erfüllt ist, gelangt deren Rechtsfolge nicht zur Awendung, sondern wird von der spezielleren Bestimmung verdrängt.

Normen können widersprüchlich sein, das heißt die ableitbaren Rechtsfolgen stehen miteinander in Konflikt. Eine Methode zur Auflösung derartiger Konflikte ist die Priorisierung der Normen, indem zeitliche Aspekte berücksichtigt werden. Für die Repräsentation solcher Metaregeln wird eine Form der anfechtbaren Logik (defeasible logic) benötigt.

In Anbetracht dieser Erkenntnisse erscheint die Repräsentation zeitlicher Aspekte und insbesondere des zeitlichen Anwendungsbereichs von Rechtsnormen besonders wichtig. Ein System muss zudem eine Form der anfechtbaren Logik implementieren, um die juristische Entscheidungsfindung adäquat zu repräsentieren.

Zeitliche Eigenschaften der Normen und die angesprochene Struktur von Gesetzen, organisiert in allgemeinen Bestimmungen und Ausnahmeregeln, können durch Metaregeln formalisiert werden. Die Konklusion einer Metaregel ist selbst wiederum eine Regel, die festlegt, welche Normen in zeitlicher Hinsicht anwendbar sind. Durch die Anwendung solcher Metaregeln werden gewissermaßen die zu einem Zeitpunkt anwendbaren Rechtsvorschriften ermittelt. Im Fall von Prioritätsrelationen zwischen Normen, beispielsweise zwischen Lex generalis und Lex specialis, ist dafür die Prüfung, ob der Sachverhalt den Tatbestand einer Norm erfüllt, erforderlich. Die Anwendung der Norm und Feststellung der Rechtsfolgen findet jedoch nicht statt.

Sobald die anzuwendenden Normen feststehen, müssen die daraus resultierenden Rechtsfolgen ermittelt werden. Gesetze und insbesondere Verwaltungsgesetze beinhalten oftmals mathematische Aspekte, die Berechnungen erforderlich machen. Es reicht damit in der Regel nicht aus zu prüfen, ob ein bestimmter Sachverhalt verboten oder erlaubt ist, sondern das „Ausmaß“ der Rechtsfolgen, wie beispielsweise die Höhe der Steuerschuld oder –gutschrift, muss vom System ermittelt werden.

Damit ergeben sich im Wesentlichen zwei Problemkreise, die bei der Formalisierung von Normen und der automatisierten Rechtsanwendung behandelt werden müssen:

(1) Zuerst gilt es, die in zeitlicher Ansicht auf einen konkreten Fall anwendbaren Rechtsvorschriften zu ermitteln. Besonderheiten, auf die Bedacht genommen werden muss, sind rückwirkend erlassene Rechtsvorschriften und die potenzielle Anwendung bereits außer Kraft getretener Normen auf Sachverhalte, die sich in der Vergangenheit ereignet haben.

Dem zeitlichen Problemkreis ebenso zugehörig ist die Bestimmung der Dauer von normativen Effekten. Man unterscheidet dauerhafte und vorübergehende Rechtsfolgen (Governatori, Rotolo, & Rubino, 2010).

Ein vollständiges Modell zur Formalisierung zeitlicher Aspekte des Rechts muss außerdem mit Aufhebungen (ex nunc) und Nichtigerklärungen (ex tunc) von Rechtsnormen umgehen können, wie sie durch verfassungsgerichtliche Entscheidungen bedingt werden.

(2) Zweitens muss die rechtliche Beurteilung des Falls durchgeführt werden. Dies umfasst einerseits die rechtliche Beurteilung des Sachverhalts und dessen deontischer Qualifikation. Das System muss aufgrund der implementierten Regeln feststellen, ob ein bestimmter Sachverhalt verboten bzw. erlaubt ist. Andererseits existieren auch komplexere Rechtsfolgen, die im Detail ermittelt werden müssen. Wie bereits weiter oben ausgeführt, sind dafür unter Umständen auch mathematische Berechnungen erforderlich.

Charakteristisch für den zweiten Schritt ist die Ableitung von Rechtsfolgen durch nicht-monotone Schlussfolgerung. Der Bestimmung der Rechtsfolgen geht selbstverständlich die Prüfung der Anwendbarkeit einer Norm auf einen Sachverhalt voraus. Sobald feststeht, dass ein Sachverhalt unter den Tatbestand einer Norm subsumiert werden kann, bedeutet dies jedoch nicht, dass tatsächlich auch deren Rechtsfolgen zur Anwendung gelangen. Vielmehr bedarf es einer Prüfung, ob (explizite) Ausnahmebestimmungen ebenfalls anwendbar sind, die der allgemeineren Norm vorgehen.

Im Wesentlichen geht es demnach beim zweiten Problemkreis um die Anwendung des juristischen Syllogismus auf den Einzelfall. Es reicht jedoch nicht aus, das Ergebnis deduktiv durch Anwendung des logischen Modus Ponens abzuleiten. Wie angemerkt, kann grundsätzlich nur nicht-monotone Logik die rechtliche Entscheidungsfindung angemessen

repräsentieren. Die rechtliche Beurteilung ist ein iterativer Prozess. Die Anwendbarkeit einer Bestimmung kann zur Anwendung weiterer Regeln führen usw.

Kernaussage dieses Abschnitts ist, dass zwischen der Formalisierung zeitlicher Aspekte von Rechtsnormen und der automatisierten Anwendung des juristischen Syllogismus unterschieden werden muss. Die konzeptuelle Trennung der beiden Problemkreise ist Voraussetzung, um beide Aspekte getrennt voneinander behandeln zu können. Damit sollte es möglich sein, die Formalisierung temporaler Aspekte auf einem abstrakteren Niveau, losgelöst von der rechtlichen Beurteilung (der Anwendung nicht-monotoner Schlussfolgerung), durchzuführen.

Die Möglichkeit der Trennung zeitlicher und inhaltlicher Aspekte bei der rechtlichen Beurteilung ist eine Hypothese, die es zu beweisen gilt. Die Machbarkeit dieser Konzeption könnte durch die Implementierung eines formalen Modells bewiesen werden.

11.3. Rechtsquellen des Verwaltungsrechts und Perspektiven der Formalisierung

Die Wissensbasis eines juristischen wissensbasierten Systems muss für eine effektive Unterstützung des Experten nicht nur die im förmlichen Gesetzgebungsverfahren zustande gekommenen Rechtsakte – das sind Gesetze und Verordnungen – repräsentieren.

Vielmehr müssen auch verwaltungsinterne Weisungen und Erlässe sowie die Entscheidungspraxis der Gerichte, Verwaltungsgerichte und Behörden berücksichtigt und entsprechend repräsentiert werden. Insbesondere die Entscheidungen der Gerichtshöfe des öffentlichen Rechts (VwGH, VfGH) entwickeln eine über den Anlassfall hinausgehende Bedeutung[161]. Entscheidungen von Behörden und Verwaltungsgerichten können aufgehoben werden, sofern sie von der Rechtsprechung der Gerichtshöfe abweichen.

Neben diesen hoheitlichen Rechtsakten existiert auch das sogenannte „Soft Law“. In diese Kategorie fallen unter anderem Önormen, die beispielsweise zur Beurteilung des „aktuellen Stands der Technik“ herangezogen werden[162]. Angemerkt sei, dass Önormen auch durch konkrete Rechtsvorschriften als verbindlich erklärt werden können. Freilich handelt es sich dann nicht mehr um „Soft Law“ im eigentlichen Sinne. Ein weiteres Beispiel sind die FIS-

[161] Dies gilt auch für die Rechtsprechung des Obersten Gerichtshofs im Zivilrecht.

[162] OGH 22.6.2010, 10 Ob 24/09s.

Regeln im Bereich des Wintersports, die von Gerichten regelmäßig zur Bestimmung des Sorgfaltsmaßstabes bei Unfällen herangezogen werden, obwohl es sich bei diesen nicht um objektives Recht handelt[163]. Je nach Zweck und Ausgestaltung eines Expertensystems müssen Rechtsquellen verschiedener Qualität in eine formale Repräsentation überführt werden.

11.3.1. Mehrschichtige Repräsentation

Bei genauerer Betrachtung von Rechtsvorschriften im Kontext der Formalisierung wird erkennbar, dass zwischen drei Ebenen unterschieden werden muss:

- Normen: Abstrakte verbindliche Befehle, die Rechte und Pflichten konstituieren.
- Rechtsvorschriften: Die textuelle Repräsentation der Normen durch Sätze, Artikel und Paragrafen.
- Regeln: Die durch Interpretation ermittelte Bedeutung der Norm, formalisiert durch logische Regeln, bestehend aus Antezedens und Konsequenz. Es ist möglich, dass eine formale Regel mehrere Rechtsvorschriften repräsentiert, aber auch, dass ein Paragraf durch mehrere formale Regeln formalisiert wird.

Im Hinblick auf die Rechtsquellen, die Rechtsvorschriften in textueller Form enthalten, ist zu bemerken, dass diese bei genauerer Analyse aus mehreren Schichten bestehen: Inhalt (Text), Textstruktur, Metadaten des Dokuments und eine Ontologie der im Text enthaltenen Begrifflichkeit. Die Rechtsquellen präsentieren demnach eine komplexe mehrschichtige Architektur. Daraus ergeben sich mehrere Perspektiven für die Untersuchung und Formalisierung (Palmirani, Ognibene, & Cervone, 2012):

- Text: Das authentische Rechtsdokument, das von der dazu befugten Autorität erlassen wurde. Dieser Rechtstext ist als einziger rechtlich verbindlich.
- Textstruktur: Sämtliche Informationen über die Struktur des Dokuments, die beschreibt, wie dieses organisiert ist.
- Metadaten: Darunter sind sämtliche Informationen zu verstehen, die nicht von der rechtlich ermächtigten Autorität erzeugt wurden. Zu solchen gehören unter

[163] Vgl. beispielsweise OGH 26.02.1997, 3 Ob 38/97b.

anderem Schlüsselwörter, die das Dokument beschreiben, sein Workflow oder die Historie des Dokuments.

- Ontologie: Sämtliche Begriffe, die im Rechtstext enthalten sind und modelliert werden müssen. Dazu gehören unter anderem auch Begriffe, die verwendet werden um zu spezifizieren, um welche Art von Dokument es sich handelt (Urteil, Gesetz etc.).
- Regeln: Die formale Repräsentation der durch Interpretation ermittelten Bedeutung des Rechtstexts. Darunter sind die in logische Regeln übersetzten Normen zu verstehen.

Sämtliche Schichten müssen auf nachvollziehbare Weise miteinander verbunden werden, um eine isomorphe Darstellung der Rechtsvorschriften zu erreichen. Ein weiterer wichtiger Aspekt ist der Umgang mit Änderungen des Gesetzes im Lauf der Zeit, die sich notwendigerweise auf das Modell auswirken. Anpassungen müssen auf sämtlichen Ebenen des Modells auf möglichst einfache Weise möglich sein.

Je nach Art und Umfang eines Systems wird der Schwerpunkt der Formalisierung nicht auf alle Ebenen gleichzeitig gelegt. Umso wichtiger ist es, dass Modelle, die auf eine bestimmte Ebene fokussieren, in eine Gesamtarchitektur integrierbar sind. Dies ermöglicht einerseits den inkrementellen Ausbau des Systems, aber auch dass die Schichten des Modells von Spezialisten erstellt und anschließend in einer gemeinsamen Wissensbasis integriert werden.

11.4. Formale Sprache, Modell und Inferenzmaschine

In diesem Abschnitt werden die Zusammenhänge von Modell, Inferenzmaschine und formaler Sprache beleuchtet. Es wird gezeigt, dass es sich dabei um unterschiedliche Problemkreise handelt, die bei der Formalisierung von Recht einer separaten Behandlung zugänglich sind.

Zunächst sei in Erinnerung gerufen, dass wissensbasierte Systeme aus zwei Komponenten bestehen, wodurch sich diese maßgeblich von herkömmlichen Programmen unterscheiden. Es muss zwischen der Wissensbasis (dem Modell) als Summe der Wissensrepräsentationen (den Teilmodellen) und der Inferenzmaschine unterschieden werden. Sofern nur eine Wissensrepräsentation existiert, ist diese ident mit der Wissensbasis des Systems.

Die dritte Komponente, die für die Repräsentation von Normen eine Rolle spielt, ist die verwendete formale Sprache (der Formalismus) selbst[164]. Im Anschluss soll das Zusammenspiel zwischen formaler Sprache, Wissensbasis (Modell) und Inferenzmaschine näher beleuchtet werden.

Mithilfe des Formalismus wird eine symbolische Repräsentation des Wissens – in diesem Fall der Normen – erstellt. Die Wissensrepräsentation als Modell umfasst der Ansicht von Newell folgend nicht nur die symbolische Abbildung des Wissens, sondern gewährleistet auch den Zugriff auf dieses (Newell, 1982, S. 114). Eine Wissensrepräsentation gemäß Newell besteht demnach aus den Komponenten „Wissen“ und „Zugriff“.

Welche Erkenntnisse können daraus für die Formalisierung abgeleitet werden?

Erstens wird deutlich, dass nicht bloß die Ausdrucksstärke des verwendeten Formalismus für die Qualität der Wissensrepräsentation ausschlaggebend ist. Vielmehr muss das formale Modell des Gesetzes sämtliche für die automatisierte Entscheidungsfindung relevanten Aspekte in maschinenlesbarer Form zur Verfügung stellen.

Zweitens können nicht alle Anforderungen an die formale Repräsentation des Wissens von der verwendeten Sprache alleine abgedeckt werden. So ist insbesondere im juristischen Kontext die Flexibilität der Wissensrepräsentation gefragt, um Novellen in das Modell einpflegen zu können. Es hätte wenig Sinn, wenn eine Novelle zwar mithilfe der formalen Sprache modelliert, dann aber nicht in die bestehende Wissensbasis integriert und von dieser zur Verfügung gestellt werden kann.

Das formale Modell der Rechtsnormen (des Wissens) wird schließlich von der Inferenzmaschine verwendet, um Schlussfolgerungen abzuleiten. In diesem Kontext enthält das Modell eine symbolische Repräsentation der Normen, die es dem Computer ermöglicht, die rechtliche Beurteilung von Fällen durchzuführen. Je nach Zweck des Modells variieren naturgemäß dessen Art und Komplexität sowie die daraus ableitbaren Antworten.

Wenn an dieser Stelle von Inferenz gesprochen wird, ist damit nicht bloß die deduktive Schlussfolgerung monotoner Logik gemeint. Vielmehr kann jede beliebige Form der Logik für die Schlussfolgerung eingesetzt werden. Um eine adäquate und effiziente Lösung juristischer

[164] Für eine eingehende Diskussion der Anforderungen an Sprachen zur Repräsentation von Normen als Regeln sei auf Kapitel 7 verwiesen.

Problemstellungen zu erreichen, ist üblicherweise der Einsatz nicht-monotoner Logik erforderlich.

Ein Aspekt, der bisher noch nicht behandelt wurde, ist die Frage nach dem adäquaten Abstraktionsniveau des Formalismus und der Wissensrepräsentation.

Newell (1982) vertrat die These der Existenz des sogenannten *Wissenslevels* (*knowledge level*). Das im Modell codierte Wissen ist als kompetenzorientierter Begriff zu verstehen und ermöglicht es dem System, bestimmte Aktionen auszuführen. Das *knowledgle level* verlangt die Verwendung der „richtigen" Abstraktionsstufe. Das Wissen bringt zum Ausdruck, „was" ein System im Stande ist zu tun, nicht „wie" im technischen Sinne eine Aufgabe zu lösen ist. Mit letzterer Frage befasst sich laut Newell das *symbol level* als eine symbolische Repräsentation des Wissens. Im *symbol level* wird das Wissen durch entsprechende Datenstrukturen codiert und den höheren Schichten zur Verfügung gestellt.

Die Wissensbasis beherbergt als formales Modell eine symbolische Repräsentation des Wissens. Diese kann von einem Computer gelesen und zur Problemlösung genutzt werden. Das Wissen in seiner symbolisierten Form und damit die Wissensbasis ist demnach dem *symbol level* nach Newell zuzuordnen.

Meines Erachtens sollte die Sprache zur Formalisierung von Normen für juristische Experten verständlich sein, sodass diese Normen selbst modellieren können. Damit könnten die erstellten Modelle von den Experten validiert werden, womit ein wichtiger Schritt zur Qualitätssicherung getan wäre. Eine Sprache zur Modellierung von Normen sollte daher einen deklarativen Ansatz verfolgen und ausdrücken, „was" getan werden soll und nicht „wie". Es stellt sich die Frage, welchen Abstraktionsgrad Sprachen zur Wissensrepräsentation aufweisen sollten, damit sie diese Anforderung erfüllen können? Boer führt zu Wissensrepräsentationssprachen in seiner Dissertation aus (Boer, 2009, S. 17):

„If a language was designed for representing knowledge we call that language a knowledge representation language. The result of the knowledge representation activity is a knowledge representation or knowledge base, depending on context."

Es bleibt offen, was unter einer Sprache zu verstehen ist, die für die Repräsentation von Wissen enworfen wurde. Die meisten Sprachen zur Wissensrepräsentation sind stark logikorientiert. Manche Sprachen werden auf den ersten Blick nicht der

Wissensrepräsentation zugeordnet. Zu diesen gehören unter anderem Musiknoten oder mathematische Formeln als Paradebeispiel. Diese Sprachen haben gemeinsam, dass sie für eine bestimmte Problemdomäne entwickelt wurden und darauf zugeschnitten sind, die relevanten Aspekte wiederzugeben. Sie operieren auf einem fachlichen Level, das von technischen Details abstrahiert, und nähern sich aus dieser Sicht dem *knowledge level* nach Newell an.

Wenn nun eine Wissensrepräsentationssprache – die auf dem Wissenslevel operiert – der Erstellung der Wissensbasis – die dem Symbollevel zuzuordnen ist – dient, läuft dies auf einen gewissen Widerspruch hinaus.

Zur Beseitigung dieser Diskrepanz kann ins Treffen geführt werden, dass die Wissensbasis als abstraktes Modell des Wissens verstanden werden sollte, das nicht an eine bestimmte symbolische Repräsentation des Wissens gebunden ist. Dem widerspricht jedoch der Umstand, dass sich die gegenständliche Untersuchung mit computergestützten Modellen von Normen befasst. Das Modell muss demnach von einer Maschine verarbeitet werden können, um eine automatisierte Entscheidungsfindung zu ermöglichen. Damit ist eine Kodierung des Wissens in Datenstrukturen erforderlich, womit das computergestützte Modell auf dem *symbol level* anzusiedeln ist. In dieser Hinsicht ist eine Untersuchung, die sich auf die höchst abstrakte Ebene des *knowledge level* beschränkt, nicht hinreichend.

Diese Ambivalenz deutet darauf hin, dass zwischen der externen und internen Repräsentation der Normen unterschieden werden muss. Das *interne* Modell auf dem *symbol level* nach Newell stellt das vom Computer verarbeitbare Modell des Gesetzes dar. Das Grundgerüst des internen Modells besteht im gegenständlichen Fall aus Klassen und Interfaces, die zur Laufzeit in Form von Objekten instanziert werden, um die Normen zu repräsentieren. Für den Fachexperten wird jedoch ein höheres Abstraktionsniveau benötigt, das technische Details verbirgt. Das *externe* Modell des Gesetzes muss für den Experten verständlich sein, sodass dieser die Struktur des Gesetzes und die einzelnen Normen darin wiederfindet. Dies deutet darauf hin, dass eine Syntax benötigt wird, die an der natürlichen Sprache orientiert ist.

Die Unterscheidung zwischen fachlichem und technischem Modell des Gesetzes wirkt auch auf die Kategorisierung des zur Modellierung verwendeten Formalismus (der Wissensrepräsentationssprache) zurück: Für den Experten muss eine

Wissensrepräsentationssprache zur Verfügung stehen, die auf einem geeigneten Abstraktionsniveau die Modellierung von Wissen der Domäne ermöglicht. Welche Aspekte die Sprache ausdrücken muss, hängt dabei maßgeblich vom Zweck des Computermodells ab. Die Sprache sollte sich jedenfalls an das *knowledge level* nach Newell annähern.

Diese Ausführungen sollen jedoch nicht darüber hinweg täuschen, dass das Modell letztendlich vom Computer verarbeitet werden muss. Dafür ist erforderlich, dass das Modell in Datenstrukturen der jeweiligen Programmiersprache übersetzt wird. Im Fall einer objektorientierten Programmiersprache wird das Wissen auf *symbol level* durch entsprechende Objekte repräsentiert. Damit muss meines Erachtens auch im Hinblick auf den Formalismus unterschieden werden, welcher Ebene dieser zuzuordnen ist, je nachdem ob dieser der Erstellung des abstrakten Wissensmodells oder des symbolischen Computermodells dient.

Es kann argumentiert werden, dass das von Newell vorgeschlagene Wissenslevel in Wahrheit nie erreicht werden kann. Sobald die Repräsentation des Wissens gelungen ist, hat man sich zwangsweise bestimmter Symbole bedient und damit streng genommen das Wissenslevel verlassen. Man darf diese Kritik jedoch nicht überwerten. Newells Wissenslevel ist mehr als Inspiration und Richtung zu verstehen als ein Ziel, das es zu erreichen gilt.

Diese Feststellung hat Auswirkungen auf die hier getroffene theoretische Unterscheidung zwischen externem und internem Modell. Es ist nämlich keineswegs denkunmöglich, dass ein Modell ein hohes Abstraktionsniveau aufweist und fachlich orientiert ist, gleichzeitig aber in den Datenstrukturen (Objekte) der jeweiligen Sprache erstellt wurde. Der Übergang zwischen den beiden Extremen ist fließend und kann sogar gänzlich entfallen. Um von einem „externen Modell“ im Sinne dieser Ausführungen zu sprechen, wird jedoch eine gewisse Mindesthöhe der Abstraktionsstufe vorausgesetzt. Umgekehrt kann nur dann von einem „internen Modell“ gesprochen werden, wenn das Modell direkt in der Syntax der jeweiligen Programmiersprache erstellt wurde. Erfüllt ein Modell beide Anforderungen, ist es zugleich „internes“ und „externes“ Modell in diesem Sinne.

Wenn externes und internes Modell mit unterschiedlichen Sprachen erstellt werden, muss letztendlich eine Übersetzung erfolgen. Diese Übersetzung des externen in das interne Modell kann automatisiert im Voraus oder ad hoc, zum Zeitpunkt der rechtlichen Beurteilung, erfolgen. Selbstverständlich ist es zudem möglich, dass die Modellierung durch

ein grafisches Werkzeug unterstützt wird, das im Hintergrund das computergestützte interne Modell „implizit“ erzeugt. In diesem Fall existiert das externe Modell als grafische Darstellung des Gesetzes. Externes und internes Modell müssen jedenfalls eine kompatible formale Semantik aufweisen, da sich ansonsten keine exakte Übersetzung bewerkstelligen lässt.

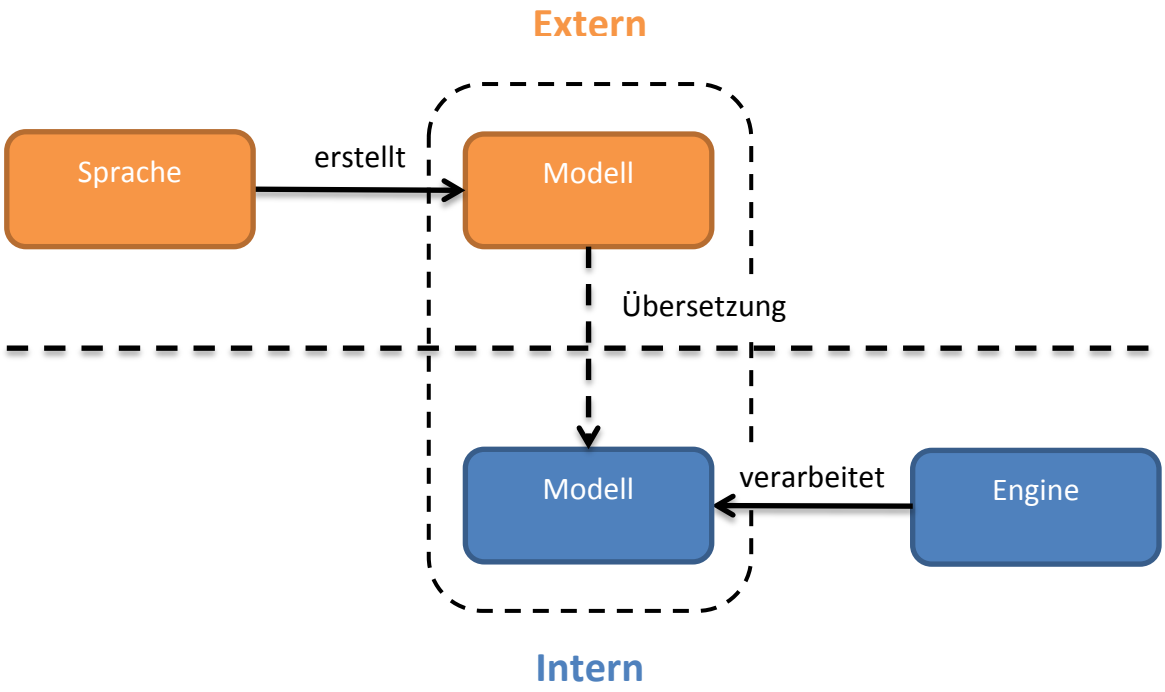

Abbildung 24: Zusammenhänge zwischen Sprache, Modell und Inferenzmaschine

Abbildung 24 zeigt die Zusammenhänge zwischen Sprache, Modell und Inferenzmaschine in einer anschaulichen Grafik. Die strichlierte horizontale Linie deutet den Übergang zwischen internem und externem Modell an, der auch gänzlich entfallen kann, wie oben ausgeführt wurde. Wie ersichtlich, muss das externe Modell des Gesetzes in die internen Datenstrukturen übersetzt werden, sodass dieses von der Inferenzmaschine verarbeitet werden kann.

Für den Benutzer eines Expertensystems, das ist in diesem Fall der juristische Experte, der das Modell erstellt, ist die Unterscheidung zwischen internem und externem Modell nicht immer augenscheinlich. Das gilt insbesondere dann, wenn die Übersetzung der internen Repräsentation der Normen in die externe automatisiert ohne manuellen Eingriff erfolgt.

Zusammengefasst handelt es sich bei der theoretischen Unterscheidung zwischen internem und externem Modell der Normen um ein theoretisches Konzept. Das interne Modell ist

technisch orientiert, in der Syntax der jeweiligen Sprache implementiert und bedient sich der vorhandenen Datenstrukturen. Es bedarf damit keiner Übersetzung in die Datenstrukturen der jeweiligen Programmiersprache und kann direkt von der Inferenzmaschine verarbeitet werden. In der Regel weist das interne Modell aus fachlicher Sicht einen niedrigen Abstraktionsgrad auf. Im Gegensatz dazu ist das externe Modell fachlich orientiert und weist einen hohen Abstraktionsgrad auf. Sofern das externe Modell sich direkt der Syntax der jeweiligen Sprache bedient, kommt es zu einer Überschneidung mit dem internen Modell[165].

Die hier vorgeschlagene Trennung ist Voraussetzung, um einzelne Probleme, die bei der Formalisierung gelöst werden müssen, besser kategorisieren und dem jeweiligen Problemkreis zuordnen zu können. Damit eröffnet sich die Möglichkeit, dass internes und externes Modell separat von dazu befähigten Experten entwickelt werden können. Dafür ist jedoch zusätzlich erforderlich, dass ein gemeinsamer Standard etabliert wird, der eine gemeinsame Semantik definiert.

Es ist durchaus möglich, dass die beiden Modelle in ihrer Ausdrucksstärke voneinander abweichen. Dies ist beispielsweise dann der Fall, wenn das externe Modell Regeln und Argumentationsstrukturen abbilden kann, das interne Modell jedoch auf die Formalisierung von Regeln beschränkt ist.

11.5. Zeitliche Aspekte von Rechtsnormen

Dieser Abschnitt befasst sich mit der Analyse der zeitlichen Aspekte von Gesetzen, um eine Grundlage für die Analyse verfügbarer Technologien und die Entwicklung eines eigenen Ansatzes zur Formalisierung der zeitlichen Dimensionen des Rechts zu schaffen.

11.5.1. Die primären zeitlichen Dimensionen

Es muss zwischen folgenden zeitlichen Dimensionen von Rechtsnormen unterschieden werden:

---

[165] Die Unterscheidung zwischen internem und externem Modell erinnert an eine ähnliche Terminologie, die im Bereich der Domain Specific Languages (DSLs) gebräuchlich ist. Dort wird zwischen internen und externen DSLs unterschieden. Alle internen DSLs haben das Charakteristikum gemeinsam, dass sie auf einer Gastsprache basieren und sich deren Infrakstruktur bedienen. Für externe DSLs ist charakteristisch, dass diese ihre Infrastruktur von Grund auf selbst entwickeln (Gosh, 2011). Zum Unterschied zur hier getroffenen Differenzierung zwischen internem und externem Modell weist eine DSL stets dasselbe Abstraktionsniveau wie die Problemdomäne auf. Die Sprache zur Erstellung des externen Modells kann daher als DSL qualifiziert werden.

Die *Existenz* (Geltung) einer Norm beginnt mit ihrer Kundmachung und endet durch einen späteren Normsetzungsakt. Existenz bedeutet, dass die Norm dem Rechtsbestand angehört. Sie endet, wenn der Norm durch eine spätere Rechtsvorschrift materiell oder formell derogiert wird. Die Geltung einer Norm ist von ihrem zeitlichen Geltungsbereich zu unterscheiden.

Der *zeitliche Geltungsbereich* ist jener Zeitraum, auf den sich der Inhalt der Norm bezieht. Er beginnt mit dem Inkrafttreten und endet mit dem Außerkrafttreten einer Norm. Inkrafttreten im Sinne des Art. 49 B-VG bedeutet, dass das Gesetz auf jene Sachverhalte anzuwenden ist, die sich nach diesem Zeitpunkt bis zum Außerkrafttreten der Norm ereignen. Sofern nichts anderes bestimmt ist, treten die Gesetze mit Ablauf des Tages ihrer Kundmachung in Kraft.

Art. 49 B-VG spricht vom Inkrafttreten von Gesetzen und meint damit undifferenziert den Beginn der verbindlichen Kraft, ohne zwischen dem Bedingungs- und Rechtsfolgenbereich zu unterscheiden. Dem entspricht auch die legistische Übung, bloß das Inkrafttreten einer Norm zu regeln. Sofern nicht anders bestimmt, beginnt und endet der Bestimmungs- und der Rechtsfolgenbereich mit dem Inkraft- bzw. Außerkrafttreten des Gesetzes. Vom Inkrafttreten ist strikt der Zeitpunkt der Kundmachung zu unterscheiden. Eine Rechtsvorschrift ist auch bereits dann Bestandteil der Rechtsordnung, wenn sie noch nicht in Kraft steht.

Entsprechend der Struktur einer Verhaltensnorm, bestehend aus Tatbestand und Rechtsfolge, muss bei genauerer Betrachtung des zeitlichen Geltungsbereichs zwischen Bedingungs- und Rechtsfolgenbereich unterschieden werden. Als *Bedingungsbereich* wird jener Zeitraum verstanden, währenddessen die im Tatbestand umschriebenen Sachverhalte verwirklicht sein müssen, damit das zur Vollziehung ermächtigte Organ die Rechtsfolge verhängen darf. Vom Bedingungsbereich ist der *Rechtsfolgenbereich* als jener Zeitraum zu unterscheiden, in dem das ermächtigte Organ die Rechtsfolge verhängen kann, d.h. jener Zeitraum, in dem die Norm im Einzelfall anzuwenden ist und daher (tatsächlich) Rechtsfolgen produziert. Die beiden Zeiträume können auch auseinanderfallen, und der Rechtsfolgenbereich kann weit über den Bedingungsbereich hinausreichen (Öhlinger & Eberhard, 2014, S. 204).

Beispielsweise ist auf das Einkommen, das im Jahr 2013 vom 1.1. bis zum 31.12. erworben wurde, das Steuerrecht zum 31.12.2013 anwendbar, da der Abgabenanspruch mit diesem Stichtag entsteht. Dies gilt selbst dann, wenn in der Zwischenzeit eine neue (für die Zukunft geltende) Regelung erlassen wurde.

Im Detail können sich recht komplizierte Konstellationen aus dem Zusammenspiel von Bedingungs- und Rechtsfolgenbereich ergeben. Im Fall einer Rückwirkung beginnt der Bedingungsbereich naturgemäß vor dem Inkrafttreten des Gesetzes und damit vor dem Rechtsfolgenbereich.

Ein Modell, das die zeitlichen Charakteristika von Normen adäquat repräsentieren will, muss mit den zeitlichen Dimensionen von Rechtsnormen – Existenz, Inkraftstehen sowie Bedingungs- und Rechtsfolgenbereich – umgehen können. Die Frage, ob eine Norm in Kraft steht und anwendbar ist, kann immer nur relativ zu einem zeitlichen Bezugspunkt beantwortet werden (Governatori, Rotolo, Riveret, Palmirani, & Sartor, 2007; Palmirani & Brighi, 2006). Dieser „Beobachtungszeitpunkt“ ist maßgeblich, da sich das Rechtssystem und dessen Normenbestand im Laufe der Zeit verändert, sodass eine allgemeingültige Antwort hinsichtlich der in Kraft stehenden Normen unmöglich ist.

Die konkrete Bedeutung der Begriffe „Geltung“, „Inkraftstehen“ usw. variiert je nach Rechtsordnung und ist von der Rechtstradition abhängig. Ferner ist es möglich, dass einzelne Dimensionen in einer Rechtsordnung unbekannt oder zusätzliche vorhanden sind (Palmirani, Governatori, & Contissa, 2010, S. 4). Dies macht die ohnehin schon anspruchsvolle Aufgabe der Repräsentation der zeitlichen Dimensionen von Rechtsnormen noch schwieriger, soll ein Modell über eine Rechtsordnung hinaus verwendbar sein.

11.5.2. Analyse der vorhandenen Literatur

In der Literatur auf dem Gebiet KI und Recht wurden die zeitlichen Aspekte des Rechts bereits intensiv diskutiert. Die verwendete Terminologie ist jedoch nicht immer einheitlich, und die Bedeutung der Begriffe scheint voneinander abzuweichen. Aus diesem Grund wird an dieser Stelle ein kurzer Überblick über die vorhandenen Untersuchungen der zeitlichen Dimensionen des Rechts auf dem Gebiet der KI und Recht gegeben.

Governatori und Rotolo sprechen von zwei Dimensionen, die es bei Änderungen des Rechtssystems zu berücksichtigen gilt (Governatori & Rotolo, 2009, S. 170):

„The reason why we have multiple versions of a normative system is that norms have different temporal dimensions: the time of validity of a norm (when the norm enters in the normative system) and the time of effectiveness (when the norm can produce legal effects).”

Der Begriff „validity“, der übersetzt so viel wie „Geltung“ bedeutet, entspricht der Existenz einer Norm. Eine Rechtsnorm ist „gültig“ in diesem Sinne, sobald sie Teil des Rechtssystems ist[166].

Unter „effectiveness“ (Effektivität) wird jener Zeitraum verstanden, in dem die Norm Rechtsfolgen erzeugen kann. Die Autoren differenzieren in diesem Zusammenhang nicht näher zwischen Bedingungs- und Rechtsfolgenbereich, sodass davon auszugehen ist, dass mit „Effektivität“ allgemein der Zeitraum bezeichnet wird, in dem die Norm in Kraft steht.

Die Autoren diskutieren im Zusammenhang mit der Aufhebung und Nichtigerklärung von Normen auch die Behandlung von „früheren“ Rechtsfolgen (Governatori & Rotolo, 2009, S. 159). Gemeint sind damit Effekte einer Norm, die einer früheren (außer Kraft getretenen) Rechtslage angehört, die bis zu einem (zukünftigen) Zeitpunkt „nachwirken“ und damit zu berücksichtigen sind. Ohne dass die Autoren selbst von einer dritten zeitlichen Dimension sprechen, entspricht diese Bedeutung weitgehend jener des Rechtsfolgenbereichs als ein Zeitraum, in dem die Rechtsfolgen einer Norm anzuwenden sind.

In Übereinstimmung mit diesen Ausführungen sprechen Governatori, Rotolo und Rubino in einem späteren Beitrag über die Implementierung der „temporal defeasible logic“ in Java von drei zeitlichen Dimensionen (Governatori et al., 2010, S. 55 f.):

„[…] typically one has to consider at least three temporal dimensions: The time of validity of the norms (when a norm is enacted), the time of force of the norm (when a norm can produce an effect), and the time of efficacy (when a norm produces an effect).”

Unzweifelhaft ist mit „validity“ jener Zeitpunkt gemeint, an dem die Norm Teil des Rechtssystems wird, in der Regel mit deren Veröffentlichung. Es geht nicht klar hervor, ob mit „time of force“ der Rechtsfolgen- bzw. Bedingungsbereich oder beides gemeint ist. Der üblichen Bedeutung entsprechend bezieht sich das Inkraftstehen auf die gesamte Norm und

[166] Der Zeitpunkt, ab dem eine Norm Teil der Rechtsordnung ist, kann nicht einheitlich bestimmt werden und variiert nach der Rechtsordnung. In Italien ist eine Norm beispielsweise erst mit dem Inkrafttreten Teil der Rechtsordnung und nicht bereits mit der Publikation in einem offiziellen Journal (wie in Österreich) (Palmirani, 2011, S. 125).

demnach auf Bedingungs- und Rechtsfolgenbereich gleichermaßen. Der Zeitraum, in dem eine Norm Rechtwirkungen erzeugt (efficacy), scheint auf den Rechtsfolgenbereich hinzudeuten.

Angemerkt sei, dass im zitierten Beitrag die „temporal defasible logic“ behandelt wird. Eine anwendbare anfechtende Regel blockiert dabei die angefochtene Bestimmung, sodass deren Rechtsfolgen nicht zur Anwendung gelangen (Governatori et al., 2010, S. 55): *“Notice that these […] norms suspend the general traffic block restrictions, but do not terminate the efficacy of the norm.”*

Das deutet darauf hin, dass mit „efficacy“ in Wahrheit der Bedingungsbereich und *nicht* der Rechtsfolgenbereich einer Norm gemeint ist. Das Zitat ist nämlich frei übersetzt so zu verstehen, dass in zeitlicher Hinsicht grundsätzlich die Anwendbarkeit der Norm fortbesteht, deren Rechtsfolgen jedoch gehemmt werden.

Palmirani et al. präsentieren in ihrem Beitrag eine Erweiterung der LKIF-Rule-Language (Boer et al., 2007; Gordon, 2008b), um die zeitlichen Dimensionen von Rechtsnormen erfassen zu können (Palmirani, Governatori, & Contissa, 2011, S. 131):

„[…] often the temporal model, mostly based on the events, is not specific for the legal domain that requires at least tree axis: time of in force, time of efficacy and time of application of the norms.”

Es fällt auf, dass statt „validity“ als dritte zeitliche Dimension „application“ angeführt wird. Die Autoren verwenden die Begrifflichkeit mit abweichender Bedeutung. Beispielsweise wird ausgeführt, dass Body (das Antezedens) und Head (die Konsequenz) einer Regel unterschiedliche Zeitpunkte des Inkrafttretens haben können. Der Zeitpunkt des Inkrafttretens wird daher in diesem Fall gleichbeutend mit dem Beginn der Anwendbarkeit von Bedingungs- und Rechtsfolgenbereich verstanden.

Folgendes Zitat über die Rückwirkung von Normen zeigt, dass die Autoren, offenbar abweichend von der in diesem Kapitel gebräuchlichen Terminologie, den Begriff „efficacy“

mit der Konsequenz einer Regel verbinden[167] (Palmirani, Governatori, & Contissa, 2011, S. 133):

„[…] we want to enter into the details of retroactive norms: the body's enter into force date is t1, the head's time of efficacy is t2 (e.g. usually in the retroactive norms t2 <t1), the rule is applied in a third time t3 that is the minor or t1 and t2 combined with other eventually conditionals (e.g. suspension, emergency external events, etc.) expressed in sentences of the body."

Die Anwendbarkeit der Norm („applicability") wird hier recht weit gesehen als ein Zeitraum, der dem Bedingungs- oder Rechtsfolgenbereich entspricht, je nachdem welcher dieser Zeiträume früher beginnt. Überdies legt der Text nahe, dass die Anwendbarkeit der Norm im Kontext eines konkreten Sachverhalts zu sehen ist. Nur dann ergibt die Betrachtung einer Norm im Zusammenhang mit anderen Regeln („emergency external events" etc.) einen Sinn.

Der Rechtsfolgenbereich im theoretischen Sinne ist jedoch (zunächst) isoliert von anderen Normen zu betrachten. Ob ein Zeitpunkt vom Rechtsfolgenbereich einer Norm erfasst wird, hängt ausschließlich von dessen zeitlicher Ausdehnung ab.

Die drei primären zeitlichen Dimensionen – force, efficacy und applicability – werden auch in jüngerer Literatur, die temporale Aspekte des Rechts behandelt, verwendet. Palmirani et al. (2012) präsentieren das Konzept eines Frameworks, das in der Lage ist, unterschiedliche Schichten des Rechts und dessen zeitliche Dimensionen abzubilden. Für die Formalisierung von Normen in Form von Regeln wird derzeit LegalRuleML entwickelt, eine Sprache, mit deren Hilfe sich die dargestellten Zeitachsen modellieren lassen (Athan et al., 2013b).

11.5.2.1. Konklusion

In der (durchwegs englischsprachigen) Literatur auf dem Gebiet AI and Law werden die Begriffe *force*, *efficacy* und *applicability* verwendet, um die zeitlichen Dimensionen von Rechtsnormen zu beschreiben. Nicht immer lässt sich deren Bedeutung direkt aus den Ausführungen ableiten. Wie aus der Abhandlung in diesem Abschnitt hervorgeht, wird gelegentlich *validity* (Geltung oder Existenz) als zeitliche Dimension angeführt. Gemeint ist

[167] Der Begriff „efficacy" wird hier offenbar als ein Zeitpunkt verstanden, zu dem die Norm „Effekte" erzeugt. Eine Novelle, die Rechtsvorschriften rückwirkend ändert, entfaltet ihre Rechtswirkungen gewissermaßen in der Vergangenheit.

damit der Zeitpunkt, ab dem eine Norm dem Rechtsbestand angehört[168]. Es ist daher richtiger, von *vier* zeitlichen Dimensionen der Rechtsnormen zu sprechen.

Unter *force* wird das Inkraftstehen einer Norm verstanden. Es ergibt sich oftmals erst aus dem Kontext, ob der Bedingungs- und/oder Rechtsfolgenbereich gemeint ist. Es ist meines Erachtens verwunderlich, dass das Inkraftstehen als eigenständige dritte Zeitachse repräsentiert wird, da der zeitliche Anwendungsbereich sich aus den zwei anderen Dimensionen zusammensetzt. Der Begriff Inkraftstehen ist daher nicht viel mehr als eine Abkürzung, wenn man sich auf Bedingungs- und Rechtsfolgenbereich gleichzeitig beziehen möchte.

Es lässt sich auch unter Berücksichtigung der Sinnzusammenhänge nicht eindeutig erschließen, ob mit *efficacy* der Bedingungsbereich einer Norm gemeint ist. Gelegentlich wird dem Begriff eine weitere Bedeutung beigemessen, als ein Zeitraum, in dem die Norm „Effekte", d.h. Rechtswirkungen, entfaltet. Dieser Zeitraum kann dabei vor und auch nach dem Zeitraum des Inkraftstehens liegen.

Die dritte Zeitachse *applicability* kann im Wesentlichen als Rechtsfolgenbereich qualifiziert werden und kennzeichnet den Zeitraum, in dem die Rechtsfolgen einer Norm anwendbar sind, d.h. vollzogen werden dürfen. In der Literatur wird der Begriff jedoch weiter gesehen und eine Norm oftmals erst dann als „anwendbar" erachtet, wenn „konkrete Bedingungen" erfüllt bzw. nicht erfüllt sind. Als Beispiele für solche „Bedingungen" werden Notsituationen oder die vorübergehende Aussetzung der Anwendbarkeit einer Norm angeführt (Palmirani et al., 2012, S. 6). Damit wird deutlich, dass unter „Bedingung" in diesem Sinne nicht ein Element des Tatbestands zu verstehen ist, sondern externe Ereignisse, die dazu führen, dass die Rechtsfolgen einer Norm im Einzelfall nicht angewendet werden[169].

Aus theoretischer Sicht kennzeichnet der Rechtsfolgenbereich einen abstrakten Zeitraum, in dem die Effekte einer Norm unabhängig vom Einzelfall grundsätzlich wirken. Der in der in diesem Abschnitt behandelten Literatur gebräuchliche Begriff „applicability" ist hingegen eher als *konkreter* Rechtsfolgenbereich zu verstehen, der sich im Einzelfall ergibt. Eine Norm

---

[168] Dies ist in Österreich der Zeitpunkt der Kundmachung auf die gesetzlich vorgesehene Art.

[169] Dieses Phänomen wird in der Literatur im Zusammenhang mit der „anfechtbaren Schlussfolgerung" untersucht.

ist in diesem Sinne „anwendbar“, sofern ihr Tatbestand im Einzelfall erfüllt ist und sie zu einem bestimmten Zeitpunkt Rechtswirkungen erzeugt. Sie erzeugt dann Rechtswirkungen, wenn ihre Rechtsfolgen nicht von einer anderen Norm blockiert werden. Dieser in der Begrifflichkeit vorhandene theoretische Bedeutungsunterschied muss bei der Nutzung und Weiterentwicklung der vorhandenen Ansätze beachtet werden.

Die hier vertretene und durch eine Analyse des Schrifttums gewonnene Ansicht wird nicht zuletzt durch einen früheren Artikel bestätigt, an dem die Autoren der diskutierten Werke beteiligt waren (Governatori et al., 2007, S. 155):

„Accordingly, also modifications have three temporal dimensions, these being attached to the conditions, to the effects, and to the overall conditional. These temporal dimensions refer to the efficacy, applicability, and force of the provision, respectively.”

11.5.2.2. Diskussion

Die *vier* zeitlichen Dimensionen einer Rechtsnorm sind *force*, *efficacy* (effectiveness), *applicability* und *validity*. Die Begriffe entsprechen im Wesentlichen der deutschsprachigen Terminologie auf dem Gebiet der Rechtstheorie, nämlich Inkraftstehen, Bedingungsbereich, Rechtsfolgenbereich und Existenz (Geltung).

Der Begriff „applicability“ wird in der Literatur regelmäßig mit der Anwendbarkeit der Norm im Einzelfall assoziiert. Im rechtstheoretischen Sinne wird der Rechtsfolgenbereich jedoch abstrakt als jener Zeitraum verstanden, in dem eine Norm anzuwenden ist, d.h. Rechtsfolgen erzeugt – unabhängig davon, ob der Tatbestand im Einzelfall erfüllt ist oder nicht.

Wie bereits ausgeführt, sollte das Inkraftstehen für sich nicht als eigenständige zeitliche Dimension bezeichnet werden, sondern als begriffliche Abkürzung, wenn auf Bedingungs- und Rechtsfolgenbereich gleichzeitig Bezug genommen werden soll. Existenz (Geltung) und Inkrafttreten stehen jedenfalls in einem engen Zusammenhang, da – sofern nichts anderes bestimmt ist – Gesetze mit Ablauf des Tages ihrer Kundmachung in Kraft treten (Art. 49 B-VG).

Im Ergebnis lässt sich festhalten, dass auf dem Gebiet KI und Recht die Formalisierung zeitlicher Aspekte des Rechts intensiv behandelt wurde. Dies zeigt sich in der großen Zahl an Beiträgen, von denen einige ausgewählte in Abschnitt 11.5.2. zitiert wurden.

Die von den Autoren verwendete Terminologie zur Bezeichnung der temporalen Dimensionen ist nicht immer einheitlich. Oftmals erschließt sich die Bedeutung eines Begriffs erst aus dem Kontext, in dem er verwendet wird. Die in diesem Abschnitt durchgeführt Analyse hatte zum Ziel, die im Schrifttum verwendete Begrifflichkeit und deren Bedeutung darzustellen sowie auf Abweichungen hinzuweisen.

Soweit ersichtlich, ist die Bedeutung der identifizierten Zeitachsen zumindest im Kern weitgehend einheitlich. Im Detail kann jedoch nicht ausgeschlossen werden, dass die Autoren gelegentlich einem Begriff eine abweichende Bedeutung zumessen. Insbesondere kann der Gehalt von „force“ und „efficacy“ an mancher Stelle nicht mit hundertprozentiger Sicherheit erschlossen werden. Auf etwaige Abweichungen ist beim Studium der vorhandenen Literatur Bedacht zu nehmen, wenn diese als Grundlage für Abhandlungen oder Implementierungen von Software, wie beispielsweise einer Inferenzmaschine, verwendet wird. Ferner sei darauf hingewiesen, dass die Bedeutung der temporalen Dimensionen von der jeweiligen Rechtsordnung abhängig ist, sodass auch in dieser Hinsicht Abweichungen auftreten können.

Vorhandene Literatur lässt sich gemäß dem Schwerpunkt der Abhandlung grob in die Kategorien „Logische Ansätze“ (Governatori et al., 2007; Governatori & Rotolo, 2009; Marín & Sartor, 1999), Sprachen (Athan et al., 2013b; Palmirani et al., 2010, 2012; Palmirani, Governatori, & Contissa, 2011) und „Implementierungen“ (Governatori et al., 2010; Rubino & Rotolo, 2009) gliedern.

Nach dem Wissen des Autors existiert nur ein einziges Projekt, das über das Stadium der theoretischen Konzeption hinausgegangen ist. Die von Rubino und Rotolo (2009) erstellte Implementierung temporaler Logik in Java scheint jedoch auf der Webseite des Projekts nicht mehr verfügbar zu sein[170].

Für die gegenständliche Abhandlung wurden möglichst aktuelle und repräsentative Werke auf dem Gebiet KI und Recht ausgewählt. Die Auswahl ist jedoch keinesfalls vollständig, es existieren einige Werke mehr. Aus Platzgründen und weil für den Zweck dieses Abschnitts,

[170] Der Link http://www.defeasible.org/implementations/TDLJava/index.html konnte am 10.7.2014 nicht mehr abgerufen werden.

der Darstellung der zeitlichen Dimensionen von Normen, kaum ein Mehrwert gewonnen werden könnte, wurde die Abhandlung auf die ausgewählte Literatur beschränkt.

11.5.3. Interne und externe Zeit

Ein Aspekt, der bislang nicht erwähnt wurde, ist die Unterscheidung zwischen externer und interner Zeit. Mit „externer Zeit“ sind in jüngerer Literatur jene Zeitangaben gemeint, die nicht direkt im Text einer Bestimmung stehen, sondern den Lebenszyklus der Norm betreffen (Existenz, Bedingungs- und Rechtsfolgenbereich). „Interne Zeit“ bezeichnet Zeitpunkte, die im Text einer Bestimmung direkt enthalten sind (Palmirani et al., 2010; Palmirani, Governatori, & Contissa, 2011).

Diese Trennung ist jedoch nicht durchgehend möglich. Folgendes Beispiel zeigt eine (fiktive) Bestimmung, deren normativer Gehalt **nicht** vom zeitlichen Bezugsbereich getrennt werden kann: *„Spikereifen dürfen nur in der Zeit vom 1. November bis 30. April des folgenden Jahres verwendet werden.“*

Die Zeitpunkte 1. November und 30. April sind als interne Zeit zu qualifizieren, da sie im Text der Bestimmung direkt stehen. Sie begrenzen den Bedingungsbereich der Norm. Im Ergebnis ist es freilich unerheblich, aus welchem Grund – externe oder interne Zeit – ein Sachverhalt vom zeitlichen Bedingungsbereich erfasst wird oder nicht.

Für die Formalisierung selbst ist diese theoretische Unterscheidung jedoch sehr wohl erheblich. Der externe Bedingungsbereich wird nämlich dem ersten Problemkreis zugezählt und wird daher vor der eigentlichen automatisierten Rechtsanwendung behandelt. Die Prüfung des im Tatbestand einer Norm zum Ausdruck gebrachten internen Bedingungsbereichs erfolgt hingegen im darauffolgenden Schritt, im Zuge der automatisierten Rechtsanwendung.

In der Literatur wird gelegentlich interner und externer Zeit eine differenzierte Bedeutung beigemessen. Auf diese abweichenden Konnotationen soll nun in geraffter Form eingegangen werden.

Marín und Sartor (1999) verstehen unter „interner Zeit“ das Subsumtionsintervall einer Norm in zeitlicher Hinsicht und meinen damit jenen Zeitraum, in dem ein Ereignis geschehen muss, sodass es von der Norm erfasst wird. Diese Bedeutung entspricht also dem Bedingungsbereich im rechtstheoretischen Sinn.

Unter „externer Zeit“ verstehen die genannten Autoren jenen Zeitraum, in dem die Norm Teil des Rechtssystems ist. Innerhalb dieses Zeitraums ist eine Norm *gültig*. Diese Bedeutung deckt sich mit der Existenz gemäß der in Abschnitt 11.5.1. getroffenen Definition. Die Geltung einer Norm wird durch andere Rechtsvorschriften explizit oder implizit durch Derogation beendet (Marín & Sartor, 1999, S. 94).

Governatori und Rotolo (2009) präsentieren ein formales Modell, das mit Veränderungen des Rechtssystems umgehen kann. Die Autoren konzentrieren sich dabei auf Änderungen, die durch Aufhebungen (ex nunc) und Nichtigerklärung (ex tunc) von Normen verursacht werden.

Dieses Modell unterscheidet zwei Zeitlinien: Eine *interne* Zeitlinie (inner time-line) innerhalb einer Version des Rechtsystems zu einem bestimmten Zeitpunkt sowie eine *äußere* Zeitlinie (outer time-line) relativ zur Entwicklung des Rechtsystems im Lauf der Zeit. Die Trennung der Zeitlinie ist notwendig, um rückwirkende Änderungen abbilden zu können. Die von Governatori und Rotolo verwendete Terminologie ist daher nicht auf eine konkrete Norm bezogen, sondern ist im Kontext des entwickelten formalen Modells zu verstehen.

Dieser kurze Überblick zeigt, dass die in der Literatur verwendete Terminologie (auch) im Hinblick auf die Begriffe „interne“ und „externe“ Zeit nicht einheitlich ist.

11.5.4. Dauer von normativen Effekten

In Abschnitt 7.4. wurde bereits ausgeführt, dass manche Rechtsfolgen nur vorübergehend anwendbar sind, solange der Tatbestand einer Norm erfüllt ist. Andere Rechtsfolgen wiederum sind persistent und dauern solange an, bis ein späteres Ereignis diese beendet.

Diesen Feststellungen ist in diesem Kontext hinzuzufügen, dass die von einem rechtskräftigen individuellen Rechtsakt – wie beispielsweise einem Bescheid – erzeugten Rechtsfolgen nicht direkt von einer Änderung der Rechtsgrundlage betroffen sind. Selbst wenn die Normen, auf denen ein Bescheid beruht, später geändert werden, beeinflusst das den normativen Inhalt der Entscheidung selbst nicht. Dies gilt selbst dann, wenn eine Norm nicht bloß *pro futuro*, sondern rückwirkend aufgehoben wird. Selbstverständlich könnte im Fall einer Novelle unter Umständen aufgrund der geänderten Rechtslage neu entschieden werden, wobei die neue Entscheidung die Rechtswirkungen des älteren Rechtsakts verdrängt.

Ein Ansatz zur Formalisierung muss demnach auf (persistente) individuelle Rechte (und Pflichten) Bedacht nehmen. Solche können bei der rechtlichen Beurteilung ähnlich den Fakten des Sachverhalts vorausgesetzt werden und fließen so in den Entscheidungsfindungsprozess ein. Die (fortdauernden) Rechtsfolgen einer individuellen Entscheidung dürfen jedoch nicht mit der generellen Anwendbarkeit einer Norm in zeitlicher Hinsicht verwechselt werden.

Manche Autoren sprechen von periodischen Rechtswirkungen und meinen damit Normen, die basierend auf einem unvorhersehbaren Ereignis, wie z.B. Schneefall, während dessen Fortbestand anwendbar sind (Governatori et al., 2010). Solche Rechtsfolgen können als „vorübergehend“ gemäß der eingangs erwähnten Unterscheidung qualifiziert werden.

Ebenso konstituieren dauerhafte Rechtsfolgen, die eine periodische Vollziehung erfordern, keine eigene Kategorie. Solche Rechtsfolgen haben gemeinsam, dass der im Rechtsakt zum Ausdruck gebrachte Sollzustand nicht durch einen einzigen Vollzugsakt hergestellt werden kann. Vielmehr sind zumeist in periodischen Abständen weitere Vollzugsakte zu setzen.

Derartige Rechtsfolgen werden beispielsweise durch einen positiven Leistungsbescheid im Sinne des KOVG bewirkt. Dem Leistungsempfänger ist monatlich die Rente in der gesetzlich vorgesehenen Höhe auszubezahlen. Ein anderes Beispiel ist der Beschluss des Gerichts im Exekutionsverfahren, in dem die Lohn- und Gehaltsexekution bewilligt wird. Der Arbeitgeber hat dabei nicht mehr an den Arbeitnehmer (den Schuldner) zu zahlen, sondern an den betreibenden Gläubiger, bis die Forderung getilgt ist. In letzterem Fall handelt es sich natürlich nicht um einen Vollzugsakt einer Behörde oder eines Gerichts.

Grundsätzlich kann in Frage gestellt werden, ob ein Formalismus zwischen transienten und persistenten Rechtsfolgen überhaupt unterscheiden muss. Governatori et al. (2010, S. 52) treten für die Notwendigkeit dieser Unterscheidung ein, da (1) der Einsatz persistenter Regeln zu einem Geschwindigkeitsgewinn führt und (2) die Begrenzung des Zeitintervalls für die Ableitung der Rechtsfolgen – beginnend mit einem Startzeitpunkt und endend mit dem Zeitpunkt der rechtlichen Beurteilung – zu falschen Ergebnissen führt.

Für komplexe Szenarien, d.h. Fälle, die sich über einen längeren Zeitraum erstrecken, in dem mehrere rechtlich relevante Ereignisse geschehen, ist es tatsächlich erforderlich, dass durch frühere Ereignisse ausgelöste Rechtsfolgen entsprechend berücksichtigt werden. Allerdings ist selbst in diesen Fällen der Zeitraum, für den die Rechtsfolgen bestimmt werden sollen,

durch ein bestimmtes Datum begrenzt. Das Ende des Zeitraums könnte beispielsweise das Datum der Antragstellung sein oder der Zeitpunkt, an dem im Falle eines Dauerdelikts das letzte Mal die Tathandlung verwirklicht wird. Aus diesem Grund scheint es meines Erachtens fraglich, ob eine Schlussfolgerung „mit offenem Zeitintervall", wie sie Governatori et al. präsentieren, tatsächlich erforderlich ist.

Im Hinblick auf persistente Rechtsfolgen gilt es außerdem zu untersuchen, ob es möglich ist, Fälle zu mehreren Zeitpunkten innerhalb eines Zeitraumes zu beurteilen, um zu einem Gesamturteil zu gelangen. In der Tat scheint es gerade im Verwaltungsrecht Anhaltspunkte für die Möglichkeit eines solchen Ansatzes zu geben. So ist beispielsweise im Steuerrecht üblicherweise die Rechtslage zum 31.12. des jeweiligen Jahres erheblich, um die Steuerschuld bzw. -gutschrift für dieses Jahr festzustellen. Soll nun das Ergebnis über mehrere Jahre hinweg festgestellt werden, so scheint es auszureichen, das Ergebnis des Vorjahres als Fakt des Sachverhalts bei der rechtlichen Beurteilung des aktuellen Jahres zu berücksichtigen.

Auch im Versorgungsrecht (HVG, KOVG etc.) erfolgt die rechtliche Beurteilung – die Berechnung von monatlichen Geldleistungen – grundsätzlich in Abschnitten. Die Leistungen sind jeweils am Monatsersten fällig[171], sodass die an diesem Tag geltende Rechtslage für die Berechnung maßgeblich ist.

11.5.5. Gültigkeit von Normen

Die Rechtsordnung ist kein statisches Gebilde, sondern häufig von Änderungen betroffen. Normen können durch Novellen und in vielen Rechtsordnungen durch ein Verfassungsgericht (rückwirkend) aufgehoben werden (Art. 140 B-VG).

Rechtsvorschriften können demnach im Lauf der Zeit ihre Gültigkeit verlieren. Unter Gültigkeit ist jener Zeitraum zu verstehen, der mit dem Inkrafttreten beginnt und mit dem Außerkrafttreten der Norm endet. Hinsichtlich des Außerkrafttretens ist zu unterscheiden, ob nur der Bedingungs-, oder auch der Rechtsfolgenbereich enden soll[172]. Eine Norm verliert in diesem Sinne ihre Gültigkeit mit dem Tag, an dem ihr Bedingungs- und/oder

[171] § 66 KOVG.

[172] Wenn einer Norm derogiert wird, muss durch Interpretation ermittelt werden, ob der Bedingungs- und/oder Rechtsfolgenbereich betroffen sind.

Rechtsfolgenbereich endet. Nach diesem Zeitpunkt ist die Verbindlichkeit der Norm beendet, sie ist nicht mehr als Teil der Rechtsordnung zu betrachten.

Im Fall der Aufhebung einer Norm *ex tunc* im Sinne einer Nichtigerklärung durch den Verfassungsgerichtshof wird diese aus dem Rechtssystem vollständig entfernt, als ob diese nie existiert hätte. Sämtliche normativen Effekte der Norm werden rückwirkend entfernt, unabhängig vom Zeitpunkt der rechtlichen Beurteilung eines Falles.

Die Auswirkungen einer Aufhebung *ex nunc* sind hingegen kontroverser. Grundsätzlich endet mit der Aufhebung ebenso die Geltung einer Norm, sie wird aus dem Rechtsbestand gewissermaßen entfernt. Der wesentliche Unterschied zur Nichtigerklärung ist, dass diese *ex nunc* wirkt und damit keine Auswirkungen auf normative Effekte zeigt, die vor der Änderung abgeleitet wurden.

Es kann argumentiert werden, dass eine Aufhebung daher die Norm nicht (vollständig) aus der Rechtsordnung entfernt. In der Tat kann eine Norm auf Fälle weiterhin anwendbar sein, die sich vor der Aufhebung ereignet haben. Selbst dann, wenn solch ein Fall erst nach der Änderung der Gesetzeslage beurteilt werden soll. Obwohl durch die Änderung der Bedingungsbereich der Norm endet, wirkt deren Rechtsfolgenbereich über den Zeitpunkt der Aufhebung hinaus. Die Norm ist daher nach wie vor ein Teil der Rechtsordnung. Für Fälle, die sich nach der Aufhebung ereignet haben, ist die Norm selbstverständlich nicht mehr anwendbar.

Im Detail können durch die Aufhebung *ex nunc* komplizierte Fragestellung entstehen, die von Juristen unterschiedlich beurteilt werden. Die Frage, welche Rechtsfolgen nun von der Aufhebung betroffen sind, soll hier nicht weiter erörtert werden. Gelegentlich wird argumentiert, dass zumindest die indirekten Rechtswirkungen einer Norm deren Aufhebung überdauern. Für eine weiterführende Diskussion sei auf die Ausführungen von Governatori und Rotolo verwiesen (2009, S. 159).

Im Hinblick auf die rechtliche Beurteilung lässt sich festhalten, dass im Fall einer Nichtigerklärung selbst Fälle, die sich vor der Änderung ereignet haben, anhand der neuen Rechtslage entschieden werden müssen. Eine Aufhebung *ex nunc* wirkt sich grundsätzlich nur auf Fälle aus, die sich nach der Änderung ereignet haben.

Anhand dieser Ausführungen wird deutlich, dass die Frage, welche Normen für die Beurteilung eines Falls maßgeblich sind, vom Zeitpunkt abhängig ist, an dem sich ein Sachverhalt ereignet hat. Ferner ist erheblich, wann der Fall beurteilt wird, da davon abhängt, welche Normen Teil der Rechtsordnung sind. Ein Urteil, das beispielsweise vor der Nichtigerklärung einer Norm gefällt wird, bleibt von der Änderung der Rechtslage grundsätzlich unberührt.

Rückwirkende Änderungen der Rechtslage können nicht nur durch verfassungsgerichtliche Urteile (Art. 139 Abs. 6, 140 Abs. 7 B-VG) bewirkt werden, sondern auch durch Novellen, soweit (verfassungs-)rechtlich zulässig[173]. Im Strafrecht verbietet üblicherweise der Grundsatz *nullum crimen sine lege* (Art. 7 EMRK) die Rückwirkung von Normen, sofern diese nicht in ihrer Gesamtschau für den Beschuldigten günstiger sind.

Änderungen von Rechtsnormen können explizit und implizit erfolgen. Im ersten Fall, der formellen Derogation, bezeichnet eine Rechtsvorschrift exakt die Normen, die aufgehoben werden sollen. Von einer impliziten Änderung, einer materiellen Derogation, spricht man, wenn eine Rechtsvorschrift die aufgehobenen Bestimmungen nicht explizit bezeichnet, mit diesen aber in Konflikt steht. Solche Normkonflikte können zwischen *lex generalis* und *lex specialis* bzw. *lex prior* und *lex posterior* entstehen, wobei Kollisionsregeln zur Lösung angewendet werden.

Unter Umständen kann es auch hier zu Konstellationen kommen, die interessante Fragestellungen aufwerfen. Welcher Norm ist zum Beispiel der Vorrang zu gewähren, wenn die Lex prior *spezieller* ist als die Lex posterior und sozusagen ein Konflikt zwischen den Kollisionsregeln vorliegt?

Abschließend sei noch darauf hingewiesen, dass abhängig vom Einzelfall unter Umständen Spezialfälle im Hinblick auf die zeitliche Anwendbarkeit einer Norm zu berücksichtigen sind. So ist die Rückwirkung auf den Anlassfall bei Aufhebung einer Norm durch den VfGH nicht ausschließbar (Art. 139 Abs. 6, 140 Abs. 7 B-VG). Unter „Anlassfall“ ist jenes Verfahren zu verstehen, in dem das Normprüfungsverfahren initiiert wurde.

---

[173] Bei Novellierung eines Gesetzes wird üblicherweise in Übergangsbestimmungen angeordnet, inwiefern anhängige Verfahren betroffen sind und ab wann die neue Rechtslage (vollständig) anwendbar ist.

11.6. Änderungen der Rechtsordnung und von Normen

Im Folgenden wird darauf eingegangen, von welchen Änderungen die Rechtsordnung und deren Normen betroffen sein können. Diese Materie steht naturgemäß in einem engen Zusammenhang mit den zeitlichen Dimensionen des Rechts, da Änderungen notwendigerweise in einer zeitlichen Reihenfolge stattfinden.

Die Änderungen des Rechtssystems und dessen Normen werden durch Ereignisse verursacht, die im Wesentlichen in folgende Kategorien eingeordnet werden können:

1. Das Ausscheiden einer Norm aus dem Rechtssystem, sobald diese außer Kraft tritt.
2. Der Eintritt einer neuen Norm in das Rechtssystem mit deren Kundmachung.
3. Änderung einer bestehenden Norm in jeglicher Richtung.

Palmirani und Brighi (2006) präsentieren in ihrem Paper ein Modell zur Repräsentation von Dokumenten, das in der Lage ist, die notwendige Versionierung von Rechtstexten abzubilden. Sie differenzieren dabei zwischen (1) Änderungen der Normen selbst und (2) Änderungen, die das Rechtssystem als Ganzes betreffen.

Der ersten Kategorie sind sämtliche Änderungen zuzuordnen, die auf die im Rechtstext zum Ausdruck gebrachten Normen selbst wirken. Der zweiten Kategorie gehören Änderungen an, die das Rechtssystem insgesamt betreffen und sich nicht auf ein bestimmtes Gesetz bzw. eine Norm beziehen. Die Kundmachung eines Staatsvertrags im Bundesgesetzblatt beispielsweise, bei der lediglich ein neues Dokument dem Rechtsbestand hinzugefügt wird, gehört der zweiten Änderungskategorie an.

Für die gegenständliche Dissertation ist die erste Kategorie – die Änderung einer Norm – wesentlich interessanter, da in weiterer Folge ein formales Modell konzipiert werden soll, das mit Änderungen der Rechtslage umgehen kann. Hinsichtlich der Änderung einer Norm kann zwischen folgenden Unterkategorien unterschieden werden (Palmirani & Brighi, 2006):

1. Inhaltliche Änderung.
2. Änderung der Reichweite der Norm.
3. Änderung des zeitlichen Geltungsbereichs.

Mit inhaltlichen Änderungen sind solche Anpassungen einer Norm gemeint, die eine Änderung der ableitbaren Rechtsfolgen mit sich bringen. Der normative Gehalt einer Bestimmung kann (1) durch eine Änderung des Texts und (2) durch eine Modifikation der

Bedeutung der Begrifflichkeit verändert werden. Die Bedeutung von Begriffen wird insbesondere durch höchstgerichtliche Entscheidungen beeinflusst. Eine Änderung des Rechtstexts schlägt zumeist auch auf die Bedeutung einer Norm durch.

In dieser Zweiteilung zeigt sich der bereits von Sartor (2009b) aufgezeigte Umstand, dass die Bedeutung von Rechtsbegriffen nicht durch einen rein terminologischen Ansatz erfasst werden kann. Die Bedeutung eines Begriffs ist in hohem Maße von den Rechtsfolgen abhängig, die aus diesem im Kontext einer Rechtsvorschrift abgeleitet werden können.

Die Modifikation der „Reichweite" einer Norm betrifft die Erweiterung oder die Einschränkung des räumlichen (örtlichen) oder persönlichen Geltungsbereichs. Selbstverständlich kann auch der zeitliche Geltungsbereich einer Norm von einer Novelle betroffen sein. Im Einzelnen muss gegebenenfalls durch Interpretation festgestellt werden, welche der zeitlichen Dimensionen einer Norm betroffen ist.

Nachdem die Taxonomie von Änderungen dargestellt wurde, soll im Folgenden ein Überblick über die hoheitlichen Rechtsakte und deren Auswirkungen auf das Rechtssystem gegeben werden.

Die Gründe für Änderungen von Normen sind vielfältig. Sie reichen von Novellen durch den Gesetzgeber über Rechtsakte der Vollziehung bis zu höchstgerichtlichen Urteilen. Die folgende Abhandlung gliedert sich nach der Staatsgewalt, der ein Rechtsakt zuzurechnen ist.

Novellen sind Rechtsakte der Gesetzgebung, die Rechtsvorschriften erlassen, aufheben und ändern können. Dem Gesetzgeber sind dabei grundsätzlich keine Grenzen gesetzt, inwiefern eine Rechtsvorschrift geändert wird. In zeitlicher Hinsicht können sowohl der Bedingungs- als auch der Rechtsfolgenbereich betroffen sein. Bei rückwirkenden Novellen sind jedoch verfassungsrechtliche Schranken zu berücksichtigen.

Der Gesetzgeber kann überdies interpretativ tätig werden und im Wege der „authentischen Interpretation"[174] eine Norm zur Klarstellung der Bedeutung einer anderen Rechtsvorschrift erlassen. Das Gesetz wird dabei vom Gesetzgeber selbst im Nachhinein interpretiert, wobei die Änderung der Bedeutung gewissermaßen zurückwirkt, so als wäre die Norm von Anfang an in diesem Sinne zu verstehen gewesen.

[174] Vgl. § 8 ABGB.

Da der Schwerpunkt dieser Dissertation die Formalisierung von Verwaltungsrecht ist, sind die Rechtsakte der Vollziehung von besonderem Interesse. Zu diesen gehören Verordnungen, „Verwaltungsverordnungen“ (auch genannt „Erlässe“, „Richtlinien“ etc.) und Weisungen[175].

Durch Verordnungen können neue Rechtsvorschriften erlassen, aber auch in bestehende eingegriffen werden. Im Versorgungsrecht (KOVG, HVG etc.) erfolgt beispielsweise die Wertanpassung der im Gesetz ausgewiesenen Beträge durch Verordnung. Sofern dies ausdrücklich im Gesetz zugelassen ist, kann eine Verordnung auch rückwirkend erlassen werden.

Das Handeln der zur Vollziehung berufenen Organe wird in der Praxis oftmals durch Verwaltungsverordnungen und Weisungen bestimmt. Verwaltungsverordnungen wirken im Gegensatz zu „Rechtsverordnungen“ nicht „nach außen“ und sind ausschließlich an unterstellte Verwaltungsorgane adressiert. Sie werden auch als „Dienstanweisungen“, „Instruktionen“, „Richtlinien“ oder „Erlässe“ bezeichnet. Im Wesentlichen handelt es sich um „generelle Weisungen“, die ausschließlich an nachgeordnete Organwalter gerichtet sind. Sie erlangen große praktische Bedeutung, da sie für die Verwaltungsbediensteten kraft Weisung verbindliche Interpretationen der Gesetze vorschreiben (Raschauer, 2009, S. 266 f.). Auf die Problematik, die entsteht, wenn eine solche generelle Weisung das Vollzugsverhalten der Behörde beeinflusst und damit faktisch „nach außen“ wirksam wird, sei hingewiesen, es soll jedoch an dieser Stelle nicht weiter darauf eingegangen werden (Raschauer, 2009, S. 267 f.).

Bei einer Weisung handelt es sich um einen hoheitlichen Befehl eines Organwalters im „Verwaltungsinnenbereich“ an nachgeordnete Organe (Organwalter). Weisungen können entweder an einen bestimmten Organwalter oder an ein Organ gerichtet werden, wenn sie jeden Organwalter binden sollen (Raschauer, 2009, S. 325 ff.).

Verwaltungsverordnungen und Weisungen haben gemeinsam, dass sie die Interpretation von Rechtstexten vorgeben können und so das Vollzugsverhalten eines nachgeordneten Organwalters (Organs) beeinflussen. Damit konkretisieren sie die Bedeutung von Normen auf einer untergesetzlichen Ebene. Änderungen des Rechtstexts von Gesetzen und Verordnungen sind jedoch ausgeschlossen. Grundsätzlich besteht kein „Bestandschutz“,

[175] Für eine eingehende Darstellung der Rechtsquellen siehe Abschnitt 11.3.

sodass Verwaltungsverordnungen und Weisungen jederzeit geändert werden können, was eine Anpassung des formalen Modells nach sich zieht.

Die Gerichtsbarkeit – dieser Begriff umfasst sowohl die ordentliche Gerichtsbarkeit als auch die Verwaltungsgerichtsbarkeit – als dritte Staatsgewalt kann ebenso Änderungen der Rechtsordnung bewirken. Insbesondere Urteile von Höchstgerichten (OGH, VwGH und VfGH) können zu einer Änderung der Bedeutung von Rechtsvorschriften führen. Der höchstgerichtlichen Rechtsprechung kommt in kontinentaleuropäischen Rechtssystemen nicht die Qualität einer objektiven Rechtsquelle zu. Trotzdem wirken höchstgerichtliche Urteile kraft ihres argumentativen Gewichts über den Einzelfall hinaus und sind von niedrigeren Instanzen zu berücksichtigen. Weicht eine untere Instanz in ihrer Entscheidung in ungerechtfertigter Weise ab, so kann das Urteil von der übergeordneten Instanz korrigiert werden. Im Bereich der Verwaltungsgerichtsbarkeit kommt dem VfGH überdies die Kompetenz zur „Normenkontrolle“ zu, womit der Gerichtshof (auch) zur Aufhebung von Rechtsvorschriften befugt ist.

Durch den Vorgang der Abstraktion lassen sich sämtliche Varianten auf die Änderung der Rechtsordnung (Eintritt und Austritt einer Norm) und die Anpassung einer bestehenden Norm reduzieren. Ein formales Modell des Gesetzes muss flexibel genug sein, um mit diesen Änderungsvarianten umgehen zu können.

11.7. Die anfechtbare Schlussfolgerung

Es herrscht weitgehend Einigkeit in der Rechtstheorie, dass juristische Schlussfolgerungen „anfechtbar“ sind (Gordon, 1988, 1993; Sartor, 2005). Wenn der Tatbestand einer Norm erfüllt ist, bedeutet das nicht zwingend, dass auch deren Rechtsfolgen zur Anwendung gelangen. Die Norm kann von einer ebenfalls anwendbaren Ausnahmebestimmung verdrängt werden.

In der Terminologie der Logik können neue Fakten bestehende Konklusionen ungültig machen. Demnach kann die Konklusion einer Regel erst dann endgültig abgeleitet werden, sofern keine stärkeren Regeln mit gegenteiliger Konsequenz vorhanden sind. Neue Informationen, die während der Beurteilung eines Falles hinzukommen, können daher dazu führen, dass bereits abgeleitete Konklusionen verworfen werden müssen.

Mithilfe der anfechtbaren Schlussfolgerung[176] (engl. defeasible reasoning) können auch mit unvollständigen Informationen Entscheidungen getroffen werden, die eventuell nur provisorisch sind und später widerrufen werden müssen. Im Fall von Konflikten zwischen Regeln wird ein skeptischer Ansatz verfolgt, d.h. keine der beiden Regeln wird angewendet, es sei denn, der Konflikt kann mithilfe von Kollisionsregeln gelöst werden.

Grundlegend muss zwischen „konklusiven" (conclusive) und „anfechtbaren" (defeasible) Schlussfolgerungsschemen unterschieden werden. Der Unterschied zwischen beiden Ansätzen lässt sich wie folgt zusammenfassen (Sartor, 2009a, S. 4):

„A conclusive reasoning schema indicates a cognitive transition that can operate regardless of any further information the agent possesses, as long as the reasoner instantiates the pre-conditions in the schema: Whenever one endorses the reason of the schema, one may safely endorse its conclusion."

Konklusive Schlussfolgerungsschemen bieten die Grundlage für die monotone Schlussfolgerung. Solange neue Informationen hinzukommen, können weitere Konklusionen abgeleitet werden. Im Gegensatz dazu bieten anfechtbare Schemen die Grundlage für die nicht-monotone Schlussfolgerung. Bereits abgeleitete Konklusionen müssen möglicherweise verworfen werden, sobald neue widersprüchliche Informationen verfügbar sind.

Daraus folgt, dass konklusive Schlussfolgerungsschemen wahrheitserhaltend sind. Sobald die Prämissen eines konklusiven Schemas wahr sind, sind notwendigerweise auch dessen Konklusionen wahr. Für nicht-monotone Schlussfolgerungsschemen trifft das nicht zu.

Welche logischen Funktionen erfüllen anfechtbare Schlussfolgerungsschemen im Allgemeinen?

Sartor (2009a, S. 9 ff.) identifiziert zwei Funktionen der anfechtbaren Schlussfolgerung, die nicht bloß auf den Kontext der automatisierten Entscheidungsfindung begrenzt sind, sondern Allgemeingültigkeit besitzen: Zum einen soll der Problemlöser in die Lage versetzt werden, provisorische Ergebnisse ableiten zu können. Aufgrund dieser provisorischen Schlüsse können Entscheidungen getroffen und Aktionen ausgeführt werden, solange keine gegenteiligen Informationen vorliegen. Zweitens soll eine strukturierte Untersuchung

[176] Auch „anfechtbares Schließen" genannt.

ermöglicht werden. Diese besteht aus der Ableitung von *pro tanto* Konklusionen und der Suche nach anfechtenden Regeln (defeater) solange, bis endgültige Resultate feststehen. *Pro tanto* Konklusionen erscheinen bei der Schlussfolgerung aufgrund der bisher vorhandenen Informationen gerechtfertigt, können jedoch durch zukünftige Informationen verworfen werden.

Dieses Vorgehen hat zwei Vorteile: (1) Die Untersuchung wird auf relevantes Wissen fokussiert und (2) der Prozess erzeugt laufend provisorische Resultate, solange dieser voranschreitet.

Drittens tragen anfechtbare Schlussfolgerungschemen dazu bei, dass das Wissen bis zu einem gewissen Grad konstant bleiben kann, während fortwährend neue Informationen gelernt werden. Diese Informationen können bestehendes Wissen anzweifeln und letztendlich dazu führen, dass in der Wissensbasis vorhandene Aussagen revidiert werden müssen.

11.7.1. Kollisionen

Die anfechtbare Schlussfolgerung ermöglicht es, die Entscheidungsfindung mit unvollständigen Informationen durchzuführen. Neue Informationen können sukzessive verarbeitet werden, sobald diese zur Verfügung stehen.

Selbstverständlich kann es vorkommen, dass Ableitungen zu inkompatiblen Konklusionen führen. Diese Art von Konflikt zwischen Regeln wird *entkräftende Kollision* (rebutting collision) genannt (Sartor, 2009a, S. 10 ff.). Von den Inferenzen, die durch die Kollision blockiert werden, wird gesagt, dass sie „angefochten“ (defeated) wurden. Die Regeln, die verhindern, dass eine Konklusion abgeleitet wird, werden „Anfechter“ (defeater) genannt.

Die Frage, ob Konklusionen inkompatibel sind und ein Konflikt vorliegt, ist nicht immer einfach zu beantworten. Oftmals lässt sich die Antwort erst durch die Betrachtung weiterer Informationen beurteilen. In der Regel kann also eine Inkompatibilität nicht sofort festgestellt werden, sondern wird im Zuge der Untersuchung mithilfe zusätzlicher Informationen aufgedeckt.

Die zweite Art, wie Regeln kollidieren können, ist die sogenannte *untergrabende Kollision* (undercutting collision) (Sartor, 2009a, S. 12). Solche Kollisionen treten auf, wenn unter den jeweiligen Umständen zweifelhaft ist, ob eine Konklusion zuverlässig anhand eines

Schlussfolgerungsschemas abgeleitet werden kann. Demnach wird die Zuverlässigkeit eines Schlussfolgerungsschemas angezweifelt. Diese Art von Kollision sagt nichts über die Richtigkeit einer Konklusion aus, sondern untergräbt bloß die zuvor abgeleitete Konklusion, ohne dass eine andere Konklusion an deren Stelle tritt.

Nachdem nun die beiden Kollisionsarten dargestellt wurden, stellt sich die Frage, wie mit Konflikten zwischen Regeln umgegangen werden soll?

Es liegt auf der Hand, dass bei kollidierenden Gründen[177] nicht beide Konklusionen abgeleitet werden können. Zumindest eine der widersprüchlichen Gründe wird angefochten. Grundsätzlich müssen zwei Fälle unterschieden werden (Sartor, 2009a, S. 13):

- Wenn ein Grund, G1, über einen anderen, G2, obsiegt, dann kann G2 verworfen werden.
- Im Gegensatz dazu, wenn keiner der beiden Gründe obsiegt, werden beide verworfen.

G1 *ficht* Grund G2 an, wenn G1 mit G2 kollidiert und G2 nicht über G1 obsiegt. Wenn zusätzlich G1 über G2 obsiegt (1. Fall oben), dann wird gesagt, dass G1 den Grund G2 *streng anficht* (G1 strictly defeats G2). In diesem Fall obsiegt G1 über G2, wird jedoch selbst nicht angefochten.

In Abhängigkeit von der Art der Kollision muss mit Konflikten unterschiedlich verfahren werden. Zusammengefasst bietet sich folgendes Lösungsschema an:

- Im Fall von *entkräftenden Kollisionen* muss die Stärke der in Konflikt stehenden Ursachen in Betracht gezogen werden. Dabei wird angenommen, dass jeder Grund, der nicht stärker ist als sein Gegenspieler, „angefochten“ wird. Im Ergebnis bedeutet das, dass im Falle eines Konflikts ausschließlich der stärkere Grund obsiegt.
- Bei *untergrabenden Kollisionen* wird hingegen angenommen, dass der untergrabende Grund immer obsiegt.

[177] Das hier zugrundeliegende Verständnis des Begriffs „Grund“ (reason) entspricht dem allgemeinen Sprachgebrauch: (1) Ein Grund A zu glauben ist, dass B geglaubt wird, (2) die Absicht A zu tun ist, dass B erwünscht ist und dadurch C erreicht werden soll. In der Rechtstheorie wird darunter jedoch oftmals ein Fakt verstanden, um eine bestimmte Handlung zu rechtfertigen.

Angemerkt sei, dass es außerordentlich schwierig sein kann festzustellen, welcher der beiden Gründe im Fall einer Kollision stärker ist. Im Recht müssen dafür oftmals weitere Informationen herangezogen werden, wie beispielsweise der durch eine interpretative Gesamtbetrachtung ermittelte „Sinn" des Gesetzes oder die zugrundeliegenden Wertungen.

11.7.2. Wiederinkraftsetzung

Ein weiteres Merkmal der anfechtbaren Schlussfolgerung wurde bisher nicht behandelt, die sogenannte *Wiederinkraftsetzung* (reinstatement) einer Regel: Wenn eine anfechtende Regel B selbst wiederum von einer späteren Konklusion C angefochten wird, dann kann die ursprünglich anwendbare Regel A wiederaufleben, sodass deren Konklusion abgeleitet werden kann[178].

Die anfechtende Regel kann dabei von einer *entkräftenden* oder einer *untergrabenden Kollision* betroffen sein und von der kollidierenden Regel verdrängt werden. Für eine weitergehende Darstellung mit Beispielen zur Illustration sei auf Sartor (2009a, S. 14 ff.) verwiesen.

11.7.3. Die Überwindung der Anfechtbarkeit juristischer Schlussfolgerungen

Rechtsvorschriften sind üblicherweise in Form von generellen Normen und Ausnahmebestimmungen strukturiert. Die Anwendung der nicht-monotonen Schlussfolgerung scheint daher naheliegend und das natürliche Mittel zu sein, um Rechtsprobleme automatisiert zu lösen bzw. juristische Experten zu unterstützen.

Trotzdem wird gelegentlich gegen die Anwendung nicht-monotonen Schließens argumentiert. Hinter dem von manchen Autoren vorgebrachten Desiderat der Überwindung der rechtlichen Anfechtbarkeit steht die Überlegung, dass das Recht in Form von konsistenten Axiomen formuliert werden sollte, die ausschließlich kompatible Resultate produzieren.

Eine solche Neuformulierung des Rechts oder zumindest von Teilen davon hätte zur Folge, dass Konflikte vermieden und so auf die anfechtbare Schlussfolgerung verzichtet werden könnte. Es ist jedoch fraglich, ob eine solche Neuformulierung des Rechts überhaupt möglich

[178] Dies entspricht der im Recht vorkommenden Situation, wenn eine Ausnahmebestimmung selbst wiederum von einer Ausnahme betroffen ist, sodass die generelle Norm Rechtswirkungen entfaltet.

ist, und selbst wenn bleibt offen, ob diese nützlich wäre. Sartor (2009a, S. 22 ff.) bringt gegen ein solches Vorhaben folgende Bedenken vor:

- Jede Regel müsste sämtliche ihrer Ausnahmen berücksichtigen. Rechtsvorschriften würden dadurch unausweichlich komplexer und damit unleserlicher werden.
- Eine solche Repräsentation der Normen könnte nicht deren innewohnende Dynamik erfassen, ohne in die Formulierung bestehender Regeln einzugreifen. Der Rechtstext müsste immer dann geändert werden, sobald neue Informationen hinzukommen, die in Konflikt stehende Regeln oder die Kriterien zur Auflösung von Kollisionen betreffen.
- Letzten Endes wäre es unmöglich, ohne die anfechtbare Schlussfolgerung bereits während der Untersuchung provisorische Ergebnisse abzuleiten, solange diese voranschreitet. Es liegt jedoch in der Natur des Rechts und der dialektischen Prozessführung, ein (vorläufiges) Ergebnis kritisch zu prüfen und gegebenenfalls wieder zu verwerfen, sobald widersprechende Informationen verfügbar sind.

Die von Sartor vorgebrachte Kritik erinnert in Grundzügen an die Beobachtungen, die im Zuge der Modellierung von Gesetzen mit Java gemacht wurden. Es sei in Erinnerung gerufen, dass es grundsätzlich zwei Möglichkeiten gibt, die anfechtbare Schlussfolgerung mit dem monotonen Ablauf eines Javaprogramms zu simulieren: (1) Es können sämtliche Bedingungen in eine einzige Wenn-Dann-Bedingung aufgenommen werden oder (2) die Reihenfolge der Ausführung der Anweisungen wird entsprechend der Priorität der Regeln festgelegt.

Die erste Lösungsmöglichkeit ist vor allem von den ersten beiden Problemen betroffen: Sobald neue Normen hinzukommen, müssen bestehende Regeln im Code um zusätzliche Bedingungen erweitert werden. Dieser Prozess ist äußerst fehleranfällig und die Lesbarkeit des Programmcodes leidet erheblich.

Hinsichtlich der zweiten Art, die nicht-monotone Schlussfolgerung gewissermaßen zu simulieren, ist zu bemerken, dass auch hier präzises Wissen über den Programmablauf und die Zusammenhänge zwischen den Anweisungen erforderlich ist, um neue Regeln hinzufügen zu können. Kern dieser Methode ist das Überschreiben von früheren Ergebnissen durch Regeln höherer Priorität. Relativ gut funktioniert diese Lösung, sofern die angefochtene (generelle) Regel nur eine einzige Konklusion hat. Sobald jedoch mehrere

Konklusionen abgeleitet werden können, wird es schwierig, wenn diese Effekte von der spezielleren Regel überschrieben werden sollen. Überdies kann die Ausführung eines Programms „Nebeneffekte“ verursachen, die ebenfalls rückgängig gemacht werden müssen, wie beispielsweise die Generierung des entsprechenden Teils der Begründung. Die Regeln werden unausweichlich starr aneinander gekoppelt, was Änderungen äußerst schwierig macht. Verschärft wird diese Problematik, wenn die weiter oben beschriebene „Wiederinkraftsetzung“ einer angefochtenen (generellen) Bestimmung ins Spiel kommt.

Diese lässt sich mit dem ersten Ansatz, der Aufnahme sämtlicher Bedingungen in eine Anweisung, nur außerordentlich schwierig bewerkstelligen. Die formulierte Bedingung wächst mit jeder zusätzlichen Norm unweigerlich an. Es lässt sich kaum nachvollziehen, auf welcher Rechtsgrundlage die Bedingungen beruhen.

Auch der zweite Lösungsansatz bietet keine zufriedenstellende Lösung. Damit die „Wiederinkraftsetzung“ implementiert werden kann, müssten die Ergebnisse der angefochtenen Norm für eine spätere Verwendung aufgehoben werden, sodass diese noch zur Verfügung stehen, wenn die anfechtende Regel selbst wiederum (erfolgreich) verdrängt wird. Die daraus resultierende Komplexität des Programmcodes sowie die de facto nicht gewährleistete Nachvollziehbarkeit des Ergebnisses sind Gründe, die schwer genug wiegen, um nach einer angemesseneren Lösung zu streben.

11.7.4. Technische Perspektive

Charakteristisch für anfechtbare Schlussfolgerungsschemen ist, dass bereits abgeleitete Konklusionen unter Umständen verworfen werden müssen, sobald neue gegenteilige Informationen vorhanden sind. Aus technischer Sicht ist es schwierig, einen nicht-monotonen Algorithmus zu parallelisieren, um die Ausführungszeit zu verringern und den Datendurchsatz zu steigern. Da die Reihenfolge der Anwendung der Regeln beachtlich ist, lässt sich das Gesamtproblem nicht so einfach in einzelne, unabhängig voneinander lösbare Teilprobleme zerlegen.

Dies läuft dem stetigen Trend der steigenden Prozessorleistung durch die Verwendung mehrerer Kerne entgegen. Moderne Prozessoren können erhebliche Geschwindigkeitsvorteile erzielen, indem mehrere Operationen parallel ausgeführt werden.

Üblicherweise werden Maßnahmen zur Steigerung des Datendurchsatzes erst dann relevant, wenn eine sehr große Anzahl von Regeln in Echtzeit verarbeitet werden muss. Das Problem scheint bei juristischen Expertensystemen eine eher untergeordnete Rolle zu spielen, da die Ausführungszeit für einfache Schlussfolgerungen, wie sie in der Praxis vorkommen, oftmals im Bereich von Millisekunden liegt. Dennoch könnte die Problematik schlagend werden, wenn beispielsweise die Vereinbarkeit von Aktionen mit einer Vielzahl von Geschäftsregeln, die auf gesetzlichen und unternehmensspezifischen Vorgaben beruhen, in Echtzeit ermittelt werden soll.

Aus Sicht der Wissensrepräsentation ist ein Vorteil der monotonen Schlussfolgerung, dass man sich nicht um die Auswirkungen der Reihenfolge der Ableitung von Inferenzen sorgen muss. Damit lassen sich mehrere Wissensrepräsentationen unproblematisch in einer Wissensbasis zusammenführen. Im Gegensatz dazu ist im Fall einer nicht-monotonen Schlussfolgerung die Reihenfolge der Ausführung der Regeln sowie des Vorbringens der Fakten und Behauptungen von Bedeutung und verändert das Ergebnis (Boer, 2009, S. 19).

Es liegt jedoch in der Natur des Rechts und der Prozessführung, dass die Reihenfolge der Vorbringen und Beweise unter Umständen das Ergebnis beeinflussen. Welchen Mehrwert erlangt man, wenn man starr an monotonen Schlussfolgerungsmodellen festhält, die abgeleiteten Konklusionen jedoch juristisch nicht haltbar sind? Meines Erachtens sollte man keinesfalls zu fixiert an einem monotonen Schlussfolgerungsmodell festhalten, nur um den Aufbau und die Integration der Wissensbasis zu erleichtern. Im Endeffekt kann die Qualität und Nützlichkeit einer Wissensrepräsentation nur anhand der aus ihr ableitbaren Ergebnisse (sinnvoll) gemessen werden.

Die Wiederverwendbarkeit einer Wissensrepräsentation könnte durch die Anwendung nicht-monotoner Schlussfolgerung angezweifelt werden. Nach der oben dargestellten Meinung hängt das Ergebnis maßgeblich von der Reihenfolge der Ausführung ab. Dadurch müsste im Fall der Verwendung der Wissensrepräsentation in einem anderen Kontext geprüft werden, an welcher Stelle im Entscheidungsfindungsprozess das neue Wissen verwendet wird. Die Nutzbarkeit modularer Wissensrepräsentationen wäre erheblich vermindert.

Meines Erachtens können Wissensrepräsentationen auch bei Anwendung nicht-monotoner Algorithmen wiederverwendet werden. Die Reihenfolge der Anwendung von Regeln bestimmt sich nach deren Priorität oder kann durch eine relative Rangordnung festgelegt

werden. Damit ist die Reihenfolge der Ausführung festlegbar oder zumindest bestimmbar, sofern dies für die Richtigkeit der Entscheidung erforderlich ist.

Damit ist eine bestimmte Reihenfolge der Ausführungen in jedem Fall beeinflussbar, sofern dies für die Richtigkeit der Entscheidung erforderlich ist. Das Problem der geringeren Wiederverwendbarkeit wird dadurch zumindest relativiert.

Intensive Tests und Überprüfungen von Expertensystemen können ohnehin auch bei der Verwendung von monotonen Schlussfolgerungsschemen nicht ausbleiben, um die Richtigkeit des Ergebnisses zu gewährleisten.

11.8. Modularität

Eine gewünschte und zugleich wichtige Eigenschaft der Wissensbasis eines (juristischen) wissensbasierten Systems ist deren Modularität. Das bedeutet, dass die formale Repräsentation des Wissens aus mehreren unabhängigen Modulen – den Wissensrepräsentationen – zusammengesetzt wird.

Im Recht ist es durchaus üblich, dass Gesetze Verweisungen enthalten. Durch eine modulare Architektur der Wissensbasis wird die größtmögliche Flexibilität erreicht. Teile der Wissensbasis können separat von Experten modelliert und gewartet werden, die dafür speziell befähigt sind. Nach Bedarf können Teile der Wissensbasis aus anderen Behörden importiert werden. Wichtig ist, dass die Wissensbasis und deren Teile (die Wissensrepräsentationen) zentral gepflegt werden, sodass diese nicht an mehreren Stellen aktualisiert werden müssen.

Wie in Abschnitt 11.3.1. ausgeführt, besteht eine Repräsentation der (schriftlichen) Rechtsquellen aus mehreren Schichten. Jede Schicht kann in diesem Kontext als eigenständiges Modul begriffen werden. Im Gegensatz zu der dort thematisierten horizontalen Abgrenzung zwischen einzelnen Schichten handelt es sich bei der Modularisierung an dieser Stelle um eine vertikale Modulbildung von nebeneinander stehenden Wissensrepräsentationen.

In der Rechtstheorie wird zwischen statischen und dynamischen Verweisungen zwischen Gesetzen unterschieden. Der verwendete Formalismus zur Modellierung der Wissensrepräsentation muss daher Verweise unterstützen, die neben der obligatorischen Kennzeichnung des Gesetzes auch optionale Parameter zur Angabe der gewünschten

Fassungen enthalten. Für Bundesgesetze bietet sich beispielsweise die Angabe der Nummer des Bundesgesetzblatts zur Bezeichnung der exakten Fassung des Gesetzes an. Je nachdem auf welcher Ebene im formalen Modell – Text, Ontologie, Regeln – die Verweisung stattfindet, können sich die verwendeten Verweise unterscheiden.

Ein solch rudimentärer Ansatz zur Repräsentation von Verweisungen, der bloß die Nummer der Fassung eines Gesetzes verwendet, reicht selbstverständlich nicht aus, um die Änderungen der Rechtsordnung abzubilden, die in Abschnitt 11.6. thematisiert wurden. Für das Konzept eines temporalen Modells, das zur Versionierung von legistischen Dokumenten entwickelt wurde und Verweise mit temporalen Parametern unterstützt, sei auf Palmirani und Brighi (2006) verwiesen.

11.9. Reihenfolge der Normanwendung

Grundsätzlich könnte man annehmen, dass die Reihenfolge der Anwendung der Paragrafen beliebig sein kann. Dem Vorgehen des Juristen entspricht es jedoch mehr, die Fallprüfung systematisch nach einem bestimmten Schema durchzuführen. Dabei kann sich der Experte auf seine praktische Erfahrung und etablierte Vorgehensweisen stützen. In vielen Rechtsgebieten existieren Fallprüfungsschemen, die den Rechtsanwender bei der Falllösung unterstützen und dabei gewissermaßen die Reihenfolge der Normprüfung vorgeben.

Diese Systematisierung der Rechtsanwendung hat nicht nur den Zweck, eine Methode zur Fallprüfung zu entwickeln, die den Blick des Rechtsanwenders auf relevante Aspekte des Falls fokussiert, sondern beschleunigt üblicherweise die Entscheidung selbst und hilft Fehler zu vermeiden. Das angewendete Fallprüfungsschema – sofern ein solches existiert – variiert naturgemäß je nach Rechtsgebiet.

Der Jurist nimmt bei der Lösung des Falles gewissermaßen eine Abkürzung und sieht von der Prüfung ohnehin aussichtsloser Ansprüche ab. Dadurch wird eine Vereinfachung der Fallprüfung erreicht. Im formalen Modell müssen jedoch sämtliche Bedingungen in eine Regel explizit aufgenommen werden, damit sie von der Maschine berücksichtigt werden können.

Im Fall der Berechnung der Beschädigtenrente nach dem KOVG beispielsweise wird die Fallprüfung beendet, sobald festeht, dass nicht einmal die Grundrente zusteht. Keinesfalls wird er zur Prüfung der anderen Voraussetzung für die Gewährung etwaiger Erhöhungen

schreiten. Der Anspruch auf Grundrente ist sozusagen eine grundlegende Bedingung für die Gewährung diverser Erhöhungen. Für den menschlichen Rechtsanwender erschließt sich diese Bedeutung ohne Schwierigkeiten aus der Systematik des Gesetzes.

Um grundlegende Anspruchsvoraussetzungen nicht an mehreren Stellen im Programmcode berücksichtigen zu müssen, wird deren Prüfung oftmals vom Programmierer an einer einzigen Stelle vorweggenommen. Die Bedingung, ob eine Grundrente zusteht, wird etwaigen Erhöhungen im Programmcode vorgelagert, um den Ablauf des Programmes gegebenenfalls vorzeitig beenden zu können.

Gelegentlich werden im Gesetz vorhandene implizite Bedingungen vom Programmierer extrahiert und in den Programmcode explizit aufgenommen. Beispielsweise lässt sich aus § 11 KOVG ableiten, dass bei einer Minderung der Erwerbsfähigkeit (MdE) von unter 20 % keine Leistung aus dem Gesetz zusteht. Diese Bedingung ist jedoch in dieser Form **nicht** explizit im Gesetzestext enthalten. § 11 KOVG listet beispielsweise nur die Geldbeträge auf, die ab einer MdE von 20 % zustehen. Obwohl solche „Kunstgriffe“ aus Sicht des Programmierers durchaus sinnvoll erscheinen, lassen sie sich nicht vollständig mit dem Prinzip des Isomorphismus vereinen.

Dem Gesetzgeber kann unterstellt werden, dass er durch die Anordnung der Paragrafen eine bestimmte Reihenfolge bei der Rechtsanwendung beabsichtigt. Bei der Formalisierung eines Gesetzes sollte daher dessen Struktur ausreichend berücksichtigt und auch der Reihenfolge der Normanwendung Beachtung geschenkt werden. Die Reihenfolge der Anwendung logischer Regeln sollte daher, falls notwendig, festgelegt werden können.

In gewisser Weise erinnert diese Problematik an die Auflösung von normativen Konflikten durch die Anwendung von Kollisionsregeln. Die speziellere Bestimmung muss beispielsweise nur dann geprüft werden, wenn auch die generelle Norm grundsätzlich zur Anwendung gelangt. Eine Inferenzmaschine könnte mithilfe von Metaregeln eine Sortierung der Regeln durchführen, um so eine bestimmte Reihenfolge festzulegen.

Ähnlichkeit weist auch die in der Literatur unter dem Schlagwort „Contrary-To-Duty Obligations“ bekannte Problematik auf. Wenn eine Norm verletzt wird, werden üblicherweise Mechanismen – in der Regel Sanktionen – ausgelöst, die den rechtskonformen Zustand wiederherstellen sollen (Governatori & Rotolo, 2006). Auch in diesem Beispiel kommen die Sanktionsnormen erst dann zur Anwendung, wenn einem Verbot bzw. Gebot

zuwidergehandelt wird, womit eine bestimmte Reihenfolge der Normanwendung vorgegeben ist.

Ein weiterer Aspekt der juristischen Rechtsanwendung, der für die Einhaltung einer gewissen Reihenfolge bei der Normprüfung spricht, ist die Notwendigkeit, dass Begründungen juristischer Entscheidungen einen möglichst sinnhaften Zusammenhang aufweisen müssen, um verständlich zu sein. Dies erfordert eine Systematik beim Aufbau der Begründung. Um beim Beispiel der Erhöhungen der Grundrente nach dem KOVG zu bleiben, wäre es sinnwidrig, wenn Ausführungen über die Erhöhung der Grundrente gemacht werden, bevor begründet wird, ob der Antragsteller überhaupt anspruchsberechtigt ist.

Die Reihenfolge der Normprüfung muss sich nicht zwingend auf die Struktur der Begründung auswirken. Die Begründung sollte jedoch den Gedankengang des Richters in nachvollziehbarer Weise wiedergeben. Eben dies scheint schwer vorstellbar, wenn die Reihenfolge der Normanwendung und der Aufbau der Begründung maßgeblich voneinander abweichen. Auch wenn die (vollkommen) automatisierte Erstellung der Begründung derzeit noch nicht möglich ist, kann die Reihenfolge der Normprüfung deren Struktur schon weitgehend vorgeben und damit nachvollziehbar machen.

Schließlich ist die Anordnung der Rechtsvorschriften im Gesetz eine Möglichkeit, die Beweislast zu verteilen (Bench-Capon & Gordon, 2009). Der Kläger muss üblicherweise die Anwendung der generellen Regel beweisen, die seinen Anspruch rechtlich stützt. Gelingt ihm das, so ist es die Aufgabe des Beklagten, die Anwendung einer Ausnahmebestimmung zu begründen oder Argumente gegen die Anwendbarkeit der vom Kläger vorgebrachten Norm zu finden. Die Beweislast kann überdies durch gesetzliche Vermutung oder explizite Beweislastregeln verschoben werden. Auf diese Möglichkeiten soll jedoch an dieser Stelle nicht weiter eingegangen werden.

Dieser prozessuale Aspekt steht auch in Zusammenhang mit der anfechtbaren Schlussfolgerung. Die Konklusion der generellen Regel wird verworfen, sobald der Beklagte die Anwendung einer Ausnahmebestimmung bewiesen hat. Damit lässt sich allerdings nur ein Teil der Dialektik eines Verfahrens erklären, die anfechtbare Schlussfolgerung stellt bloß die logische Grundlage zur Verfügung.

Die Struktur des Gesetzes hat wichtige Funktionen und muss in einer isomorphen Repräsentation berücksichtigt werden. Diese Forderung wurde bereits auf der ersten

Konferenz über Künstliche Intelligenz und Recht (ICAIL 1987) vorgebracht (Gordon, 1987), fand jedoch bis heute in der Praxis der Verwaltungsautomatisierung keine Beachtung.

11.10. Hypothetische Schlussfolgerung

Die in Kapitel 9 dargestellte Formalisierung des KOVG mit Java hat gezeigt, dass ein Expertensystem in der Lage sein muss, simple Hypothesen zu bilden. Konkret geht es um die Berechnung der Familienzulage gemäß § 16 KOVG. Wie in Abschnitt 9.5.9.6. ausführlich behandelt, besteht zwischen § 16 („Familienzulage“) und § 12 („Zusatzrente“) KOVG eine Wechselwirkung. Die Familienzulage ist um jenen Teil zu kürzen, um den die Zusatzrente wegen des Anspruchs auf Familienzulage erhöht wird.

Zur Ermittlung des Betrags, um den die Familienzulage gekürzt werden soll, muss das System eine fiktive Berechnung der Zusatzrente ohne die Erhöhung durchführen. Die Software muss also eine hypothetische Zusatzrente berechnen, wobei die Erhöhung aufgrund von Familienzulagen außer Betracht zu lassen ist. Anschließend wird die hypothetische Zusatzrente mit der tatsächlichen verglichen und die Differenz ermittelt, die sodann zu gleichen Teilen von den Beträgen der Familienzulage abgezogen wird.

Als weiteres Beispiel für die Notwendigkeit einer Hypothesenbildung sei das Strafrecht erwähnt. Das im Strafrecht geltende Günstigkeitsprinzip besagt, dass sich der Täter nach der Rechtslage zum Zeitpunkt der Tat verantworten muss, es sei denn, die neue Rechtslage ist für den Täter günstiger[179]. Damit kommt es zu einer Durchbrechung des im Strafrecht geltenden strengen Rückwirkungsverbots, das nur diese Ausnahme zulässt.

Das Günstigkeitsprinzip (*lex mitior*) ist die einzig zulässige Durchbrechung des strengen Rückwirkungsverbots (*nulla poena sine lege*) im Strafrecht und ein Ausfluss des generellen Prinzips des *favor rei* (zu verstehen als „zum Vorteil des Beschuldigten“). Aus dem Prinzip des *favor rei* ergeben sich auch der Grundsatz *in dubio pro reo*, das Analogieverbot und das Verbot einer exzessiven Interpretation zu Lasten des Beschuldigten.

Für die weiteren Erläuterungen soll von folgendem fiktiven Fall ausgegangen werden: Die Tat ereignet sich 2010 und wird im Jahr 2012 bei Gericht anhängig, wobei im Jahr 2011

[179] Ob die neue Rechtslage „günstiger“ ist, muss in einer Gesamtschau beurteilt werden, dabei sind die alte und die neue Rechtslage einander gegenüberzustellen (VwGH 16.12.1997, 96/09/0149). Das Günstigkeitsprinzip gilt jedoch nicht für verfahrensrechtliche Bestimmungen (VwGH 26.01.2012, 2009/07/0039).

zwischenzeitlich eine Novelle des Strafrechts stattgefunden hat. Der Richter – und auch das Expertensystem – muss aufgrund des Günstigkeitsprinzips die Rechtslage zum Zeitpunkt der Tat (2010) mit jener der Entscheidung (2012) vergleichen und die Rechtslage anwenden, die für den Beschuldigten günstiger ist.

In den hier skizzierten Beispielen geht es um die Bildung von (einfachen) Hypothesen, um das finale Ergebnis ermitteln zu können. Wie die Beispiele zeigen, geht mit der Hypothesenbildung oftmals ein Vergleich der Konklusionen einher, um zum endgültigen Ergebnis zu gelangen. Es bleibt offen, ob und wie solche Hypothesen und Wechselbeziehungen zwischen Normen (wie im ersten Beispiel) am besten modelliert werden können. Vorzugsweise sollten auch komplexe Beziehungen dieser Art von einer Sprache ausgedrückt werden können. Aus technischer Sicht geht es bei der hier diskutierten Hypothesenbildung im Wesentlichen um eine Parametrisierung von Methoden, die Teile der rechtlichen Beurteilung durchführen.

Die hier diskutierten einfachen Fälle der Hypothesenbildung dürfen nicht mit den Forschungen von Ashley (1990) verwechselt werden. Ashleys HYPO ist in der Lage, Ähnlichkeiten[180] zwischen Fällen zu berücksichtigen und Hypothesen zu bilden, die in weiterer Folge einem Anwalt helfen sollen, die Stärke von Argumenten für und gegen eine Entscheidung im Licht neuer Fakten bewerten zu können.

[180] HYPO basiert auf einfachen, auf Fakten basierenden Analogien, im Sinne von zwischen Fällen geteilten bzw. nicht geteilten Faktoren.

12. State-of-the-Art-Technologien

12.1. JBoss Drools

12.1.1. Überblick

JBoss Drools ist ein sogenanntes „Business Rule Management System“ und liegt derzeit (August 2014) in Version 6.1.0 vor. Die Software wurde unter der „Apache Software License 2.0“ veröffentlicht, einer liberalen Open-Source-Lizenz, die eine kommerzielle Verwendung erlaubt.

Drools verwendet Produktionsregeln als primäre Datenstruktur und ist demnach als regelbasiertes System zu kategorisieren[181]. Dabei wird ein deklarativer Ansatz verfolgt. Der Entwickler soll sich darum kümmern, „was“ getan werden soll, und nicht „wie“ ein bestimmtes Problem gelöst werden soll, wie das bei imperativen Programmiersprachen (bspw. Java) der Fall ist. Deklarative Ansätze eigenen sich daher grundsätzlich besser, um die Modellierung durch den Fachexperten zu unterstützen.

Ein wesentlicher Unterschied zwischen Regeln und Methoden in Java ist der Umstand, dass Regeln nicht direkt aufgerufen und auch nicht direkt miteinander kommunizieren können (Mylopoulos, 1980, S. 7). Der Aufruf der Regeln erfolgt ausschließlich durch die Inferenzmaschine. Daraus folgt, dass eine Regel stets sämtliche Bedingungen für ihren Aufruf enthalten muss. Die Reihenfolge der Ausführung der Regeln ist grundsätzlich nicht vorbestimmt.

Version 5 von Drools basierte auf dem Algorithmus „ReteOO“, einer erweiterten und optimierten Version für objektorientierte Systeme des ursprünglich von Forgy (1982) entwickelten Rete-Algorithmus. Seit Version 6 wird ein verbesserter Algorithmus namens „PHREAK“ verwendet, der die Probleme des Rete-Algorithmus nicht mehr aufweist.

Rete und PHREAK gehören den *monotonen* Algorithmen an, die aufgrund der Fakten aus der Wissensbasis abgeleiteten Konklusionen können daher nicht verworfen werden, sobald gegenteilige Informationen zur Verfügung stehen. Ob Drools trotzdem für die Lösung juristischer Probleme eingesetzt werden kann, wird in Abschnitt 12.1.4. diskutiert.

---

[181] Programme wie Drools werden daher oft auch „Rules Engine“, „Rule Engine“ oder „Business Rules Engine“ genannt.

Drools ist eine sogenannte „Business Logic Integration Platform“, die neben der Rule Engine unter anderem auch eine Workflow Engine enthält. Die eigentliche Rule Engine – Drools Expert – ist in diese Plattform eingebettet und kann nahtlos mit den anderen Komponenten integriert werden. Sofern im Folgenden von „Drools“ gesprochen wird, ist die Rule Engine Drools Expert gemeint, sofern nicht anders angegeben[182].

Die Rule Engine Drools Expert beherrscht datengetriebenes *forward chaining* und seit Version 5 auch zielorientiertes *backward chaining* im Stil von Prolog. Aus diesem Grund wird die Engine von deren Entwicklern als „Hybrid Reasoning System“ bezeichnet.

Derzeit beherrscht Drools nur den Umgang mit eindeutigen Werten, eine Erweiterung, um mit unvollständigen Informationen umzugehen, ist jedoch bereits auf dem Weg. Anfänglich wird diese Erweiterung in einer Komponente bestehen, die mit Fuzzylogik umgehen kann. Ebenso ist eine Integration von OWL-basierten Ontologien geplant. Für weitere Informationen sei auf das sehr gute Handbuch von Drools verwiesen[183].

12.1.2. Die Verwendung von Drools im juristischen Bereich

Obwohl Rule Engines in Unternehmen seit vielen Jahren zur Automatisierung diverser Agenden, wie beispielsweise zur Prüfung der Einhaltung von Vorgaben (policy conformance checking), verwendet werden, haben sie sich bisher im öffentlichen Bereich zur Implementierung von juristischen Expertensystemen nicht durchgesetzt.

Von Palmirani et al. (2012) wurde Drools 5.4 verwendet und eine Bestimmung aus dem US-Urheberrecht modelliert. Die folgenden Ausführungen stützen sich auf die Erkenntnisse aus dieser Untersuchung.

---

[182] Drools Expert wird hier stellvertretend für die Kategorie der Rule Engines dargestellt. Es existieren zahlreiche Rule Engines, die unter verschiedenen Lizenzmodellen angeboten werden. Die Palette reicht von Open-Source-Software bis zu kommerziellen Produkten. Als prominenter Vertreter kommerzieller Systeme sei die „Oracle Policy Automation“ von Oracle genannt. Die zugehörige Webseite ist unter folgendem Link verfügbar: http://www.oracle.com/us/products/applications/oracle-policy-automation/policy-automation/overview/index.html.
Die Technologie der „Policy Automation“ wurde ursprünglich von einem Unternehmen namens SoftLaw entwickelt (Borsari et al., 2005; P. Johnson & Mead, 1991), das später Ruleburst und dann Haley hieß, bevor es letztendlich von Oracle gekauft wurde. Da es sich um ein kommerzielles Produkt handelt, sind detaillierte Informationen nicht frei zugänglich, was eine eingehende wissenschaftliche Untersuchung des Programms erschwert. Aus diesem Grund wurde von einer eingehenden Untersuchung der „Oracle Policy Automation“ Abstand genommen.

[183] *JBoss Inc.*, Drools Reference Manual 6.1.0. http://docs.jboss.org/drools/release/6.1.0.Final/drools-docs/pdf/drools-docs.pdf aufgerufen: 12.8.2014 (2013).

12.1.3. Das temporale Modell

Das temporale Modell von Drools basiert auf der Theorie von James Allen (1981, 1983) und ist demnach intervallbasiert. Drools unterstützt darüber hinaus auch Zeitpunkte, die als Intervalle ohne Dauer (0 ms) repräsentiert werden.

Dieser Ansatz ist nur effektiv, wenn „Geschäftsregeln" repräsentiert werden sollen. Damit sind Ereignisse gemeint, die sich in der Gegenwart abspielen und in der Zukunft enden. Im juristischen Kontext kommt es jedoch oftmals zu einem komplexen Mix aus Vergangenheit, Gegenwart und Zukunft. Für eine detaillierte Analyse sei auf Abschnitt 11.5. verwiesen.

Aus diesem Grund sind die in Drools für Regeln verfügbaren Attribute „date-effective" und „date-expires" unzureichend, um die zeitlichen Dimensionen des Rechts zu erfassen. Diese Limitierung wurde von Palmirani et al. (2012) teilweise umgangen, indem zeitliche Eigenschaften in die Bedingung der Regeln aufgenommen wurden.

Eine andere Möglichkeit ist, temporale Eigenschaften von Normen als Metaregeln zu betrachten, die die Ausführung der Regeln kontrollieren. Zu diesem Zweck wurde von Palmirani et al. Drools RuleFlow und jBPM[184] verwendet. RuleFlow ist ein Feature von Drools, das Kontrolle über die Ausführung von Regeln erlaubt.

Die Rule Engine wird mithilfe von XOR-Bedingungen[185] dazu gebracht, nur jene Regeln zu aktivieren, die den jeweiligen zeitlichen Parametern entsprechen. Dieser einfache Ansatz erlaubt es, Novellen relativ rasch in das formale Modell einfügen zu können.

Diese Variante ist jedoch nicht besonders effektiv, wenn mehrere Versionen einer Norm zum selben Zeitpunkt anwendbar sind. Das ist beispielsweise dann der Fall, wenn für bereits anhängige Verfahren die alte Rechtslage, für neue aber die novellierten Bestimmungen anwendbar sind. Solche Fälle könnten grundsätzlich durch den Einsatz von OR-Bedingungen[186] gelöst werden. Die Regeln stehen jedoch oftmals in komplexen Beziehungen zueinander, sodass der modellierte Workflow eine Vielzahl von Bedingungen prüfen muss, um die anwendbaren Regeln abzuleiten.

---

[184] RuleFlow und jBPM (eine Workflow Engine) wurden in Version 5.2.0 von Drools verschmolzen.

[185] Ausschließende Disjunktion, auch genannt „exklusives Oder".

[186] Nicht ausschließende Disjunktion, auch genannt „inklusives Oder".

12.1.4. Die anfechtbare Schlussfolgerung

Wie eingangs bereits erwähnt wurde, ist Drools eine hybride Rule Engine, die *forward* und *backward chaining* beherrscht. Die Konklusionen aus der Regelbasis – darunter ist die Wissensbasis als Summe aller Regeln zu verstehen – werden dabei *monoton* abgeleitet. Sobald das Antezedens einer Regel von den Fakten erfüllt ist, folgt daraus zwingend die Ableitung der Konklusion. Neue Informationen, die später hinzukommen, können bestehende Konklusionen nicht verwerfen.

Der im Zusammenhang mit temporalen Aspekten diskutierte Einsatz von jBPM zur Steuerung der Ausführung hilft hier nicht weiter. Mit diesem Ansatz lassen sich weder die Rangordnung von Regeln festlegen noch Konflikte zwischen Normen auflösen. Ebenso scheint die „Wiederinkraftsetzung“ (siehe Abschnitt 11.7.2.) von Normen im Zuge der juristischen Rechtsanwendung auf diese Weise nicht implementierbar zu sein.

Drools bietet verschiedene Möglichkeiten, mit Konflikten zwischen Regeln umzugehen. Die wohl beste Möglichkeit, die von Palmirani et al. (2012) aufgegriffen wurde, ist die Anbringung des Attributs „salience“ auf Regeln. Mit diesem Attribut wird die Priorität einer Regel festgelegt, und damit die Reihenfolge der Ausführung. Das Attribut muss jedoch explizit und im Vorhinein auf jeder Regel angebracht werden, es ist nicht möglich, die Priorität dynamisch mithilfe einer Funktion zur Laufzeit abzuleiten.

In Drools ist darüber hinaus möglich, dass im Attribut „salience“ Ausdrücke mit gebundenen Variablen verwendet werden. Der Wert dieser Variable kann zur Laufzeit dynamisch gesetzt werden. Für jede Regel wäre eine eigene Variable notwendig, die deren Priorität repräsentiert. Dieser Ansatz erscheint unbrauchbar, da die Anzahl der Variablen und damit die Wartungsintensivität der Regelbasis linear mit der Menge der Regeln ansteigt.

Drools simuliert die nicht-monotone anfechtbare Schlussfolgerung mithilfe der Prioritäten. Die generelle Regel wird vor der spezielleren ausgeführt, wobei letztere die Konklusion der allgemeinen Regel überschreibt. Die Wiederinkraftsetzung für den Fall, dass die anfechtende Regel selbst wiederum angefochten wird, scheint sich nur schwer bewerkstelligen zu lassen.

12.1.5. Constraint-Programmierung

Die Constraint-Programmierung ist ein Programmierparadigma, das seit Mitte der 1980er entwickelt wird und sich als Weiterentwicklung der logischen Programmierung versteht. Es

geht um die Beschreibung von Beziehungen zwischen Variablen mit sogenannten „Constraints“. Im Gegensatz zu den Anweisungen einer imperativen Programmiersprache geben Constraints gewissermaßen die Eigenschaften der Lösung wieder, die gefunden werden soll.

In Java ist die Constraint-Progammierung als zusätzliche Bibliothek verfügbar. Der zugehörige Standard heißt JSR-331 Constraint Programming API[187]. Dieser Standard wurde auch von Herstellern von Rule Engines aufgegriffen und implementiert. OpenRules veröffentlichte deren constraint-basierte Inferenzmaschine „RuleSolver“ im März 2012 (Feldman, 2011).

Im Gegensatz zu den gängigen Ansätzen anderer Business Rule Engines verfolgte der constraint-basierte Ansatz, der von OpenRules gewählt wurde, schon jeher die Unterscheidung zwischen *hard* und *soft constraints*. Das erinnert an die in der *defeasible logic* übliche Differenzierung zwischen *strict* und *defeasible rules*.

Die Erweiterung von Rule Solver um Features zur Beherrschung von defeasible logic war daher naheliegend. Es wurde ein Ansatz entwickelt, der die Rangordnung von Regeln in Entscheidungstabellen zum Ausdruck bringt. Dabei wird den soft rules eine Wahrscheinlichkeit zugewiesen. Dadurch entsteht ein absolutes Ranking zwischen Regeln und es wird vermieden, dass sämtliche Relationen im Vorhinein explizit festgelegt werden müssen (Feldman, 2013a, 2013b).

Dieses Feature scheint Kollisionen zwischen Rechtsnormen in Grundzügen lösen zu können. Komplexere Beziehungen zwischen Normen und die damit mögliche „Wiederinkraftsetzung“ einer bereits angefochtenen Norm können jedoch nicht dargestellt werden. Explizite Ausnahmen zwischen Normen, die oftmals in Gesetzen enthalten sind, gehen durch die absolute Priorisierung verloren. Die isomorphe Darstellung der Rechtsnormen kann daher nicht gewährleistet werden.

Grundsätzlich ist dieser Ansatz jedenfalls den sonst üblichen multi-hit Entscheidungstabellen zur Repräsentation von Prioritätsrelationen zwischen Regeln überlegen.

---

[187] JSR-331 Constraint Programming API. https://jcp.org/aboutJava/communityprocess/final/jsr331/index.html aufgerufen: 13.8.2014 (2013).

12.1.6. Experimentelle Ansätze

Am 1. August 2013 schrieb der Teamleiter von Drools Mark Proctor online[188]:

„Davide just got a first cut of Defeasible Logic in Drools. This is like a super charged TMS implementation, that allows users to handle conflicts in logical insertions. It introduces the following rule annotations

@Defeater

@Defeasible

@Defeats("rule1", "rule2", "rule3) // comma separated list of rules it defeats"

Es scheint demnach nur eine Frage der Zeit zu sei, bis Drools neben dem monotonen PHREAK-Algorithmus auch die nicht-monotone anfechtbare Schlussfolgerung beherrscht.

Abgesehen davon, dass in der aktuellen Version von Drools dieses neue Feature noch nicht verfügbar ist, sei angemerkt, dass die verwendete Annotation „Defeats" sämtliche Regeln, die von der aktuellen Regel angefochten werden, explizit anführen muss. Dies scheint nur dann sinnvoll zu sein, wenn diese Ausnahmen auch explizit im Gesetzestext in dieser Form zum Ausdruck gebracht werden.

Die manuelle und statische Angabe von Ausnahmen kann zu erheblichem Wartungsaufwand führen, sobald die Relationen von Normen durch eine Novelle geändert werden.

12.1.7. Abschließende Bemerkungen

Drools ist eine interessante Plattform, die neben einer Rule Engine viele nützliche Features wie Ereignisverarbeitung, Workflowmanagement etc. bietet. Selbst die Ausführung von tausenden Regeln in Echtzeit bereiten der Engine keine Probleme.

Im Hinblick auf den Einsatz im rechtlichen Bereich weist Drools jedoch einige Defizite auf. Zum einen beherrscht Drools ausschließlich monotone Algorithmen, sodass die Schlussfolgerungen der Engine erst der rechtlichen, nicht-monotonen Entscheidungsfindung angenähert werden müssen.

Zweitens ist das temporale Modell von Drools nur bedingt für den rechtlichen Bereich geeignet. Die temporale Theorie von James Allen (1983, 1984), auf der Drools basiert, geht

---

[188] *Proctor, M.*, Defeasible Logic. http://drools.46999.n3.nabble.com/Defeasible-Logic-tt4025258.html aufgerufen: 13.8.2014 (2013).

von einer diskreten Reihenfolge von Ereignissen aus. Rückwirkende Effekte von Rechtsnormen können daher nicht adäquat repräsentiert werden.

Für den rechtlichen Bereich scheint Drools am besten für die Entwicklung von Expertensystemen geeignet zu sein, die auf ein spezielles Anwendungsszenario fokussiert sind. So ist es möglich, die Features von Drools bestmöglich zu nutzen.

Offen bleibt jedoch, ob Drools auch als Grundlage für die Implementierung von „erweiterten“ juristischen Anforderungen geeignet ist. Zum Beispiel könnte ein Expertensystem zwischen Behauptungen und Fakten unterscheiden und aufzeigen, welche Partei die Beweislast für die Anwendung einer Regel trifft etc.

Solch spezielle Anforderungen können nur erfüllt werden, wenn die Wissensbasis des Expertensystems entsprechend ausdrucksstark ist. In Drools müssten Regeln wohl mit speziellen Attributen versehen werden, um solche Wünsche zu erfüllen. Darüber hinaus wären wohl auch Erweiterungen der Inferenzmaschine selbst notwendig, damit diese die entsprechenden Antworten ableiten kann.

Aus technischer Sicht ist Drools aufgrund seines modularen Aufbaus mit definierten Schnittstellen sehr gut geeignet, als Rule Engine in ein Expertensystem eingebettet zu werden. Damit bietet Drools ein gutes Fundament für die Entwicklung eines juristischen Expertensystems.

Es sollte jedoch nicht vergessen werden, dass Drools bedingt durch die zahlreichen Features relativ komplex ist. Je mehr in bestehende Funktionalitäten wie die Inferenzmaschine eingegriffen werden muss, desto schwieriger wird die Implementierung.

Bevor mit der Entwicklung eines Expertensystems begonnen wird, sollte daher eingehend überlegt werden, welche Anforderungen dieses erfüllen soll, und Drools entsprechend bewertet werden. Abhängig von den jeweiligen Rahmenbedingungen könnte auch ein Prototyp einer eigenen simplen juristischen Rule Engine entwickelt werden, deren Datenmodell und Inferenzmaschine speziell auf die jeweiligen Anforderungen zugeschnitten sind.

12.2. SPINdle

12.2.1. Überblick

Der „SPIN Defeasible Logic Reasoner“ (kurz „SPINdle“) ist eine regelbasierte Inferenzmaschine und basiert auf einem nicht-monotonen Schlussfolgerungsmodell. SPINdle beherrscht in Version 2.2.x *defeasible logic* und *modal defeasible logic* (Lam & Governatori, 2009).

SPINdle ist vollständig in Java implementiert und kann als eigenständiger Theorembeweiser verwendet oder in eine andere Anwendung als *defeasible logic rule engine* eingebettet werden. Die Inferenzmaschine ist in der Lage, Theorien mit mehreren tausend Regeln in wenigen Sekunden auszuführen. Die Ausführungszeit steigt dabei linear zum Verhältnis der Anzahl der Regeln an. Unter einer „Theorie“ ist im Kontext von SPINdle eine Menge von Aussagen zu verstehen.

Das Design von SPINdle verfolgt unter anderem das Ziel, offen für Erweiterungen zu sein. Die Architektur basiert auf einem Plugin-System, das es erlaubt, dass Module falls notwendig nachgeladen werden können. Das System kann so besser an zukünftige Einsatzzwecke und Anforderungen angepasst werden.

Diese Flexibilität ist wohl ein wichtiger Grund, warum SPINdle auch in jüngster Zeit in verschiedensten Bereichen eingesetzt wird. Von Lam und Governatori (2012) wurde SPINdle mit einer numerischen Rechenmaschine kombiniert, um die Steuerung von Fahrzeugen in einer urbanen Umgebung zu implementieren.

Im juristischen Bereich wurde SPINdle beispielsweise verwendet, um Lizenzen auf Anfrage des Benutzers automatisiert durch die Anwendung von Heuristiken zu kombinieren (Governatori, Lam, Rotolo, Villata, & Gandon, 2013).

12.2.2. Grundlagen der anfechtbaren Logik

Sämtliche Regelarten der *defeasible logic*, die im Folgenden „anfechtbare Logik“ genannt wird, werden von SPINdle beherrscht, namentlich *facts*, *strict rules*, *defeasible rules* und *defeaters*. Außerdem kann die *superiority relation* zwischen Regeln abgebildet werden.

Fakten sind unstrittige Aussagen, die entweder in der Form von Situationen oder Aktionen, die stattgefunden haben, repräsentiert werden. Eine Regel beschreibt hingegen den

Zusammenhang zwischen einer Menge von Literalen (Prämisse) und einem Literal (Konklusion).

Die drei Regelarten – *strict*, *defeasible* und *defeater* – variieren nach ihrer Stärke. *Strict rules* sind Regeln im klassischen Sinne. Sobald deren Prämisse erfüllt ist, folgt daraus notwendigerweise die Ableitung der Konklusion. *Defeasible rules* hingegen können durch entgegengesetzte Beweise angefochten werden. Defeaters sind die schwächsten Regeln. Es handelt sich um spezielle Regeln, aus denen selbst keine Konklusion abgeleitet werden kann. Sie blockieren jedoch die Konklusion von *defeasible rules*, indem sie Beweise für deren Nichtanwendung produzieren.

Defeasible logic ist eine „skeptische", nicht-monotone Logik. Widersprüchliche Konklusionen sind nicht zulässig. Wenn sowohl Gründe für A als auch ¬A vorhanden sind, wird keine der beiden Konklusionen abgeleitet. Im Falle von widersprüchlichen Regeln kann demnach keine Konklusion abgeleitet werden, solange die Regeln nicht priorisiert wurden. Für die Priorisierung wird eine *superiority relation* zwischen Regeln verwendet, sodass die stärkere Regel die Konklusion der schwächeren aufhebt. Eine *superiority relation* drückt den Vorrang der stärkeren Regel relativ zur schwächeren aus. Zum Beispiel bedeutet die Relation r' > r, dass r' im Fall einer Kollision Vorrang vor r hat.

12.2.3. Das temporale Modell

Das temporale Modell von SPINdle basiert auf der von Governatori, Rotolo und Sartor (2005) entwickelten theoretischen Grundlage, die auf die dynamische Behandlung von Verpflichtungen im zeitlichen Kontext ausgerichtet ist und in Verbindung mit der Modellierung von gerichteten Obligationen steht. Eine *gerichtete normative Aussage* in diesem Zusammenhang ist eine normative Aussage, die darauf gerichtet ist (das Ziel hat), dem Interesse einer bestimmten Person zu dienen (Governatori, Rotolo, et al., 2005, S. 26).

Die zeitlichen Dimensionen von Rechtsnormen können jedoch nicht von SPINdle repräsentiert werden. Das Modell von SPINdle ist nicht in der Lage, mit Novellen und rückwirkenden Effekten von Normen umzugehen. Ebenso wenig können die zeitliche Verzögerung des Inkrafttretens (z.B. im Fall von Übergangsbestimmungen) einer Rechtsvorschrift oder die (vorübergehende) Aussetzung der Anwendbarkeit (engl.

suspension) einer Norm und das mögliche Wiederaufleben derselben in der Zukunft repräsentiert werden.

12.2.4. Abschließende Bemerkungen

SPINdle implementiert sämtliche Regelarten der defeasible logic und baut auf einer modularen Architektur auf, die es erlaubt, die Inferenzmaschine problemlos in andere Anwendungen einzubetten. Trotz dieser Eigenschaften, die SPINdle als Inferenzmaschine für juristische Expertensysteme zunächst geeignet erscheinen lassen, gibt es Aspekte, die diskussionsbedürftig sind.

(1) Das temporale Modell von SPINdle ist nicht in der Lage, die zeitlichen Dimensionen des Rechts zu repräsentieren. Dies ist aber zwingend notwendig, da ein juristisches Expertensystem mit Änderungen des Rechts im Lauf der Zeit umgehen können muss.

(2) Die Syntax der Sprache von SPINdle zur Modellierung ist mit einigen gewichtigen Einschränkungen behaftet. Die Syntax ist stark an einfachen Literalen orientiert. Variable in atomaren Ausdrücken werden derzeit von SPINdle nicht ersetzt.

Seit Version 2.1.0 unterstützt SPINdle benutzerdefinierte Variablen – genannt *literale Variable* – deren Name mit einem „@“-Zeichen beginnen. Boolesche Funktionen können verwendet werden, um den Wert von literalen Variablen zu setzen. Im Body eine Regel (dem Bedingungsteil) dürfen boolesche Funktionen und literale Variablen vorkommen.

(3) Es ist jedoch nicht zulässig, literale Variable und boolesche Funktionen im Head einer Regel zu verwenden. Mit anderen Worten darf die Konklusion einer Regel keine literalen Variablen oder boolsche Funktionen enthalten. Die Konklusion einer Regel ist ein einfaches Literal bzw. dessen Negation.

(4) SPINdle erlaubt keine logischen Operatoren zur Formulierung der Bedingung im Body einer Regel. Daher lassen sich keine komplexen Bedingungen formulieren. Die im Body einer Regel enthaltenen Literale werden implizit mit einer Und-Verknüpfung behandelt und müssen daher alle wahr sein, damit das Antezedens der Regel erfüllt ist. Negierte Literale sind jedenfalls auch zulässig.

Der Sprachumfang von SPINdle umfasst zudem einfache Datums- und Zahlenfunktionen. Diese werden durch spezielle literale Variable repräsentiert werden, denen innerhalb der nachfolgenden Klammer Werte übergeben werden können.

(5) Für eine vollständige Abdeckung sämtlicher mathematischer Funktionen – wie Rundungsregeln, Prozentrechnung etc. – müsste der Funktionsumfang von SPINdle erweitert werden. Wie bereits oben angemerkt, sind literale Variable nicht in der Konklusion einer Regel zulässig. Wie in dieser Arbeit demonstriert wurde, sind jedoch oftmals für die Bestimmung der Rechtsfolgen einer Norm mathematische Operationen notwendig.

Manche der diskutierten Einschränkungen könnten überwunden werden, indem SPINdle durch eigene Module ergänzt wird, die eine Übersetzung der Anfragen des Systems in die Syntax von SPINdle durchführen und die Antwort von SPINdle wieder entsprechend zurück übersetzen. Die Komplexität eines solchen Ansatzes müsste gegen eine Erweiterung der Syntax von SPINdle selbst abgewogen werden.

12.3. LegalRuleML

12.3.1. Überblick

Juristische Expertensysteme, die auf Technologien der Künstlichen Intelligenz basieren, sollten auf gemeinsamen offenen Standards basieren. Mit diesem Ziel wurden Anstrengungen unternommen, um einen gemeinsamen Standard zu definieren, der LKIF-Rules (Gordon, 2008a) um temporale und deontische Eigenschaften erweitern sollte – die Sprache LKIF++ war geboren (Palmirani et al., 2010; Palmirani, Governatori, & Contissa, 2011)[189].

Bald wurde jedoch festgestellt, dass für einen Standard zur syntaktischen Repräsentation von Normen eine neue Sprache benötigt wird. Das OASIS-Konsortium startete die Entwicklung einer vollkommen neuen Syntax zur Modellierung von Rechtsnormen. Die Entwicklung stützte sich dabei explizit auf die Schichten des Semantic-Web-Stacks.

Aus diesen Bemühungen entstand die Sprache LegalRuleML (Athan et al., 2013b; Palmirani, Governatori, Rotolo, et al., 2011). LegalRuleML ist eine auf XML basierende Erweiterung von RuleML, die speziell zur Modellierung von Normen entworfen wurde und auf einer formalen Semantik basiert.

[189] Palmirani et al. bezeichneten die Erweiterung von LKIF-Rules selbst nicht als „LKIF++“. Dieser Begriff wurde erstmals von Ceci in dessen Dissertation verwendet (Ceci, 2013, S. 107).

12.3.2. Features

Gemäß der juristischen Ausrichtung unterstützt LegalRuleML fundamentale Features zur Modellierung von Rechtsnormen wie Isomporphismus, anfechtbare Logik, Jurisdiktion und rechtssetzende Autorität, temporale Parameter, deontische Operatoren, Qualifikationen, Negation und die Modellierung von Verhaltensweisen (engl. behaviours). Zudem ist die Integration der modellierten Normen mit einer Ontologie möglich, sodass die in der Ontologie repräsentierte Begrifflichkeit verwendet werden kann.

12.3.3. Abschließende Bemerkungen

LegalRuleML befindet sich derzeit noch in Entwicklung, es existiert daher noch keine finale Version. Es fehlt bislang an einer Inferenzmaschine, die LegalRuleML ausführen kann. Die von Palmirani et al. (2012) vorgeschlagene Erweiterung von Drools wurde nach derzeitigem Wissensstand noch nicht implementiert.

Die Syntax von LegalRuleML scheint ausdrucksstark genug zu sein, um einen Großteil der Normen repräsentieren zu können. Die isomorphe Darstellung von Normen wird durch entsprechende Verweise auf den authentischen Rechtstext gewährleistet.

Offen bleibt, ob auch spezifische Wechselwirkungen zwischen Normen, die über das Verhältnis Obligation/Verbot und Reparation hinausgehen, repräsentiert werden können. Dieses Erfordernis steht in engem Zusammenhang mit der Bildung von (einfachen) Hypothesen, siehe Abschnitt 11.10.

Prozessuale Aspekte scheinen derzeit unterrepräsentiert zu sein. Es fehlt an geeigneten Sprachkonstrukten zur Repräsentation von Verfahrensabläufen, Argumenten sowie Beweismaßen, Beweislastregeln und Vermutungen[190].

Da LegalRuleML auf XML basiert, sollte sich der Sprachumfang relativ einfach erweitern lassen. Neue in XML geschriebene Sprachmodule können damit implementiert werden, um die Syntax von LegalRuleML zu erweitern. Solche Erweiterungen bewegen sich jedoch außerhalb des Standards, was den Austausch und die Wiederverwendbarkeit des formalen Modells erschwert.

---

[190] Es sei auf die Forschungslinie hingewiesen, die der Entwicklung formaler Modelle zur Repräsentation von Beweisen und der Beweislastverteilung gewidmet ist (Bex & Verheij, 2011, 2013; Gordon & Walton, 2009b).

13. rOWLer – Eine hybride Rule Engine zur juristischen Entscheidungsfindung

13.1. Einleitung

In diesem Kapitel wird rOWLer[191] präsentiert, eine hybride Rule Engine zur juristischen Entscheidungsfindung. Die Engine kombiniert die Ausdrucksstärke von Regeln und Ontologien, um die rechtliche Entscheidungsfindung durchzuführen. Das formale Modell ist für die Verwendung in der Verwaltung (Steuerrecht, Pensionsrecht, Sozialleistungen etc.) optimiert und basiert auf einer flexiblen Architektur, die eine Anpassung an unterschiedliche Anforderungen erlaubt.

Das formale Modell ist in der Lage, Änderungen des Rechts im Lauf der Zeit zu bewältigen, und gewährleistet eine isomorphe Repräsentation der Regeln. Für die Inferenz kommt ein nicht-monotoner Algorithmus zum Einsatz, wodurch rOWLer *defeasible reasoning* beherrscht.

Die Entwicklung von rOWLer basiert auf Erfahrungen, die während der Formalisierung von Normen mit Java und OWL 2 gemacht wurden. Dieses Kapitel versucht, den Spalt zwischen der syntaktischen Repräsentation von Normen (in XML oder anderen Formaten) und dem Bedürfnis der Verwaltung nach einer leistungsstarken und flexiblen Rule Engine, die trotzdem einfach zu verwenden ist, zu überbrücken.

Die Architektur von rOWLer ist am Semantic Web Stack ausgerichtet und kompatibel zu LegalRuleML (Athan et al., 2013b), einem aufkommenden Standard für die Modellierung von Normen. Dieses Kapitel zeigt, dass die Berücksichtigung rechtstheoretischer Überlegungen zu einem formalen Modell führt, das aktuellen Systemen im Bereich der öffentlichen Verwaltung überlegen ist.

13.1.1. Motivation

Obwohl die derzeit in der Praxis eingesetzten Modelle des Rechts nützlich sind und weitgehend akzeptiert werden, weisen sie doch einige schwerwiegende Unzulänglichkeiten auf. Insbesondere verletzen sie das Prinzip des Isomorphismus in einem sich ständig ändernden rechtlichen Umfeld, wodurch Anpassungen und Wartungen zu einer

[191] Der Name „rOWLer“ ist ein Neologismus, der aus den englischen Begriffen „ruler“, im Sinne von Herrscher, der die Regeln vorgibt, und „OWL“ besteht.

Herausforderung werden. Die derzeitigen Modelle sind nicht in der Lage, die Dynamiken des Rechts – das heißt dessen Veränderung im Lauf der Zeit durch Novellen etc. – angemessen zu verarbeiten. Systeme im öffentlichen Bereich basieren zudem auf einem monotonen Schlussfolgerungsmodell, das die anfechtbare Schlussfolgerung bis zu einem gewissen Grad simulieren kann. Dieser Ansatz ist jedoch sehr limitiert und kann nur wenige Aspekte der rechtlichen Entscheidungsfindung erfassen.

Es existiert keine leichtgewichtige Rules Engine für den öffentlichen Sektor, die mit den temporalen Dimensionen des Rechts umgehen kann und *defeasible reasoning* beherrscht.

In diesem Kapitel wird die Rule Engine „rOWLer“ präsentiert, für die ein solides temporales Modell entwickelt wurde, das in der Lage ist, mit Änderungen des Rechts effizient umzugehen. Der Algorithmus kann überdies die zu einem bestimmten Zeitpunkt auf einen Fall anwendbaren Rechtsnormen ermitteln. Das formale Modell erlaubt die Entwicklung von klaren und gut strukturierten Modellen des Rechts, was zu einer Reduktion der Kosten für Anpassungen führt.

Die technische Architektur von rOWLer folgt einem modularen Ansatz und basiert auf Best Practices aus dem Bereich des Softwareengineerings. In diesem Sinne kann rOWLer auch als der erste Schritt in Richtung eines flexiblen Frameworks für die Entwicklung von juristischen Expertensystemen gesehen werden. Solch ein Framework kann die Entwicklung unterstützen, fördert die Wiederverwendung und hilft Fehler zu vermeiden.

Die Forschung und Entwicklung von rOWLer komplementiert Anstrengungen, einen anerkannten Standard für den Rule Layer des Semantik Web Stacks zu etablieren.

13.1.2. Anforderungen

A1) Ermittlung der anwendbaren Rechtslage: Das System muss in der Lage sein, die auf einen Fall anwendbare Rechtslage festzustellen.

A1.1) Temporale Dimensionen von Normen: Das Modell muss die vier temporalen Dimensionen von Rechtsnormen repräsentieren können.

A1.2) Repräsentation der aktuellen Rechtslage: Das Modell muss die Rechtslage zum aktuellen Zeitpunkt repräsentieren. Dies muss auf eine Art und Weise geschehen, die es erlaubt, den aktuellen Zeitpunkt zu verändern, ohne dass die Wissensbasis tiefgreifend verändert werden muss.

A2) Aktualisierbarkeit: Eine Besonderheit des Rechts ist dessen Änderungshäufigkeit. Anpassungen werden insbesondere durch Novellen bewirkt. Das Modell muss in der Lage sein, flexibel auf solche Änderungen zu reagieren. Dabei sollen Modifikationen eine möglichst geringe Auswirkung auf die Struktur des Modells haben.

A3) Repräsentation der historischen Gesetzeslage: Im Recht ist es eine Notwendigkeit, dass Fälle unter Umständen nach einer älteren Rechtslage beurteilt werden müssen, die bereits außer Kraft getreten ist. Das kann aus rechtlichen Gründen erforderlich sein oder etwa wenn eine, Entscheidung für eine Überprüfung der Richtigkeit wiederholt werden muss. Die historische Gesetzeslage muss daher erhalten bleiben und vom System bereitgestellt werden können. Damit ist gewährleistet, dass das System die für einen Fall maßgebliche Rechtslage anwenden kann.

A4) Anfechtbare Schlussfolgerung (engl. defeasible reasoning): Die Inferenzmaschine muss in der Lage sein, die anfechtbare Schlussfolgerung durchzuführen. Dabei muss das System zumindest die Anfechtung von Regeln durch *entkräftende Kollisionen* und die *Wiederinkraftsetzung* angefochtener Regeln beherrschen.

A5) Spezifikation von Ausnahmen zwischen Regeln: Um der Struktur von Gesetzen möglichst zu entsprechen, muss die Spezifikation von *expliziten* Ausnahmen zu einer Regel möglich sein. Andererseits existieren auch *implizite* Ausnahmerelationen zwischen Bestimmungen, sodass das formale Modell auch diese repräsentieren muss.

A5.1) Sortierung von impliziten Ausnahmen: Das System sollte in der Lage sein, implizite Ausnahmen zu Regeln zu sortieren. Dies erleichtert die Modellierung erheblich, hilft Fehler zu vermeiden und entspricht besser der Struktur von Gesetzen.

A6) Isomorphismus: Um die Validierung und Wartung zu vereinfachen, muss eine direkte Beziehung zwischen den Regeln des formalen Modells und den Teilen des authentischen Rechtstexts bestehen, wie insbesondere den Paragrafen.

A7) Reifikation: Normen sind Objekte mit Eigenschaften wie Jurisdiktion, rechtserzeugende Autorität, temporale Attribute etc. Das Modell muss demnach eine adäquate Repräsentationsform von Normen unterstützen, an die diverse Eigenschaften geknüpft werden können. Nur so ist eine (automatisierte) effektive juristische Entscheidungsfindung möglich.

A8) Begründung der Entscheidung: Ein menschlicher Benutzer soll in der Lage sein, die Entscheidungen des Systems nachvollziehen zu können. Das System muss erklären, welche Schritte zum jeweiligen Ergebnis geführt haben. Die bloße Erklärung, wie eine (automatisierte) Entscheidung zustande gekommen ist, reicht jedoch im juristischen Bereich nicht aus. Um eine rechtsgültige Begründung zu generieren, ist erforderlich, dass das System seine Entscheidungen „rechtfertigt". Es muss demnach ersichtlich sein, warum eine bestimmte Entscheidung bei der Falllösung getroffen wurde. Sämtliche Entscheidungen müssen auf einer Rechtsgrundlage basieren und durch entsprechende Bestimmungen gerechtfertigt werden. In der Begründung muss die Verbindung zwischen Entscheidung und Rechtsgrundlage hinreichend erläutert werden.

A9) Änderbarkeit der Entscheidung: Eine vom System erzeugte Entscheidung muss vom juristischen Experten geändert werden können, sollten Anpassungen oder Korrekturen erforderlich sein. Dies gilt auch für einzelne Entscheidungsschritte, die während der Schlussfolgerung vom System verarbeitet werden.

A10) Präsentation von Entscheidungsalternativen: Sofern zu einer Norm unterschiedliche Interpretationen vorhanden sind, soll das System dem Juristen diese Entscheidungsalternativen aufzeigen, sodass dieser wählen kann. Der Jurist erhält nach Wahl einer Interpretation sofort das Ergebnis angezeigt, sodass die für den Fall geeignetste Lösung gefunden werden kann.

13.1.3. Designprinzipien

DP1) Modularität: Das System soll durch die Verwendung von Interfaces modular aufgebaut werden. Dies reduziert die Komplexität des Systems und gewährleistet dessen einfachere Wartung. Die Modularität ist überdies Grundvoraussetzung, um das System offen für Erweiterungen zu machen.

DP2) Erweiterbarkeit: Das System soll so konzipiert werden, dass es mithilfe definierter Schnittstellen durch Module erweitert werden kann. Um dieses Ziel zu erreichen, ist eine klare und präzise API (Application Programming Interface) erforderlich. Das Programm muss sogenannte „Hooks" – etwa in Form von Schablonenmethoden (template methods) – anbieten, damit der Programmablauf individualisiert werden kann.

DP3) Wiederverwendbarkeit: Beim Design des Systems soll auf die Wiederverwendbarkeit der Module Bedacht genommen werden. Es soll über seinen ursprünglich angedachten Einsatzweck hinaus möglichst flexibel nutzbar sein. Dazu ist erforderlich, dass die Teile des Systems mit einem möglichst hohen Abstraktionsniveau entworfen werden. Spezielle Module, die auf einen bestimmten Anwendungszweck zugeschnitten sind, müssen konfigurierbar sein oder ausgetauscht werden können.

13.2. Architektur

Das theoretische Modell des in diesem Kapitel dargestellten Expertensystems zur semi-automatischen Entscheidungsfindung und -unterstützung besteht aus drei Hauptschichten, komplementiert durch eine elektronische Dokumentensammlung, nämlich (1) der Prozessschicht, (2) der Regelschicht und (3) der ontologischen Schicht.

Jede der drei Schichten repräsentiert eine andere Sicht auf das Recht und Normen mit einer unterschiedlichen Abstraktionsstufe. Das Recht wird dabei als dialektischer Prozess, als logische Regeln und als eine Sammlung von Begriffen verstanden.

Die drei Hauptschichten sowie die elektronische Dokumentensammlung werden im Anschluss einer näheren Betrachtung unterzogen.

13.2.1. Die Prozessschicht

Die Prozessschicht weist den höchsten Abstraktionsgrad auf und formalisiert das rechtliche Verfahren. Sie ist für die Durchführung des Dialogs zwischen dem Antragsteller und der Behörde verantwortlich, ist für die Ablaufsteuerung zuständig und koordiniert die einzelnen Verfahrensschritte. Während des Verfahrens werden die relevanten Fakten automatisiert und manuell gesammelt, dabei wird die Regelschicht aufgerufen, um kontinuierlich das (vorläufige) Ergebnis bereitzustellen. Soll eine „schwierige“ Regel angewendet werden, kann das System den juristischen Experten um Entscheidung bitten.

Der Ablauf des Verwaltungsverfahrens wird in Grundzügen durch das formelle Recht vorgegeben. Die Ausgestaltung im Detail obliegt jedoch der jeweiligen Behörde bzw. dem verfahrensleitenden Organ. Aufgabe der Prozessschicht ist es, den Dialog zwischen

Antragsteller und Behörde bzw. zwischen den Parteien in einem Mehrparteienverfahren zu formalisieren[192].

In der ersten Phase, dem sogenannten Ermittlungsverfahren, geht es um die Erhebung der rechtserheblichen Tatsachen. Diese werden vor allem von den Parteien vorgebracht, können jedoch auch von Amts wegen erhoben werden. Das System kann bei der Behörde bereits vorhandene Daten nutzen sowie externe Register abfragen, um weitere Daten zu erhalten. Gegebenenfalls kann die Genehmigung zur automatisierten Ermittlung von Daten von den Parteien eingeholt werden.

Neben manuellen sind demnach auch automatisierte Schritte möglich, die das System selbstständig ausführt. Durch die Kombination von manuellen und automatisierten Arbeitsschritten wird das Zusammenwirken von Mensch und Maschine in einem semi-automatischen Ablauf erreicht.

Schon während der Sachverhaltsfeststellung kann die Regelschicht aufgerufen werden, um laufend das Ergebnis der rechtlichen Beurteilung zu ermitteln und zu präsentieren. Schwierige Fälle, d.h. wenn die Anwendbarkeit einer Norm unsicher ist, werden dem menschlichen Benutzer vorgelegt, um das Problem zu lösen. Ferner ist denkbar, dass zur Lösung solcher Fälle komplexere Argumentationssysteme integriert werden, die zumindest dem Benutzer verschiedene Lösungsalternativen aufzeigen können.

Neben der Durchführung der materiellrechtlichen Beurteilung des Falles berücksichtigt das System auch formellrechtliche Aspekte. So ist es von größter Wichtigkeit, dass zwischen Behauptungen und Fakten unterschieden wird. Eine Behauptung wird durch die Vorlage adäquater Beweismittel zum Fakt. Jeder Beweis ist mit einem Status versehen, der anzeigt, ob der jeweilige Beweismaßstab erfüllt wurde und der Beweis damit erbracht ist. Die formalen Regeln werden dazu mit entsprechenden Beweisen verknüpft, jeder Beweis wiederum mit Beweislastregeln und dem zu erfüllenden Beweismaß. Je nach Beweisart und Beweismaß kann ein Beweis vom System automatisiert oder vom Sachbearbeiter akzeptiert werden.

[192] Es sei darauf hingewiesen, dass die Architektur derzeit als Single-Agent-System konzipiert und für den Dialog zwischen Behörde und Antragsteller optimiert ist.

13.2.2. Die Regelschicht

Diese Schicht des Systems repräsentiert die Normen in Form von formalen Regeln und führt die (juristische) Schlussfolgerung mithilfe der Inferenzmaschine durch. Die rechtliche Entscheidungsfindung wird durchgeführt, indem die notwendigen Informationen aus der Ontologie gewonnen und der Prozessschicht zur Verfügung gestellt werden.

Normen sind die durch juristische Interpretation ermittelte Bedeutung des Rechtstexts. Die Normen werden durch logische Regeln in eine formale Repräsentation überführt.

Es existieren mehrere Arten von Regeln. Im Bereich der KI und Recht sind Regeln folgender Definition von Interesse (Jewell & Abate, 2001): *„One of a set of explicit or understood regulations or principles governing conduct within a particular sphere of activity."*

Weiters wird in Übereinstimmung mit der Rechtstheorie davon ausgegangen, dass Normen logisch gesehen eine konditionale Struktur aufweisen, die aus Antezedens und Konsequenz besteht (Kelsen, 1979). Die Konsequenz einer Regel ist nicht notwendigerweise eine deontische Qualifikation.

Die Regelschicht des Systems unterstützt daher verschiedene Arten von Regeln. Für die Schlussfolgerung kommt ein nicht-monotoner Algorithmus zum Einsatz, der auf *defeasible logic* basiert. Die Inferenzmaschine beherrscht sämtliche Regelarten der *defeasible logic*.

Das Objektmodell unterstützt sowohl die Modellierung von impliziten als auch von expliziten Ausnahmeregelungen. Die Priorität von Regeln kann absolut durch Vergabe einer Gewichtung oder relativ im Verhältnis zu einer anderen Regel angegeben werden. Damit wird dem Aufbau und der Struktur von Gesetzen unter dem Aspekt des Isomorphismus am besten entsprochen.

Im ersten Schritt analysiert das System die formalen Regeln und versucht die Relationen zwischen generellen und speziellen Normen automatisiert zu ermitteln. Das Ergebnis dieser Analyse ist ein mit Prioritätsrelationen angereichertes Modell, in dem neben den explizit modellierten Ausnahmen auch die impliziten Beziehungen (*Lex specialis* und *Lex generalis*) zwischen den Normen festgelegt sind.

Nicht immer ist die Anwendbarkeit einer Regel oder eines Tatbestandselements eindeutig. Soll vom System eine solche „schwierige Regel" angewendet werden, so muss auf ausgefeiltere Strategien zurückgegriffen werden als der bloße Abgleich der Bedingungen mit

dem Sachverhalt. Dazu sendet die Regelschicht eine entsprechende Anforderung an die Prozessschicht zurück und bittet um Lösung des Problems. Aufgabe der Prozessschicht ist nun, die „schwierige“ Regel zu behandeln. Es wird versucht, eine Lösung für das Problem zu finden, indem die vorhandenen Problemlösungsstrategien geprüft werden.

Wird eine Strategie gefunden, die mit der fraglichen Regel umgehen kann, so wird diese verwendet, um das Problem zu lösen. Eine solche Strategie kann unter anderem der Aufruf eines auf „Faktoren und Dimensionen“ (Ashley, 1990; Sartor, 2006b) basierenden Moduls sein, das in der Lage ist zu klären, ob ein vager Gesetzesbegriff im Einzelfall anwendbar ist oder nicht. Für die Behandlung von „schwierigen“ Fällen kommen jedoch nicht nur automatisierte Problemlösungsstrategien in Frage, sondern es kann auch der menschliche Benutzer in den Problemlösungsprozess eingebunden werden. Das System kann den Benutzer beispielsweise fragen, ob ein vages Tatbestandselement im Einzelfall erfüllt ist oder nicht.

Je nach Möglichkeit kann die Software den Benutzer unterstützen und mehrere Entscheidungsmöglichkeiten zur Behandlung einer „schwierigen“ Norm vorschlagen. Dabei steht es dem Benutzer offen, die zu einer Norm vorhandenen Kommentare und Entscheidungen zu sichten, um zu einer Entscheidung zu gelangen. Damit kann auch in Fällen Unterstützung geboten werden, zu denen mehrere vertretbare Interpretationen existieren oder keine einheitliche höchstgerichtliche Rechtsprechung vorhanden ist. Damit das System erkennen kann, ob eine Regel einen „schwierigen“ Fall repräsentiert, muss diese im Objektmodell entsprechend gekennzeichnet werden.

13.2.3. Die ontologische Schicht

Die ontologische Schicht unterstützt die Prozess- und die Regelschicht durch „oberflächliche“ Schlussfolgerungen, basierend auf der Wissensbasis, soweit die Ausdrucksstärke von OWL 2 reicht. Das modellierte Wissen wird so für komplexere Schlussfolgerungen, die in Folge von der Regelschicht durchgeführt werden, vorbereitet. Zu den Aufgaben der Ontologie gehören unter anderem die Vervollständigung von Daten sowie die Ableitung von Inferenzen aus dem Sachverhalt (Behauptungen, Fakten und Beweise) und den Rechtsbegriffen.

Wie der Name andeutet, besteht diese Schicht aus mehreren in OWL 2 modellierten Ontologien: (1) Einer Core-Ontologie (LKIF Core), (2) einer Domänenontologie mit den

Begriffen des Gesetzes sowie Legaldefinitionen und (3) einer anwendungsspezifischen Ontologie, die die Daten des Einzelfalls enthält.

Der Aufbau von Ontologien entspricht recht gut der Struktur von Gesetzen, die Begriffe oftmals in Hierarchien anordnen. Ontologien ohne Regeln repräsentieren jedoch nicht viel mehr als eine Taxonomie (Sartor, 2009b). Beschreibungslogik ist selbst für „einfache“ Fälle zu ausdrucksschwach, um die rechtliche Beurteilung durchzuführen zu können.

Legaldefinitionen können jedoch grundsätzlich mithilfe von Ontologien modelliert werden. Taxonomien, also hierarchische Anordnungen von Begriffen, können relativ gut in einer Ontologie modelliert werden. Die Ontologie bereitet so die Begrifflichkeit und Beziehungen auf und trägt so maßgeblich zum besseren Verständnis der Domäne bei.

Die Anwendungsontologie enthält sämtliche verfahrensrelevanten Daten des Einzelfalls. Zu diesen Verfahrensdaten gehören die Informationen über die Parteien, der Antrag, der Sachverhalt sowie die vorgelegten Beweismittel etc.

Der Antrag beinhaltet das Begehren sowie die Sachverhaltsdarstellung aus Sicht des Antragstellers. Von letzterer ist die „akzeptierte“ Sachverhaltsdarstellung zu unterscheiden, die sämtliche Fakten umfasst, deren Richtigkeit durch einen entsprechenden Beweis bescheinigt wurde.

Der große Vorteil von Ontologien ist der Umstand, dass es sich um semi-strukturierte Datensammlungen handelt. Die gespeicherten Daten müssen nicht immer vollständig sein. Dies ist ein wesentlicher Unterschied zu Datenbankmodellen, die üblicherweise durch zahlreiche Integritätsbedingungen eingeschränkt werden. Der Nachteil solcher stark strukturierter Modelle ist die Notwendigkeit, stets nur vollständige Datensätze einfügen zu können. Vorteilhaft ist wiederum, dass so eine Mindestqualität der Daten gewährleistet werden kann. Das soll jedoch nicht bedeuten, dass weniger stark strukturierte Datensammlungen zwangsweise eine geringere Qualität aufweisen. Bedingungen und Einschränkungen können an anderer Stelle, zum Beispiel bei der Erfassung der Daten, etabliert werden, ohne die Vorteile semi-strukturierter Datenmodelle zu verlieren.

Für juristische Expertensysteme ist ein stark strukturiertes Datenmodell jedoch nicht unproblematisch, liegt es doch in der Natur eines Verfahrens, dass die rechtserheblichen Tatsachen und Rechtsfolgen schrittweise ermittelt werden. Ontologien bieten den

immensen Vorteil, dass relevante Daten nach und nach hinzugefügt werden können, sobald diese zur Verfügung stehen.

Die an einem Verfahren beteiligten Parteien, die vorgebrachten Tatsachenbehauptungen und Beweise stehen zueinander in wechselseitigen Beziehungen. Ein juristischer Fall kann abstrakt als miteinander vernetzte Objekte betrachtet werden. Domänen, deren Daten naturgemäß in Netzwerken organisiert sind, können mit relationalen Datenbankmodellen nur unzureichend erfasst werden (Florescu, 2005). Ontologien eignen sich aufgrund ihrer strukturellen Beschaffenheit besser für die Darstellung von Entitäten und deren Beziehungen.

Experimente mit OWL 2 haben gezeigt, dass Legaldefinitionen grundsätzlich in Ontologien modelliert werden können. Begriffe werden dabei durch entsprechende beschreibungslogische Restriktionen formalisiert. So kann beispielsweise ausgedrückt werden, dass unter einem „Angehörigen“ im Sinne des Gesetzes der Ehepartner und die Kinder des Antragstellers zu verstehen sind. Diese Definition variiert je nach Gesetz, sodass denkbar wäre, dass auch die nicht verheiratete Lebensgefährtin des Antragstellers unter den Begriff „Angehöriger“ zu subsumieren ist.

Die von der Inferenzmaschine durchgeführten Schlussfolgerungen stehen sofort zur Verfügung und können von den Regeln der Regelschicht verwendet werden. Eine Regel, die zur Bedingung hat, dass eine Person ein Angehöriger des Antragstellers ist, kann diese Information direkt aus der Ontologie beziehen.

Die Kommunikation zwischen Regelschicht und Ontologie ist bidirektional, sodass neue aus den Regeln abgeleitete Konklusionen der Inferenzmaschine zur Verfügung gestellt werden. Die Interaktion zwischen den beiden Schichten ist jedoch im Detail sehr komplex, nicht zuletzt aufgrund der unterschiedlichen Semantik von Regeln und Ontologien. Ontologien folgen nämlich der „open world assumption“, wohingegen regelbasierte Formalismen grundsätzlich auf der „closed world assumption“ beruhen, womit neben technischen Schwierigkeiten auch der semantische Graben zwischen den Technologien überwunden werden muss.

13.2.4. Elektronische Dokumentensammlung

Flankiert werden die anderen Schichten und damit das formale Modell des Gesetzes von der sogenannten „Elektronischen Dokumentensammlung“, die eine elektronische Version juristischer Dokumente bereitstellt. Die elektronischen Dokumente können beispielsweise in Akoma Ntoso (Palmirani & Vitali, 2011) oder CEN MetaLex (Boer & van Engers, 2011) erstellt werden. Entitäten der anderen Schichten, wie Regeln und Begriffe etc., werden mithilfe von IRIs mit dem Rechtstext verknüpft. Damit kann die Entscheidungsfindung durch den Juristen unterstützt werden, indem diesem Gesetze, Verordnungen und Urteile etc. vom System zur Verfügung gestellt werden. Ferner wird die isomorphe Darstellung der Regeln gefördert, indem diese mit deren Rechtsgrundlage verknüpft werden.

Zu den vorhandenen elektronischen Dokumenten gehören Gesetze, Verordnungen, Urteile, Erlässe, aber auch nicht amtliche Dokumente wie Kommentare und Lehrbücher.

Die elektronischen Dokumente unterstützen weniger die automatisierte Entscheidungsfindung, sondern sind die Grundlage für ein integriertes Rechtsinformationssystem. Sämtliche Entitäten der anderen Schichten, insbesondere Regeln, können auf die zugehörigen elektronischen Dokumente verweisen. Auch die elektronischen Dokumente sind untereinander verlinkt. Zu den in einem Kommentar zitierten Urteilen ist beispielweise eine Verknüpfung hinterlegt, sodass diese bei Bedarf aufgerufen werden können. Damit wird die Grundlage für ein interaktives Rechtsinformationssystem geschaffen.

Bei Bedarf kann der Benutzer den zu einer Regel gehörenden (authentischen) Rechtstext aufrufen oder weitergehende Informationen im elektronischen Kommentar sichten sowie die Regel betreffende Urteile abrufen. Ferner kann der Dokumentenbestand anhand von Schlüsselwörtern durchsucht werden. Der Jurist kann so das Ergebnis der rechtlichen Beurteilung überprüfen, indem er die zu einer Regel vorhandenen Kommentare und Entscheidungen heranzieht.

Um die elektronischen Ressourcen zu adressieren und referenzieren, kommt ein auf IRIs basierendes Modell zum Einsatz. Damit können die Verweise von Regeln auf den Rechtstext etc. erzeugt werden. Das System ist in der Lage, solche Verweise automatisiert aufzulösen und die adressierte elektronische Ressource abzurufen.

Eine Regel kann mit mehreren elektronischen Dokumenten verknüpft werden. Dies gilt grundsätzlich auch für alle anderen Verweise zwischen den Dokumenten selbst. Um diese Anforderung zu formalisieren, verweist eine Regel auf die Instanz in der Ontologie, die eine Norm repräsentiert. Die Instanz in der Ontologie dient gewissermaßen als Anker, von dem aus auf sämtliche elektronischen Dokumente verwiesen wird, die mit der Vorschrift in Zusammenhang stehen. Damit wird vermieden, dass die Regelschicht unnötig mit Metadaten (den Verweisen) überladen wird.

Das auf IRIs basierende Modell ermöglicht eine nahtlose Integration des Expertensystems mit elektronischen Dokumenten, die eindeutig adressiert werden können. Die elektronische Fassung von Kommentaren etc. kann damit auch von einem vorhandenen Rechtsinformationssystem zur Verfügung gestellt werden. Vorausgesetzt, dass die intern verwendeten IRIs geeignet sind, die Dokumente des Rechtsinformationssystems und deren Inhalt zu adressieren. Die elektronischen Dokumente müssen aus mehreren hierarchisch angeordneten Komponenten bestehen, sodass diese von der Struktur einer IRI erfasst werden können.

Auf Grundlage der hier skizzierten Architektur und insbesondere der Ontologie wäre ein ausgereifterer elektronischer Kommentar implementierbar. Dieser wäre nicht bloß auf eine Volltextsuche beschränkt, sondern die Suche könnte mithilfe der Ontologie auf bedeutungsgleiche und bedeutungsnahe Begriffe automatisch ausgeweitet werden.

13.3. Schlussfolgerungsmodul und Algorithmus

An dieser Stelle wird ein Überblick über das Schlussfolgerungsmodul gegeben. Technisch wird der Algorithmus zur Entscheidungsfindung vom Schlussfolgerungsmodul gekapselt, das die Inferenzmaschine mit dem restlichen System integriert und auch das temporale Modell umschließt. Das Modul kann als dünne Integrationsschicht gesehen werden, die die für den Aufruf der Inferenzmaschine notwendigen Parameter aus den Fakten ableitet, wie zum Beispiel das „signifikante“ Datum.

Oftmals ist es notwendig Situationen zu beurteilen, die sich über einen längeren Zeitraum hinziehen[193]. Solche Szenarien werden ebenso vom Schlussfolgerungsmodul behandelt, das mit der Inferenzmaschine interagiert, um das Gesamtergebnis der Beurteilung zu ermitteln.

In den folgenden Unterabschnitten wird der Algorithmus zur Entscheidungsfindung präsentiert. Es wird jedoch lediglich ein Überblick gegeben und keine vollständige Logik entwickelt.

Der hier vorgeschlagene Algorithmus besteht in Grundzügen aus zwei voneinander unabhängigen Schritten: (1) Ermittlung der in zeitlicher Hinsicht auf den Sachverhalt anwendbaren Normen und (2) der Durchführung der rechtlichen Beurteilung durch Anwendung der Regeln auf die Sachverhaltselemente.

Die Trennung zwischen zeitlichen Aspekten und der rechtlichen Beurteilung selbst ermöglicht eine separate Behandlung beider Problemkreise. Technisch werden beide Schritte von einer entsprechenden Schnittstelle gekapselt, für die jeweils eine unabhängige Implementierung vorhanden ist.

Dieser Ansatz reduziert die Komplexität des Algorithmus, indem das Gesamtproblem in kleinere Teile zerlegt wird, und unterstützt gleichzeitig die bessere Integrierbarkeit, Wartung und Testbarkeit.

13.3.1. Die Ermittlung der anwendbaren Rechtslage

Zuerst gilt es, die auf einen Fall in zeitlicher Hinsicht anwendbaren Normen zu ermitteln. Dazu ist (1) die Ermittlung des *maßgeblichen Zeitpunkts* notwendig, der für das Auffinden der anwendbaren Rechtslage ausschlaggebend ist, und (2) ist der *Zeitpunkt der rechtlichen Beurteilung* selbst zu berücksichtigen.

Das System muss demnach feststellen, welche Normen zum *Zeitpunkt der rechtlichen Beurteilung* t'', das ist der Beobachtungszeitpunkt der Rechtsordnung, auf den Sachverhalt zum *maßgeblichen Zeitpunkt* t', anzuwenden sind.

---

[193] Beispielsweise sind nach der Rechtsprechung des Obersten Gerichtshofs (OGH 1 Ob 9/96) Dauersachverhalte (wie Versicherungsverträge) vor der Novelle gemäß der alten und danach entsprechend der neuen Rechtslage zu beurteilen. Es finden demnach auf einen Sachverhalt mehrere Rechtslagen Anwendung und sind bei der Gesamtbeurteilung zu berücksichtigen.

Welches Datum als *maßgeblicher Zeitpunkt* t' zu qualifizieren ist, hängt vom jeweiligen Rechtsbereich ab. Der für die rechtliche Beurteilung maßgebliche Zeitpunkt kann beispielsweise im Verwaltungsrecht das Antragsdatum oder im Strafrecht der Zeitpunkt der Tat sein.

Es geht um die Beantwortung der Frage, zu welchem Zeitpunkt die Tatbestandsvoraussetzungen vorliegen müssen, damit eine Norm auf den Sachverhalt anwendbar ist. Damit die Anwendbarkeit einer Norm auf einen Sachverhalt gegeben ist, muss der maßgebliche Zeitpunkt t' innerhalb des zeitlichen Bedingungsbereichs der Norm liegen.

Im zweiten Schritt ist der *Zeitpunkt der rechtlichen Beurteilung* zu berücksichtigen. Damit ist abstrakt jener Zeitpunkt gemeint, an dem ein bestimmter Sachverhalt der rechtlichen Prüfung unterzogen und die Entscheidung gefällt wird. Dieser Zeitpunkt ist gleichbedeutend mit dem Beobachtungszeitpunkt der Rechtsordnung (siehe Abschnitt 13.4.5.1.). Die Rechtsfolgen einer Norm dürfen nur dann verhängt werden, wenn der Zeitpunkt der rechtlichen Beurteilung t'' innerhalb des zeitlichen Rechtsfolgenbereichs der Norm liegt.

Die Ableitung von Rechtsfolgen zum Zeitpunkt der rechtlichen Beurteilung kann auch automatisiert erfolgen. Zum Beispiel kann ein E-Government-System einen Antrag entgegen nehmen und der Antragsteller erhält augenblicklich ein (vorläufiges) Ergebnis präsentiert. Dabei könnte das System das aktuelle Datum als Zeitpunkt der rechtlichen Beurteilung interpretieren.

Welche Rechtslage für einen Fall maßgeblich ist, lässt sich nicht generell feststellen. Im Strafrecht ist beispielsweise grundsätzlich jene Rechtslage zum Zeitpunkt der Tatbegehung anwendbar, es sei denn, die Normen der neueren Rechtslage sind für den Beschuldigten in ihrer Gesamtschau günstiger.

Wie in Abschnitt 11.5.1. ausführlich dargestellt wurde, müssen sich das Inkraftstehen, der Bedingungs- und Rechtsfolgenbereich einer Norm nicht zwingend decken. Oftmals wird mit „in Kraft“ und „außer Kraft treten“ undifferenziert Beginn und Ende des Bedingungs- und auch des Rechtsfolgenbereichs gemeint. Decken sich Bedingungs- und Rechtsfolgenbereich in zeitlicher Hinsicht, so ist bereits der erste Schritt des hier skizzierten Algorithmus hinreichend, um die in zeitlicher Hinsicht anwendbaren Normen zu identifizieren.

Anders verhält es sich, wenn Bedingungs- und Rechtsfolgenbereich auseinander fallen. In diesen Fällen kann nicht undifferenziert vom Inkraftstehen einer Norm gesprochen werden. Vielmehr muss unterschieden werden, welcher der beiden Bereiche gemeint ist.

Der hier skizzierte – auf den Rechtsfolgenbereich abstellende – Ansatz für die Ermittlung der anwendbaren Rechtslage zum Zeitpunkt der rechtlichen Beurteilung scheint für beide Varianten geeignet zu sein. Für weitere Details zur Ermittlung der anwendbaren Normen sei auf Abschnitt 13.4.5. verwiesen.

Die Ermittlung der in zeitlicher Hinsicht anwendbaren Rechtsvorschriften stellt sich jedoch nicht immer so einfach dar. In der Regel enthalten Gesetze Übergangsbestimmungen, die das Inkrafttreten von (einzelnen) Bestimmungen verzögern. Solche Übergangsbestimmungen werden eingesetzt, wenn die Effekte eines Gesetzes auf die Normadressaten derart schwerwiegend sind, dass es angebracht erscheint, der Gesellschaft Zeit zu geben, ihr Verhalten an die neuen Vorschriften anzupassen.

Derartige Besonderheiten variieren je nach Gesetz und können nicht von einem generellen Algorithmus abgedeckt werden. Ein Framework, das den vorgeschlagenen Algorithmus implementiert, muss daher geeignete Erweiterungsmöglichkeiten anbieten, die es erlauben, den Algorithmus im Einzelfall zu modifizieren.

13.3.2. Die rechtliche Beurteilung

Nach dem ersten Schritt stehen sämtliche Rechtsnormen fest, die in zeitlicher Hinsicht *in abstracto* auf den Sachverhalt anwendbar sind. Die temporalen Aspekte des Sachverhalts wurden vorab verwendet, um generell eine Vorauswahl der Normen zu treffen.

Welche Normen *in concreto* aufgrund der Faktenlage anwendbar sind, gilt es nun im zweiten Schritt zu ermitteln. Ob für die rechtliche Beurteilung ein auf monotoner oder nicht-monotoner Logik basierender Algorithmus verwendet wird, ist nicht vorherbestimmt und kann nach Zweckmäßigkeit variiert werden. Ebenso kann die Inferenzmaschine eine datengetriebene Vorwärtsverkettung, eine zielorientierte Rückwärtsverkettung oder eine Kombination aus beiden als hybriden Ansatz implementieren, je nachdem welchem Zweck das Expertensystem dient.

Juristische Expertensysteme, gleich welchem Zweck diese dienen, müssen grundsätzlich mit unvollständigen und eventuell widersprüchlichen Informationen umgehen. Es ist Wesensart

kontradiktorischer Verfahren, die Sach- und Rechtslage, vorangetrieben durch das Vorbringen der Parteien, zu erörtern. Es handelt sich um einen inkrementellen Prozess, wobei sich die für die rechtliche Beurteilung maßgeblichen Umstände erst nach und nach verdichten, sodass eine Entscheidung gefällt werden kann.

Wie auch bereits weiter oben ausgeführt (siehe Abschnitt 11.7.), empfiehlt es sich für juristische Expertensysteme, auf einem nicht-monotonen Ansatz der Schlussfolgerung aufzubauen. Die einfachste Variante ist die Implementierung eines *anfechtbaren* Modus Ponens (defeasible modus ponens). Im Gegensatz zum *monotonen* Modus Ponens können neue Informationen bereits abgeleitete Konklusionen ungültig machen (Walton, 2005, S. 81 f., 107).

An dieser Stelle soll ein nicht-monotoner, vorwärtsverkettender Algorithmus, der auf dem Modus Ponens beruht, in Grundzügen skizziert werden. Auf technische Details wird an dieser Stelle zur besseren Lesbarkeit bewusst verzichtet. Normen werden dabei als konditionale Strukturen verstanden, bestehend aus dem Antezedens und der Konsequenz (auch genannt Sukzedens).

Die Inferenzmaschine prüft zuerst – analog zum juristischen Syllogismus – das Antezedens jeder Regel. Sind sämtliche Bedingungen des Antezedens erfüllt, so wird die Regel als „aktiv“ markiert, deren Konklusion jedoch vorerst noch nicht abgeleitet.

Im zweiten Schritt muss das System nach anfechtenden Regeln (defeaters) suchen, die die im ersten Schritt als „aktiv“ markierten Regeln angreifen. Wird eine solche anfechtende Regel gefunden und hat diese höhere Priorität als die angefochtene Regel, so wird letztere als „angefochten“ markiert und zunächst deaktiviert. Selbst diese Deaktivierung einer grundsätzlich anwendbaren Regel muss nicht endgültig sein. Die angefochtene Regel kann wiederaufleben, wenn die anfechtende Regel selbst wiederum erfolgreich attackiert wird usw. Dieses Phänomen wird als Wiederinkraftsetzung bezeichnet, für weitere Ausführungen sei auf Abschnitt 11.7.2. verwiesen.

Erst wenn feststeht, dass keine weiteren anfechtenden Regeln (mit höherer Priorität) vorhanden sind, kann im dritten Schritt die Konklusion einer Regel endgültig abgeleitet werden. So werden nach und nach weitere Konklusionen abgeleitet, wobei in schwierigen Fällen auch der Benutzer des Systems unterstützend mitwirken könnte.

Der Algorithmus wird beendet, wenn keine weiteren Regeln zur Prüfung zur Verfügung stehen. Mit Voranschreiten des Verfahrens können neue Fakten hinzukommen, die eine erneute rechtliche Beurteilung erforderlich machen, sodass der Algorithmus erneut ausgeführt werden muss.

13.3.3. Abschließende Bemerkungen

Die in Abschnitt 11.5.3. thematisierte Unterscheidung hinsichtlich der internen und externen zeitlichen Dimension von Normen findet sich in diesem Abschnitt wieder.

Der erste Schritt des vorgeschlagenen Algorithmus – die Feststellung der anwendbaren Rechtslage – befasst sich mit den externen zeitlichen Dimensionen von Rechtsnormen. Darunter sind jene Zeitangaben zu verstehen, die einer Rechtsvorschrift als Attribute anhaften – wie insbesondere die Kundmachung, das Inkraft- und Außerkrafttreten – und die zeitliche Ausdehnung der Norm, ihren Lebenszyklus, bestimmen. Diese Eigenschaften sind den Normen gewissermaßen auf einer Metaebene zuzurechnen, sind jedoch nicht Teil des Rechtstexts selbst.

Im Gegensatz dazu werden als interne Zeit solche Zeitangaben verstanden, die direkt im Rechtstext der jeweiligen Bestimmung enthalten sind. In diesen Fällen kann die zeitliche Dimension nicht von der Norm getrennt werden. Die Behandlung von internen Zeitangaben ist daher dem zweiten Schritt vorbehalten, in dem die eigentliche rechtliche Beurteilung stattfindet.

Dies scheint der Struktur des Gesetzes im Hinblick auf eine isomorphe Darstellung der Normen besser zu entsprechen. Es wäre wenig sinnvoll, eine Norm bloß aufgrund ihres temporalen Inhalts bei der eigentlichen rechtlichen Beurteilung zu ignorieren und deren Konklusion als gegeben hinzunehmen. Die Nachvollziehbarkeit der (automatisierten) Entscheidung wäre erheblich beeinträchtigt.

Die prinzipielle Trennung der Behandlung von zeitlichen Aspekten in Schritt Eins und der eigentlichen rechtlichen Beurteilung in Schritt Zwei schafft die Voraussetzungen, um beide Problemkreise separat voneinander behandeln zu können. Damit kann für jeden Problemkreis ein eigener Algorithmus verwendet werden. Technisch wird jeder der beiden Schritte in einem eigenen Modul mithilfe einer definierten Schnittstelle abstrahiert.

13.4. Das temporale Modell

13.4.1. Überblick

In diesem Unterkapitel wird ein temporales Modell zur Versionierung von Gesetzen und Normen beschrieben. Das Modell abstrahiert dabei von der konkreten Repräsentationsform der Norm, sie wird als Objekt mit temporalen Eigenschaften verstanden. Das Versionierungsmodell im Detail wird in Abschnitt 13.4.4. dargestellt.

Neben der grundlegenden Repräsentation der einzelnen Fassungen eines Gesetzes ist das Modell in der Lage, die auf einen Fall anwendbaren Normen zu ermitteln. Diesem zweiten Aspekt widmet sich Abschnitt 13.4.5.

13.4.2. Versionierung legislativer Ressourcen

In diesem Abschnitt wird ein Modell, das in der Lage ist, mit Änderungen von legislativen Ressourcen umzugehen, beschrieben.

Grundsätzlich gibt es drei Möglichkeiten, die dem Gesetzgeber offen stehen, um den Aufwand und damit die Kosten, die mit legislativem Änderungsmanagement verbunden sind, zu reduzieren:

1. Kodifizierung: Sämtliche Novellen einer Rechtsvorschrift werden in einem Gesetzesbuch zusammengeführt.
2. Konsolidierung: Verfolgt wie die Kodifizierung den Zweck, die Textmenge zu reduzieren, die konsolidierte Fassung ist jedoch im Gegensatz zur Kodifizierung nicht authentisch und damit nicht rechtlich verbindlich.
3. Neuerlassung: Anstatt das bestehende Gesetz zu verändern, wird ein neues Gesetz erlassen, das sämtliche Novellen zusammenführt.

Welcher Ansatz auch immer vom Gesetzgeber gewählt wird, es bedarf stets einer Methode, um verschiedene Versionen legislativer Ressourcen zu unterscheiden und um Änderungen zu bewältigen.

Grundsätzlich gibt es verschiedene Vorgehensweisen, um mit legislativen Änderungen umzugehen (Palmirani, 2011, S. 103 ff.). Das hier vorgestellte Modell basiert auf der „direkten Methode“.

Eine Novelle – das modifizierende Dokument – wird direkt auf das bestehende Gesetz angewendet, um eine modifizierte und konsolidierte Version des Gesetzes zu erzeugen.

Die „direkte Methode“ zur Bewältigung normativer Änderungen kann auf ein solides rechtstheoretisches Fundament gestützt werden. Die theoretische Grundlage ist die Annahme, dass das Rechtssystem durch amtliche Dokumente verändert wird, die von einer dazu ermächtigten Institution erlassen werden. Diese Methode kommt typischerweise nach dem legislativen Verfahren zur Rechtserzeugung zum Einsatz, um die konsolidierte Version eines Dokuments zu einem bestimmten Zeitpunkt zu erzeugen. Sie ermöglicht eine Point-in-Time-Versionierung eines Gesetzes nach der offiziellen Kundmachung der Novelle.

Jede Änderung ist die Konsequenz einer Ursache, wie beispielsweise eines authentischen Dokuments, eines Rechtsakts etc., und jede Modifikation soll in Übereinstimmung mit dem Grundsatz *lex posterior derogat legi priori* erfolgen.

Die konsolidierte Version des Gesetzes ist nicht immer authentisch (wie in Österreich die konsolidierte Version des Bundesrechts) und hat daher keine rechtsverbindliche Wirkung. Es gibt jedoch auch Gesetzgeber, die neben der Novelle auch eine authentische konsolidierte Fassung des Gesetzes erlassen. Letzteres ist beispielsweise im Land Niederösterreich der Fall.

13.4.3. Methoden der computergestützten Versionierung von Normen

Die Gründe für die Änderungen von Rechtstexten sind vielfältig. In der Regel erfolgen Anpassungen von Gesetzen durch den Gesetzgeber selbst, indem dieser eine Novellierung vornimmt. Wie im vorherigen Abschnitt dargelegt wurde, stehen dem Gesetzgeber mehrere Varianten offen, um die Novellierung von Rechtstexten durchzuführen. Sämtlichen Methoden ist gemein, dass eine alte und eine neue Fassung sowie ein modifizierendes Dokument vorhanden sind. Sie unterscheiden sich jedoch in der zeitlichen Reihenfolge und der Art, wie diese Dokumente entstehen.

Das in diesem Abschnitt diskutierte computergestützte Modell orientiert sich an der weiter oben beschriebenen direkten Methode zur Versionierung von gesetzlichen Änderungen. Dies ist eine Designentscheidung, da grundsätzlich jeder beliebige Ansatz gewählt hätte werden können. Wie oben ausgeführt wurde, lässt sich die direkte Methode auf eine solide rechtstheoretische Grundlage stützen, womit ihr der Vorzug zu geben war.

Um Novellen in die Wissensbasis eines Expertensystems einzupflegen, kommen grundsätzlich zwei Methoden in Frage:

(1) Inkrementelle Versionierung: Jede Novelle (das modifizierende Dokument) wird in eine formale Repräsentation überführt. Durch Anwendung der formalen Repräsentation der Novelle auf die Stammfassung des Gesetzes entsteht die novellierte Version.

Logisch kann dieser Zusammenhang wie folgt ausgedrückt werden: $G_{n+1} = N_{n+1}(G_n)$. G_n repräsentiert eine bestimmte Version des Gesetzes zum Zeitpunkt n. Eine Novelle N zum Zeitpunkt $n+1$ wird als Funktion beschrieben, die auf die aktuelle Version des Gesetzes zum Zeitpunkt n angewendet wird. Die neue und konsolidierte Version des Gesetzes, die die modifizierten Normen enthält, wird automatisiert durch Anwendung der Bestimmungen der Novelle erstellt.

Die Rechtsordnung R kann als eine Menge der Stammfassung und deren Novellen beschrieben werden: $R = \{G_1, N_2(G_1), N_3\big(N_2(G_1)\big), \dots, N_n(N_{n-1}(G_1))$.

Der inkrementelle Ansatz hat den Nachteil der höheren Komplexität, da die neue Version des Gesetzes automatisiert durch das Expertensystem erstellt wird.

(2) Absolute Versionierung: Bei diesem Ansatz arbeitet das temporale Modell mit der formalen Repräsentation der konsolidierten Fassungen des Gesetzes. Neben der Stammfassung wird die konsolidierte Fassung des Gesetzes, die unter Anwendung der Modifikationen entsteht, in der Wissensbasis gespeichert. So entstehen mehrere zeitlich linear geordnete Versionen des Gesetzes. Die Konsolidierung der Stammfassung und der Novelle liegt dabei nicht in der Zuständigkeit des Systems, sondern wird beispielsweise von einem juristischen Experten im Vorfeld durchgeführt.

Im Gegensatz zur inkrementellen Versionierung ist die Anwendung der Novelle als logische Funktion auf die aktuelle Fassung nicht notwendig. Die Rechtsordnung kann daher recht simpel als eine Menge der Stammfassung und der konsolidierten Versionen beschrieben werden: $R = \{G_1, G_2, \dots, G_n\}$.

Das in diesem Abschnitt beschriebene Modell basiert auf der absoluten Versionierungsmethode. Es wird davon ausgegangen, dass jede Änderung des Rechtssystems eine konsolidierte Version des Gesetzes erzeugt. Die konsolidierte Version ist insofern

„vollständig“, da sie die Normen der alten Fassung und der Novelle zusammenführt. Jede Version wird durch das Datum des Inkraft- und Außerkrafttretens begrenzt.

13.4.4. Versionierungsmodell

13.4.4.1. Theoretische Grundlage

Der direkten Methode zur Versionierung legislativer Ressourcen und dem Paradigma der absoluten Versionierung folgend wird davon ausgegangen, dass jede Novelle eine neue Fassung des Gesetzes – das heißt eine neue Version in der Terminologie des Modells – erzeugt, die sämtliche Normen enthält.

Die novellierte Fassung setzt alle Normen der vorherigen Fassung außer Kraft. Dies mag aus rechtstheoretischer Sicht nicht vollständig überzeugen, da eine Novelle grundsätzlich nur jenen Bestimmungen (materiell) derogiert, die von Änderungen betroffen sind. Keine derartigen Bedenken bestehen jedoch, wenn die konsolidierte Fassung mit sämtlichen Normen authentisch vom Gesetzgeber erlassen wird. In letzterem Fall derogieren die neuen Bestimmungen (zumindest materiell) den älteren.

Die hier getroffene Annahme, dass die novellierte Fassung eines Gesetzes stets sämtliche Normen enthält, führt zu einer Vereinfachung des computergestützten Modells. Ansonsten müsste nämlich ermittelt werden, welche Normen im Einzelnen von Änderungen betroffen sind und daher außer Kraft treten.

Für weitere Details hinsichtlich der formalen Semantik der Versionierung und von retroaktiven Modifikationen sei auf die folgenden Abschnitte in diesem Kapitel verwiesen.

Die theoretische Grundlage des hier dargestellten Modells orientiert sich an der Theorie zur Versionierung legislativer Dokumente. Die vorhandenen Ansätze werden, falls notwendig, erweitert, um die Versionierung von Normen, repräsentiert von formalen Regeln, zu bewerkstelligen.

Um mit Änderungen des Rechts umgehen zu können, müssen zwei Aspekte berücksichtigt werden: (1) Eine solide Namenskonvention zur Benennung von Gesetzen und Regeln, und (2) ein Versionierungsmodell, das die temporalen Dimensionen des Rechts formalisiert.

Das in diesem Kapitel dargestellte Modell ist an den FRBR-Standard (Saur, 2009) für bibliographische Ressourcen angelehnt und verwendet eine vereinfachte Version der HTTP-

basierten Syntax für IRIs von Akoma Ntoso (Francesconi, 2011), kompatibel mit CEN MetaLex (Boer & van Engers, 2011). Ein Gesetz wird abstrakt als „Werk“ und jede Fassung als konkrete „Expression“ qualifiziert, dazu mehr in Abschnitt 13.4.4.3.

13.4.4.2. Repräsentation der Zeit

Das formale Modell muss neben der Versionierung von Normen auch in der Lage sein zu ermitteln, welche Normen zu einem bestimmten Zeitpunkt in Kraft stehen und auf einen Fall anwendbar sind. Abstrakt müssen die Beziehungen zwischen Zeitpunkten und Intervallen (dem Zeitraum des Inkraftstehens etc.) hergestellt werden können. Das gegenständliche Modell muss damit sowohl mit Zeitpunkten als auch Intervallen umgehen können. Ein Intervall wird dabei grundsätzlich durch einen Anfangs- und Endzeitpunkt oder eine bestimmte Dauer begrenzt.

Die Zeit wird als diskrete lineare Ordnung von Zeitpunkten verstanden. Vorweggenommen sei, dass es jedoch nicht darum geht, eine vollständige temporale Logik zu entwickeln.

Ein temporales Modell kann grundsätzlich auf Intervallen oder Zeitpunkten basieren. Diese Unterscheidung betrifft die zugrundeliegende atomare Struktur, die zur Repräsentation der Zeit verwendet wird. Die beiden Repräsentationsformen sind jedoch miteinander kompatibel, sodass ein Modell beide Strukturen kombinieren kann, um zeitliche Aspekte zu repräsentieren. Für das gegenständliche Modell wird ein solch „hybrider“ Ansatz gewählt.

13.4.4.3. Benennung und Identifikation legislativer Ressourcen

Das computergestützte Modell ist in der Lage, Gesetze und Normen in Form von Regeln zu repräsentieren. Dabei wird auf die zeitliche Dimension des Rechts Bedacht genommen, sodass das Modell mit mehreren Versionen einer Norm umgehen kann.

Damit eine Norm in einer bestimmten Fassung referenziert werden kann und auch statische bzw. dynamische Verweisungen zwischen Gesetzen aufgelöst werden können, bedarf es einer Namenskonvention mit einem soliden Mechanismus, um legislative Ressourcen zu identifizieren und zu adressieren. Einmal vergebene Bezeichnungen und Verweise müssen im Lauf der Zeit konstant bleiben und resistent gegen Änderungen der Rechtsordnung sein. Nur so kann gewährleistet werden, dass Verweise im Modell jederzeit korrekt aufgelöst werden können.

Die Architektur des Modells orientiert sich an den „Functional Requirements for Bibliographic Records“ (FRBR). Dies gewährleistet die Kompatibilität mit den Standards CEN MetaLex (Boer & van Engers, 2011) und Akoma Ntoso (Palmirani & Vitali, 2011) zur Beschreibung von legislativen Dokumenten, da letztere ebenfalls auf dem FRBR-Modell aufbauen.

Im Folgenden werden die wichtigsten Begriffe des FRBR-Standards (Saur, 2009) im Überblick dargestellt. Die Begrifflichkeit der FRBR-Ontologie beschreibt unterschiedliche Abstraktionsebenen eines bibliographischen Werks, die auch von CEN MetaLex (und Akoma Ntoso) verwendet werden, um juristische Dokumente (Gesetze, Verordnungen, aber auch Urteile) zu beschreiben (Francesconi, 2011, S. 50 ff.):

- Work: Ein abstrakter Begriff, der ein bibliographisches Werk, in diesem Fall eine Rechtsquelle (Gesetz, Verordnung etc.), beschreibt, z.B.: Kriegsopferversorgungsgesetz 1957, BGBl. Nr. 152/1957.
- Expression: Jede Version eines Werks (Work), dessen Inhalt bestimmt ist und sich von anderen aus beliebigem Grund, wie Sprache, Version etc. unterscheidet. Als „Expression“ ist unter anderem die novellierte Fassung eines Gesetzes zu verstehen, z.B.: Kriegsopferversorgungsgesetzes 1957, BGBl. Nr. 152/1957 in der Fassung BGBl. I Nr. 81/2013.
- Manifestation: Jede elektronische oder physikalische Erscheinungsform einer Expression, wie Microsoft Word, OpenOffice, XML, TIFF, PDF etc. Beispielsweise eine PDF-Datei der Stammfassung des Kriegsopferversorgungsgesetzes 1957.
- Item: Eine bestimmte physikalische Kopie einer Manifestation in Form einer Datei, oder ein bestimmtes Buch im Regal einer Bibliothek etc. Beispielsweise eine PDF-Datei mit dem Namen kovg1957.pdf auf dem Computer des Autors, die das Kriegsopferversorgungsgesetz in der Fassung vom 1.6.2013 enthält. Eine Kopie der Datei erzeugt ein neues Item, jedoch handelt es sich um dieselbe Manifestation.

Alle Dokumente (ein Gesetz, eine Verordnung etc.) auf jeder Ebene können aus mehreren Unterelementen bestehen, die einzelne Teile (Paragrafen, Sätze etc.) repräsentieren. Die Summe aller Komponenten formt ein Dokument so, wie es sich für den Leser darstellt. Auch die formale Repräsentation der Normen durch Regeln folgt dieser Struktur.

Der Umgang mit Änderungen legislativer Ressourcen erfordert eine solide Namenskonvention zur Benennung von Ressourcen und deren Versionen. Eine einmal vergebene Bezeichnung muss im Lauf der Zeit konstant bleiben, sodass die adressierte Ressource – beispielsweise ein Dokument – jederzeit aufgefunden werden kann. Frühere Entscheidungen basieren auf einer älteren Rechtslage, sodass die darin enthaltenen Referenzen selbstverständlich auf diese und nicht die aktuelle Fassung verweisen müssen.

Für die Adressierung von Ressourcen der jeweiligen Ebene (Work, Expression, Manifestation, Item) kommen Internationalized Resource Identifier (IRIs) – die internationalisierte Form der Uniform Resource Identifier (URIs) – zum Einsatz. IRIs kommen auch im Semantic Web und in Ontologien zum Einsatz, sodass die Integration des Systems mit diesen Technologien möglich ist.

Die verwendete Namenskonvention orientiert sich an der in Akoma Ntoso zum Einsatz kommenden Syntax für Bezeichner, die auf dem HTTP-Protokoll bzw. dessen Syntax für URIs basiert. Damit können Dokumente, Normen und deren formale Repräsentation mit vorhandenen Mechanismen zur Adressierung von Ressourcen im Web adressiert werden. Für die Benennung von Ressourcen im computergestützten Modell kann somit eine einheitliche Methode verwendet werden.

In Gesetzestexten finden sich in Form von dynamischen Verweisungen oftmals Verweise auf ein Gesetz (ein Werk in der Terminologie des Modells) in seiner jeweils geltenden Fassung und nicht auf eine bestimmte Version desselben. Umgekehrt sind jedoch auch statische Verweisungen auf eine konkrete Fassung eines Gesetzes üblich.

Das im Folgenden dargestellte Modell zur Benennung von Ressourcen und Verweisen kann beide Varianten abbilden. Dabei ist das konkrete Format der jeweiligen Expression (XML, Objekte im Speicher etc.), also die Manifestation der Expression, in diesem Kontext eher nebensächlich. Vorwiegend muss mit IRIs auf dem Work- und Expressionlevel umgegangen werden.

Die beschriebene Namenskonvention basiert auf der in Akoma Ntoso verwendeten, am HTTP-Protokoll orientierten Syntax (Francesconi, 2011, S. 63 ff.), wurde aber für die gegenständlichen Zwecke vereinfacht. Beispielsweise ist es nicht notwendig in der IRI die Sprache der legislativen Ressource zu unterscheiden, sodass dieser Teil weggelassen wird.

Die IRI eines Werks – im Folgenden genannt „Worklevel-IRI“ – bildet die Grundlage zur Adressierung von Ressourcen auf dem Expressionlevel. Eine Worklevel-IRI besteht aus folgenden Komponenten:

- `<base-url>`: Basis URL der namensgebenden Autorität, die in der Lage ist, eine IRI aufzulösen. Dieser Teil fehlt bei relativen IRIs und ist für die Namenskonvention nicht relevant.
- `<doc-type>`: Die Art des Dokuments, z.B. „act“ für ein Gesetz.
- `<creation-date>`: Das Erzeugungsdatum des Dokuments, ausgedrückt im Format `yyyy-mm-dd` oder nur `yyyy`, wenn das Jahr zur Identifikation ausreichend ist.
- `<title>`: Nummer oder Titel, die zur eindeutigen Identifizierung des Werks verwendet werden.

Sämtliche Komponenten werden durch einen Schrägstrich getrennt, sodass relative Verweise möglich sind. Eine Worklevel-IRI, die auf das KOVG 1957 verweist, könnte beispielsweise so aussehen: `/act/1957/KOVG`. Im Beispiel wurde auf die Angabe der `<base-url>` verzichtet, diese könnte z.B. http://ris.bka.gv.at lauten.

In der Regel wird auf eine bestimmte Fassung eines Gesetzes bzw. auf einzelne Paragrafen verwiesen. Solche Referenzen finden im Modell auf dem Expressionlevel statt. Eine Expressionlevel-IRI besteht aus folgenden Komponenten:

- Die Worklevel-IRI (siehe oben).
- Das „@“-Zeichen: Dieses Zeichen ist erforderlich und kennzeichnet eine IRI auf dem Expressionlevel.
- Versionsangabe im folgenden Format: `<type>=<version>;<view-date>`. Die Angabe des Versionsbezeichners sowie jedes einzelnen Teils ist optional. Werden die Version als Ganzes oder einzelne Teile davon nicht angegeben, kommen Standardwerte zum Einsatz.
- Inhaltsspezifische Angaben, die einen bestimmten Teil einer Expression referenzieren, z.B. „par6“ für § 6.

Wie bei einer Worklevel-IRI werden sämtliche Komponenten der Expressionlevel-IRI durch einen Schrägstrich getrennt. Das „@-Zeichen“ kennzeichnet eine IRI auf dem Expressionlevel, danach können Angaben zur Version und zum gewünschten Inhalt folgen.

Der Aufbau des Versionsbezeichners ist in diesem Kontext von herausragender Bedeutung. Wie bereits angemerkt wurde, ist die Angabe eines Versionsbezeichners sowie der einzelnen Teile optional.

Fehlt die Version ganz, so wird standardmäßig die Fassung des Gesetzes ermittelt, die zum Zeitpunkt der Auflösung der IRI (dem aktuellen Datum) existiert und in Kraft steht. Eine Norm steht „in Kraft“, wenn sie zum jeweiligen Datum „anwendbar“ ist, im Sinne des Abschnitts 13.4.5.3. Das Attribut `<version>` entspricht dabei dem maßgeblichen Zeitpunkt und `<view-date>` dem Beobachtungszeitpunkt in der Terminologie des verwiesenen Abschnitts.

Das Attribut `<type>` gibt an, wie `<version>` zu interpretieren ist. Dabei sind zwei Werte möglich, nämlich „force“, womit das Inkraftstehen, und „promulgation“, womit das Kundmachungsdatum gemeint ist. Mithilfe des Attributs `<type>` lässt sich somit eine bestimmte Fassung über das offizielle Kundmachungsdatum oder das Datum des Inkraftstehens referenzieren. Dies erlaubt sehr flexible Abfragen, die zur Repräsentation von statischen und dynamischen Verweisungen notwendig sind, vgl. Abschnitt 13.4.5.5. Wird kein Typ für das Versionsdatum angegeben, so wird dieses als „Datum des Inkraftstehens“ („force“) interpretiert.

Der gewünschte Zeitpunkt kann mithilfe des Attributs `<version>` im Format `yyyy-mm-dd` spezifiziert werden. Wird das Attribut nicht angegeben, so wird das aktuelle Tagesdatum verwendet. Wenn der Typ des Datum „promulgation“ ist, wird eine Norm selektiert, wenn das Datum der Kundmachung kleiner gleich dem Versionsdatum ist und letzteres kleiner als das Datum des Außerkrafttretens. Ist der Typ hingegen „force“ so gilt, dass eine Norm selektiert wird, wenn diese zum gegebenen Versionsdatum in Kraft steht und „anwendbar“ im Sinne des Abschnitts 13.4.5.3. ist.

Zusätzlich zu den Attributen `<type>` und `<version>,` aber auch ohne diese, kann mithilfe von `<view-date>` der Beobachtungszeitpunkt der Rechtsordnung im Format `yyyy-mm-dd` angegeben werden. Das Attribut wird immer von einem Semikolon eingeleitet, damit dieses von der Angabe des Versionsdatums unterschieden werden kann. Das Attribut `<view-date>` wird benötigt, um die Rechtslage vor und nach einer rückwirkenden Modifikation zu referenzieren. Wird beispielsweise die Fassung des Gesetzes, die am 1. Jänner 2012 in Kraft steht, am 30. Oktober 2014 rückwirkend modifiziert, so macht es einen Unterschied, ob die

Abfrage vor oder nach dem 30. Oktober ausgeführt wird. Wie beim Versionsattribut wird das aktuelle Datum verwendet, wenn der Beobachtungszeitpunkt nicht explizit angegeben wird.

Beispielsweise zeigt die Expressionlevel-IRI `/act/1957/KOVG@force=2014-07-01;2014-10-09` auf das Kriegsopferversorgungsgesetz in der Fassung in Kraft am 7. Juli 2014, wobei als Beobachtungszeitpunkt der Rechtsordnung der 9. Oktober 2014 gewählt wird.

Der letzte Teil der Expressionlevel-IRI ist die Angabe des Pfades zu einer Komponente der Expression. Ein (legislatives) Dokument besteht grundsätzlich aus Abschnitten, Paragrafen, Nummern und Sätzen etc. Diese Teile werden durch die Komponenten der Expression abstrahiert und mit einer hierarchischen Pfadangabe adressiert.

Das obige Beispiel einer IRI kann erweitert werden, um § 6 aus der gewählten Fassung des Kriegsopferversorgungsgesetzes zu selektieren: `/act/1957/KOVG@force=2014-07-01;2014-10-09/par6`.

13.4.4.4. Temporale Dimensionen

In Übereinstimmung mit der Rechtstheorie unterscheidet das Modell folgende temporale Dimensionen von Normen (Palmirani et al., 2010; Palmirani, Governatori, & Contissa, 2011):

- Existenz: Die Periode, in der die Norm Teil des Rechtsystems ist, beginnend mit dem Datum der Publikation (in einem offiziellen Journal) und endend durch einen späteren normativen Akt. Im Fall von Bundesrecht ist dies die Kundmachung im Bundesgesetzblatt. Nach ordnungsgemäßer Kundmachung ist die Norm Teil der Rechtsordnung.
- Inkraftstehen: Der Zeitraum, begrenzt durch Beginn- und Enddatum, in dem die Norm in Kraft steht und daher „vollständig“ Teil der Rechtsordnung ist. Dieser Zeitraum beginnt üblicherweise nach dem Tag der Publikation, kann jedoch durch Legisvakanz verzögert werden.
- Effektivität (Wirksamkeit): Der Zeitraum, begrenzt durch Beginn- und Enddatum, in dem die Norm operativ ist und der Richter die Norm grundsätzlich anwenden darf. Der Sachverhalt muss sich innerhalb dieses Zeitraums ereignen, damit die Norm grundsätzlich anwendbar ist. Eine Norm, die sistiert wird, ist weiterhin Teil der Rechtsordnung, jedoch (vorübergehend) nicht effektiv.

- Anwendbarkeit: Das Intervall der Anwendbarkeit einer Norm ist jener Zeitraum, in dem im Einzelfall die Norm angewendet wird und damit ihre Rechtsfolgen entfaltet. Diese Dimension bezieht sich auf die Rechtsanwendung durch den Richter, wenn dieser die abstrakten Rechtsvorschriften auf den konkreten Fall anwendet.

Die Existenz (Geltung) einer Norm beginnt mit dem Datum der Kundmachung (`publication`) und endet mit ihrer Aufhebung durch einen späteren Normsetzungsakt. Der Zeitraum, in dem eine Norm existiert, wird in der Terminologie des Modells passend mit „Existenzzeitraum“ bezeichnet.

Vom Kundmachungsdatum ist das Inkrafttreten zu unterscheiden. Grundsätzlich treten Rechtsvorschriften mit Ablauf des Tages ihrer Kundmachung in Kraft. Es kann jedoch eine längere Periode der Legisvakanz vorgesehen oder eine Rechtsvorschrift rückwirkend erlassen werden.

Der Zeitraum, in dem ein Gesetz in Kraft steht, wird zeitlicher Geltungsbereich genannt. Dieser ist vom Zeitpunkt der Existenz (Geltung) und dem Existenzzeitraum zu unterscheiden. Der zeitliche Geltungsbereich beginnt mit dem Inkrafttreten und endet mit dem Außerkrafttreten der Norm. Durch die Verwendung der Attribute `startForce` und `endForce` lässt sich das Intervall begrenzen und kann ermittelt werden, welche Rechtsvorschriften zu einem bestimmten Zeitpunkt in Kraft stehen.

Es existieren jedoch auch Gesetze, die vor und nach ihrem Inkrafttreten Rechtswirkungen erzeugen und daher effektiv sind. Um auch solche atypische Fälle zu erfassen, werden zwei weitere temporale Dimensionen benötigt, nämlich der Zeitraum der Effektivität und der Zeitraum der Anwendbarkeit des Gesetzes. Diese beiden Intervalle werden durch die Attribute `startEfficacy` und `endEfficacy` bzw. `startApplicability` und `endApplicability` beschrieben.

In vielen Fällen decken sich die Zeiträume des Inkraftstehens, der Effektivität und der Anwendbarkeit. Wird für ein Gesetz bloß der Zeitraum des Inkraftstehens angegeben, so gilt, dass Effektivität und Anwendbarkeit exakt diesem Intervall entsprechen. Das ist jedoch nicht zwingend der Fall, sodass die Intervalle der Effektivität und Anwendbarkeit auch abweichend spezifiziert werden können. Sofern Effektivität und Anwendbarkeit vom Zeitraum des Inkraftstehens abweichen, müssen diese Dimensionen bei der Berechnung des Gültigkeitszeitraums berücksichtigt werden.

Die Terminologie des Modells ist an der englischsprachigen Literatur und den dort entwickelten theoretischen Modellen orientiert. Die Intervalle Effektivität (engl. efficacy) und Anwendbarkeit (engl. applicability) entsprechen in ihrer Semantik dem Bedingungs- bzw. Rechtsfolgenbereich der deutschsprachigen rechtstheoretischen Literatur. Auf Abweichungen sei jedoch hingewiesen. So wird gelegentlich mit „efficacy“ recht weit jener Zeitraum bezeichnet, in dem die Norm Rechtswirkungen entfaltet, wobei damit sowohl der Bedingungs- als auch der Rechtsfolgenbereich gemeint ist.

13.4.4.5. Vererbung der temporalen Dimensionen

Dem temporalen Modell liegt eine zweischichtige hierarchische Struktur zugrunde, die das Verhältnis zwischen dem Gesetz und den zugehörigen Normen wiedergibt. Ein Gesetz wird als eine Menge von Normen betrachtet und besitzt dieselben temporalen Dimensionen wie letztere.

Ein Gesetz[194] hat demnach ebenso eine dreidimensionale temporale Ausdehnung. Es gilt damit die Regel, dass der Zeitraum des Inkraftstehens mit der Effektivität und Anwendbarkeit der Norm ident ist, sofern für letztere keine abweichenden Angaben gemacht werden.

Hinter dem formalen Modell steht ein Vererbungsmechanismus. Das bedeutet, dass die temporalen Dimensionen des Gesetzes an die Normen „vererbt“ werden, sofern diese keine spezielleren temporalen Eigenschaften besitzen.

Der Vererbungsmechanismus trägt dem Umstand Rechnung, dass in Gesetzen oftmals Normen vorkommen, die von den anderen Bestimmungen abweichende zeitliche Eigenschaften besitzen, beispielsweise später in Kraft treten.

Neben der Vereinfachung des formalen Modells wird damit der legistischen Übung entsprochen, das allgemeine Inkrafttreten des Gesetzes in einer Schlussbestimmung zu regeln und bei Bedarf Sonderregelungen für einzelne Paragrafen vorzusehen.

---

[194] Der Begriff „Gesetz“ steht hier für eine beliebige Rechtsquelle, wie beispielsweise eine Verordnung oder ein Vertrag.

13.4.4.6. Versionierung von Gesetzen und Normen

Das im Folgenden beschriebene Versionierungsmodell baut auf den in den vorherigen Unterkapiteln beschriebenen temporalen Dimensionen auf und erweitert den von Palmirani und Brighi (2006) beschriebenen Ansatz, um nicht bloß mit Dokumenten, sondern formalen Regeln umgehen zu können.

Das Modell basiert auf dem Axiom, dass jede Fassung des Gesetzes zumindest ein Datum des Inkrafttretens aufweist. Für sämtliche Versionen[195] bis auf die letzte – das ist die aktuelle Fassung – gilt, dass diese auch ein Datum des Außerkrafttretens besitzen. So lässt sich ermitteln, in welchem Zeitraum ein Gesetz (mit allen seinen Versionen) insgesamt in Kraft steht bzw. stand. Für die Effektivität und Anwendbarkeit gilt hingegen, dass die Angabe des Beginn- und Enddatums nicht obligatorisch ist.

Für die eindeutige Kennzeichnung jeder Version wird intern eine eindeutige Identifikationsnummer vergeben, z.B.: v1, v2 usw. Jede Version wird überdies durch das Publikationsdatum gekennzeichnet. Das Publikationsdatum ist das offizielle Datum der Kundmachung der Stammfassung bzw. der Novelle (des konsolidierten Rechtstexts). Wenn in weiterer Folge von „offiziellem Versionsdatum“ oder „Datum des Dokuments“ gesprochen wird, dann ist damit das Publikationsdatum gemeint, sofern nicht explizit anders angegeben. Wird daher von einem Dokument in der Fassung vom 12.12.2014 gesprochen, so ist damit die letzte Version gemeint, die vor oder am 12. Dezember kundgemacht wurde.

Mithilfe des Publikationsdatums können die Versionen eines Gesetzes linear in aufsteigender Reihenfolge geordnet werden. Die erste Version ist dabei stets die Stammfassung eines Gesetzes. Alle nachfolgenden Fassungen werden nach dem Datum der Kundmachung des modifizierenden Dokuments angeordnet. Damit kann das Modell die zu einem beliebigen Zeitpunkt existierende Fassung eines Gesetzes ermitteln.

Die Veränderung einer Norm geschieht zu dem Zeitpunkt, an dem die modifizierende Bestimmung auf das Gesetz angewendet wird, und so ihre Effekte auf den Rechtstext wirken. Oftmals ist das Datum des Inkrafttretens der Novelle identisch mit dem Datum der Änderung des Rechtstexts. Abweichungen sind jedoch möglich, beispielsweise wenn die

[195] Unter einer „Version“ wird die formale Repräsentation einer bestimmten Fassung eines Gesetzes verstanden.

Novelle zwar in Kraft tritt, deren Anwendbarkeit jedoch von einem Ereignis abhängig gemacht wird oder die Modifikation retroaktiv auf eine frühere Fassung wirkt.

Dem temporalen Modell liegt die Annahme zugrunde, dass die novellierte Fassung eines Gesetzes grundsätzlich zu jenem Zeitpunkt in Kraft tritt, an dem die Novelle ihre Rechtswirkungen auf die ursprüngliche Fassung entfaltet. Die Versionierung wird demnach vom Datum der Anwendung der Novelle auf den Rechtstext angetrieben. Von dieser Annahme gibt es jedoch auch Ausnahmen, insbesondere wenn ein Gesetz, bevor dieses in Kraft tritt, von einer Novelle modifiziert wird. In diesem Fall wird das Gesetz zwar modifiziert, tritt jedoch nicht sogleich zu diesem Zeitpunkt in Kraft.

Die Fassungen eines Gesetzes werden anhand des Datums des Inkrafttretens (`startForce`) linear in sogenannten „Versionierungsketten“ angeordnet und im Modell durch das Interface `VersioningChain` repräsentiert. Wird eine neue Fassung kundgemacht, so wird diese am Ende der Kette eingefügt. Dadurch entsteht eine sogenannte „Versionierungskette“ (`VersioningChain`), die sämtliche Versionen eines Gesetzes im Lauf der Zeit enthält. Dieser Ansatz erlaubt es, Konflikte zwischen Normen anhand der temporalen Attribute zu lösen, indem der Grundsatz *lex posterior derogat legi priori* angewendet wird, und steht damit in Einklang mit der Rechtstheorie. Dem Modell liegt implizit die Annahme zugrunde, dass sich der Zeitraum des Inkraftstehens zweier Fassungen niemals überlappt. Dies gewährleistet die Richtigkeit der linear geordneten Versionierungsketten.

13.4.4.7. Retroaktive Modifikationen

Manche Änderungen der Rechtsordnung verändern den Effekt einer Norm grundlegend. Zu solchen Änderungen gehören unter anderem:

- Rückwirkende Modifikationen des Rechtstexts.
- Rückwirkende Aufhebungen von Normen (z.B. durch den Verfassungsgerichtshof).
- Authentische Interpretationen durch den Gesetzgeber.

Retroaktive Modifikationen verändern die Rechtslage zu einem Zeitpunkt, der vor dem Beginn der Existenz des modifizierenden Rechtsakts (Novelle, Urteil etc.) liegt. Eine Novelle kann beispielsweise Auswirkungen auf eine frühere Gesetzeslage haben, obwohl sie (lange) nach dieser kundgemacht wird und in Kraft tritt. Die Rechtsordnung wird durch solche Änderungen demnach in der Vergangenheit verändert.

Es ist jedoch (zumindest nach dem heutigen Stand der Wissenschaft) physisch nicht möglich, die Vergangenheit zu ändern. Aus diesem Grund wird ein Mechanismus benötigt, der es erlaubt, die Rechtsordnung aus heutiger Sicht zu betrachten, aber so, als wäre die Änderung in der Vergangenheit geschehen.

Eine rückwirkende Änderung führt dazu, dass aus heutiger Sicht eine neue Fassung des Gesetzes in der Vergangenheit geschaffen wird, so als hätte die originale Fassung zum damaligen Zeitpunkt nie existiert. Derartige Neuerungen haben unter Umständen auch Auswirkungen auf nachfolgende Novellen eines Gesetzes, die dieselben Paragrafen betreffen wie die Rückwirkung.

Wie diese Ausführungen zeigen, macht es im Fall von retroaktiven Modifikationen einen Unterschied, zu welchem Zeitpunkt die Rechtsordnung betrachtet wird.

Den Ausführungen von Palmirani und Brighi (2006) folgend muss die Zeitlinie gewissermaßen in der Vergangenheit gespalten werden, um eine Rechtslage herzustellen, die zum damaligen Zeitpunkt ursprünglich nicht existiert hat. Um tiefgreifende Veränderungen des temporalen Modells im Fall von retroaktiven Modifikationen zu vermeiden, wird die dort vorgeschlagene Lösung erweitert und angepasst, sodass die Notwendigkeit der Teilung von Versionierungsketten entfällt.

Jede retroaktive Modifikation der Rechtslage erzeugt eine neue Versionierungskette, die durch das Publikationsdatum der Novelle identifiziert wird und zugleich das Datum beschreibt, ab dem die Kette gültig ist und damit die „aktuelle“ Rechtslage repräsentiert. Wenn eine retroaktive Modifikation stattfindet, wird die letzte Kette dupliziert, wobei die neue Kette die modifizierten Rechtsvorschriften enthält, beginnend mit dem Datum zu dem die retroaktiven Modifikationen durchgeführt wurden.

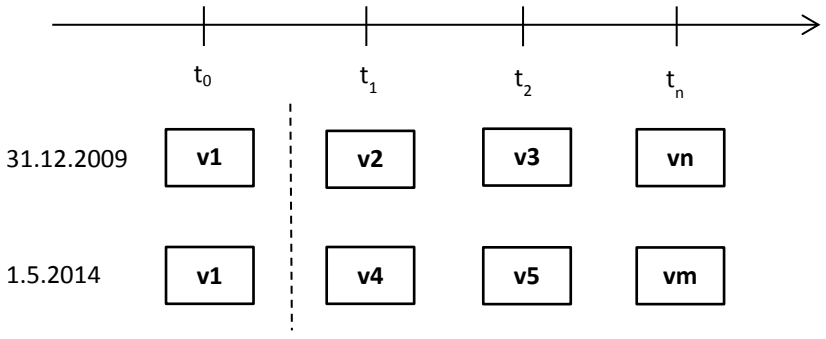

Abbildung 25: Beispiel einer retroaktiven Modifikation

Abbildung 25 zeigt das Beispiel einer Novelle, kundgemacht am 1.5.2014, die rückwirkend die Fassung v2 zum Zeitpunkt t_1 modifiziert und dadurch eine neue Kette erzeugt, die unverändert v1, gefolgt von den modifizierten Versionen v4 bis v5, enthält.

Im Beispiel existieren folgende Versionen vor der rückwirkenden Novelle:

- v1@1.1.2010: Stammfassung des Gesetzes, in Kraft ab 1.1.2010.
- v2@1.1.2011: Novellierte Fassung, in Kraft ab 1.1.2011.
- v3@1.1.2012: Novellierte Fassung, in Kraft ab 1.1.2012.

Am 1.5.2014 wird nun eine Novelle kundgemacht, die rückwirkend in Kraft tritt und die Fassung v2 modifiziert, sodass sich folgende Rechtslage ergibt:

- v1@1.1.2010: Unveränderte Stammfassung des Gesetzes, in Kraft ab 1.1.2010.
- v4@1.1.2011: Rückwirkend novellierte Fassung, in Kraft ab 1.1.2011.
- v5@1.1.2012: Rückwirkend novellierte Fassung, in Kraft ab 1.1.2012.

Die Zeitlinie wird gewissermaßen nach der Fassung v1 geteilt, die nicht von der Modifikation betroffen ist, wie von der strichlierten Linie angedeutet wird. Angenommen, die rückwirkende Modifikation betrifft § 10 der Fassung v2, so führt dies in weiterer Folge zu einer Anpassung der nachfolgenden Fassungen, die ebenfalls diesen Paragrafen enthalten. Damit entsteht die im Beispiel dargestellte Situation.

Die Versionierungsketten erlauben es der Inferenzmaschine, die Rechtsfrage abzufragen, wie sie vor dem und am (oder nach dem) 1. Mai 2014 beschaffen war, an dem die rückwirkende Novelle kundgemacht und Teil des Rechtssystems wurde. Ferner ist es möglich, auf die „aktuelle“ Rechtslage Bezug zu nehmen, indem eine entsprechende Variable der letzten Kette zugewiesen wird.

Wenn die aktuelle Kette aufgrund einer retroaktiven Modifikation außer Dienst gestellt werden muss, reicht es aus, wenn die Variable „current“ von nun an auf die neue Kette verweist, ohne dass der Rest des Modells davon betroffen ist. Im Gegensatz zum Modell von Palmirani und Brighi (2006) besteht keine Notwendigkeit, existierende Ketten zu teilen. Der Änderungsaufwand des formalen Modells bei retroaktiven Modifikationen bleibt auf ein Minimum beschränkt.

Der von Palmirani und Brighi (2006) gewählte Ansatz der Teilung der Ketten im Fall von retroaktiven Modifikationen hat den Vorteil, dass das Modell etwas anschaulicher wirkt. Das hier entwickelte Modell enthält jedoch alle Informationen, die notwendig sind, um eine Darstellung der Versionierungsketten mit Teilung zu generieren, sollte dies zwecks Anschaulichkeit erforderlich sein.

13.4.5. Die Ermittlung der anwendbaren Rechtsnormen

Basierend auf dem Versionierungsmodell und den Überlegungen, die in den vorhergehenden Abschnitten angestellt wurden, konnte eine temporale Inferenzmaschine entworfen werden, die in der Lage ist, die auf einen Fall zeitlich anwendbaren Normen zu ermitteln. Die Ermittlung der anwendbaren Normen ist der erste Schritt der automatisierten rechtlichen Beurteilung, bevor materiellrechtliche Aspekte des Falles behandelt werden. Der Algorithmus berücksichtigt die „externen" zeitlichen Dimensionen von Rechtsnormen, die deren Lebenszyklus bestimmen, und nicht die „internen" Zeitangaben, die im Rechtstext selbst ausgedrückt sind, z.B. ab welchem Datum Winterreifen verwendet werden müssen.

Bevor näher auf den bereits in Abschnitt 13.3.1. in Grundzügen dargestellten Algorithmus zur Ermittlung der auf einen Fall anwendbaren Normen eingegangen wird, sollen ein paar grundlegende Überlegungen angestellt werden.

Für die weiteren Ausführungen wird von der Struktur einer Verhaltensnorm, bestehend aus Tatbestand und Rechtsfolge, ausgegangen. Weiter wird angenommen, dass sämtliche Normen – unabhängig von deren Art – in eine formale Repräsentation, bestehend aus Antezedens und Konsequenz (Sukzedens), überführt werden können.

Ausgehend von dieser Struktur der Normen muss die Inferenzmaschine, um herauszufinden, welche Regeln auf einen Fall anwendbar sind, das Intervall der Effektivität und der Anwendbarkeit berücksichtigen. Dementsprechend muss das temporale Modell mit zwei Daten abgefragt werden: (1) Dem Beobachtungszeitpunkt der Rechtsordnung, und (2) dem „signifikanten" Datum, das verwendet wird, um die anwendbaren Regeln zu ermitteln.

Letzteres ist üblicherweise abhängig vom Inhalt und der Art des Rechts (formelles oder materielles Recht). Im Strafrecht ist beispielsweise das signifikante Datum jener Zeitpunkt, an dem das Verbrechen begangen wurde, ausschlaggebend für die anwendbare Rechtslage.

Diese überblicksmäßige Darstellung der wichtigsten Aspekte der Ermittlung der anwendbaren Rechtsnormen soll nun durch weitere Erläuterungen vertieft werden.

Die Effektivität einer Norm bezieht sich auf jenen Zeitraum, in dem die Norm operativ ist und vom Richter grundsätzlich verwendet werden darf. Damit ist jenes Intervall gemeint, in dem sich Sachverhalte verwirklichen müssen, auf die sich die Norm bezieht (Bedingungsbereich).

Von der Effektivität ist das Intervall der Anwendbarkeit zu unterscheiden. Das ist jener Zeitraum, in dem Rechtsfolgen an sich verwirklichende Sachverhalte geknüpft werden sollen (Rechtsfolgenbereich). Üblicherweise können Rechtsfolgen mit Beginn der Effektivität einer Norm verhängt werden. Dies ist jedoch nicht zwingend so. Die Anwendbarkeit einer Norm kann später beginnen, aber auch früher enden als ihre Effektivität.

Eine Norm ist zeitlich auf einen Fall anwendbar, wenn sich der Sachverhalt innerhalb des Zeitraums der Effektivität (Bedingungsbereich) verwirklicht hat und der Zeitpunkt der rechtlichen Beurteilung (das Urteil, der Beschluss etc.) innerhalb des Zeitraums der Anwendbarkeit (Rechtsfolgenbereich) liegt.

Diese beiden Intervalle sind oftmals identisch mit dem Zeitraum des Inkraftstehens der Norm, sodass sie im Gesetzestext nicht explizit geregelt werden. In solch einem Fall wird angenommen, dass sich die Intervalle Effektivität und Anwendbarkeit mit dem Zeitraum des Inkraftstehens decken. Mithilfe von Effektivität und Anwendbarkeit kann damit hinreichend festgestellt werden, ob eine Norm in zeitlicher Hinsicht auf einen Fall anwendbar ist[196].

Es gilt aber auch Besonderheiten zu berücksichtigen. Zu diesen gehören unter anderem rückwirkende Änderungen der Rechtslage oder Übergangsbestimmungen, die die zeitliche Anwendbarkeit einer Norm verzögern. Übergangsbestimmungen können auf konkrete Merkmale des Sachverhalts abstellen, beispielsweise wann das Verfahren bei der Behörde anhängig wurde. Letztere können verhindern, dass eine grundsätzlich anwendbare Norm im Einzelfall auch tatsächlich angewendet wird.

[196] Oftmals wird undifferenziert von der „Anwendbarkeit“ einer Norm gesprochen und damit gemeint, dass diese im Einzelfall Rechtswirkungen entfaltet. In diesem Abschnitt wird jedoch unter „Anwendbarkeit“ eine temporale Dimension verstanden, nämlich jener Zeitraum, in dem die Rechtsfolgen einer Norm angewendet werden können (Rechtsfolgenbereich), und der vom Zeitraum der Effektivität (Bedingungsbereich) zu unterscheiden ist.

Das weiter oben dargestellte Versionierungsmodell ist in der Lage, die Rechtsordnung vor und nach einer retroaktiven Modifikation darzustellen. Für die Abfrage der Rechtslage vor einer retroaktiven Modifikation muss der Beobachtungszeitpunkt der Rechtsordnung bei der Abfrage des Modells verwendet werden.

Um die auf einen Fall anwendbaren Rechtsnormen zu ermitteln, benötigt der Algorithmus grundsätzlich zwei Parameter: (1) Den Beobachtungszeitpunkt der Rechtsordnung (i.d.R. der Zeitpunkt der rechtlichen Beurteilung), und (2) den für den Fall maßgeblichen Zeitpunkt (i.d.R. der Zeitpunkt der Verwirklichung des Sachverhalts).

Aus diesen Überlegungen ergeben sich grob folgende Schritte, um die auf einen Fall zeitlich anwendbaren Normen zu ermitteln:

1. Abfrage der existierenden Normen: Feststellung, welche Normen zum Beobachtungszeitpunkt Teil der Rechtsordnung sind.
2. Berechnung der temporalen Dimensionen: Berechnung des Intervalls des Inkraftstehens, der Effektivität und der Anwendbarkeit der im ersten Schritt ermittelten Normen.
3. Ermittlung der anwendbaren Normen: Prüfung, ob die Normen zeitlich auf den Sachverhalt (maßgeblicher Zeitpunkt) zum Zeitpunkt der rechtlichen Beurteilung (Beobachtungszeitpunkt) anwendbar sind.
4. Sistierung von Normen: Verarbeitung von Übergangsbestimmungen etc., die zur Nichtanwendung von Normen führen können.
5. Auflösung von Verweisungen: Auflösung von statischen und dynamischen Verweisungen, die zur Vergrößerung der Menge der anwendbaren Normen führen.

Jeder der Schritte, mit Ausnahme der Schritte Zwei und Fünf, kann abstrakt als Filter gesehen werden, der die Menge der in Frage kommenden Normen verkleinert. Vor Schritt Eins sind sämtliche Normen der Rechtsordnung potentiell anwendbar, nach Schritt Vier nur mehr jene, deren zeitlicher Geltungsbereich mit dem Fall übereinstimmt.

Die oberflächliche Darstellung des Algorithmus wird in den folgenden Abschnitten vertieft. Dabei wird auf jeden der Schritte separat eingegangen, und es werden die wichtigsten Aspekte bei der Diskussion aufgegriffen.

13.4.5.1. Die Ermittlung der existierenden Normen

Im ersten Schritt geht es um die Feststellung, welche Normen zum Beobachtungszeitpunkt existierten und damit bei der rechtlichen Beurteilung grundsätzlich zu berücksichtigen sind. Der Begriff „Existenz“ ist in diesem Unterkapitel **nicht** im rechtstheoretischen Sinn zu verstehen als jener Zeitraum, der mit der Kundmachung der Norm beginnt und mit ihrem Außerkrafttreten endet.

Eine Norm „existiert“ in diesem Sinn, wenn sie spätestens zum Beobachtungszeitpunkt der Rechtsordnung kundgemacht wurde und damit Teil der normativen Gesamtmenge ist. Demnach sind dieser Gesamtmenge auch historische, im rechtstheoretischen Sinn nicht mehr existierende Gesetze zuzurechnen.

Es sei in Erinnerung gerufen, dass das im Abschnitt 13.4.4. beschriebene Modell mit (1) dem Beobachtungszeitpunkt (t') – dieser ist gleichbedeutend mit dem Zeitpunkt der rechtlichen Beurteilung – und (2) dem für den Fall maßgeblichen Zeitpunkt (t'') – das ist in der Regel der Zeitpunkt der Verwirklichung des Sachverhalts – abgefragt werden muss. Es bietet eine solide Grundlage, um mithilfe der beiden Zeitpunkte die auf einen Fall anwendbaren Normen zu ermitteln.

Der erste Schritt ist daher die Ableitung eines Standbildes der Rechtsordnung, wie sie zum Beobachtungszeitpunkt beschaffen war. Diese Momentaufnahme beinhaltet dabei sämtliche Fassungen der Gesetze und Normen bis zum Beobachtungszeitpunkt. Es wird angenommen, dass der Beobachtungszeitpunkt jener Moment ist, in dem Rechtsfolgen an einen Sachverhalt geknüpft und abgeleitet werden sollen.

Wie im Unterkapitel 13.4.4.7. ausführlich dargelegt wurde, ist der Beobachtungszeitpunkt insbesondere dann erheblich, wenn die Rechtsordnung von rückwirkenden Novellen modifiziert wurde.

Mithilfe des Attributs `publication` kann recht einfach festgestellt werden, welche Normen zu einem bestimmten Zeitpunkt existieren und Teil der Rechtsordnung sind (bzw. waren). Eine Norm ist zu einem Zeitpunkt Teil der Rechtsordnung, wenn das Datum ihrer Kundmachung vor oder am Tag der Betrachtung liegt. Die beschriebene Versionierungskette ermöglicht eine Point-in-Time-Versionierung der Rechtslage und die Bereitstellung eines Schnappschusses der existierenden Normen zu einem bestimmten Zeitpunkt.

Formal kann dies wie folgt dargestellt werden: $G(t') = \{N_i | i\ ist\ ein\ Datum \leq t'\}$. Die Funktion G(t') liefert eine Menge von Normen, deren Datum der Kundmachung i kleiner gleich dem Datum der Beobachtung t' ist.

Die Menge N_t, repräsentiert sämtliche Normen, die zum Zeitpunkt t' existieren, d.h. ordnungsgemäß kundgemacht wurden.

13.4.5.2. Die Berechnung der temporalen Intervalle

Im zweiten Schritt geht es um die Berechnung der temporalen Intervalle von sämtlichen Normen, die in N_t, enthalten sind. Diese Intervalle sind notwendig, damit festgestellt werden kann, ob ein Sachverhalt vom zeitlichen Geltungsbereich einer Norm erfasst wird.

Angemerkt sei, dass diese Berechnung grundsätzlich auch vor dem ersten Schritt erfolgen könnte. Da jedoch im ersten Schritt (zumindest theoretisch) eine Verkleinerung der Gesamtmenge an Normen stattfindet, müssen im zweiten Schritt nicht mehr alle Normen betrachtet werden. Dies führt zu einem geringeren Ressourcenverbrauch des computergestützten Modells und der Schlussfolgerung.

Die formale Repräsentation von Normen besitzt temporale Attribute, die statische Markierungen im Lebenszyklus einer Norm sind. Die Zeitpunkte `startForce` und `endForce` markieren beispielsweise Beginn und Ende des Zeitraums des Inkraftstehens einer Norm.

Bevor also festgestellt werden kann, ob eine Norm in zeitlicher Hinsicht auf einen Fall anwendbar ist, müssen die temporalen Intervalle berechnet werden.

Es ist jedoch unzureichend, bloß auf den Zeitraum des Inkraftstehens abzustellen, da eine Norm bereits vor ihrem Inkrafttreten (retroaktive Effektivität) bzw. nach ihrem Außerkrafttreten (prospektive Effektivität) Rechtswirkungen erzeugen kann. Die Intervalle können demnach voneinander abweichen.

Grundsätzlich gilt, wenn keine abweichenden Angaben für den Zeitraum der Effektivität und der Anwendbarkeit gemacht werden, so fallen diese mit dem Zeitraum des Inkraftstehens zusammen.

Beginn und Ende der Effektivität bzw. Anwendbarkeit können demnach explizit spezifiziert oder implizit aus dem Zeitraum des Inkraftstehens abgeleitet werden. Im

computergestützten Modell sind daher in jedem Fall alle drei temporalen Dimensionen einer Norm belegt, sodass deren zeitliche Ausdehnung berechnet werden kann.

Unter Umständen wird das Inkrafttreten einer Rechtsvorschrift etc. von Bedingungen abhängig gemacht, sodass das Intervall, in dem die Norm in Kraft steht, berechnet werden muss. Ebenso könnte der Zeitraum der Anwendbarkeit mit einer bestimmten Dauer und nicht durch fixe Zeitpunkte bestimmt werden, z.B.: „Diese Bestimmung soll für die Dauer von 6 Monaten nach dem Inkrafttreten angewendet werden."

Im Gegensatz zu den der Norm anhaftenden statischen Zeitpunkten ist die zeitliche Ausdehnung der drei temporalen Dimensionen dynamisch. Sie ist vom Beobachtungszeitpunkt sowie den Attributen des Einzelfalls abhängig. Zum Beispiel kann das Inkrafttreten eines Gesetzes von Bedingungen abhängig gemacht werden, sodass für die Anwendung der Normen maßgeblich ist, ob diese zum Beobachtungszeitpunkt erfüllt sind.

Die temporalen Intervalle können daher nicht in jedem Fall vorberechnet werden, sondern sind von konkreten Umständen abhängig. Wird ein Intervall durch fixe Zeitpunkte begrenzt, so steht einer unabhängigen Vorberechnung nichts im Weg.

Zusammengefasst prüft das System in diesem Schritt etwaige Bedingungen, berechnet die temporalen Intervalle der Norm und stellt die Ergebnisse für weitere Schlussfolgerungen zur Verfügung.

13.4.5.3. Die Ermittlung der zeitlich anwendbaren Normen

Die in den vorherigen Abschnitten beschriebenen Schritte dienen der Vorbereitung der eigentlichen Prüfung des zeitlichen Geltungsbereichs der Normen. Es steht die Menge der Normen fest, die zum Beobachtungszeitpunkt existiert, und die temporalen Intervalle sind vollständig berechnet.

Wenn an dieser Stelle von „Ermittlung der zeitlich anwendbaren Normen" gesprochen wird, so geht es um die Beantwortung der zentralen Frage, ob sich eine Norm auf den maßgeblichen Zeitpunkt des Sachverhalts bezieht und Rechtsfolgen zum Beobachtungszeitpunkt an diesen knüpft.

Als „maßgeblicher Zeitpunkt" ist regelmäßig jener Moment zu qualifizieren, an dem die Verwirklichung eines Sachverhalts stattgefunden hat – dies variiert jedoch nach Rechtsgebiet und Rechtsart (materielles oder formelles Recht). Der „Beobachtungszeitpunkt" ist

üblicherweise der Zeitpunkt der rechtlichen Beurteilung, an dem beispielsweise ein Richter sein Urteil fällt.

Die Effektivität einer Norm (der Bedingungsbereich) bezieht sich auf einen Sachverhalt, wenn der maßgebliche Zeitpunkt (t'') innerhalb dieses Intervalls liegt. Sie knüpft Rechtsfolgen an einen Sachverhalt, wenn der Beobachtungszeitpunkt (t') innerhalb ihrer Anwendbarkeit (dem Rechtsfolgenbereich) liegt.

Formal kann die Bedingung, ob eine Norm auf einen Sachverhalt anwendbar ist, wie folgt notiert werden: $(t_{startEfficacy} \leq t'' \leq t_{endEfficacy}) \wedge (t_{startApplicability} \leq t' \leq t_{endApplicability})$.

In der Literatur wird der Zeitraum, in dem eine Norm in Kraft steht und abstrakt Rechtswirkungen erzeugen kann, als zeitlicher Geltungsbereich bezeichnet.

Für die Beurteilung, ob eine Norm in einem konkreten Fall Rechtswirkungen erzeugen kann, sind jedoch stets zwei Zeitpunkte maßgeblich: (1) Der Zeitpunkt, der für die Ermittlung der anwendbaren Rechtslage maßgeblich ist, und (2) der Zeitpunkt, zu dem Rechtsfolgen an einen Fall geknüpft werden sollen. Letzterer ist der Beobachtungszeitpunkt, in der Terminologie des Modells der Zeitpunkt der rechtlichen Beurteilung.

Mithilfe der Intervalle Effektivität und Anwendbarkeit kann hinreichend festgestellt werden, ob eine Norm im Einzelfall in zeitlicher Hinsicht anwendbar ist. Vom theoretischen Standpunkt aus betrachtet könnte es sogar Fälle geben, in denen sich der Zeitraum der Effektivität und der Anwendbarkeit gar nicht berühren. Nur wenn sich die beiden Zeiträume decken, kann einheitlich von einem zeitlichen Geltungsbereich gesprochen werden, der dem Inkraftstehen einer Norm entspricht.

Im Modell wird aus diesen Gründen zwischen Effektivität und Anwendbarkeit differenziert und nicht von einem einheitlichen zeitlichen Geltungsbereich ausgegangen.

13.4.5.4. Die temporäre Nichtanwendung von Normen

Im vorherigen Schritt wurde die generelle zeitliche Anwendbarkeit der Normen geprüft. Damit ist jedoch noch nicht endgültig festgestellt, ob eine Norm auch tatsächlich im Einzelfall anwendbar ist. In diesem Abschnitt soll auf solche Ausnahmen und deren Behandlung im Kontext des computergestützten Modells eingegangen werden.

Gesetze enthalten oftmals Übergangbestimmungen, die die Anwendbarkeit einer Norm hinauszögern. Es handelt sich dabei um eine legistische Entwurfstechnik, um temporäres Recht zu schaffen. Die Notwendigkeit, temporäre Gesetze zu schaffen, entsteht vor allem aus zwei Gründen: (1) Eine Angelegenheit ist komplex und dringend, sodass eine temporäre Lösung gefunden werden muss, und (2) wenn die Gesellschaft Zeit benötigt, um sich auf die neue Gesetzeslage einzustellen.

Die rationale Grundlage von Übergangsbestimmungen ist die Überlegung, dass manche Änderungen so starke Auswirkung auf die Rechtsunterworfenen haben, dass es angebracht erscheint, diesen Zeit zu geben, um sich den neuen Gegebenheiten anzupassen.

Übergangsbestimmungen sind üblicherweise zeitlich beschränkt, es ist jedoch auch möglich, dass diese ohne festes Enddatum versehen sind und deren Außerkrafttreten von einem Ereignis – wie dem Inkrafttreten einer anderen Rechtsvorschrift – abhängig gemacht wird.

Ferner existieren Normen, die Übergangsbestimmungen in ihren Rechtsfolgen ähneln, jedoch eine dauerhafte zeitliche Anwendbarkeit aufweisen. Solche Rechtsvorschriften setzen in Abhängigkeit von bestimmten Umständen des Einzelfalls sich selbst oder andere Bestimmungen außer Kraft und ordnen die Anwendung von Normen einer älteren Rechtslage an. Als Beispiel sei § 8 KOVG angeführt, der die Anwendung des § 8 „in der bis zum 31. Dezember 1997 geltenden Fassung“ auf Verfahren anordnet, „in denen der Antrag auf Gewährung oder Neubemessung der Beschädigtenrente vor dem 1. Jänner 1998 eingebracht wurde und das Verfahren noch nicht rechtskräftig abgeschlossen ist.“

Neben Übergangsbestimmungen, die explizit eine Norm sistieren, ist auch eine implizite Sistierung möglich, wenn eine Bestimmung von einer höherrangigen Rechtsvorschrift verdrängt wird. Im Unterschied zu Übergangsbestimmungen sind solche Normen nicht Teil desselben Gesetzes. Wenn eine Rechtsvorschrift die andere sistiert, führt dies zum Einfrieren letzterer und zur Aufhebung ihrer Anwendbarkeit. Wenn die sistierende Norm selbst außer Kraft tritt, kommt es zum Wiederaufleben der aufgehobenen Norm und deren Rechtswirkungen. Es sind auch komplexere Zusammenhänge, wie die Sistierung einer sistierenden Bestimmung usw., möglich.

Um solche impliziten Sistierungen modellieren zu können, werden entsprechende kontextuale Informationen benötigt, wie die rechtssetzende Autorität und der Rang der Vorschrift (einfaches Gesetz, Verfassungsgesetz etc.). Grundsätzlich können solche

Informationen im computergestützten Modell abstrakt durch eine Zahl repräsentiert werden, die die Priorität einer Norm ausdrückt. Dabei geht jedoch der Sinnzusammenhang der Sistierung mit deren rechtlicher Grundlage weitgehend verloren.

Als Beispiel sei das in der Verfassung verankerte Notverordnungsrecht des Bundespräsidenten gemäß Art. 18 Abs. 3 B-VG erwähnt. Eine solche Notverordnung könnte vorübergehend die Anwendung einer einfachgesetzlichen Bestimmung verhindern.

Ein Sonderfall, der aus Gründen der Vollständigkeit hier erwähnt werden soll, ist die nicht ausschließbare Rückwirkung einer Normaufhebung durch den Verfassungsgerichtshof für den Anlassfall. Der Anlassfall ist stets im Sinne einer „Ergreiferprämie“ nach der bereinigten Rechtslage zu entscheiden. Als Anlassfall ist jenes Verfahren zu verstehen, im Zuge dessen das Normprüfungsverfahren beim VfGH anhängig gemacht wurde. Das System muss in der Lage sein, solche Ausnahmeregeln zu repräsentieren, die von den Umständen des Einzelfalls abhängig sind.

Ein computergestütztes Modell muss die dargestellten Varianten bei der Ermittlung der zeitlich anwendbaren Normen berücksichtigen. Wie die Ausführungen verdeutlich haben, handelt es sich dabei um eine Art der nicht-monotonen Schlussfolgerung. Eine zeitlich grundsätzlich anwendbare Norm wird durch eine andere Bestimmung angefochten und deren Anwendung bei erfolgreicher Anfechtung verhindert.

13.4.5.5. Die Behandlung von Verweisungen

Nach dem vierten Schritt des Algorithmus, der im vorherigen Abschnitt dargestellt wurde, steht die Grundmenge der zeitlich anwendbaren Normen fest. Zur endgültigen Auswahl müssen im letzten Schritt Verweisungen im Gesetzestext behandelt werden[197].

Dazu sind zuerst sämtliche Normen zu verarbeiten, die Verweisungen enthalten. Eine Verweisung verkörpert im Modell eine Anweisung zum Import von Normen aus einem anderen Gesetz.

---

[197] Das Modell trifft keine Annahmen darüber, wie die Verweisungen aus dem Rechtstext extrahiert werden. Denkbar ist eine manuelle, eine semi-automatisierte oder eine automatisierte Extraktion. In jüngster Zeit wurden vielversprechende Versuche unternommen, um automatisiert Verweisungen aus Rechstexten zu extrahieren (Tran, Ngo, Nguyen, & Shimazu, 2013, 2014). Bislang existiert jedoch kein automatisierter Ansatz mit ausreichender Präzision, der praktisch eingesetzt werden könnte.

In der Rechtstheorie werden zwei Arten unterschieden, nämlich statische und dynamische Verweisungen: Bei einer *statischen Verweisung* wird auf eine andere Norm in einer bestimmten Fassung verwiesen. Dabei sind spätere Änderungen oder das Außerkrafttreten der verwiesenen Norm aus Sicht der verweisenden Norm unerheblich. Eine *dynamische Verweisung* referenziert hingegen eine andere Norm in ihrer jeweils geltenden Fassung (Öhlinger & Eberhard, 2014, S. 68).

Der Importmechanismus des Systems muss daher in Übereinstimmung mit der rechtstheoretischen Grundlage die beiden Verweisungsarten korrekt behandeln. Die Behandlung einer statischen Verweisung erfolgt so, als wäre die verwiesen Norm tatsächlich gemeinsam mit der verweisenden Norm erlassen worden. Sie wird daher in ihrer jeweiligen Form fester Bestandteil des verweisenden Gesetzes. Damit stellt sich jedoch die Frage, wie die zeitlichen Dimensionen der „importierten" Norm behandelt werden sollen?

In Übereinstimmung mit den obigen Ausführungen, dass beispielsweise das Außerkrafttreten aus Sicht der verweisenden Norm unerheblich ist, muss davon ausgegangen werden, dass die verwiesene Norm die zeitlichen Dimensionen des verweisenden Gesetzes bzw. der verweisenden Norm gewissermaßen erbt. Der Lebenszyklus der importierten Norm kann sich demnach von ihrer ursprünglichen zeitlichen Ausdehnung unterscheiden. Dies hat selbstverständlich keinen Einfluss auf das verwiesene Gesetz und dessen Normen selbst.

Daraus können interessante Konstellationen entstehen, wenn die verwiesene Norm ihrerseits von Normen abhängig ist, diese aber mit dem verwiesenen Gesetz außer Kraft treten. In solch einem Fall „lebt" die verwiesene Norm im verweisenden Gesetz gewissermaßen weiter, die anderen Normen sind jedoch bereits außer Kraft getreten. Auf diese Problematik sei hingewiesen, sie kann aber an dieser Stelle nicht weiter vertieft werden.

Statische Verweisungen werden von nachträglichen Änderungen der verwiesenen Norm nicht berührt, das gilt auch für retroaktive Modifikationen. Da es sich damit um einen unveränderlichen „Import" der verwiesenen Norm handelt, sind nach der Rechtsprechung des VfGH statische Verweisungen auch auf Normen zulässig, die von einer anderen rechtssetzenden Autorität erlassen wurden.

Das System muss bei der Verarbeitung einer statischen Verweisung dafür Sorge tragen, dass die verwiesene Norm in der ursprünglichen und unveränderten Fassung importiert wird. Eine bestimmte Fassung einer Norm (eines Gesetzes) kann im Lauf der Zeit eindeutig mithilfe des Datums der Kundmachung (`publication`) und dem Beobachtungszeitunkt (`viewDate`) der Rechtsordnung identifiziert werden. Der Beobachtungszeitpunkt muss bei der Auflösung der statischen Verweisung berücksichtigt werden, da unter Umständen die originale Fassung der Norm in Zukunft von einer rückwirkenden Modifikation betroffen sein kann, die auf die Rechtsordnung zum damaligen Zeitpunkt wirkt.

Im Gegensatz dazu referenzieren dynamische Verweisungen eine Norm „in der jeweils geltenden Fassung“. Damit werden auch zukünftige Änderungen der verwiesenen Norm übernommen. Eine Änderung der verwiesenen Norm wirkt sich damit automatisch auf den Inhalt der verweisenden Norm aus.

Im Gegensatz zu einer statischen Verweisung wird nicht eine bestimmte Fassung einer Norm importiert. Das System muss daher einen anderen Importmechanismus anwenden als bei der Auflösung von statischen Verweisungen.

Es ist davon auszugehen, dass für die Auflösung einer dynamischen Verweisung zunächst mithilfe des Beobachtungszeitpunkts sämtliche Fassungen der verwiesenen Norm selektiert werden. Danach werden die temporalen Intervalle berechnet und die zeitliche Anwendbarkeit der verwiesenen Normen mithilfe des Beobachtungszeitpunkts und des maßgeblichen Zeitpunkts geprüft. Sodann muss geprüft werden, ob etwaige Übergangsbestimmungen etc. dazu führen, dass eine grundsätzlich anwendbare Norm nicht angewendet wird. Schließlich können die verwiesenen Normen selbst wiederum Verweisungen enthalten, sodass auch diese aufgelöst werden müssen, usw.

Wie ersichtlich, werden die verwiesenen Normen in die Menge anwendbarer Normen importiert und vergrößern diese. Bei der (automatisierten) Auflösung von Verweisungen wird der in diesem Abschnitt beschriebene Algorithmus auf die neu hinzukommenden Normen rekursiv angewendet. Dies wiederholt sich, solange Verweisungen vorhanden sind. Auf Probleme, die aus zirkulären Verweisungen entstehen, sei hingewiesen, die Diskussion soll diesbezüglich jedoch nicht vertieft werden.

13.4.6. Abschließende Bemerkungen

In diesem Unterkapitel wurde ein Modell vorgestellt, das in der Lage ist, die temporalen Dimensionen von Gesetzen und Normen zu repräsentieren. Dabei liegt der Fokus auf der Modellierung der externen zeitlichen Dimensionen (dem zeitlichen Geltungsbereich) von Normen. Nicht in der Zuständigkeit des Modells liegt hingegen die Formalisierung von Ereignissen, die in einer Rechtsvorschrift selbst zum Ausdruck gebracht werden, z.B. wann eine Person ihre Steuererklärung abgeben muss.

Das temporale Modell von rOWLer basiert demnach auf der Annahme, dass zwischen externer und interner Zeit von Rechtsnormen unterschieden werden kann. Wie in Abschnitt 11.5.3. gezeigt wurde, kann diese Unterscheidung nicht immer aufrechterhalten werden. Selbst in solchen Fällen, in denen der zeitliche Bezugsbereich direkt in der Norm zum Ausdruck gebracht wird, ist es jedoch möglich, dass interne Zeitangaben im Modell als externe Zeit behandelt werden. Damit sollte der vorgeschlagene Algorithmus in jedem Fall eingesetzt werden können.

Kern dieses Modells sind sogenannte „Versionierungsketten“, mit deren Hilfe eine Point-in-Time-Versionierung der Rechtsordnung ermöglicht wird. Flankiert wird das temporale Modell von einer soliden Namenskonvention, die eine eindeutige Bezeichnung und Identifikation von legislativen Ressourcen gewährleistet.

Die Konvention ist an den FRBR-Standard zur Repräsentation bibliographischer Werke angelehnt, die auch Akoma Ntoso zugrunde liegt. Damit ist gewährleistet, dass Bezeichnungen und IRIs zu den State-of-the-Art-Technologien kompatibel sind. Das formale Modell kann daher ohne Probleme mit der in Akoma Ntoso ausgeführten elektronischen Repräsentation eines Gesetzes integriert werden.

Durch die Anlehnung des Modells an die Struktur von juristischen Dokumenten lässt sich eine möglichst isomorphe Darstellung des Rechts erreichen. Das Gefüge des Gesetzes wird vom formalen Modell adäquat wiedergegeben.

Mithilfe von IRIs lassen sich Gesetze, Paragrafen, Absätze und sogar einzelne Sätze adressieren. Im gegenständlichen Anwendungsfall werden jedoch überwiegend Paragrafen adressiert, die durch formale Regeln repräsentiert werden. Um der temporalen Dynamik des Rechts zu entsprechen, können in die IRI entsprechende Angaben aufgenommen werden,

die eine bestimmte Fassung einer Norm eindeutig adressieren. Mit diesem Ansatz können sowohl statische als auch dynamische Verweisungen formalisiert werden.

Obwohl das gegenständliche Versionierungsmodell und die Namenskonvention für die Versionierung und Adressierung von formalen Regeln konzipiert sind, verwaltet das Modell intern abstrakte „Ressourcen“. Eine solche Ressource ist ein beliebiges Objekt, das mit temporalen Attributen versehen ist. Das Modell ist daher grundsätzlich unabhängig von der konkreten Form der verwalteten Objekte und deren Semantik.

Die Verwendung von Abstraktionen ermöglicht die größtmögliche Flexibilität, sodass das Modell in unterschiedlichsten Anwendungsszenarios eingesetzt werden kann. Der Algorithmus „zur Ermittlung der anwendbaren Rechtslage“, der in Abschnitt 13.4.5. dargestellt wurde, ist daher nur eine Möglichkeit, wie das temporale Modell verwendet werden kann. Das Modell ist ausdrucksstark genug, um beispielsweise als Grundlage für die Implementierung eines juristischen Rechtsinformationssystems zu dienen.

Ferner handelt es sich um ein theoretisches Modell, das unabhängig von einer konkreten Implementierung ist. Das vorgestellte Konzept macht keine Einschränkungen hinsichtlich der Technologie für die Implementierung.

13.5. Grundpfeiler des formalen Modells

Die weiteren Überlegungen folgen Kelsen (1960), nach dem eine Norm der Sinn eines menschlichen Willensaktes ist, der anderen Personen ein Verhalten gebietet, verbietet, erlaubt oder sie zur Setzung von Normen ermächtigt (Koller, 1997, S. 151 ff.). Normen werden üblicherweise schriftlich durch textuelle Klauseln – Sätze, Absätze, Paragrafen in Gesetzen – zum Ausdruck gebracht.

Eine Regel ist die logische Formalisierung einer textuellen Bestimmung, bestehend aus dem Antezedens und der Konsequenz, repräsentiert im Modell durch die Abstraktion[198] `Rule`.

Gelegentlich kommt es vor, dass mehrere textuelle Rechtsvorschriften durch eine einzige Regel formalisiert werden, aber auch umgekehrt, dass eine Bestimmung mehrere formale

[198] Wenn in diesem Kapitel von „Abstraktion“ gesprochen wird, ist im technischen Sinn ein Interface gemeint. Interfaces sind neben Klassen die wesentlichen „Bausteine“ eines objektorientierten Programms.

Regeln beinhaltet. Es ist von größter Wichtigkeit, diese Zusammenhänge im Modell sichtbar zu machen, damit dieses durch einen Juristen verifiziert und validiert werden kann.

13.5.1. Isomorphismus

Eines der schwierigsten Probleme bei der Formalisierung von Rechtsvorschriften ist eine isomorphe Repräsentation zu erreichen – damit ist die Verbindung der formalen Regeln und der rechtlich verbindlichen textuellen Aussagen zu verstehen, die von den Regeln modelliert werden.

Immer dann, wenn sich die Rechtsquellen im Lauf der Zeit verändern, müssen auch die formalen Regeln entsprechend aktualisiert werden, wobei üblicherweise kein automatisierter Mechanismus vorhanden ist, um Änderungen zu verfolgen und in das formale Modell einzupflegen. Wenn das Prinzip des Isomorphismus nicht angemessen berücksichtigt wird, ist es schwierig festzustellen, welche formalen Regeln von der Änderung betroffen sind (Bench-Capon & Coenen, 1992).

Ferner ist von größter Wichtigkeit, dass der juristische Experte (Richter, Sachbearbeiter, Jurist etc.) nachvollziehen kann, ob die formalen Regeln im Hinblick auf den originalen rechtsverbindlichen Text korrekt modelliert wurden. Die Verknüpfung der formalen Regeln mit der Rechtsgrundlage ist auch die Voraussetzung, um die Ergebnisse der Inferenzmaschine auf ihre rechtliche Richtigkeit prüfen zu können.

Das Objektmodell bietet Unterstützung für die feingranulare Verknüpfung der formalen Regeln mit deren Rechtsgrundlage, basierend auf IRIs und der weiter oben dargestellten Namenskonvention, um elektronische Ressourcen zu adressieren. Jeder Teil einer Regel kann mithilfe einer IRI mit seiner Rechtsquelle verknüpft werden. Dies gilt für das Antezedens, die Konsequenz und auch für jede einzelne Bedingung des Antezedens.

Die Verbindung zwischen formaler Regel und Rechtsquelle wird durch das Interface `IsomorphicLegalEntity` im Modell repräsentiert. Das Interface kann von einer beliebigen Klasse implementiert werden und definiert nur eine Methode „getSourceIRIs“, die eine oder mehrere IRIs liefert, um auf die jeweiligen Rechtsquellen zu verweisen. Eine IRI kann dabei direkt auf eine elektronische Ressource oder einen Begriff (Klasse) verweisen, die in einer Ontologie modelliert ist.

Das Interface `IsomorphicLegalEntity` erlaubt eine n:m-Beziehung zwischen mehreren formalen Regeln und einer textuellen Rechtsvorschrift, aber auch zwischen mehreren textuellen Rechtsvorschriften und einer formalen Regel[199].

Um die Struktur von Gesetzen adäquat im formalen Modell zu repräsentieren, werden neben der Repräsentation von ersteren durch das Interface `Statute` auch Teile eines Gesetzes (genannt „Teil", „Abschnitt" etc.) durch das Interface `Section` abstrahiert. Die Bedeutung der Abstraktion `Section` ist in einem weiten Sinn zu sehen, sie ist – ebenso wie `Statute` – abstrakt als eine Menge von Regeln zu verstehen und erfasst „Teile", „Abschnitte" etc. gleichermaßen. Implementierungen des Interfaces `Statute`, die demnach ein Gesetz repräsentieren, können sich wiederum aus mehreren „Sektionen" (Abschnitten) zusammensetzen. Ferner werden vom formalen Modell keine Einschränkungen hinsichtlich der Verschachtelungstiefe von „Sektionen" gemacht.

Wie einzelne Regeln können Klassen vom Typ `Statute` und `Section` auch das Interface `IsomorphicLegalEntity` implementieren, um auf die jeweilige Rechtsgrundlage zu verweisen.

Mithilfe der hier dargestellten Interfaces lässt sich die Struktur von Gesetzen äußerst genau nachbilden, was nicht nur dem Prinzip des Isomorphismus entspricht, sondern auch legistische Gründe hat. Beispielsweise erklärt § 99 Abs. 1 Pensionsgesetz (PG), BGBl. Nr. 340/1965 i.d.F. BGBl. I Nr. 46/2014, den Abschnitt XIII nur für Beamte anwendbar, die bestimmte Kriterien erfüllen.

13.5.2. Temporale Dimensionen

Die Repräsentation der zeitlichen Dimensionen von Rechtsnormen erfolgt durch das Interface `TemporalLegalEntity`, das folgende Signatur aufweist:

```
public interface TemporalLegalEntity {
     public Publication getPublication();
     public Force getForce();
     public Efficacy getEfficacy();
     public Applicability getApplicability();
}
```

[199] Je nach Möglichkeit sollte eine 1:1-Beziehung zwischen formaler Regel und Rechtsvorschrift angestrebt werden, um das Modell einfach zu halten.

Bei `Publication`, `Force`, `Efficacy` und `Applicability` handelt es sich wiederum um Abstraktionen (Interfaces), die grundsätzlich ein Intervall beschreiben, begrenzt durch Start- und Enddatum. Es wäre jedoch in Zukunft denkbar, dass eine konkrete Implementierung einer der Dimensionen auf bestimmte Ereignisse abstellt, wie beispielsweise das Inkrafttreten einer anderen Rechtsvorschrift, von denen der Zeitraum des Inkraftstehens abhängig ist.

Neben der Kapselung von Implementierungsdetails bietet die Verwendung von höherwertigen Interfaces den Vorteil, dass die Interfaces mit Metadaten versehen werden können, die die jeweilige Dimension beschreiben. Im formalen Modell werden die temporalen Interfaces mit einer IRI versehen, die auf die Beschreibung der temporalen Dimension in einer Ontologie verweist.

Klassen vom Typ `Rule` und `Statute` implementieren zusätzlich das Interface `TemporalLegalEntity`, um das in Abschnitt 13.4.4. beschriebene zweischichtige temporale Modell umzusetzen.

13.5.3. Metadaten

Das formale Modell besteht aus mehreren Schichten und unterscheidet zwischen Metadaten und formalen Regeln.

Unter „Metadaten“ werden zusätzliche Daten verstanden, die nicht von der rechtlich zuständigen Autorität (dem Gesetzgeber) stammen, sondern von einem beliebigen anderen Autor (Palmirani & Vitali, 2011, S. 78). Metadaten beschreiben zusätzliche Informationen zur formalen Regel, wie die Beschreibung[200], den Autor, die Jurisdiktion, die rechtserzeugende Autorität sowie allfällige Urteile und Kommentare zu einer Norm.

Die Metadaten reichern die formale Regel um zusätzliche Informationen an, die von der Engine bei Bedarf verwendet werden können und auch als Grundlage für die Anzeige von weiterführenden Informationen zur Norm herangezogen werden.

Für die Validierung der Richtigkeit des Modells und der darauf beruhenden (automatisierten) Entscheidungen werden darüber hinaus Informationen hinterlegt, wie der Autor die Norm interpretiert und die formale Regel daraus abgeleitet hat. Diese Beschreibung ist rein

[200] Darunter ist ein unförmlicher Kommentar zu verstehen, der die formale Regel näher erläutert.

textueller Natur und gewissermaßen ein Kommentar des Autors zur formalen Regel. Diese Information wird durch die Methode `getImplementationDescription` des Interfaces `Metadata` bereitgestellt.

Technisch werden die Metadaten durch das Interface, treffend genannt `Metadata`, repräsentiert. Implementierungen des Interfaces kapseln die Metadaten zu einer Regel und bieten entsprechende Zugriffsmethoden, wie `getAuthor`, `getAuthority` etc.

Regeln, die Metadaten bereitstellen können, implementieren zusätzlich zu `Rule` das Interface `MetadataAware`. Dieses Interface definiert die Methode `getMetadata`, die verwendet wird, um die Metadaten der Regel abzurufen.

13.6. Die Modellierung von Normen

Kelsen (1979) folgend und in Übereinstimmung mit der Rechtstheorie wird angenommen, dass Normen grundsätzlich folgende logische Struktur aufweisen: Wenn A1,…,A2 dann B; wobei „A1,…,A2“ die Bedingungen der Norm sind, „B“ die Rechtswirkungen und „wenn…dann“ ist der *normative Konditional* (engl. *normative conditional*).

Die Normen werden in einem objektorientierten Modell[201] durch Regelobjekte formalisiert, bestehend aus Antezedens und Konsequenz. Im technischen Sinne werden Regeln durch das Interface `Rule` repräsentiert:

```
public interface Rule {
     public Antecedent getAntecedent();
     public Consequent getConsequent();
}
```

Wie das Codebeispiel zeigt, kapselt eine Instanz vom Type `Rule` das Antezedens und die Konsequenz einer Regel. Antezedens und Konsequenz formalisieren den Tatbestand bzw. die Rechtsfolgen einer Norm. Das Antezedens besteht aus einer Menge von Bedingungen in der Form $\{C_1, \dots, C_n\}$[202]. Die Bedingungen können mit logischen Operatoren (und, oder, nicht) verknüpft werden. Ebenso wird die Konsequenz einer Norm als eine Menge von Rechtsfolgen begriffen und weist dem entsprechend die Form $\{R_1, \dots, R_n\}$[203] auf.

---

[201] Das in diesem Abschnitt beschriebene Modell wurde in Java implementiert. Die Ausführungen können jedoch auf eine beliebige objektorientierte Sprache übertragen werden.

[202] Ein Element „C“ steht dabei für „condition“.

[203] Ein Element „R“ steht dabei für „result“.

13.6.1. Unterstützung verschiedener Normtypen

Wie bereits oben ausgeführt, wird davon ausgegangen, dass sämtliche Normen logisch als normative Konditionale, bestehend aus Antezedens und Konsequenz, beschrieben werden können. Die üblichen deontischen Qualifikationen werden durch die Interfaces `Permission`, `Obligation` und `Prohibition` repräsentiert, die von `Rule` abgeleitet sind.

Die Konklusion von normativen Konditionalen muss nicht zwingend deontisch sein. In der Rechtstheorie werden grundsätzlich konstitutive und präskriptive Normen unterschieden. Darüber hinaus gibt es jedoch auch komplexere Normtypen.

Das System muss demnach in der Lage sein, mit verschiedenen Normtypen umzugehen. Das Interface `Rule` bietet dafür einen ausgezeichneten Ansatzpunkt. Es können beliebige Klassen und Subinterfaces erzeugt werden, die eine erweiterte Semantik zur Repräsentation spezieller Normtypen aufweisen.

Um die verschiedenen Normtypen aus technischer Sicht verarbeiten zu können, werden Erweiterungspunkte (Schnittstellen) im Modell benötigt, die bei Bedarf mit speziellen Implementierungen versorgt werden können.

Die wichtigste Abstraktion in diesem Zusammenhang ist die `NormHandlingStrategy`, deren Aufgabe es ist, die materiellrechtliche Anwendbarkeit einer Norm zu prüfen und gegebenenfalls die Rechtsfolgen abzuleiten. Die von der Engine verwendete Standardimplementierung `DefaultNormHandlingStrategy` bedient sich zur Bewältigung ihrer angedachten Aufgaben weiterer Abstraktionen, nämlich `NormMatchingStrategy` – zur Prüfung des Antezedens der Regel – und `NormExecutionStrategy` – zur Ableitung der Rechtsfolgen.

Die `NormMatchingStrategy` ist verantwortlich zu prüfen, ob das Antezedens einer Regel – das ist die formale Repräsentation des Tatbestands – im Einzelfall erfüllt ist. Im einfachsten Fall beschränkt sich diese Prüfung auf die Feststellung der direkten Übereinstimmung der Bedingungen des Antezedens und den Sachverhaltselementen. Für diesen Zweck existiert die `SimpleNormMatchingStrategy` im System.

Spiegelbildlich zur `NormMatchingStrategy` ist die `NormExecutionStrategy` zu sehen. Letztere ist für die Ableitung der Rechtsfolgen einer Regel zuständig, sobald diese als

„anwendbar“ qualifiziert wurde[204]. Eine solche „Ableitung“ von Rechtsfolgen kann die deontische Feststellung sein, dass ein bestimmter Sachverhalt verboten, geboten oder erlaubt ist, aber auch die Durchführung einer Berechnung (wie im Steuer- oder Pensionsrecht) etc. Anhand dieser Beispiele zeigt sich, dass eine konkrete Implementierung der `NormExecutionStrategy` den Typ der Konklusion der Regel interpretieren und ausführen muss.

Die Engine stellt den Regeln und den diversen Strategy-Implementierungen während der Durchführung der rechtlichen Beurteilung eine Instanz vom Typ `ReasonerContext` zur Verfügung. Der `ReasonerContext` kann vereinfacht als eine Sammlung von Schlüssel- und Wertepaaren verstanden werden. Die Ergebnisse der Anwendung einer Regel beispielsweise können im `ReasonerContext` zur weiteren Verwendung abgelegt werden. Ferner bietet er Methoden zur Kommunikation mit der Engine an.

Das Interface `NormHandlingStrategy` im Zusammenspiel mit `NormMatchingStrategy` und `NormExecutionStrategy` ermöglicht eine äußerst hohe Flexibilität hinsichtlich der Unterstützung verschiedenster Normtypen, ohne dass der Kern des Systems und der Algorithmus verändert werden müssten. So ist es möglich, spezielle Prüfungen des Antezedens zu realisieren und Rechtsfolgen abzuleiten, die einer besonderen Behandlung der Konsequenz einer Regel bedürfen.

Mithilfe einer speziellen `NormMatchingStrategy` könnten auch „schwierige“ Regeln einer Behandlung zugeführt werden. Eine solche Behandlung der Regel kann automatisiert erfolgen oder an den menschlichen Benutzer delegiert werden. Beispielsweise könnte eine auf „Dimensions and Factors“ (Bench-Capon, 2012; Sartor, 2006b) basierende Implementierung verwendet werden, um vage Begriffe des Tatbestands einer automatisierten Behandlung zugänglich zu machen.

Durch Entkoppelung der Prüfung des Antezedens und der Behandlung der Konsequenz einer Regel vom Algorithmus wird die größtmögliche Flexibilität erreicht, die erforderlich ist, um neben Geboten, Verboten und Erlaubnissen auch komplexere Normtypen modellieren zu

[204] Dabei darf nicht vergessen werden, dass eine Regel im Sinn der „defeasible logic“ erst dann angewendet wird, wenn keine (höherrangigen) anfechtenden Regeln („defeaters“) diese blockieren.

können wie jene, die von Hohfeld (1913, 1917) entwickelt wurden. Es sei weiters auf die gelungene Darstellung verschiedener Rechtsbegriffe von Sartor (2006a) verwiesen.

13.6.2. Normative Ausnahmen und Prioritäten

Im Recht kommen implizite (*Lex specialis*, *Lex posterior* und *Lex superior*) und explizite Ausnahmen zwischen Normen vor. Ein computergestütztes Modell des Rechts muss in der Lage sein, beide Arten von Ausnahmen zu repräsentieren, um die Struktur des Gesetzes zu reflektieren, bestehend aus allgemeinen Regeln und Ausnahmen.

Auf dem Gebiet der Künstlichen Intelligenz und Recht wurden verschiedene Methoden entwickelt, um Konflikte zwischen Regeln zu lösen, nämlich *Spezifität* (engl. specifity) (Prakken & Schrickx, 1991; van de Ven, Breuker, et al., 2008), *Gewichtung* (engl. weight oder auch salience) (Walton & Gordon, 2012) und *Präferenzrelation* (auch genannt *Superioritätsrelation*) (Johnston & Governatori, 2003; Lam & Governatori, 2009). Das formale Modell repräsentiert Gewichtungen und Präferenzrelationen zwischen Regeln durch das Interface `WeightedRule` bzw. `PreferenceRelation`.

Konflikte zwischen Regeln werden vom System gelöst, indem diese mithilfe einer Implementierung des Interface `RuleOrderingStrategy` geordnet werden. Die Standardimplementierung dieses Interface inspiziert sämtliche Regeln um diese zu ordnen, wobei sämtliche oben beschriebene Methoden unterstützt und implizite und explizite Informationen genützt werden. Die Regeln werden in einem Netzwerk angeordnet, das die Rangordnung zwischen diesen repräsentiert, und dabei von einem dynamischen Proxy[205] zur Laufzeit umhüllt, der das Interface `SuperiorityRelation` implementiert.

Das Interface `SuperiorityRelation` repräsentiert ein abstraktes Konzept, das die duale Beziehung zwischen zwei Regeln beschreibt[206]. Entsprechend den Methoden zur Repräsentation von Prioritäten zwischen Regeln – Spezifität, Gewichtung und Präferenzrelation – existiert eine Implementierung für jeden Ansatz.

---

[205] Die Architektur des Systems baut durchgehend auf Interfaces auf, die es erlauben, den Mechanismus zur dynamischen Erzeugung von Proxys in Java zu verwenden.

[206] In dieser Hinsicht ähnelt eine Superioritätsrelation der Präferenzrelation, im Gegensatz zu letzterer ist sie jedoch eine Abstraktion, deren Instanzen zur Laufzeit von der Engine erzeugt werden.

Die Abstraktion `SuperiorityRelation` ermöglicht die dynamische Erzeugung beliebiger Relationen zwischen Regeln, z.B. zwischen *Lex superior* und *Lex inferior*, indem die Rangordnung der rechtserzeugenden Institutionen aus der Ontologie verwendet wird, die die jeweiligen Regeln erlassen haben[207]. Der Ansatz von rOWLer folgt damit den Überlegungen von Prakken und Sartor (1997) und Antoniou (2004), dass Prioritäten zwischen Regeln aus dem Schlussfolgerungsprozess dynamisch abgeleitet werden (können) und überträgt diese auf ein objektorientiertes Modell. Eine dynamische Berechnung von Prioritäten ist freilich optional und könnte in Zukunft dahingehend erweitert werden, dass die abgeleiteten Werte selbst „anfechtbar" sind und durch einen nicht-monotonen Algorithmus berechnet werden.

Das Modell von rOWLer gibt keine festen Prioritätslevels vor, wie diese zunächst erforderlich erscheinen, um den Stufenbau der Rechtsordnung unter Einbeziehung des supranationalen Rechts der EU nachzubilden. Falls erforderlich, können Stufen simuliert werden, indem allen Regeln einer Stufe dieselbe Priorität zugeordnet wird. Wie bereits oben ausgeführt, steht einer speziellen Implementierung von `SuperiorityRelation` nichts im Wege, um auch diese Anforderung abzudecken.

Das mit Superioritätsrelationen angereicherte Modell bildet die Grundlage, um die Regeln als *defeater*, *defeasible* und *strict* im Sinne der defeasible logic zu qualifizieren. Ferner ermöglicht es die Verwendung von Inferenzmaschinen wie SPINdle (Lam & Governatori, 2009), die auf defeasible logic basieren sowie die Implementierung einer eigenen Inferenzmaschine, die auf nicht-monotonen Algorithmen wie dem Delores Algorithmus (Maher, 2004) basiert.

13.6.3. Unterstützung mehrerer Interpretationen

Nicht immer ist die Bedeutung einer Norm eindeutig und die Anwendbarkeit im Einzelfall unzweifelhaft – es bedarf der Interpretation der fraglichen Rechtsvorschrift durch den Juristen. Anders als dies bei Expertensystemen – insbesondere im öffentlichen Sektor – sonst üblich ist, unterstützt rOWLer mehrere Varianten einer Regel, die verschiedenen Interpretationen einer Norm entsprechen.

---

[207] Damit ist es unter anderem möglich, den Vorrang von Verfassungsrecht vor einfachen Gesetzen zu modellieren.

Technisch wird eine mehrdeutige Norm durch das Interface `AmbiguousRule` repräsentiert. Dieses Interface ist von `Rule` abgeleitet und definiert die Methode `getAlternativeInterpretations`. Ihrem Namen entsprechend liefert die Methode die Varianten einer formalen Regel.

Wie üblich implementiert die Regel selbst sowie deren Varianten das Interface `IsomorphicLegalEntity` und verweist mithilfe von `Metadata` auf den Autor der Regel, der die alternative Interpretation vertritt, die Rechtsquelle und zugehörige Entscheidungen (Urteile etc.).

Es bleibt dem Autor des formalen Modells überlassen, welche Regel er als „Standardinterpretation" modelliert und welche Regeln als Interpretationsvarianten. Um im formalen Modell die „Standardregel" zum Ausdruck zu bringen, muss das Interface `DefaultInterpretation` implementiert werden, das bloß zur Markierung der Regel dient.

Ein Axiom des Modells ist die Annahme, dass nur eine Standardinterpretation einer Norm als formale Regel existieren kann.

13.6.4. Sanktion und Wiedergutmachung

Charakteristisch für Normen ist die Möglichkeit, dass diesen zuwidergehandelt werden kann. Üblicherweise führt dies in weiterer Folge zur Verhängung einer Sanktion, die den rechtmäßigen Zustand wiederherstellen soll, um das geschehene Unrecht zu kompensieren.

Der Zusammenhang zwischen der Norm und den zugehörigen Sanktionen kann als Liste von deontischen Formeln verstanden werden. Dieses Konzept wurde von Governatori und Rotolo (2006) im Zusammenhang mit der Modellierung von Sanktionsnormen („contrary-to-duty obligations") entwickelt und wird in Athan et al. (2013b) „suborder list" genannt. Im Folgenden werden diese Listen schlicht „Regellisten" genannt.

Charakteristisch für eine Sanktionsnorm ist, dass sie nur dann anwendbar ist, wenn die primäre Norm verletzt wurde. Die Intuition der Formalisierung von Sanktionsnormen, der gefolgt wird, ist, dass diese als spezielle Form von normativen Ausnahmen begriffen werden, die keinen Anwendungsfall der „anfechtbaren Schlussfolgerung" darstellen (Governatori & Rotolo, 2006; Prakken & Sergot, 1996).

Die Funktionsweise einer Regelliste sei anhand folgenden Beispiels erläutert: [OBL] A, [OBL] B, [FORB] C, [PERM] D

Dieser Ausdruck bedeutet, dass A geboten ist, wenn A verletzt wird – d.h. wenn die Situation ~A[208] eintritt – dann ist existiert Verpflichtung B, um die Verletzung von A zu kompensieren. Wenn auch die Verpflichtung B verletzt wird, dann ist das Verbot von C anwendbar. Sofern trotzdem die Situation C eintritt, kommt die Erlaubnis von D ins Spiel.

In Rechtstexten wie Gesetzen, aber auch Verträgen, existiert üblicherweise ein Abschnitt mit Strafen für den Fall, dass Normen verletzt werden. Eine Strafnorm kann dabei durchaus dafür gedacht sein, die Verletzung verschiedener Bestimmungen zu kompensieren.

Mithilfe der beschriebenen Regellisten lassen sich recht simpel die primäre Verpflichtung und die zugehörigen Strafnormen, bei gleichzeitiger Berücksichtigung des Isomorphismus, modellieren. Das formale Modell wird nun weiter ausgebaut, um primäre Normen und Sanktionen mithilfe von Regellisten zu formalisieren.

Regellisten werden im formalen Modell durch das Interface `SuborderList` repräsentiert, das die folgende Signatur aufweist:

```
public interface SuborderList {
     public List<Rule> getRules();
}
```

Die primäre Regel, deren Verletzung durch zumindest eine Sanktionsnorm kompensiert werden soll, implementiert das Interface `Reparable`, das nur eine Methode `getSuborderList` definiert, die ihrem Namen entsprechend die zugehörige `SuborderList` liefert.

Zur Behandlung von Regeln vom Typ `Reparable` wird eine spezielle `NormHandlingStrategy` verwendet, nämlich die Klasse `ReparableAwareNormHandlingStrategy`. Die Strategy ist in der Lage zu erkennen, wenn die primäre Norm verletzt wird, und ruft die zugehörigen Sanktionsnormen auf. Sollte dieser ebenfalls zuwidergehandelt werden, wird die nächste Sanktionsnorm in der Liste verhängt usw., bis die letzte Norm der Regelliste erreicht wird.

[208] ~A bedeutet „nicht A“, d.h. eine Situation, die der Obligation A widerspricht.

Die in einer Regelliste enthaltenen Normen werden grundsätzlich nicht vom Algorithmus der Engine verarbeitet, da sie eine spezielle Art der Ausnahme repräsentieren und daher nicht direkt an der „anfechtbaren Schlussfolgerung“ teilnehmen.

Zukünftige Forschungen könnten sich mit der Frage beschäftigen, ob die hier dargestellten Regellisten auch für die Abbildung von Berechnungsschemen geeignet sind. Insbesondere im Steuer- und Pensionsrecht sind ziemlich komplexe Berechnungen für die Bestimmung der Rechtsfolgen notwendig, vgl. die Analyse in Abschnitt 11.9. Die Normen bilden dabei gemeinsam ein Berechnungsschema, dem gefolgt werden muss, um zum Endergebnis zu gelangen. In vielen Fällen gilt es dabei, zunächst einen Grundanspruch zu berechnen, und erst wenn dieser gegeben ist, macht es Sinn, etwaige Erhöhungen etc. zu prüfen.

Diese Struktur erinnert gewissermaßen an die oben dargestellte Formalisierung von Sanktionsnormen, die erst dann zu berücksichtigen sind, wenn die primäre Norm verletzt wurde. Freilich liegt im Fall einer Anspruchsberechnung keine Normverletzung vor, sodass die Regelliste eine andere Semantik aufweist, die im Fall von Berechnungen durch das Interface `Formula` repräsentiert werden könnte.

Aufbauend auf diesen Überlegungen könnte untersucht werden, ob die Vorgabe der Reihenfolge der Normprüfung im Sinne eines Fallprüfungsschemas vorteilhaft wäre. Die automatisierte Entscheidung könnte vereinfacht und besser nachvollziehbar werden, da die Reihenfolge der Normprüfung mehr jener des Juristen entspricht. Regellisten könnten daher zu einem abstrakten Fallprüfungsschema ausgebaut werden.

13.7. Verwandte Arbeiten

13.7.1. JBoss Drools

JBoss Drools[209] ist eine Business Rule Engine, die unter der „Apache Software License 2.0“, einer Open-Source-Lizenz, veröffentlicht wurde. Die grundlegende Datenstruktur, auf der Drools operiert, sind sogenannte „Produktionsregeln“. Seit Version 6 basiert Drools auf einem monotonen Algorithmus namens „PHREAK“, der Vorwärts- und Rückwärtsverkettung beherrscht (Red Hat, 2013).

[209] Mit „Drools“ ist an dieser Stelle „Drools Expert“ gemeint, die Rule Engine der Drools Plattform.

Obwohl Drools in der Lage ist, mit tausenden von Regeln umzugehen, und erlaubt, Regeln in einem deklarativen Stil zu erstellen, ist es nur bedingt für den juristischen Bereich geeignet. Erstens unterstützt Drools nur monotones Schließen und kommt daher nicht mit unvollständigen Daten zurecht. Zweitens ist das temporale Modell von Drools nicht in der Lage, die zeitlichen Dimensionen des Rechts abzubilden, und müsste demnach erweitert werden, um mit Änderungen der Rechtslage im Lauf der Zeit umzugehen.

Verglichen mit Drools basiert rOWLer auf einem nicht-monotonen Modell des Schließens. Das formale Modell von rOWLer kann die zeitlichen Dimensionen von Rechtsnormen adäquat repräsentieren und darüber hinaus mehrere Fassungen (Versionen) eines Gesetzes verwalten.

Drools ist eine mächtige Rule Engine mit vielen Features, was eine gewisse Komplexität des Codes bedingt. Anpassungen und Erweiterungen von Drools, um die Engine für juristische Probleme zu optimieren, erscheinen schwierig. Die Architektur von rOWLer ist von vornherein optimiert, um Normen formalisieren zu können und verzichtet auf nicht benötigte Features. Daraus resultiert ein klareres und einfacheres System, das vorzüglich erweitert werden kann.

13.7.2. SPINdle

SPINdle[210] (Lam & Governatori, 2009) ist eine andere Open-Source-Rule-Engine, die „defeasible logic“ und auch „modal defeasible logic“ (Governatori & Rotolo, 2008) beherrscht. Im Gegensatz zu Drools, das auf einem monotonen Algorithmus basiert, verwendet SPINdle einen nicht-monotonen Algorithmus und ist damit in der Lage, die anfechtbare Schlussfolgerung mit Theorien[211], bestehend aus tausenden von Regeln, durchzuführen.

Zeitpunkte und Intervalle werden von SPINdle grundsätzlich unterstützt, es ist jedoch nicht in der Lage, mit den temporalen Dimensionen von Rechtsnormen (Inkraftstehen, Effektivität, Anwendbarkeit) umzugehen. Die Grundlage von rOWLer ist hingegen ein ausgeklügeltes Versionierungsmodell, das mit Änderungen umgehen kann und temporale

[210] Die vollständige Bezeichnung lautet „The SPIN Defeasible Logic Reasoner“.

[211] Eine „Theorie“ im Sinne von SPINdle ist ein Tripel (F, R, >), wobei F und R eine abgeschlossene Menge von Fakten bzw. Regeln sind, und > ist eine Superioritätsrelation auf R.

Schlussfolgerungen ermöglicht, um die zu einem bestimmten Zeitpunkt anwendbaren Rechtsvorschriften festzustellen.

Die Regeln in SPINdle basieren stark auf Literalen. Vereinfacht gesagt ist die Konsequenz einer Regel ein Literal oder dessen Negation. Um verschiedene Normarten zu formalisieren, wird jedoch eine Repräsentationsform benötigt, die komplexere Rechtsfolgen zulässt, wie beispielsweise die Durchführung einer Berechnung oder die Inklusion anderer Rechtsnormen im Fall der Auflösung von Referenzen. Aus diesem Grund unterstützt rOWLer ein ausdrucksstärkeres Objektmodell, das unterschiedliche Arten von Regeln zulässt, die von entsprechenden Strategy-Implementierungen verarbeitet werden.

Wie auch SPINdle basiert rOWLer auf einer modularen Architektur, die zahlreiche Erweiterungsmöglichkeiten bietet. Beispielsweise könnte SPINdle als Rule Engine in rOWLer eingebettet werden, um die Schlussfolgerung durchzuführen. In diesem Fall müsste für SPINdle eine Integrationsschicht entwickelt werden, damit SPINdle das komplexere Objektmodell von rOWLer verarbeiten kann. Die Integration von SPINdle hätte den Vorteil, das dessen auf einer soliden Basis beruhende, Algorithmus zur Durchführung der Inferenz verwendet werden könnte.

13.7.3. Carneades

Das Carneades Argumentation Framework ist sowohl ein formales mathematisches Argumentationsmodell als auch eine Software zur Evaluierung, Konstruktion und Visualisierung von Argumenten (Gordon, 2010a). Argumente können dabei aus Ontologien, Regeln und Fällen konstruiert werden (Gordon, 2011a, 2011b).

Das Argumentationsmodell basiert auf den neuesten Erkenntnissen auf dem Gebiet der Argumentationstheorie, beschrieben von Walton (2006) sowie von Gordon und Walton (2009a), und wendet Beweismaße an, um die Akzeptanz von Aussagen im Einzelfall zu prüfen (Bench-Capon, Dunne, Gordon, Prakken, & Walton, 2007). Carneades ist damit in der Lage, den Benutzer zu informieren, ob eine Behauptung ein bestimmtes Beweismaß im Lichte der von den Parteien vorgelegten Beweise und anderen Argumenten erfüllt. Der Benutzer muss dabei nicht die mathematischen Details kennen, um Carneades verwenden zu können (Gordon & Walton, 2006).

Im Moment fehlt es Carneades an einer Untrstützung der temporalen Dimensionen des Rechts und an deontischen Operatoren (Ceci, 2013, S. 199 ff.). Im Gegensatz zu Carneades ist das formale Modell von rOWLer für die automatisierte Beurteilung einfacher Fälle optimiert, die weitgehend „mechanisch“ entschieden werden können (Hart, 2009). Das Modell basiert jedoch auf einer soliden theoretischen Grundlage zur Repräsentation der temporalen Dimensionen des Rechts und kann mit Änderungen des Gesetzes im Lauf der Zeit umgehen.

13.7.4. Temporal Defeasible Logic

Die sogenannte „Temporal Defeasible Logic“ (TDL) ist eine Erweiterung der Defeasible Logic (Johnston & Governatori, 2003; Nute, 2003), um temporale Informationen zu repräsentieren. Grundsätzlich existieren intervallbasierte (Allen, 1984; Augusto & Simari, 2001; Governatori & Terenziani, 2007) und punktbasierte Ansätze (Governatori, Palmirani, Riveret, Rotolo, & Sartor, 2005; Governatori et al., 2007, 2010; Governatori & Rotolo, 2009), um Schlussfolgerungen mit temporalen Informationen durchzuführen (Bohlen, Busatto, & Jensen, 1998). Die Mehrzahl der Ansätze der Temporal Defeasible Logic basiert auf einem punktbasierten temporalen Modell.

Neben dieser grundsätzlichen Differenzierung, ob die Zeit in Form von Punkten oder Intervallen repräsentiert wird, kann die Forschungslinie der TDL noch nach einem weiteren Kriterium unterschieden werden, nämlich ob das logische Modell Änderungen der Rechtsordnung repräsentiert (Governatori, Palmirani, et al., 2005; Governatori et al., 2007; Governatori & Rotolo, 2009) oder die Logik verwendet wird, um temporale Eigenschaften der Fakten des Sachverhalts und der Normen zu repräsentieren (Governatori et al., 2010; Governatori & Terenziani, 2007; Rubino & Rotolo, 2009). Im zweiten Fall werden die Regeln auf einen Fall angewendet, und die Rechtsfolgen zu verschiedenen Zeitpunkten ermittelt.

Die erste Forschungslinie der TDL beschäftigt sich mit der Repräsentation von Änderungen der Rechtsordnung im Lauf der Zeit. Die Logik wird dabei eingesetzt, um die Rechtswirkungen einer Norm nach deren Aufhebung zu blockieren. Dabei wird zwischen rückwirkenden Modifikationen (*ex tunc*) und solchen, die ab dem Zeitpunkt der Aufhebung wirken (*ex nunc*), unterschieden.

Das temporale Modell von rOWLer ist nicht konzipiert, um die Änderungen von Normen im Lauf der Zeit, d.h. die Novellen, logisch zu repräsentieren, sondern geht vielmehr von konsolidierten Fassungen des Gesetzes aus. Es wird damit ausschließlich die konsolidierte Fassung einer Gesetzesnovelle in das formale Modell von rOWLer übertragen und nicht die Novelle selbst.

Die zweite Forschungslinie der TDL befasst sich mit logischen Modellen, die in der Lage sind zu ermitteln, welche Rechtsfolgen zu einem bestimmten Zeitpunkt ableitbar sind. Die Prüfung des zeitlichen Geltungsbereichs und die materiellrechtliche Subsumtion werden dabei gemeinsam durchgeführt. Es liegt jedoch nicht im Fokus dieser Forschungen zu ermitteln, welche Normen *in abstracto* in zeitlicher Hinsicht auf einen Sachverhalt anwendbar sind. Es ist zu bemerken, dass diese Forschungslinie nicht zwischen der Behandlung von interner und externer Zeit (siehe Abschnitt 11.5.3.) von Normen unterscheidet.

Der Schwerpunkt des temporalen Modells von rOWLer liegt auf der Ermittlung der auf einen Fall anwendbaren Normen zu einem bestimmten Zeitpunkt. Das temporale Modell befasst sich demnach mit der externen zeitlichen Dimension von Rechtsnormen, die deren Lebenszyklus bestimmt. Nicht im Fokus liegt jedoch die interne Zeit, das sind temporale Angaben, die direkt im Rechtstext der Norm zum Ausdruck gebracht werden. Die temporalen Eigenschaften von Normen und die Subsumtion werden getrennt voneinander behandelt. Inwieweit TDL der zweiten Forschungslinie genutzt werden könnte, um zu ermitteln, welche Normen zu einem bestimmten Zeitpunkt *in abstracto* anwendbar sind, bietet Raum für zukünftige Forschungen.

Das temporale Modell von rOWLer basiert nicht auf einer vollständigen Logik. Die TDL ist daher ausdrucksstärker und weist eine formale Semantik auf. Dafür ist das Modell von rOWLer simpler und damit besser zu verstehen.

13.7.5. Norma-System

Das Norma-System (Palmirani & Benigni, 2007) ist ein juristisches Informationssystem, das den Rechtserzeugungsprozess und den gesamten Lebenszyklus eines legistischen Dokuments unterstützt. Sämtliche Schritte des Rechtserzeugungsverfahrens vom Entwurf, der

Überarbeitung, der Konvertierung in XML, der Archivierung, der Abfrage bis zur Veröffentlichung werden begleitet. Das Norma-System basiert auf dem in Palmirani und Brighi (2006) entwickelten temporalen Modell zur Verwaltung legislativer Dokumente.

Das System kann mit rückwirkenden Modifikationen umgehen und diese in seiner Datenbank speichern. Norma-System integriert das Frontoffice mit dem Backoffice und kann aber auch als Content-Management- und Document-Management-System eingesetzt werden.

Insgesamt geht der Funktionsumfang von rOWLer über jenen des Norma-Systems hinaus. In diesem Sinne unterstützt rOWLer nicht nur die temporalen Dimensionen des Rechts, sondern ist vielmehr Rule Engine und flexibles Framework zugleich. Der Entwurf von legistischen Dokumenten und das Dokumentenmanagement gehören jedoch nicht zu den Aufgaben von rOWLer.

Das temporale Modell von rOWLer ist nicht bloß auf Dokumente, sondern auf die Versionierung von formalen Regeln spezialisiert, und unterstützt wie das Norma-System alle temporalen Dimensionen des Rechts. Ferner kann rOWLer die auf einen Fall anwendbaren Normen automatisiert ermitteln. Flankiert wird das temporale Modell von einer soliden Namenskonvention und einem Mechanismus zur Behandlung von Verweisungsnormen, um eine bestimmte Fassung eines Gesetzes bzw. einer Regel im Lauf der Zeit adressieren zu können.

13.7.6. Akoma Ntoso

Akoma Ntoso ist ein Standard zur Modellierung von legistischen Dokumenten, aber auch Urteilen. Zu diesem Zweck wird ein XML-Schema verwendet, mit dessen Hilfe der Rechtstext mit semantischem Markup versehen werden kann. Ebenso definiert Akoma Ntoso ein Schema zum Aufbau von Verweisen auf Dokumente in einer bestimmten Fassung (Francesconi, 2011).

Die Syntax der im temporalen Modell verwendeten IRIs zur Identifikation und Adressierung von formalen Ressourcen (Gesetzen und Regeln) basiert auf der Akoma-Ntoso-Syntax für Verweise. Die Syntax der IRIs wurde aber für die gegenständlichen Zwecke vereinfacht. Hinsichtlich der temporalen Attribute wurde die Syntax in zweifacher Hinsicht erweitert: (1)

Der Beobachtungszeitpunkt der Rechtsordnung kann spezifiziert werden, und (2) der Typ des Versionsdatums angegeben werden, um dessen Semantik klarzustellen.

Durch die Möglichkeit, den Beobachtungszeitpunkt anzugeben, eignet sich die entwickelte Syntax auch, um historische Gesetze vor der rückwirkenden Modifikation zu selektieren.

Die elektronische Dokumentensammlung von rOWLer könnte mit Akoma Ntoso realisiert werden, nicht zuletzt da die auf IRIs basierenden Verweismechanismen zueinander kompatibel sind.

13.7.7. LegalRuleML

LegalRuleML (Athan et al., 2013b) ist eine auf XML basierende Sprache und Erweiterung von RuleML, die es ermöglicht, Normen in Form von formalen Regeln zu modellieren. Neben grundlegenden Features wie Isomorphismus und Anfechtbarkeit (defeasibility) können Regeln auch mit temporalen Parametern versehen werden.

Die Semantik des temporalen Modells von rOWLer ist kompatibel zu LegalRuleML. Sämtliche in LegalRuleML vorhandenen temporalen Attribute (force, efficacy, applicability) werden unterstützt.

Mithilfe einer entsprechenden Erweiterung könnten in LegalRuleML modellierte Regeln eingelesen und vom temporalen Modell verarbeitet werden. Derzeit existiert noch keine Implementierung, die LegalRuleML unterstützt. Das skizzierte theoretische Modell stellt eine exzellente Grundlage für eine solche Implementierung zur Verfügung.

13.8. Diskussion

In diesem Kapitel wurde rOWLer dargestellt, eine Rule Engine, die für die Formalisierung von Rechtsnormen entworfen und optimiert wurde. Kern der Architektur sind ein temporales Modell und ein ausdrucksstarkes und flexibles Objektmodell, das zahlreiche Erweiterungsmöglichkeiten durch Interfaces bietet.

Verglichen mit aktuellen Systemen im öffentlichen Sektor beruht rOWLer auf einer soliden rechtstheoretischen Grundlage und unterstützt neben temporalen Aspekten die anfechtbare Schlussfolgerung. Das Modell fördert dabei den Isomorphismus von formalen Regeln und den Rechtsquellen. Aktuellen Systemen, die in der Praxis verwendet werden, fehlt ein

ausgereiftes temporales Modell, um mit Änderungen umzugehen, was zu erhöhter Komplexität des Codes und in weiterer Folge zu Wartungsproblemen führt.

Um mit Änderungen des Rechts im Lauf der Zeit fertig zu werden, wurde für rOWLer ein ausgereiftes temporales Modell entwickelt, das die zeitlichen Dimensionen von Rechtsnormen formalisiert und darüber hinaus in der Lage ist, die auf einen Fall anwendbaren Normen zu einem bestimmten Zeitpunkt zu ermitteln. Durch die Möglichkeit der Angabe eines Beobachtungszeitpunktes ist das Modell auch in der Lage, mit retroaktiven Modifikationen umzugehen, und kann die historische und aktuelle Rechtslage nach der Novelle bereitstellen.

Im Moment ist rOWLer als ein Einzel-Agentensystem konzipiert und die Inferenzmaschine ist optimiert, um mit Rechtsvorschriften mit einem mathematischen Inhalt umzugehen, also mit „einfachen" Fällen[212] in der Terminologie von Hart (2009).

Vom konkreten Regeltyp wurde in rOWLer mithilfe des Interface `Rule` abstrahiert. Das Model von rOWLer ist damit flexibel und kann in der Zukunft erweitert werden, sodass auch „schwierige" Fälle behandelt werden können, z.B. indem dem Juristen verschiedene Entscheidungsalternativen vorgeschlagen werden oder fortgeschrittene Argumentationssysteme wie Carneades (Gordon, 2010a) eingesetzt werden.

Normen haben Eigenschaften, wie das Datum des Inkraftretens, die Jurisdiktion und die Autorität, die die Norm erlassen hat. Aus diesem Grund können Normen nicht adäquat durch materiale Implikationen der Prädikatenlogik repräsentiert werden (Ceci, 2013, S. 152). Um dieses Problem zu vermeiden, werden Normen im formalen Modell von rOWLer durch Objekte reifiziert. Damit ist es möglich, eine Norm als Objekt mit Eigenschaften zu behandeln und Aussagen über diese zu machen. Das Modell unterscheidet damit, in Einklang mit rechtstheoretischen Überlegungen, zwischen der Norm und deren Aussage (Breuker et al., 2006, S. 173). Regeln sind meines Erachtens, in Übereinstimmung mit Bench-Capon (1994a), eine wichtige Form der Repräsentation von Normen und schaffen die Grundlage für

[212] „Einfache" Fälle können weitgehend „mechanisch" entschieden werden, indem das Ergebnis aus den Regeln und den Fakten abgeleitet wird. „Schwierige" Fälle für Hart sind solche, wenn die Fakten in das „Zwielicht" der Bedeutung der Begriffe des Tatbestands fallen. Solche Fälle machen es erforderlich, dass der Richter von dem ihm eingeräumten Ermessen Gebrauch macht (Hart, 2009).

praktisch nützliche Expertensysteme. Die Regelschicht eines formalen Modells kann dabei wichtige Ergebnisse liefern, die von einer höheren Schicht genutzt werden können.

Das interne Modell von rOWLer ist kompatibel zu LegalRuleML, einem aufkommenden Standard zur Modellierung von Normen. Die Entwicklung von rOWLer komplementiert damit Anstrengungen, einen akzeptierten Standard für die Regelschicht des Semantic Web Stacks zu schaffen. Durch die Unterstützung von LegalRuleML können vorhandene Werkzeuge, wie der RAWE-Editor (Palmirani, Cervone, Bujor, & Chiappetta, 2013; Palmirani, Cervone, & Bujor, 2013), zur Modellierung verwendet und zukünftige Entwicklungen und Verbesserungen auf dem Gebiet der Computerlinguistik zur automatisierten Regelextraktion genutzt werden. Den verfügbaren NLP-Methoden[213] (de Maat & Winkels, 2010; Schweighofer, Rauber, & Dittenbach, 2001; Schweighofer & Winiwarter, 1993) zur semantischen Extraktion von Normen fehlt es bislang allerdings an der notwendigen Präzision, sodass die Erstellung computergestützter Modelle des Rechts von einem Experten durchgeführt werden muss (Palmirani, Cervone, Bujor, et al., 2013, S. 4).

In der Zukunft muss das konzeptionelle Modell weiterentwickelt werden, insbesondere im Hinblick auf die Repräsentation von Normen und den nicht-monotonen Algorithmus zur Schlussfolgerung. Für die Durchführung der Schlussfolgerung wird ein (logischer) Algorithmus benötigt, sofern nicht ein vorhandenes System wie SPINdle verwendet werden kann. Ferner muss an der konzeptuell einwandfreien Integration von OWL 2 und den Regeln gearbeitet werden. Das ist jedoch keine einfache Aufabe, da OWL 2 (wie auch OWL) der Open-World Assumption folgt und regelbasierte Sprachen grundsätzlich der Closed-World Assumption folgen. Damit unterscheidet sich die formale Semantik von OWL 2 und von Regeln erheblich, eine formal einwandfreie Kombination der Semantiken ist daher ein schwieriges Unterfangen (Eiter, Ianni, Polleres, Schindlauer, & Tompits, 2006; Hitzler & Parsia, 2009).

Obwohl das temporale Modell von rOWLer intern mit Abstraktionen arbeitet, ist es für die Versionierung von formalen Regeln optimiert. Ein ganzheitlicher Ansatz muss jedoch auch die anderen Schichten, wie insbesondere das ontologische Modell, einbeziehen. In dieser Hinsicht könnten in Zukunft vorhandene Ansätze zur Repräsentation zeitlicher Aspekte in

[213] „NLP“ steht für Natural Language Processing. Eine ausgezeichnete Einführung in NLP geben Jackson und Moulinier (2007).

Ontologien evaluiert und das ontologische Modell von rOWLer dahingehend verbessert werden (Peroni, Shotton, & Vitali, 2012; Scheuermann, Motta, Mulholland, Gangemi, & Presutti, 2013).

Damit ein Expertensystem in seiner Gesamtheit mit mehreren Fassungen eines Gesetzes umgehen kann, ist überdies erforderlich, dass nicht nur das formale Modell mehrere Versionen einer Norm repräsentieren kann, sondern auch, dass Datenstrukturen (Objekte) und die Benutzeroberfläche damit zurechtkommen. Der zweite Aspekt wurde jedoch in diesem Diskurs bewusst ausgeklammert.

Die Machbarkeit des bislang mit Ausnahme einiger weniger Klassen und Interfaces, die bereits implementiert wurden, rein theoretischen Modells muss evaluiert werden, indem ein objektorientierter Prototyp in Java entwickelt wird. Im Zuge der Entwicklung wird sich auch zeigen, wie weit vorhandene Systeme und Technologien nützlich sind.

14. Konklusion

14.1. Zusammenfassung

Diese Dissertation untersuchte die Möglichkeiten der Formalisierung von Normen und der automatisierten Entscheidungsfindung im Kontext des Forschungsgebiets Künstliche Intelligenz und Recht. Der Schwerpunkt des Diskurses wurde dabei auf die Ontologie, ein spezielles computergestütztes Modell, gelegt und die Eignung von OWL 2 zur Durchführung (einfacher) Schlussfolgerungen bewertet. Anhand eines ausgewählten Gesetzes wurde gezeigt, dass – entgegen der gelegentlich vertretenen Meinung – Beschreibungslogik nicht ausreicht, um selbst einfache juristische Problemstellungen zu lösen. Basierend auf diesen Erkenntnissen wurde die Untersuchung punktuell vertieft, um schließlich das Konzept eines neuartigen Expertensystems, genannt „rOWLer“ zu entwicklen, das die Ausdrucksstärke von Ontologien und Regeln kombiniert und in der Lage ist, Änderungen der Rechtsgrundlage im Lauf der Zeit effizient zu handhaben.

Dreizehn aufeinander aufbauende Kapitel mit unterschiedlichen Schwerpunkten begleiten den Leser und führen den Diskurs hin zur Konklusion der Dissertation. Die Motivation und die Gründe zur Verfassung dieser Dissertation sowie die zu untersuchenden Hypothesen und Forschungsfragen werden ausführlich in Kapitel 1 beleuchtet. Kapitel 2 steckt den Rahmen ab, in den sich Expertensysteme der Verwaltung einfügen müssen, und gibt dazu eine Einführung in das Thema E-Government. Der aktuelle Stand der Verwaltungsautomatisierung und offene Probleme gängiger Ansätze zur Formalisierung von Normen werden in Kapitel 3 präsentiert, um die Darstellung des Kontexts der Dissertation abzurunden.

Kapitel 4 markiert den Beginn der Untersuchung der theoretischen Grundlagen der Formalisierung und automatisierten Entscheidungsfindung. Insbesondere erfolgen eine Begriffsbestimmung der Formalisierung und die Untersuchung von deren Grenzen im Hinblick auf die Repräsentationen von Normen. Schließlich wird argumentiert, dass die vorhandenen Hindernisse nicht dadurch gelöst werden können, indem der Gesetzgeber selbst ein formales Modell von Gesetzen erlässt. Diesem Ansatz stehen schwerwiegende rechtstheoretische und verfassungsrechtliche Bedenken entgegen. Mit den Grundlagen der Wissensrepräsentation befasst sich Kapitel 5 und stellt verschiedene Möglichkeiten der Repräsentation von juristischem Wissen dar, dabei wird im Zuge der Untersuchung im Detail

auf offene Probleme eingegangen. Das daran anschließende Kapitel 6 beschäftigt sich mit dem Semantic Web und Ontologien und schafft so die wichtigste Grundlage für die Erstellung einer Ontologie zur Repräsentation von Normen und Unterstützung der juristischen Entscheidungsfindung. Die Anforderungen an Sprachen zur Formalisierung von Verwaltungsrecht werden in Kapitel 7 thematisiert. Dabei wird vorausgesetzt, dass Normen durch formale Regeln, bestehend aus Antezedens und Konsequenz, repräsentiert werden können. Das Kapitel bietet eine vorzügliche Grundlage für die Evaluierung von Sprachen zur Modellierung von Normen. Die Überleitung vom theoretischen Teil zu den praktischen Modellierungsversuchen mit Java und OWL erfolgt durch Kapitel 8, in dem ein Konzept eines Expertensystems zur semi-automatisierten Vollziehung skizziert wird.

Stellvertretend wird das Kriegsopferversorgungsgesetz ausgewählt und in Kapitel 9 mit Java formalisiert. Dabei wird insbesondere auf eine möglichst isomorphe Darstellung der Normen im Programmcode geachtet und es werden verschiedene Möglichkeiten diskutiert, um Normen und Ausnahmebestimmungen zu repräsentieren. Das in Java implementierte Modell zeigt zwar grundsätzlich recht gute Ergebnisse, weist jedoch – trotz Berücksichtigung der Theorie – auch erhebliche Unzulänglichkeiten auf: (1) Die Repräsentation der Normen ist nicht isomorph zur Rechtsgrundlage, (2) es fehlt an einem abstrakten Konzept zur Behandlung der Beziehung zwischen generellen und speziellen Normen, und (3) zeitliche Aspekte des Rechts können nicht adäquat modelliert werden. Insgesamt kann mit Java eine nur sehr oberflächliche Repräsentation des Rechts erreicht werden. Das in Java geschriebene Modell der Normen kann als Referenz zur Bewertung experimenteller Ansätze dienen.

Kapitel 10 widmet sich der Erstellung einer Ontologie des Kriegsopferversorgungsgesetzes, modelliert mit der Web Ontology Language (OWL 2). Ziel der Untersuchung ist es, die Ausdrucksstärke von OWL auszureizen und zu zeigen, ob ein ausschließlich auf Beschreibungslogik basierender Ansatz zur Automatisierung juristischer Entscheidungsfindung zielführend ist. Für die Evaluierung werden zwei Ontologien entwickelt, wobei eine auf der LKIF-Core Ontologie basiert.

Das Zwischenergebnis der Dissertation, die Erkenntnisse aus der Formalisierung von Normen mit Java und OWL 2, wird in Kapitel 11 dargestellt. Das Kapitel beschränkt sich jedoch nicht bloß auf die Zusammenfassung der Ergebnisse, sondern vertieft punktuell das theoretische Wissen über die Formalisierung, um so eine Grundlage für die Entwicklung eines eigenen

Ansatzes zu schaffen. Die Abhandlung geht dabei eingehend auf die temporalen Dimensionen des Rechts ein und analysiert vorhandene Literatur auf dem Gebiet der Künstlichen Intelligenz und Recht. Ein weiterer Schwerpunkt wird auf die Darstellung der anfechtbaren Schlussfolgerung[214] gelegt und ferner argumentiert, dass die Reihenfolge der Normprüfung bei der Subsumtion ein wichtiger Aspekt ist, dem bei der Automatisierung juristischer Entscheidungsfindung bisher zu wenig Beachtung geschenkt wurde.

Einen Überblick über die State-of-the-Art-Technologien gibt Kapitel 12, die im Lichte der im vorherigen Kapitel ausgeführten (erweiterten) Anforderungen diskutiert werden. Die Untersuchung zeigt, dass der Repräsentation der temporalen Eigenschaften des Rechts zu wenig Beachtung geschenkt wurde. Keine der dargestellten Technologien ist in der Lage, mit mehreren Fassungen eines Gesetzes umzugehen und die auf einen Fall in zeitlicher Hinsicht anwendbaren Normen zu ermitteln. Dies bestätigt, dass tatsächlich Bedarf nach einem formalen Modell besteht, das mit den zeitlichen Dimensionen des Rechts umgehen kann.

Den wichtigsten wissenschaftlichen Beitrag der Dissertation und das Ende des Diskurses präsentiert Kapitel 13. Auf den im Laufe der Untersuchung gewonnenen Einsichten wird das Konzept einer Rule Engine, genannt „rOWLer“, dargestellt, die anfechtbare Schlussfolgerung beherrscht und auf einem soliden temporalen Modell basiert, das in der Lage ist, mit Änderungen der Rechtslage im Lauf der Zeit umzugehen. Die Architektur von rOWLer basiert auf mehreren Schichten, die unterschiedliche Sichtweisen auf das Recht repräsentieren: (1) Die Prozessschicht formalisiert den Dialog zwischen Antragsteller und Behörde, (2) die Regelschicht repräsentiert Normen durch anfechtbare Regeln und führt die Schlussfolgerung durch, und (3) die ontologische Schicht modelliert Begriffe, Fakten, Parteien und Beweise und führt einfache Schlussfolgerungen auf der Wissensbasis aus. Die drei Hauptschichten werden von einer elektronischen Dokumentensammlung flankiert, die eine elektronische Repräsentation des Rechtstexts zur Verfügung stellt. Das System ist modular aufgebaut und kann flexibel erweitert werden, um an unterschiedliche Anforderungen angepasst zu werden. Insofern kann das System auch als flexibles Framework zur Entwicklung juristischer Expertensysteme verstanden werden.

---

[214] Auch genannt „anfechtbares Schließen“.

14.2. Wichtigste Ergebnisse

Ein Schwerpunkt der Dissertation beschäftigte sich mit der Frage, ob Ontologien (in OWL 2) ausdrucksstark genug sind, um Normen zu repräsentieren und die rechtliche Beurteilung von Fällen zu unterstützen.

Diese Frage ist nicht unbedingt neu, wurde jedoch von Forschern auf dem Gebiet Künstliche Intelligenz und Recht kontrovers diskutiert: Auf der einen Seite des Spektrums steht die Meinung, dass OWL – zumindst für manche Bereiche – ausreicht, um die rechtliche Beurteilung durchzuführen (Boer, 2009; van de Ven, Breuker, et al., 2008; van de Ven, Hoekstra, et al., 2008). Dem steht die gegenteilige Meinung auf der anderen Seite des Spektrums entgegen, wonach Ontologien ohne Regeln, die auf die Begriffe angewendet werden, im Wesentlichen bloß eine Taxonomie konstituieren (Athan et al., 2013a, 2013b; Sartor, 2005, 2009b). Die Ursache dafür wird insbesondere in der „Anfechtbarkeit" juristischer Schlussfolgerungen und Normen ausgemacht (Gordon et al., 2009; Gordon, 1994b).

Zur Klarstellung dieser Kontroverse für den Bereich des Verwaltungsrechts wurde ein ontologisches Modell des Kriegsopferversorgungsgesetzes erstellt, dass die erforderliche Detailtiefe aufweist, um die automatisierte Beurteilung von Fällen zu ermöglichen. Der Inhalt des Kriegsopferversorgungsgesetzes besteht vor allem aus der Berechnung monetärer Werte und ähnelt damit dem Steuer- und Pensionsrecht. Dies rechtfertigt die Annahme, dass sich die Erkenntnisse der Untersuchung auch auf diese Rechtsbereiche übertragen lassen und damit breitere Relevanz besitzen.

Die Ontologie besteht aus mehreren Modulen und enthält die Fakten des Falls sowie die formale Repräsentation der Normen in Form von Generic Cases (GC) (Valente, 1995). Ein Generic Case beschreibt die Situation, die der Tatbestand einer Norm zum Ausdruck bringt. Die Inferenzmaschine kann so ermitteln, welche Normen auf einen Sachverhalt anwendbar sind. Damit ist das ontologische Modell grundsätzlich ausdrucksstark genug, sodass die Inferenzmaschine ableiten kann, welche Leistungen dem Anstragsteller zustehen. Es ist jedoch nicht möglich, mithilfe der Beschreibungslogik Berechnungen durchzuführen. Aus diesem Grund kann der Betrag einer Leistung nicht ermittelt werden.

Basierend auf der beschreibungslogischen Semantik können nur deduktive Schlussfolgerungen von der Inferenzmaschine (Modus Ponens) durchgeführt werden. Wie

ausführlich in Kapitel 10 diskutiert wurde, können nicht alle Aspekte der juristischen Entscheidungsfindung von einem monotonen Schlussfolgerungsmodell repräsentiert werden. Es kann nur sehr rudimentär das Verhältnis zwischen genereller und spezieller Norm modelliert und berücksichtigt werden. Die speziellere Norm wird dabei von der Inferenzmaschine als Subklasse der generelleren klassifiziert. Damit entsteht eine Klassenhierarchie zwischen Lex generalis und Lex specialis, wobei die unterste Subklasse die anwendbare Norm repräsentiert. Die Ontologie kann damit nicht direkt beantworten, welche Norm letztendlich auf einen Sachverhalt anwendbar ist, sondern die Klassenhierarchie muss von der Software entsprechend interpretiert werden, um das korrekte Ergebnis abzuleiten. Der Ansatz ist jedenfalls ungeeignet, das normative Muster „Ausnahme von einer Ausnahme“ zu repräsentieren, das unter Umständen zum Wiederaufleben der ursprünglich anwendbaren generellen Bestimmung führt. Die Klassenhierarchie bietet für solche Konstellationen zu wenige Informationen um zu beurteilen, welche Norm letztendlich anwendbar ist.

Insgesamt lässt sich feststellen, dass Ontologien und eine ausschließlich auf Beschreibungslogik basierende deduktive Schlussfolgerung nicht ausreichen, um die rechtliche Beurteilung durchzuführen. Ontologien sollten meines Erachtens zur Strukturierung komplexer Entitäten und deren Beziehungen verwendet werden. Der Einsatz von Ontologien kann maßgeblich zum besseren Verständnis eines Problembereichs beitragen. Eine Inferenzmaschine kann das in der Ontologie modellierte Wissen für komplexere Schlussfolgerungen mit Regeln oder für den Einsatz von Argumentationssystemen vorbereiten.

Basierend auf den aus der Modellierung von Normen in Java und OWL 2 gewonnenen Erkenntnissen wurde das Konzept einer Rule Engine entwickelt, die auf einem soliden temporalen Modell basiert und anfechtbare Schlussfolgerung beherrscht. Im Gegensatz zu den gängigen Systemen handelt es sich um ein wissensbasiertes System, was bedeutet, dass die Wissensbasis vom Algorithmus getrennt ist. Die Rule Engine kombiniert die Ausdrucksstärke von OWL 2 und Regeln, es handelt sich demnach um ein hybrides System. Die Ontologie dient der Strukturierung und Aufbereitung der Daten, um diese für komplexere Schlussfolgerungen in der Regelschicht vorzubereiten. Die Vorzüge von Ontologien kommen insbesondere bei der Strukturierung komplexer Sachverhalte zur

Geltung. Aus der Ontologie kann die Inferenzmaschine neues Wissen ableiten, womit die Wissensbasis insgesamt „intelligenter“ als herkömmliche relationale Datenbankmodelle ist. Das in der Ontologie modellierte Wissen eignet sich vorzüglich zur Wiederverwendung. Durch die formale Semantik von OWL 2 wird gewährleistet, dass eine korrekt implementierte Inferenzmaschine unter allen Umständen dieselben Konklusionen ableitet.

Die Rechtsnormen selbst werden durch formale Regeln in der auf der Ontologie aufbauenden Regelschicht repräsentiert. Das Objektmodell von rOWLer ist mit der Semantik von LegalRuleML kompatibel. Es können sowohl implizite (materielle Derogation) als auch explizite Ausnahmen (formelle Derogation) zwischen Normen dargestellt werden. Damit kann die anfechtbare Schlussfolgerung durch Ausführung eines nicht-monotonen Algorithmus auf dem Objektmodell durchgeführt werden. Durch den Einsatz geeigneter Schnittstellen für Erweiterungen unterstützt rOWLer verschiedene Regelarten, womit auch komplexere Normtypen repräsentiert werden können. Weiters unterstützt das Modell die Repräsentation von alternativen Interpretationen einer Norm. Dies schafft die Voraussetzungen zur Entscheidungsunterstützung, da dem Benutzer verschiedene Entscheidungsmöglichkeiten vorgeschlagen werden können. Das Prinzip des Isomorphismus wird erfüllt, indem eine solide Namenskonvention zur Adressierung unterschiedlicher Fassungen einer Norm zum Einsatz kommt. Aus technischer Sicht werden IRIs für die Identifikation einer bestimmten Version einer Regel verwendet.

Durch das solide temporale Modell und die flexible Architektur von rOWLer sollte der Aufwand für Änderungen der Wissensbasis auf ein Minimum reduziert werden können. Die Entwicklung von rOWLer komplementiert Anstrengungen, einen anerkannten Standard für den Rule Layer des Semantic Web Stacks zu erreichen.

14.3. Zukünftige Forschungen

Obwohl sehr intensive Anstrengungen unternommen wurden, den Einsatz von künstlicher Intelligenz in der Verwaltung zu untersuchen, sind dennoch offene Probleme verblieben, die Inspiration für zukünftige Forschungen bieten.

Die entwickelte Ontologie des Kriegsopferversorgungsgesetzes fokussierte auf die Untersuchung der automatisierten Vollziehung, um den Funktionsumfang gängiger Systeme zu erreichen. In dieser Hinsicht lässt sich die Ontologie noch verfeinern, indem Aktionen und

Abläufe detaillierter dargestellt werden. Beispielsweise könnten die Ereignisse, die zu einer Gesundheitsschädigung geführt haben, modelliert werden, um die Voraussetzungen für die Beurteilung derselben als Dienstbeschädigung zu schaffen. Mit dieser Feststellung ist die Beantwortung schwieriger Fragen verbunden, unter anderem, ob eine selbst zugefügte Verletzung als Dienstbeschädigung zu werten ist.

Für die Beantwortung solcher Fragestellungen ist die Abwägung von Argumenten für und gegen eine Entscheidung erforderlich. In diese Beurteilung fließen auch Wertungen mit ein, die aus den Zielen des Gesetzes und Moralvorstellungen abgeleitet werden. Man begibt sich damit zwangsläufig auf das Gebiet der computergestützten juristischen Argumentation und der Repräsentation juristischer Argumente. Die Ontologie des Kriegsopferversorgungsgesetzes in ihrer derzeitigen Form ist zu ausdrucksschwach und nicht geeignet, Argumente zu modellieren.

Eine interessante Untersuchung wäre die Repräsentation von Argumenten der Gerichtshöfe des öffentlichen Rechts in einem ontologischen Modell. Die entwickelte Ontologie des Kriegsopferversorgungsgesetzes könnte für ein solches Modell als Grundlage dienen, müsste jedoch umfangreich erweitert werden. Basierend auf dieser Ontologie könnte das Carneades Argumentation Framework (Gordon & Walton, 2006; Gordon, 2010a) zur Visualisierung und Verarbeitung der Argumente eingesetzt werden. Carneades bietet sich insbesondere deshalb für solch eine Untersuchung an, da es in der Lage ist, Argumente aus einer Ontologie zu konstruieren (Gordon, 2010a, S. 8 f.). Unabhängig davon, welcher Rechtsbereich für diese Untersuchung gewählt wird, sollte darauf geachtet werden, dass das ontologische Modell mit dem Gedanken der Wiederverwendbarkeit konstruiert wird. Damit kann das Modell in einem anderen Rechtsbereich eingesetzt und so stetig verbessert werden.

Eine weitere Möglichkeit, juristische Argumente zu formalisieren, sind auf „Dimensionen und Faktoren“ basierende Modelle (Araszkiewicz et al., 2013; Ashley, 1990; Bench-Capon, 2012; Sartor, 2006b). Solche Modelle könnten evaluiert und mit den Möglichkeiten von Carneades verglichen werden.

Da der Einsatz von Ontologien alleine nicht ausreicht, um die juristische Entscheidungsfindung durchzuführen, wurden temporale Aspekte des ontologischen Modells nicht behandelt. Das bedeutet jedoch nicht, dass temporale Aspekte in Ontologien nicht von Bedeutung wären. Im Gegenteil, die Modellierung von komplexen Abläufen zur

Darstellung von Sachverhalten ist im Recht unerlässlich. Überdies kann sich auch die Bedeutung von Rechtsbegriffen im Lauf der Zeit ändern, sodass ein Versionierungskonzept für die in der Ontologie modellierten Klassen erforderlich erscheint. Dieser Problemaufriss bietet genügend Substanz für zukünftige Untersuchungen, die jedenfalls auf die bereits vorhandenen Erkenntnisse in diesem Bereich eingehen sollten (Klarman et al., 2008; Peroni et al., 2012; Scheuermann et al., 2013).

Das in der Dissertation entwickelte Konzept der Rule Engine „rOWLer“ existiert weitgehend nur in der Theorie. Am detailliertesten wurden die Repräsentation der temporalen Dimensionen des Rechts und die Ermittlung der anwendbaren Normen behandelt. Der zweite Schwerpunkt des Systems, die Durchführung der anfechtbaren Schlussfolgerung, wurde nur in Grundzügen dargestellt. In dieser Hinsicht müsste das Modell jedenfalls verfeinert werden.

Mit Ausnahme einiger weniger Klassen und Interfaces wurde das theoretische Modell noch nicht implementiert. Zur Beurteilung der Machbarkeit und praktischen Eignung sollte ein Prototyp in Java entwickelt werden. Der modulare Aufbau von rOWLer ermöglicht dabei eine Umsetzung in iterativen Zyklen, sodass je nach Verfügbarkeit von Ressourcen einzelne Module implementiert werden können. Begonnen werden könnte mit einem in LegalRuleML modellierten Gesetz, das von der Engine eingelesen und auf einen Fall angewendet wird. Dafür ist zunächst die Implementierung eines geeigneten Parsers für LegalRuleML erforderlich, der die XML-Daten in das interne Objektmodell von rOWLer übersetzt. Am geeignetsten für solche Experimente scheint ein prominentes Gesetz zu sein, wie das Einkommenssteuergesetz (EStG) oder Pensionsberechnungen nach dem Allgemeinen Sozialversicherungsgesetz (ASVG). Die Syntax von LegalRuleML müsste höchstwahrscheinlich erweitert werden, um mathematische Berechnungen zu modellieren, die charakteristisch für die genannten Gesetze sind.

Neben einer geeigneten Repräsentation von Normen, Fällen etc. und einem System (einer Engine) zur Verarbeitung des computergestützten Modells wird eine Methode zur Modellierung von juristischem Wissen benötigt. Die Extraktion von formalen Regeln kann manuell, semi-automatisiert oder vollautomatisiert erfolgen. Der computergestützten Extraktion von Begriffen und Regeln widmeten sich unterschiedlichste Forschungen (Buabuchachart, Metcalf, Charness, & Morgenstern, 2013; de Maat & Winkels, 2010; Le, Le

Nguyen, & Shimazu, 2013; Poudyal & Quaresma, 2012; Šavelka, Grabmair, & Ashley, 2014; R. Winkels & Hoekstra, 2012; Wyner & Peters, 2011; Yoshida et al., 2013). Derzeit existiert jedoch kein Ansatz zur vollautomatisierten Extraktion von Normen, der zufriedenstellende Ergebnisse liefert. Aus diesem Grund scheint eine Methode, die einen semi-automatisierten Ansatz präferiert, am erfolgversprechendsten. Der Jurist könnte beispielsweise von einem grafischen Editor zur Annotation von Rechtstexten bei der Modellierung formaler Regeln angeleitet werden. Je nach Möglichkeit könnte das System den Rechtstext bereits vorverarbeiten und versuchen, Regeln zu extrahieren, die dem Benutzer sodann zur weiteren Bearbeitung vorgeschlagen werden. Das Spektrum der Untersuchung sollte die Methoden zur Extraktion juristischen Wissens sowie Werkzeuge zur Unterstützung der Modellierung umfassen.

Die Dissertation beschäftigte sich, wie oben bereits angemerkt, primär nicht mit der Repräsentation juristischer Argumentation. Dementsprechend wurde die Modellierung von Argumenten als eine Form der juristischen Wissensrepräsentation zwar dargestellt, jedoch nicht weiter vertieft. Die Formalisierung juristischer Argumente mit Ontologien und Regeln bietet reichlich Inspiration für zukünftige Forschungen. Wie bereits vorgeschlagen, könnte das ontologische Modell um die Repräsentation von höchstgerichtlichen Entscheidungen und Argumenten erweitert werden. Die Ontologie sollte nur solche Axiome enthalten, die allgemein wahr und in einem Bereich unbestritten sind. Aus diesem Grund muss eine eingehende Untersuchung juristischer Argumentation im Kontext der Verwaltung auch auf die Repräsentation von Wissen in Form von anfechtbaren Regeln zurückgreifen.

14.4. Abschließende Bemerkungen

In dieser Dissertation wurde ziemlich umfangreich der Einsatz von künstlicher Intelligenz in der Verwaltung untersucht. Der Diskurs entwickelte sich dabei im Lauf der Zeit über den ursprünglichen Schwerpunkt, die Evaluierung von Ontologien zur Unterstützung der automatisierten Vollziehung durch Verwaltungsorgane, hinaus. Infolgedessen fand eine eingehende Auseinandersetzung mit dem zeitlichen Anwendungsbereich von Normen und der anfechtbaren Schlussfolgerung statt, die schließlich zur Konzeption eines neuartigen Expertensystems, genannt „rOWLer“, führte.

Die Arbeit intendiert, das Interesse der Verwaltung zu wecken, sich nicht vor technologischen Innovationen zu verschließen und zugänglicher für die Entwicklungen auf

dem Gebiet der Künstlichen Intelligenz und Recht zu sein. Ein verstärkter Dialog zwischen dem Forschungsbereich Künstliche Intelligenz und Recht und der Verwaltung könnte durchaus für beide Seiten von Vorteil sein. Für die Verwaltung kann der Einsatz innovativer Technologien zu einer Effizienzsteigerung und Erhöhung der Qualität von Expertensystemen führen. Andererseits können die auf dem Gebiet der Künstlichen Intelligenz und Recht entwickelten theoretischen Konzepte durch praktische Anwendung evaluiert werden.

Oftmals sind die von den Forschungen hervorgebrachten Modelle äußerst komplex, sodass deren praktischer Einsatz erschwert wird. Der Fokus der Untersuchungen auf dem Gebiet der Künstlichen Intelligenz und Recht wurde vor allem auf ausgefeilte computergestützte Modelle zur Repräsentation juristischer Argumentation gelegt. Auf den Bedarf der Praxis nach möglichst einfachen Modellen zur Bewältigung der täglichen Aufgaben wurde weitgehend nicht Bedacht genommen[215]. Diese Dissertation ist der erste Schritt, um diese Versäumnisse nachzuholen, indem ein einfaches Expertensystem konzipiert wurde, das die dringendsten Anforderungen der Praxis berücksichtigt.

Trotz potentieller Schwierigkeiten, den der Einsatz innovativer Technologien mit sich bringt, attestiert der Autor die Notwendigkeit der Verbesserung juristischer Expertensysteme in der Verwaltung. Zum einen steigt die Anzahl der Gesetze stetig an, wobei auch Rechtsakte auf europäischer und internationaler Ebene vermehrt bei der rechtlichen Beurteilung zu berücksichtigen sind. Zudem nehmen die Komplexität und der Umfang von Fällen kontinuierlich zu, wobei das Problem zusätzlich dadurch verschärft wird, dass in gleicher Zeit eine größere Anzahl von Fällen bearbeitet werden muss. Expertensysteme können einen wichtigen Beitrag zur Unterstützung des Juristen leisten, indem vorhandenes Wissen besser strukturiert und aufbereitet wird; wenn möglich, können auch Entscheidungsvarianten vorgeschlagen werden.

Die Extraktion und Übersetzung von Wissen in ein computergestütztes Modell muss auch heute noch weitgehend von einem Menschen durchgeführt werden. Die Modellierung von Gesetzen und Fällen etc. ist sehr arbeitsintensiv, sodass nur selten der gewünschte Nutzen durch den Einsatz von Expertensystemen erzielt werden kann. Der Einsatz von

[215] Insofern decken sich die Erkenntnisse mit jenen von Oskamp und Lauritsen (2002), die zum damaligen Zeitpunkt ebenso kein Beispiel für die erfolgreiche Anwendung „echter“ künstlicher Intelligenz in der juristischen Praxis finden konnten.

computergestützten Modellen lohnt sich derzeit nur, wenn eine Vielzahl von Fällen bei geringerem Ressourcenverbrauch vom System erledigt werden können. Die Überwindung dieses sogenannten „Knowledge Acquisition Bottlenecks“ ist meines Erachtens erforderlich, sollen sich juristische Expertensysteme in Zukunft in der Verwaltung und anderen Bereichen stärker durchsetzen.

Aus diesem Grund sollten sich die Anstrengungen der „AI and Law Community“ verstärkt auf diesen Bereich konzentrieren. Ein Durchbruch auf dem Gebiet der Verarbeitung natürlicher Sprache[216], insbesondere in anderen Sprachen als Englisch, scheint derzeit allerdings nicht in Sicht zu sein. Vielversprechender ist die Entwicklung von Werkzeugen, die den Juristen bei der semantischen Annotation von Rechtstexten, wie Gesetzen und Fällen, unterstützen, sodass die intellektuell herausfordernde Aufgabe der Modellierung von juristischem Wissen schneller und effizienter durchgeführt werden kann. Dies schließt jedoch nicht aus, dass Rechstexte gegebenenfalls mithife von Technologien zur Verarbeitung natürlicher Sprache aufbereitet werden, sodass der Jurist das Modell weiter verfeinern kann. In der Tat widmen sich auf dem Gebiet Künstliche Intelligenz und Recht gewichtige Forschungen dieser Aufgabenstellung (Palmirani, Cervone, Bujor, et al., 2013; Šavelka et al., 2014).

Der Erfolg von juristischen Expertensystemen wird maßgeblich von deren Benutzerfreundlichkeit und der Fähigkeit, den Juristen bei der Modellierung von Wissen zu unterstützen, abhängig sein.

[216] Natural Language Processing (NLP).

Literaturverzeichnis

Al-Abdulkarim, L., Atkinson, K., & Bench-Capon, T. J. M. (2014). Abstract Dialectical Frameworks for Legal Reasoning. In *Legal Knowledge and Information Systems, JURIX 2014: The Twenty-Seventh Annual Conference* (S. 61–70). Amsterdam: IOS Press. doi:10.3233/978-1-61499-468-8-61

Allemang, D., & Hendler, J. (2008). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL* (1. Aufl.). Burlington: Morgan Kaufmann.

Allemang, D., & Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL* (2. Aufl.). Burlington: Morgan Kaufmann.

Allen, J. F. (1981). An interval-based representation of temporal knowledge. In *Proceedings of the 7th international joint conference on Artificial intelligence - IJCAI'81* (S. 221–226). San Francisco, California, USA: Morgan Kaufmann Publishers Inc.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, *26*(11), 832–843. doi:10.1145/182.358434

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, *23*(2), 123–154. doi:10.1016/0004-3702(84)90008-0

Amgoud, L., Maudet, N., & Parsons, S. (2000). Modelling dialogues using argumentation. In *Proceedings Fourth International Conference on MultiAgent Systems* (S. 31–38). IEEE Comput. Soc. doi:10.1109/ICMAS.2000.858428

Angele, J., Kifer, M., & Lausen, G. (2009). Ontologies in F-Logic. In S. Staab & R. Studer (Hrsg.), *Handbook on Ontologies* (S. 45–70). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-92673-3

Antoniou, G. (2004). Defeasible logic with dynamic priorities. *International Journal of Intelligent Systems*, *19*(5), 463–472. doi:10.1002/int.20008

Araszkiewicz, M. (2011). Analogy, Similarity and Factors. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law - ICAIL '11* (S. 101–105). New York: ACM Press. doi:10.1145/2018358.2018372

Araszkiewicz, M., Łopatkiewicz, A., & Zienkiewicz, A. (2013). Factor-based parent plan support system. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law - ICAIL '13* (S. 171–175). New York: ACM Press. doi:10.1145/2514601.2514621

Ashley, K. D. (1990). *Modeling Legal Argument: Reasoning with Cases and Hypotheticals*. Cambridge: The MIT Press.

Ashley, K. D. (2009). Ontological requirements for analogical, teleological, and hypothetical legal reasoning. In *Proceedings of the 12th International Conference on Artificial Intelligence and Law - ICAIL '09* (S. 1–10). New York, New York, USA: ACM Press. doi:10.1145/1568234.1568236

Ashley, K. D., Desai, R., & Levine, J. (2002). Teaching Case-Based Argumentation Concepts Using Dialectic Arguments vs. Didactic Explanations. In S. Cerri, G. Gouardères, & F. Paraguaçu (Hrsg.), *Intelligent Tutoring Systems* (Bd. 2363, S. 585–595). Berlin: Springer Berlin Heidelberg. doi:10.1007/3-540-47987-2_60

Ashley, K. D., & Walker, V. R. (2013). From information retrieval (IR) to argument retrieval (AR) for legal cases: Report on a baseline study. In K. D. Ashley (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2013: The Twenty-Sixth Annual Conference* (Bd. 259, S. 29–38). Amsterdam: IOS Press. doi:10.3233/978-1-61499-359-9-29

Athan, T., Boley, H., Governatori, G., Palmirani, M., Paschke, A., & Wyner, A. (2013a). LegalRuleML: From Metamodel to Use Cases. In L. Morgenstern, P. Stefaneas, F. Lévy, A. Wyner, & A. Paschke (Hrsg.), *Theory, Practice, and Applications of Rules on the Web SE - 4* (Bd. 8035, S. 13–18). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-642-39617-5_4

Athan, T., Boley, H., Governatori, G., Palmirani, M., Paschke, A., & Wyner, A. (2013b). OASIS LegalRuleML. In *Proceedings of the Fourteenth International Conference on Artificial*

Intelligence and Law - ICAIL '13 (S. 3–12). New York, USA: ACM Press. doi:10.1145/2514601.2514603

Atkinson, K., Bench-Capon, T. J. M., Prakken, H., & Wyner, A. (2013). Argumentation Schemes for Reasoning about Factors with Dimensions. In K. D. Ashley (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2013: The Twenty-Sixth Annual Conference* (S. 39–48). Amsterdam: IOS Press. doi:10.3233/978-1-61499-359-9-39

Augusto, J. C., & Simari, G. R. (2001). Temporal Defeasible Reasoning. *Knowledge and Information Systems*, *3*(3), 287–318. doi:10.1007/PL00011670

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., & Patel-Schneider, P. F. (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge: Cambridge University Press.

Baader, F., Horrocks, I., & Sattler, U. (2008). Description Logics. In F. van Harmelen, V. Lifschitz, & B. Porter (Hrsg.), *Handbook of Knowledge Representation* (Bd. 3, S. 135–179). New York: Elsevier. doi:10.1016/S1574-6526(07)03003-9

Bader, K. S. (1955). Das Ermessen des Strafrichters. *JuristenZeitung*, *10*(17), 525–526. doi:10.2307/20804010

Balzert, H. (2001). *UML kompakt*. Heidelberg: Spektrum Akademischer Verlag.

Bench-Capon, T. J. M. (1989). Deep models, normative reasoning and legal expert systems. In *Proceedings of the second international conference on Artificial intelligence and law - ICAIL '89* (S. 37–45). New York, New York, USA: ACM Press. doi:10.1145/74014.74020

Bench-Capon, T. J. M. (1994a). In Defence of Rule-based representations for Legal Knowledge-based systems. *Information & Communications Technology Law*, *3*(1), 15–28. doi:10.1080/13600834.1994.9965689

Bench-Capon, T. J. M. (1994b). Legal Theory and Legal KBS: A Computer Scientist's Perspective. In H. Prakken, A. J. Muntjewerff, & A. Soeteman (Hrsg.), *Legal knowledge*

based systems JURIX 94: The Foundation for Legal Knowledge Systems (S. 33–42). Lelystad: Koninklijke Vermande.

Bench-Capon, T. J. M. (2012). Representing Popov v Hayashi with dimensions and factors. *Artificial Intelligence and Law*, *20*(1), 15–35. doi:10.1007/s10506-012-9118-7

Bench-Capon, T. J. M., Araszkiewicz, M., Ashley, K. D., Atkinson, K., Bex, F., Borges, F., … Wyner, A. (2012). A history of AI and Law in 50 papers: 25 years of the international conference on AI and Law. *Artificial Intelligence and Law*, *20*(3), 215–319. doi:10.1007/s10506-012-9131-x

Bench-Capon, T. J. M., & Coenen, F. P. (1992). Isomorphism and legal knowledge based systems. *Artificial Intelligence and Law*, *1*(1), 65–86. doi:10.1007/BF00118479

Bench-Capon, T. J. M., & Dunne, P. E. (2007). Argumentation in artificial intelligence. *Artificial Intelligence*, *171*(10), 619–641.

Bench-Capon, T. J. M., Dunne, P., Gordon, T. F., Prakken, H., & Walton, D. (2007). The Carneades model of argument and burden of proof. *Artificial Intelligence*, *171*(10), 875–896.

Bench-Capon, T. J. M., & Gordon, T. F. (2009). Isomorphism and Argumentation. In *Proceedings of the 12th International Conference on Artificial Intelligence and Law - ICAIL '09* (S. 11–20). New York, New York, USA: ACM Press. doi:10.1145/1568234.1568237

Bench-Capon, T. J. M., & Prakken, H. (2006). Argumentation. In A. Lodder & A. Oskamp (Hrsg.), *Information Technology and Lawyers* (S. 61–80). Dordrecht: Springer Netherlands. doi:10.1007/1-4020-4146-2_3

Bench-Capon, T. J. M., Prakken, H., Wyner, A., & Atkinson, K. (2013). Argument Schemes for Reasoning with Legal Cases Using Values. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law - ICAIL '13* (S. 13–22). New York: ACM Press. doi:10.1145/2514601.2514604

Bench-Capon, T. J. M., & Visser, P. R. S. (1997). Ontologies in legal information systems; The Need for Explicit Specifications of Domain Conceptualisations. In *Proceedings of the sixth international conference on Artificial intelligence and law - ICAIL '97* (S. 132–141). New York, New York, USA: ACM Press. doi:10.1145/261618.261646

Benjamins, V. R., Casanovas, P., Breuker, J., & Gangemi, A. (2005). Law and the Semantic Web, an Introduction. In V. R. Benjamins, P. Casanovas, J. Breuker, & A. Gangemi (Hrsg.), *Law and the Semantic Web* (Bd. 3369, S. 1–17). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-540-32253-5_1

Bentahar, J., Moulin, B., & Bélanger, M. (2010). A taxonomy of argumentation models used for knowledge representation. *Artificial Intelligence Review*, *33*(3), 211–259. doi:10.1007/s10462-010-9154-1

Berners-Lee, T. (2000). Semantic Web on XML.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, *284*(5), 34–43.

Bex, F., & Verheij, B. (2011). Legal Shifts in the Process of Proof. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law - ICAIL '11* (Bd. 21, S. 253–278). New York: ACM Press. doi:10.1007/s10506-012-9137-4

Bex, F., & Verheij, B. (2013). Legal stories and the process of proof. *Artificial Intelligence and Law*, *21*(3), 253–278. doi:10.1007/s10506-012-9137-4

Biasiotti, M. A., Francesconi, E., & Sartor, G. (2008). *Legal Informatics and Management of Legislative Documents* (No. 2). (G. Sartor, Hrsg.). Rome, Italy.

Birnstiel, A., Bungenberg, M., & Heinrich, H. (Hrsg.). (2013). *Europäisches Beihilfenrecht*. Wien: Facultas.WUV.

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, *5*(3), 1–22. doi:10.4018/jswis.2009081901

Bloch, J. (2008). *Effective Java* (2. Aufl.). Amsterdam: Addison-Wesley Longman.

Boer, A. (2009). *Legal Theory, Sources of Law & the Semantic Web*. University of Amsterdam.

Boer, A., Gordon, T. F., van den Berg, K., Di Bello, M., Förhécz, A., & Vas, R. (2007). *Specification of the legal knowledge interchange format. Deliverable 1.1*. Estrella.

Boer, A., Hoekstra, R., & Winkels, R. (2002). METALex: Legislation in XML. In *Legal Knowledge and Information Systems - JURIX 2002: The Fifteenth Annual Conference* (S. 1–10). Amsterdam: IOS Press.

Boer, A., & van Engers, T. M. (2011). A MetaLex and Metadata Primer: Concepts, Use, and Implementation. In G. Sartor, M. Palmirani, E. Francesconi, & M. A. Biasiotti (Hrsg.), *Legislative XML for the Semantic Web* (Bd. 4, S. 131–149). Dordrecht: Springer Netherlands. doi:10.1007/978-94-007-1887-6_8

Bohlen, M. H., Busatto, R., & Jensen, C. S. (1998). Point-versus interval-based temporal data models. In *Proceedings 14th International Conference on Data Engineering* (S. 192–200). IEEE Comput. Soc. doi:10.1109/ICDE.1998.655777

Borsari, G., Cevenini, C., Contissa, G., Morini, S., Sartor, G., & Still, P. (2005). Hare: An Italian Application of SoftLaw's STATUTE Expert Technology. In *Proceedings of the 10th international conference on Artificial intelligence and law - ICAIL '05* (S. 225–229). doi:10.1145/1165485.1165521

Bratt, S. (2008). Semantic Web, and other technologies to watch.

Brennan, A. (2012). Necessary and Sufficient Conditions. In (E. N. Zalta, Hrsg.)*Stanford Encyclopedia of Philosophy*.

Breuker, J., Boer, A., Hoekstra, R., & van den Berg, K. (2006). Developing Content for LKIF: Ontologies and Frameworks for Legal Reasoning. In T. M. van Engers (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2006: The Nineteenth Annual Conference* (S. 169–174). Amsterdam: IOS Press.

Breuker, J., Hoekstra, R., Boer, A., van den Berg, K., Rubino, R., Sartor, G., … Bench-Capon, T. J. M. (2007). *OWL ontology of basic legal concepts. Deliverable 1.4.* Estrella.

Breuker, J., van de Ven, S., El Ali, A., Bron, M., Hoekstra, R., Klarman, S., … Förhécz, A. (2008). *Developing HARNESS. Deliverable 4.6*. Estrella.

Bron, M., El Ali, A., Ji, X., & Klarman, S. (2007). *Modeling Nomic in LKIF-core ontology*.

Broy, M., Crane, M., Dingel, J., Hartman, A., Rumpe, B., & Selic, B. (2007). 2nd UML 2 Semantics Symposium: Formal Semantics for UML. In T. Kühne (Hrsg.), *Models in Software Engineering* (Bd. 4364, S. 318–323). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-540-69489-2_39

Buabuchachart, A., Metcalf, K., Charness, N., & Morgenstern, L. (2013). Classification of regulatory paragraphs by discourse structure, reference structure, and regulation type. In K. D. Ashley (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2013: The Twenty-Sixth Annual Conference* (Bd. 259, S. 59–62). Amsterdam: IOS Press. doi:10.3233/978-1-61499-359-9-59

Buey, M., Garrido, Á., & Ilarri, S. (2014). An Approach for Automatic Query Expansion Based on NLP and Semantics. In H. Decker, L. Lhotská, S. Link, M. Spies, & R. Wagner (Hrsg.), *Database and Expert Systems Applications* (Bd. 8645, S. 349–356). Berlin: Springer International Publishing. doi:10.1007/978-3-319-10085-2_32

Bullinger, M. (1984). Das Ermessen der öffentlichen Verwaltung - Entwicklung, Funktionen, Gerichtskontrolle. *JuristenZeitung*, *39*(22), 1001–1009.

Büschenfeldt, M., & Scholl, M. (2014). Die Archetypen des Web 2.0 als Referenzmodell für das E-Government 2.0. In D. Lück-Schneider, T. F. Gordon, S. Kaiser, M. G. Löhe, J. von Lucke, E. Schweighofer, & M. A. Wimmer (Hrsg.), *Beiträge zur Verwaltungsinformatik* (S. 117–139). Berlin.

Bydlinski, F. (2005). *Grundzüge der juristischen Methodenlehre*. Wien: WUV-Univ.-Verl.

Caminada, M. (2008). On the Issue of Contraposition of Defeasible Rules. In P. Besnard, S. Doutre, & A. Hunter (Hrsg.), *Computational Models of Argument. Proceedings of COMMA 2008* (S. 109–115). Amsterdam: IOS Press.

Cardoso, J. (2007). The Semantic Web Vision: Where Are We? *IEEE Intelligent Systems*, *22*(5), 84–88. doi:10.1109/MIS.2007.4338499

Casanovas, P., Sartor, G., Biasiotti, M. A., & Fernández-Barrera, M. (2011). Introduction: Theory and Methodology in Legal Ontology Engineering: Experiences and Future Directions. In G. Sartor, P. Casanovas, M. A. Biasiotti, & M. Fernández-Barrera (Hrsg.), *Approaches to Legal Ontologies* (S. 1–14). New York: Springer.

Casellas, N. (2011). *Legal Ontology Engineering*. Dordrecht: Springer Netherlands. doi:10.1007/978-94-007-1497-7

Ceci, M. (2013). *Interpreting Judgements using Knowledge Representation Methods and Computational Models of Argument*. University of Bologna.

Chandrasekaran, B., Johnson, T. R., & Smith, J. W. (1992). Task-structure analysis for knowledge modeling. *Communications of the ACM*, *35*(9), 124–137. doi:10.1145/130994.131002

Chun, S. A., Shulman, S., Sandoval, R., & Hovy, E. (2010). Government 2.0: Making connections between citizens, data and government. *Information Polity*, *15*(1-2), 1–9. doi:10.3233/IP-2010-0205

Conetta, O., & Schafer, B. (2013). LKIF in Commercial Legal Practice: Transaction Configuration from Eurobonds to Copyright. In K. D. Ashley (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2013: The Twenty-Sixth Annual Conference* (S. 63–72). Amsterdam: IOS Press. doi:10.3233/978-1-61499-359-9-63

Cornides, T. (1975). Logik der Normen. In G. Winkler (Hrsg.), *Rechtstheorie und Rechtsinformatik* (S. 67–101). Wien: Springer.

Davies, J., Potter, M., Richardson, M., Stinčić, S., Domingue, J., Pedrinaci, C., … González-Cabero, R. (2009). Towards the open service web. *BT Technology Journal*, *26*(2), 153–164.

Davis, R., Shrobe, H., & Szolovits, P. (1993). What Is a Knowledge Representation? *AI Magazine*, *14*(1), 17–33. doi:10.1609/aimag.v14i1.1029

De Jong, T., & Ferguson-Hessler, M. G. M. (1996). Types and qualities of knowledge. *Educational Psychologist*, *31*(2), 105–113. doi:10.1207/s15326985ep3102_2

De Maat, E., & Winkels, R. (2010). Automated Classification of Norms in Sources of Law. In E. Francesconi, S. Montemagni, W. Peters, & D. Tiscornia (Hrsg.), *Semantic Processing of Legal Texts* (Bd. 6036, S. 170–191). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-642-12837-0_10

De Mulder, R., van Noortwijk, K., & Combrink-Kuiters, L. (2010). Jurimetrics Please! *European Journal of Law and Technology*, *1*(1).

Delgado, J., Gallego, I., Llorente, S., & García, R. (2003). IPROnto: An Ontology for Digital Rights Management. In D. Bourcier (Hrsg.), *Legal Knowledge and Information Systems - JURIX 2003: The Sixteenth Annual Conference* (S. 111–120). Amsterdam.

Dentler, K., Cornet, R., ten Teije, A., & de Keizer, N. (2011). Comparison of reasoners for large ontologies in the OWL 2 EL profile. *Semantic Web*, *2*(2), 71–87. doi:10.3233/SW-2011-0034

Dijkstra, E. W. (1982). On the role of scientific thought. In *Selected writings on Computing: A Personal Perspective.* (S. 60–66). New York: Springer.

Dung, P. M. (1995). On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, *77*(2), 321–357. doi:10.1016/0004-3702(94)00041-X

Eiter, T., Ianni, G., Polleres, A., Schindlauer, R., & Tompits, H. (2006). Reasoning with Rules and Ontologies. In P. Barahona, F. Bry, E. Franconi, N. Henze, & U. Sattler (Hrsg.),

Reasoning Web (Bd. 4126, S. 93–127). Berlin: Springer Berlin Heidelberg. doi:10.1007/11837787_4

Engisch, K. (1963). *Logische Studien zur Gesetzesanwendung* (3. Aufl.). Heidelberg: Winter.

European Commission. (2014). *eGovernment Benchmark*. doi:10.2759/48856

Feldman, J. (2011). Representing and Solving Rule-Based Decision Models with Constraint Solvers. In F. Olken, M. Palmirani, & D. Sottara (Hrsg.), *Rule - Based Modeling and Computing on the Semantic Web* (Bd. 7018, S. 208–221). Berlin Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-24908-2_23

Feldman, J. (2013a). Solving Rule Conflicts - Part 1. Abgerufen von http://openrules.wordpress.com/2013/08/14/solving-rule-conflicts-part-1/

Feldman, J. (2013b). Solving Rule Conflicts - Part 2. Abgerufen von http://openrules.wordpress.com/2013/08/14/solving-rule-conflicts-part-2/

Fellinger, P. F., & Philipps, L. (2003). Fallvergleich im Sterndiagramm. In E. Schweighofer (Hrsg.), *Zwischen Rechtstheorie und e-Government - Aktuelle Fragen der Rechtsinformatik* (S. 51–62). Wien: Verlag Österreich.

Ferber, R. (2003). *Information Retrieval. Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web*. Heidelberg: dpunkt.

Ferrajoli, L. (2007). *Principia iuris, vol. I, Teoria del diritto.* Laterza: Roma-Bari.

Fiedler, H. (2004). Formalisierung im Recht. In E. Schweighofer, D. Liebwald, G. Kreuzbauer, & T. Menzel (Hrsg.), *Informationstechnik in der juristischen Realität* (S. 29–37). Wien: Verlag Österreich.

Florescu, D. (2005). Managing semi-structured data. *Queue*, *3*(8), 18–24. doi:10.1145/1103822.1103832

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, *19*(1), 17–37. doi:10.1016/0004-3702(82)90020-0

Fowler, M. (2000). POJO. Abgerufen von http://www.martinfowler.com/bliki/POJO.html

Fowler, M. (2005). Inversion of Control. Abgerufen von http://martinfowler.com/bliki/InversionOfControl.html

Francesconi, E. (2011). Naming Legislative Resources. In G. Sartor, M. Palmirani, E. Francesconi, & M. A. Biasiotti (Hrsg.), *Legislative XML for the Semantic Web* (Bd. 4, S. 49–74). Dordrecht: Springer Netherlands. doi:10.1007/978-94-007-1887-6_5

Franklin, J. (2012). Discussion paper: how much of commonsense and legal reasoning is formalizable? A review of conceptual obstacles. *Law, Probability and Risk*, *11*(2-3), 225–245. doi:10.1093/lpr/mgs007

Freeman, E., & Freeman, E. (2004). *Head First Design Patterns*. Cambridge: O'Reilly Media.

Gabbay, D., Horty, J. F., Parent, X., van der Meyden, R., & van der Torre, L. (Hrsg.). (2013). *Handbook of Deontic Logic and Normative Systems*. London: College Publications.

Gangemi, A. (2007). Design patterns for legal ontology constructions. In P. Casanovas, M. A. Biasiotti, E. Francesconi, & M. T. Sagri (Hrsg.), *Proceedings of the 2nd Workshop on Legal Ontologies and Artificial Intelligence Techniques - LOAIT 2007* (S. 65–85). Stanford, California, USA.

García, J. M., Ruiz, D., & Ruiz-Cortés, A. (2012). Improving Semantic Web Services Discovery Using SPARQL-Based Repository Filtering. *Web Semantics: Science, Services and Agents on the World Wide Web*, *17*(2012).

García, R., & Delgado, J. (2005). An Ontological Approach for the Management of Rights Data Dictionaries. In *Legal Knowledge and Information Systems - JURIX 2005: The Eighteenth Conference on Legal Knowledge and Information Systems* (S. 137–146). Brüssel.

Gegick, M., & Barnum, S. (2013). Failing Securely. Abgerufen von https://buildsecurityin.us-cert.gov/articles/knowledge/principles/failing-securely

Gettier, E. L. (1963). Is Justified True Belief Knowledge? *Analysis*, *23*(6), 121. doi:10.2307/3326922

Gordon, T. F. (1987). Oblog-2: A hybrid knowledge representation system for defeasible reasoning. In *Proceedings of the first international conference on Artificial intelligence and law - ICAIL '87* (S. 231–239). New York, New York, USA: ACM Press. doi:10.1145/41735.41761

Gordon, T. F. (1988). The importance of nonmonotonicity for legal reasoning. In H. Fiedler (Hrsg.), *Expert systems in law: Impacts on legal theory and computer law* (S. 111–126). Tübingen: Attempto.

Gordon, T. F. (1991). An abductive theory of legal issues. *International Journal of Man-Machine Studies*, *35*(1), 95–118. doi:10.1016/S0020-7373(07)80009-5

Gordon, T. F. (1993). *The Pleadings Game - An Artificial Intelligence Model of Procedural Justice*. Technical University of Darmstadt.

Gordon, T. F. (1994a). From Jhering to Alexy - Using Artificial Intelligence Models in Jurisprudence. In H. Prakken, A. J. Muntjewerff, & A. Soeteman (Hrsg.), *Legal knowledge based systems JURIX 94: The Foundation for Legal Knowledge Systems* (S. 19–32). Lelystad: Koninklijke Vermande.

Gordon, T. F. (1994b). The Pleadings Game. *Artificial Intelligence and Law*, *2*(4), 239–292. doi:10.1007/BF00871972

Gordon, T. F. (2008a). Constructing Legal Arguments with Rules in the Legal Knowledge Interchange Format (LKIF). In P. Casanovas, G. Sartor, N. Casellas, & R. Rubino (Hrsg.), *Computable Models of the Law* (Bd. 4884, S. 162–184). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-85569-9_11

Gordon, T. F. (2008b). *The Legal Knowledge Interchange Format (LKIF). Deliverable 4.1*. Estrella.

Gordon, T. F. (2010a). An Overview of the Carneades Argumentation Support System. In C. Reed & C. W. Tindale (Hrsg.), *Dialectics, Dialogue and Argumentation. An Examination of Douglas Walton's Theories of Reasoning* (S. 145–156). London: College Publications.

Gordon, T. F. (2010b). An Overview of the Legal Knowledge Interchange Format. In W. Abramowicz, R. Tolksdorf, & K. Węcel (Hrsg.), *Business Information Systems Workshops* (Bd. 57, S. 240–242). Berlin Heidelberg: Springer. doi:10.1007/978-3-642-15402-7_30

Gordon, T. F. (2011a). Analyzing open source license compatibility issues with Carneades. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law - ICAIL '11* (S. 51–55). New York: ACM Press. doi:10.1145/2018358.2018364

Gordon, T. F. (2011b). Combining Rules and Ontologies with Carneades. In *Proceedings of the 5th International RuleML2011@ BRF Challenge*. Fort Lauderdale, Florida, USA.

Gordon, T. F., Governatori, G., & Rotolo, A. (2009). Rules and Norms: Requirements for Rule Interchange Languages in the Legal Domain. In G. Governatori, J. Hall, & A. Paschke (Hrsg.), *Rule Interchange and Applications* (S. 282–296). New York: Springer.

Gordon, T. F., & Walton, D. (2006). The Carneades Argumentation Framework - Using Presumptions and Exceptions to Model Critical Questions. In P. E. Dunne & T. J. M. Bench-Capon (Hrsg.), *Computational Models of Argument. Proceedings of COMMA 2006* (S. 195–207). Amsterdam, United Kingdom: IOS Press.

Gordon, T. F., & Walton, D. (2009a). Legal reasoning with argumentation schemes. In *Proceedings of the 12th International Conference on Artificial Intelligence and Law - ICAIL '09* (S. 137). New York, New York, USA: ACM Press. doi:10.1145/1568234.1568250

Gordon, T. F., & Walton, D. (2009b). Proof Burdens and Standards. In G. Simari & I. Rahwan (Hrsg.), *Argumentation in Artificial Intelligence* (S. 239–258). New York: Springer. doi:10.1007/978-0-387-98197-0_12

Gosh, D. (2011). *DSLs in Action*. Greenwich: Manning.

Gosling, J., Joy, B., Steele, G., Bracha, G., & Buckley, A. (2013). *The Java Language Specification* (7. Aufl.). Redwood City: Oracle America.

Governatori, G., Lam, H.-P., Rotolo, A., Villata, S., & Gandon, F. (2013). Heuristics for Licenses Composition. In K. D. Ashley (Hrsg.), *Legal Knowledge and Information Systems, JURIX*

2013: The Twenty-Sixth Annual Conference (S. 77–86). Amsterdam: IOS Press. doi:10.3233/978-1-61499-359-9-77

Governatori, G., Palmirani, M., Riveret, R., Rotolo, A., & Sartor, G. (2005). Norm Modifications in Defeasible Logic. In *Proceedings of the 2005 Conference on Legal Knowledge and Information Systems: JURIX 2005: The Eighteenth Annual Conference* (S. 13–22). Amsterdam, The Netherlands, The Netherlands: IOS Press.

Governatori, G., & Rotolo, A. (2006). Logic of Violations: A Gentzen System for Reasoning with Contrary-To-Duty Obligations. *Australasian Journal of Logic*, *4*, 193–215.

Governatori, G., & Rotolo, A. (2008). BIO logical agents: Norms, beliefs, intentions in defeasible logic. *Autonomous Agents and Multi-Agent Systems*, *17*(1), 36–69. doi:10.1007/s10458-008-9030-4

Governatori, G., & Rotolo, A. (2009). Changing legal systems: legal abrogations and annulments in Defeasible Logic. *Logic Journal of IGPL*, *18*(1), 157–194. doi:10.1093/jigpal/jzp075

Governatori, G., Rotolo, A., Riveret, R., Palmirani, M., & Sartor, G. (2007). Variants of temporal defeasible logics for modelling norm modifications. In *Proceedings of the 11th international conference on Artificial intelligence and law - ICAIL '07* (S. 155–159). New York, New York, USA: ACM Press. doi:10.1145/1276318.1276347

Governatori, G., Rotolo, A., & Rubino, R. (2010). Implementing Temporal Defeasible Logic for Modeling Legal Reasoning. In K. Nakakoji, Y. Murakami, & E. McCready (Hrsg.), *New Frontiers in Artificial Intelligence* (S. 45–58). Berlin: Springer. doi:10.1007/978-3-642-14888-0_5

Governatori, G., Rotolo, A., & Sartor, G. (2005). Temporalised normative positions in defeasible logic. In *Proceedings of the 10th international conference on Artificial intelligence and law - ICAIL '05* (S. 25). New York, New York, USA: ACM Press. doi:10.1145/1165485.1165490

Governatori, G., & Terenziani, P. (2007). Temporal Extensions to Defeasible Logic. In M. A. Orgun & J. Thornton (Hrsg.), *AI 2007: Advances in Artificial Intelligence* (Bd. 4830, S. 476–485). Berlin Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-76928-6_49

Grau, B. C., Horrocks, I., Parsia, B., Patel-Schneider, P., & Sattler, U. (2006). Next Steps for OWL. In B. C. Grau, P. Hitzler, C. Shankey, & E. Wallace (Hrsg.), *Proceedings of the Second OWLED Workshop on OWL: Experiences and Directions (OWLED 2006)*. Athens, Georgia, USA.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *5*(2), 199–220. doi:10.1006/knac.1993.1008

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies*, *43*(5-6), 907–928. doi:10.1006/ijhc.1995.1081

Gruninger, M., & Fox, M. S. (1994). The Role of Competency Questions in Enterprise Engineering. In *Proceedings of the IFIP WG5.7 Workshop on Benchmarking—Theory and Practice*. Trondheim.

Guarino, N., Oberle, D., & Staab, S. (2009). What Is an Ontology? In S. Staab & R. Studer (Hrsg.), *Handbook on Ontologies* (S. 1–17). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-540-92673-3_0

Hadrbolec, M. (2011). Von isolierten Datenfriedhöfen zur übergreifenden effizienten Umweltdatenauswertung. In E. Schweighofer & F. Kummer (Hrsg.), *Tagungsband des 14. Internationalen Rechtsinformatik Symposions - IRIS 2011* (S. 355–359). Wien: Österreichische Computer Gesellschaft.

Hart, H. L. A. (2009). *Der Begriff des Rechts*. (C. Möllers, Übers.). Berlin: Suhrkamp.

Heck, P. (1914). Gesetzesauslegung und Interessenjurisprudenz. *Archiv für die civilistische Praxis*, *112*(1), 1–11.

Hitzler, P., Krötzsch, M., Rudolph, S., & Sure, Y. (2008). *Semantic Web*. Berlin: Springer. doi:10.1007/978-3-540-33994-6

Hitzler, P., & Parsia, B. (2009). Ontologies and Rules. In S. Staab & R. Studer (Hrsg.), *Handbook on Ontologies* (S. 111–132). Berlin Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-92673-3_5

Hoekstra, R. (2009). *Ontology Representation: Design Patterns and Ontologies that Make Sense*. University of Amsterdam.

Hoekstra, R., & Breuker, J. (2008). Polishing Diamonds in OWL 2. In A. Gangemi & J. Euzenat (Hrsg.), *Knowledge Engineering: Practice and Patterns* (Bd. 5268, S. 64–73). Springer Berlin Heidelberg. doi:10.1007/978-3-540-87696-0_8

Hoekstra, R., Breuker, J., Di Bello, M., & Boer, A. (2007). The LKIF Core Ontology of Basic Legal Concepts. In P. Casanovas, M. A. Biasiotti, E. Francesconi, & M. T. Sagri (Hrsg.), *Proceedings of the 2nd Workshop on Legal Ontologies and Artificial Intelligence Techniques - LOAIT 2007* (S. 43–63). Stanford, California, USA.

Hoekstra, R., Breuker, J., Di Bello, M., & Boer, A. (2009). LKIF Core: Principled Ontology Development for the Legal Domain. In J. Breuker, P. Casanovas, M. C. A. Klein, & E. Francesconi (Hrsg.), *Law, Ontologies and the Semantic Web* (S. 21–52). Amsterdam: IOS Press. doi:10.3233/978-1-58603-942-4-21

Hohfeld, W. N. (1913). Some Fundamental Legal Conceptions as Applied in Judicial Reasoning. *The Yale Law Journal*, *23*(1), 16. doi:10.2307/785533

Hohfeld, W. N. (1917). Fundamental Legal Conceptions as Applied in Judicial Reasoning. *The Yale Law Journal*, *26*(8), 710. doi:10.2307/786270

Horridge, M., & Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. *Semantic Web*, *2*(1), 11–21. doi:10.3233/SW-2011-0025

Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., & Wang, H. H. (2006). The Manchester OWL Syntax. In B. C. Grau, P. Hitzler, C. Shankey, & E. Wallace (Hrsg.),

Proceedings of the Second OWLED Workshop on OWL: Experiences and Directions (OWLED 2006). Athens, Georgia, USA.

Horrocks, I., Kutz, O., & Sattler, U. (2006). The Even More Irresistible SROIQ. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)* (S. 57–67). Lake District, United Kingdom.

Horrocks, I., Parsia, B., Patel-Schneider, P., & Hendler, J. (2005). Semantic Web Architecture: Stack or Two Towers? In F. Fages & S. Soliman (Hrsg.), *Principles and Practice of Semantic Web Reasoning* (Bd. 3703, S. 37–41). Berlin: Springer Berlin Heidelberg. doi:10.1007/11552222_4

Hötzendorfer, W., & Schweighofer, E. (2014). Safe Harbor in der „Post-Snowden-Ära“. In D. Lück-Schneider, T. F. Gordon, S. Kaiser, J. von Lucke, E. Schweighofer, M. A. Wimmer, & M. G. Löhe (Hrsg.), *Gemeinsam Electronic Government ziel(gruppen)gerecht getalten und organisieren - Gemeinsame Fachtagung Verwaltungsinformatik und Fachtagung Rechtsinformatik 2014* (S. 125–136). Bonn: Gesellschaft für Informatik.

Houy, C., Niesen, T., Calvillo, J., Fettke, P., & Loos, P. (2014). Konzept und Architektur eines Software-Werkzeuges zur automatisierten Identifikation und Analyse von Argumentationsstrukturen. In D. Lück-Schneider, T. F. Gordon, S. Kaiser, J. von Lucke, E. Schweighofer, M. A. Wimmer, & M. G. Löhe (Hrsg.), *Gemeinsam Electronic Government ziel(gruppen)gerecht getalten und organisieren - Gemeinsame Fachtagung Verwaltungsinformatik und Fachtagung Rechtsinformatik 2014* (S. 113–124). Bonn: Gesellschaft für Informatik.

Hürsch, W. L., & Lopes, C. V. (1995). *Separation of Concerns*. Boston, MA.

Iannone, L., & Rector, A. (2008). Calculations in OWL. In C. Dolbear, A. Ruttenberg, & U. Sattler (Hrsg.), *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions (OWLED 2008)*. Karlsruhe, Germany.

Jackson, P., & Moulinier, I. (2007). *Natural Language Processing for Online Applications: Text Retrieval, Extraction and Categorization* (2. Aufl.). Amsterdam: John Benjamins Publishing Company.

Jansana, R. (2011). Propositional Consequence Relations and Algebraic Logic. In *The Stanford Encyclopedia of Philosophy*.

Jeckle, M., Rupp, C., Zengler, B., Queins, S., & Hahn, J. (2004). UML 2.0—Neue Möglichkeiten und alte Probleme. *Informatik-Spektrum*, *27*(4), 323–331. doi:10.1007/s00287-004-0416-7

Jewell, E. J., & Abate, F. (2001). *The New Oxford American Dictionary*. New York: Oxford University Press.

Joerden, J. C. (2010). *Logik im Recht* (2. Aufl.). Berlin: Springer.

Johnson, P., & Mead, D. (1991). Legislative knowledge base systems for public administration: some practical issues. In *Proceedings of the third international conference on Artificial intelligence and law - ICAIL '91* (S. 108–117). New York, New York, USA: ACM Press. doi:10.1145/112646.112660

Johnson, R. E., & Foote, B. (1988). Designing Reuseable Classes. *Journal of object-oriented programming*, *2*(1), 22–35.

Johnston, B., & Governatori, G. (2003). Induction of defeasible logic theories in the legal domain. In *Proceedings of the 9th international conference on Artificial intelligence and law - ICAIL '03* (S. 204). New York, New York, USA: ACM Press. doi:10.1145/1047788.1047834

Jothilakshmi, R., Shanthi, N., & Babisaraswathi, R. (2013). A Survey on Semantic Query Expansion. *Journal of Theoretical and Applied Information Technology*, *57*(1), 128–138.

Karning, B. (2004). *E-Government im Bereich des BMLV*. Universität Wien.

Kelsen, H. (1960). *Reine Rechtslehre* (2. Aufl.). Wien: Verlag Franz Deuticke.

Kelsen, H. (1979). *Allgemeine Theorie der Normen*. Wien: Manz.

Klarman, S., Hoekstra, R., & Bron, M. (2008). Versions and applicability of concept definitions in legal ontologies. In K. Clark & P. F. Patel-Schneider (Hrsg.), *Proceedings of the Fourth*

OWLED Workshop on OWL: Experiences and Directions (OWLED 2008DC). Washington, DC, USA.

Klug, U. (1982). *Juristische Logik*. Berlin: Springer.

Knyrim, R. (2013). Entwurf der neuen EU-Datenschutzrecht-Grundverordnung. In M. Scholz & A. Funk (Hrsg.), *DGRI Jahrbuch 2012* (S. 25–38). Köln: Verlag Dr. Otto Schmidt.

Koller, P. (1997). *Theorie des Rechts* (2. Aufl.). Wien: Böhlau Verlag.

Kolodner, J. L. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, *6*(1), 3–34. doi:10.1007/BF00155578

Kowalski, R. A. (1995). Legislation as Logic Programs. In Z. Bankowski, I. White, & U. Hahn (Hrsg.), *Informatics and the Foundations of Legal Reasoning* (Bd. 21, S. 325–356). Dordrecht: Springer Netherlands. doi:10.1007/978-94-015-8531-6_14

Kowalski, R. A. (2014). Logic Programming. In D. M. Gabbay, F. J. Pelletier, & J. Woods (Hrsg.), *Handbook of the History of Logic* (Bd. Volume 9, S. 523–569). Amsterdam: Elsevier. doi:http://dx.doi.org/10.1016/B978-0-444-51624-4.50012-5

Kowalski, R. A., & Sergot, M. (1990). The Use of Logical Models in Legal Problem Solving. *Ratio Juris*, *3*(2), 201–218. doi:10.1111/j.1467-9337.1990.tb00058.x

Kutschera, A. (2004). e-Finanz - der Steuerpflichtige im elektronischen Zeitalter. In E. Schweighofer, D. Liebwald, G. Kreuzbauer, & T. Menzel (Hrsg.), *Informationstechnik in der juristischen Realität* (S. 331–338). Wien: Verlag Österreich.

Lackes, R. (2013). Fuzzy Logic. In *Gabler Wirtschaftslexikon*. Springer.

Lam, H.-P., & Governatori, G. (2009). The Making of SPINdle. In G. Governatori, J. Hall, & A. Paschke (Hrsg.), *Rule Interchange and Applications* (S. 315–322). New York: Springer. doi:10.1007/978-3-642-04985-9_29

Lam, H.-P., & Governatori, G. (2012). Towards a model of UAVs navigation in urban canyon through defeasible logic. *Journal of Logic and Computation*, *23*(2), 373–395. doi:10.1093/logcom/exr028

Le, T. T. N., Le Nguyen, M., & Shimazu, A. (2013). Unsupervised keyword extraction for japanese legal documents. In K. D. Ashley (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2013: The Twenty-Sixth Annual Conference* (Bd. 259, S. 97–106). Amsterdam: IOS Press. doi:10.3233/978-1-61499-359-9-97

Lenat, D. B., & Feigenbaum, E. A. (1991). On the thresholds of knowledge. *Artificial Intelligence*, *47*(1-3), 185–250. doi:10.1016/0004-3702(91)90055-O

Levesque, H. J., & Brachman, R. J. (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, *3*(1), 78–93. doi:10.1111/j.1467-8640.1987.tb00176.x

Liebwald, D. (2009). An ontology for the implementation of the EU Services Directive. In G. Governatori (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2009: The Twenty-Second Annual Conference* (Bd. 205, S. 100–105). Amsterdam: IOS Press. doi:10.3233/978-1-60750-082-7-100

Lindholm, T., Yellin, F., Bracha, G., & Buckley, A. (2013). *The Java Virtual Machine Specification* (7. Aufl.). Redwood City: Oracle America.

Liu, S., Liu, Y., André, É., Choppy, C., Sun, J., Wadhwa, B., & Dong, J. (2013). A Formal Semantics for Complete UML State Machines with Communications. In E. Johnsen & L. Petre (Hrsg.), *Integrated Formal Methods* (Bd. 7940, S. 331–346). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-642-38613-8_23

Loevinger, L. (1962). Jurimetrics: Science and Prediction in the Field of Law. *MULL: Modern Uses of Logic in Law*, *3*(3), 187–205.

Maher, M. J. (2004). Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming*, *1*(06), 691–711. doi:10.1017/S1471068401001168

Marín, R. H., & Sartor, G. (1999). Time and norms: a formalisation in the event-calculus. In *Proceedings of the seventh international conference on Artificial intelligence and law - ICAIL '99* (S. 90–99). New York, New York, USA: ACM Press. doi:10.1145/323706.323719

Mc Ilraith, S. A., Son, T. C., & Zeng, H. (2001). Semantic Web services. *IEEE Intelligent Systems*, *16*(2), 46–53. doi:10.1109/5254.920599

McGuinness, D. L. (2003). Ontologies Come of Age. In D. Fensel, J. Hendler, H. Lieberman, & W. Wahlster (Hrsg.), *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential* (S. 171–194). Massachusetts: The MIT Press.

Miller, C., & Smith, P. (2011). Web Technologies and Reasoning Communities. In J. Yearwood & A. Stranieri (Hrsg.), *Technologies for Supporting Reasoning Communities and Collaborative Decision Making* (S. 397–411). New York: IGI Global. doi:10.4018/978-1-60960-091-4.ch021

Miller, J. (2008). Patterns in Practice: Cohesion And Coupling. *MSDN Magazine*, *2008*(10).

Modgil, S., & Prakken, H. (2014). The ASPIC + framework for structured argumentation: a tutorial. *Argument & Computation*, *5*(1), 31–62. doi:10.1080/19462166.2013.869766

Mokyr, J. (2002). *The Knowledge Society: Theoretical and Historical Underpinnings*. New York: United Nations.

Moles, R. N. (1992). Expert Systems – The Need for Theory. In *Legal knowledge based systems JURIX 92: The Foundation for Legal Knowledge Systems* (S. 113–122).

Morscher, E. (1988). Möglichkeiten und Grenzen der Formalisierung im Recht. In H. Schäffer (Hrsg.), *Theorie der Rechtssetzung* (S. 85–123). Wien: Manz.

Motik, B., Cuenca Grau, B., & Sattler, U. (2008). Structured Objects in OWL: Representation and Reasoning. In *Proceeding of the 17th international conference on World Wide Web - WWW '08* (S. 555–564). New York, New York, USA: ACM Press. doi:10.1145/1367497.1367573

Mylopoulos, J. (1980). An overview of Knowledge Representation. In *Proceedings of the 1980 workshop on Data abstraction, databases and conceptual modeling* - (S. 5–12). New York, New York, USA: ACM Press. doi:10.1145/800227.806869

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, *18*(1), 87–127. doi:10.1016/0004-3702(82)90012-1

Noy, N. F., & McGuinness, D. L. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology*.

Nute, D. (2003). Defeasible Logic. In O. Bartenstein, U. Geske, M. Hannebauer, & O. Yoshie (Hrsg.), *Web Knowledge Management and Decision Support* (S. 151–169). Berlin: Springer Berlin Heidelberg. doi:10.1007/3-540-36524-9_13

Nygard, M. T. (2007). *Release It!: Design and Deploy Production-Ready Software*. Frisco: The Pragmatic Bookshelf.

Öhlinger, T., & Eberhard, H. (2014). *Verfassungsrecht* (10. Aufl.). Facultas.WUV.

Ollmann, G. (2007). Writing secure code. *Network Security*, *2007*(5), 16–20. doi:10.1016/S1353-4858(07)70037-5

Opalek, K., & Wolenski, J. (1975). Das Problem der Axiomatisierung des Rechts. In G. Winkler (Hrsg.), *Rechtstheorie und Rechtsinformatik* (S. 51–66). Wien: Springer.

Oskamp, A., & Lauritsen, M. (2002). AI in law practice? So far, not much. *Artificial Intelligence and Law*, *10*(4), 227–236. doi:10.1023/A:1025402013007

Palmirani, M. (2011). Legislative Change Management with Akoma-Ntoso. In G. Sartor, M. Palmirani, E. Francesconi, & M. A. Biasiotti (Hrsg.), *Legislative XML for the Semantic Web* (Bd. 4, S. 101–130). Dordrecht: Springer Netherlands. doi:10.1007/978-94-007-1887-6_7

Palmirani, M., & Benigni, F. (2007). Norma-System: A Legal Information System for Managing Time. In C. Biagioli, E. Francesconi, & G. Sartor (Hrsg.), *Proceedings of the V Legislative XML Workshop*. European Press Academic Publishing.

Palmirani, M., & Brighi, R. (2006). Time Model for Managing the Dynamic of Normative System. In M. A. Wimmer, H. Scholl, Å. Grönlund, & K. V. Andersen (Hrsg.), *Electronic Government* (Bd. 4084, S. 207–218). Berlin Heidelberg: Springer Berlin Heidelberg. doi:10.1007/11823100_19

Palmirani, M., Cervone, L., & Bujor, O. (2013). Legal Rules Modelling Meets the Web. In M. Palmirani & G. Sartor (Hrsg.), *JURIX 2013 Doctoral Consortium and Poster Sessions*. Bologna, Italy.

Palmirani, M., Cervone, L., Bujor, O., & Chiappetta, M. (2013). RAWE: An Editor for Rule Markup of Legal Texts. In *RuleML2013@Challenge*. Seattle, Washington.

Palmirani, M., Contissa, G., & Rubino, R. (2009). Fill the Gap in the Legal Knowledge Modelling. In G. Governatori, J. Hall, & A. Paschke (Hrsg.), *Rule Interchange and Applications* (S. 305–314). New York: Springer. doi:10.1007/978-3-642-04985-9_28

Palmirani, M., Governatori, G., & Contissa, G. (2010). Temporal Dimensions in Rules Modelling. In R. G. F. Winkels (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2010: The Twenty-Third Annual Conference* (S. 159 – 162). Amsterdam: IOS Press. doi:10.3233/978-1-60750-682-9-159

Palmirani, M., Governatori, G., & Contissa, G. (2011). Modelling temporal legal rules. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law - ICAIL '11* (S. 131–135). New York: ACM Press. doi:10.1145/2018358.2018378

Palmirani, M., Governatori, G., Rotolo, A., Tabet, S., Boley, H., & Paschke, A. (2011). LegalRuleML: XML-Based Rules and Norms. In F. Olken, M. Palmirani, & D. Sottara (Hrsg.), *Rule - Based Modeling and Computing on the Semantic Web* (Bd. 7018, S. 298–312). Berlin, Heidelberg: Springer. doi:10.1007/978-3-642-24908-2_30

Palmirani, M., Ognibene, T., & Cervone, L. (2012). Legal Rules, Text and Ontologies Over Time. In H. Aït-Kaci, Y.-J. Hu, G. J. Nalepa, M. Palmirani, & D. Roman (Hrsg.), *RuleML2012@ECAI Challenge*. Montpellier, France.

Palmirani, M., & Vitali, F. (2011). Akoma-Ntoso for Legal Documents. In G. Sartor, M. Palmirani, E. Francesconi, & M. A. Biasiotti (Hrsg.), *Legislative XML for the Semantic Web* (Bd. 4, S. 75–100). Dordrecht: Springer Netherlands. doi:10.1007/978-94-007-1887-6_6

Parsia, B., & Smith, M. (2008). Quantities in OWL. In C. Dolbear, A. Ruttenberg, & U. Sattler (Hrsg.), *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions (OWLED 2008)*. Karlsruhe, Germany.

Peroni, S., Shotton, D., & Vitali, F. (2012). Scholarly publishing and linked data. In *Proceedings of the 8th International Conference on Semantic Systems - I-SEMANTICS ´12* (S. 9–16). New York, New York, USA: ACM Press. doi:10.1145/2362499.2362502

Philipps, L. (2011). Anschauliche Normlogik. Zugleich eine Erinnerung an Bindings Normentheorie. *Jusletter IT*, *2011*(4).

Philipps, L. (2012). *Endliche Rechtsbegriffe mit unendlichen Grenzen*. Bern: Editions Weblaw.

Poudyal, P., & Quaresma, P. (2012). An hybrid approach for legal information extraction. In B. Schafer (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2012: The Twenty-Fifth Annual Conference* (Bd. 250, S. 115–118). Amsterdam: IOS Press. doi:10.3233/978-1-61499-167-0-115

Pradeesh Hosea, S., Harikrishnan, V., & Rajkuma, K. (2011). Artificial Intelligence. In *3rd International Conference on Electronics Computer Technology* (S. 124–129). Kanyakumari: IEEE.

Prakken, H. (2008). Formalising ordinary legal disputes: a case study. *Artificial Intelligence and Law*, *16*(4), 333–359. doi:10.1007/s10506-008-9069-1

Prakken, H. (2010). An abstract framework for argumentation with structured arguments. *Argument & Computation*, *1*(2), 93–124. doi:10.1080/19462160903564592

Prakken, H., & Sartor, G. (1996). A dialectical model of assessing conflicting arguments in legal reasoning. *Artificial Intelligence and Law*, *4*(3-4), 331–368. doi:10.1007/BF00118496

Prakken, H., & Sartor, G. (1997). Argument-based extended logic programming with defeasible priorities. *Journal of Applied Non-Classical Logics*, *7*(1-2), 25–75. doi:10.1080/11663081.1997.10510900

Prakken, H., & Sartor, G. (1998). Modelling Reasoning with Precedents in a Formal Dialogue Game. *Artificial Intelligence and Law*, *6*(2-4), 231–287. doi:10.1023/A:1008278309945

Prakken, H., & Sartor, G. (2007). Formalising arguments about the burden of persuasion. In *Proceedings of the 11th international conference on Artificial intelligence and law - ICAIL '07* (S. 97). New York, New York, USA: ACM Press. doi:10.1145/1276318.1276338

Prakken, H., & Sartor, G. (2013). Formalising arguments about norms. In K. D. Ashley (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2013: The Twenty-Sixth Annual Conference* (Bd. 259, S. 121–130). Amsterdam: IOS Press. doi:10.3233/978-1-61499-359-9-121

Prakken, H., & Schrickx, J. (1991). Isomorphic models for rules and exceptions in legislation. In J. Breuker, R. V. De Mulder, & J. C. Hage (Hrsg.), *Legal knowledge based systems JURIX 91: The Foundation for Legal Knowledge Systems* (S. 17–27). Lelystad: Koninklijke Vermande.

Prakken, H., & Sergot, M. (1996). Contrary-to-duty obligations. *Studia Logica*, *57*(1), 91–115. doi:10.1007/BF00370671

Preiß, T. (2012). Die Bedeutung der Risikoanalyse für den Rechtsschutz bei automatisierten Verwaltungsstrafverfahren. In E. Schweighofer, F. Kummer, & W. Hötzendorfer (Hrsg.), *Tagungsband des 15. Internationalen Rechtsinformatik Symposions - IRIS 2012* (S. 91–98). Salzburg: Österreichische Computer Gesellschaft.

Priddle-Higson, A. (2010). *Computational models of ontology evolution in legal reasoning*. University of Edinburgh.

Promberger, K., Bernhart, J., & Früh, G. (2010). *E-Government* (1. Aufl.). Wien: Linde.

Püttner, G. (2008). Ermessen und Ermessensausübung. *Zeitschrift für öffentliches Recht*, *63*(3), 345–357. doi:10.1007/s00708-008-0007-1

Rademakers, T. (2012). *Activiti in Action*. Shelter Island: Manning.

Raschauer, B. (2009). *Allgemeines Verwaltungsrecht*. Wien: Springer. doi:10.1007/978-3-211-99262-3

Raschauer, B. (2013). *Allgemeines Verwaltungsrecht* (4. Aufl.). Verlag Österreich.

Rector, A., Drummond, N., Horridge, M., Rogers, J., Knublauch, H., Stevens, R., … Wroe, C. (2004). OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors and Common Patterns. In E. Motta, N. R. Shadbolt, A. Stutt, & N. Gibbins (Hrsg.), *Engineering Knowledge in the Age of the Semantic Web* (S. 63–81). Berlin: Springer Berlin Heidelberg. doi:10.1007/978-3-540-30202-5_5

Red Hat. (2013). *Drools Documentation - Version 6.0.1*. Raleigh: Red Hat Inc.

Reisinger, L. (1975). Probleme der Symbolisierung und Formalisierung im Recht. In G. Winkler (Hrsg.), *Rechtstheorie und Rechtsinformatik* (S. 22–50). Wien: Springer.

Rissland, E. L. (1988). Artificial Intelligence and Legal Reasoning: A Discussion of the Field and Gardner's Book. *AI Magazine*, *9*(3), 45–55. doi:10.1609/aimag.v9i3.942

Rissland, E. L., & Ashley, K. D. (1987). A case-based system for trade secrets law. In *Proceedings of the first international conference on Artificial intelligence and law - ICAIL '87* (S. 60–66). doi:10.1145/41735.41743

Rissland, E. L., Ashley, K. D., & Loui, R. P. (2003). AI and Law: A fruitful synergy. *Artificial Intelligence*, *150*(1-2), 1–15. doi:10.1016/S0004-3702(03)00122-X

Ritter, J. (Hrsg.). (1976). *Historisches Wörterbuch der Philosophie*. Darmstadt: Wissenschaftliche Buchgesellschaft.

Röchert-Voigt, T., & Gronau, N. (2010). Web 2.0 für die interne Vernetzung der Verwaltung: Ein Wiki zur Fortschreibung einer Rechtsvorschrift. In M. A. Wimmer, U. Brinkhoff, S. Kaiser, D. Lück-Schneider, E. Schweighofer, & A. Wiebe (Hrsg.), *Vernetzte IT für einen effektiven Staat - Gemeinsame Fachtagung Verwaltungsinformatik und Fachtagung Rechtsinformatik 2010* (S. 123–132). Bonn: Gesellschaft für Informatik.

Rödig, J. (1980). *Schriften zur juristischen Logik*. Berling: Springer.

Rodríguez-Doncel, V., Santos, C., & Casanovas, P. (2014). A model of Air Transport Passenger Incidents and Rights. In R. Hoekstra (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2014: The Twenty-Seventh Annual Conference* (S. 55–60). Amsterdam: IOS Press.

Ropohl, G. (2012). *Allgemeine Systemtheorie: Einführung in transdisziplinäres Denken*. Berlin: edition sigma.

Rountree, D. (2013). *Federated Identity Primer*. Oxford: Elsevier.

Routen, T., & Bench-Capon, T. J. M. (1991). Hierarchical formalizations. *International Journal of Man-Machine Studies*, *35*(1), 69–93. doi:10.1016/S0020-7373(07)80008-3

Rubino, R., & Rotolo, A. (2009). A Java Implementation of Temporal Defeasible Logic. In G. Governatori, J. Hall, & A. Paschke (Hrsg.), *Rule Interchange and Applications* (S. 297–304). New York: Springer.

Rubino, R., Rotolo, A., & Sartor, G. (2006). An OWL Ontology of Fundamental Legal Concepts. In T. M. van Engers (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2006: The Nineteenth Annual Conference* (S. 101–110). Amsterdam: IOS Press.

Russel, S., & Norvig, P. (2010). *Artificial Intelligence - A Modern Approach*. New York: Prentice Hall.

Salhofer, P., Stadlhofer, B., & Tretter, G. (2009). Ontology Driven E-Government. In *2009 Fourth International Conference on Software Engineering Advances* (S. 378–383). IEEE. doi:10.1109/ICSEA.2009.61

Sandro, P. (2011). An Axiomatic Theory of Law. *Res Publica*, *17*(4), 343–354. doi:10.1007/s11158-011-9168-9

Sartor, G. (2005). *Legal Reasoning, A Cognitive Approach to the Law*. Dordrecht: Springer.

Sartor, G. (2006a). Fundamental legal concepts: A formal and teleological characterisation. *Artificial Intelligence and Law*, *14*(1-2), 101–142. doi:10.1007/s10506-006-9009-x

Sartor, G. (2006b). Reasoning with Factors. *Argumentation*, *19*(4), 417–432. doi:10.1007/s10503-005-0509-8

Sartor, G. (2009a). *Defeasibility in Legal Reasoning*.

Sartor, G. (2009b). Legal concepts as inferential nodes and ontological categories. *Artificial Intelligence and Law*, *17*(3), 217–251. doi:10.1007/s10506-009-9079-7

Sartor, G., Walton, D., Macagno, F., & Rotolo, A. (2014). Argumentation Schemes for Statutory Interpretation: A Logical Analysis. In *Legal Knowledge and Information Systems, JURIX 2014: The Twenty-Seventh Annual Conference* (S. 11–20). Amsterdam: IOS Press. doi:Argumentation Schemes for Statutory Interpretation: A Logical Analysis

Saur, K. G. (2009). *Functional Requirements for Bibliographic Records: Final report.* München: IFLA Study Group on the Functional Requirements for Bibliographic Records.

Šavelka, J., Grabmair, M., & Ashley, K. D. (2014). Mining Information from Statutory Texts in Multi-Jurisdictional Settings. In R. Hoekstra (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2014: The Twenty-Seventh Annual Conference* (S. 133–142). Amsterdam: IOS Press. doi:10.3233/978-1-61499-468-8-133

Scheuermann, A., Motta, E., Mulholland, P., Gangemi, A., & Presutti, V. (2013). An empirical perspective on representing time. In *Proceedings of the seventh international conference on Knowledge capture - K-CAP '13* (S. 89–96). New York, New York, USA: ACM Press. doi:10.1145/2479832.2479854

Schrems, I. (2001). Der Begriff der „Logik“ in der Rechtssprache. In E. Schweighofer (Hrsg.), *Auf dem Weg zur ePerson - Aktuelle Fragestellungen der Rechtsinformatik* (S. 219–228). Wien: Verlag Österreich.

Schweighofer, E. (1999). *Rechtsinformatik und Wissensrepräsentation: Automatische Textanalyse im Völkerrecht und Europarecht*. Wien: Springer.

Schweighofer, E. (2010). Semantic Indexing of Legal Documents. In E. Francesconi, S. Montemagni, W. Peters, & D. Tiscornia (Hrsg.), *Semantic Processing of Legal Texts* (Bd. 6036, S. 157–169). Berlin Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-12837-0_9

Schweighofer, E. (2011). Indexing as an Ontological Support for Legal Reasoning. In J. Yearwood & A. Stranieri (Hrsg.), *Technologies for Supporting Reasoning Communities and Collaborative Decision Making* (S. 213–236). New York: IGI Global. doi:10.4018/978-1-60960-091-4.ch012

Schweighofer, E., & Hötzendorfer, W. (2012). Elektronische Identitäten - Öffentliche und private Initiativen. In J. von Lucke, C. P. Geiger, S. Kaiser, E. Schweighofer, & M. A. Wimmer (Hrsg.), *Auf dem Weg zu einer offenen, smarten und vernetzten Verwaltungskultur - Gemeinsame Fachtagung Verwaltungsinformatik und Fachtagung Rechtsinformatik 2012* (S. 137–148). Bonn: Gesellschaft für Informatik.

Schweighofer, E., & Liebwald, D. (2007). Advanced lexical ontologies and hybrid knowledge based systems: First steps to a dynamic legal electronic commentary. *Artificial Intelligence and Law*, *15*(2), 103–115. doi:10.1007/s10506-007-9029-1

Schweighofer, E., Rauber, A., & Dittenbach, M. (2001). Automatic text representation, classification and labeling in European law. In *Proceedings of the 8th international conference on Artificial intelligence and law - ICAIL ’01* (S. 78–87). New York, New York, USA: ACM Press. doi:10.1145/383535.383544

Schweighofer, E., & Winiwarter, W. (1993). Legal expert system KONTERM — Automatic representation of document structure and contents. In V. Mařík, J. Lažanský, & R.

Wagner (Hrsg.), *Database and Expert Systems Applications* (Bd. 720, S. 486–497). Berlin Heidelberg: Springer Berlin Heidelberg. doi:10.1007/3-540-57234-1_49

Sergot, M., Sadri, F., Kowalski, R. A., Kriwaczek, F., Hammond, P., & Cory, H. T. (1986). The British Nationality Act as a logic program. *Communications of the ACM*, *29*(5), 370–386. doi:10.1145/5689.5920

Shadbolt, N., Berners-Lee, T., & Hall, W. (2006). The Semantic Web Revisited. *IEEE Intelligent Systems*, *21*(3), 96–101. doi:10.1109/MIS.2006.62

Sirin, E., & Parsia, B. (2007). SPARQL-DL: SPARQL Query for OWL-DL. In C. Golbreich, A. Kalyanpur, & B. Parsia (Hrsg.), *Proceedings of the Third OWLED Workshop on OWL: Experiences and Directions (OWLED 2007)*. Innsbruck, Austria.

Smith, B. (2004). Ontology. In L. Floridi (Hrsg.), *The Blackwell Guide to the Philosophy of Computing and Information* (S. 155–166). Oxford: Blackwell Publishing.

Smith, B. (2011). Preface. In G. Sartor, P. Casanovas, M. A. Biasiotti, & M. Fernández-Barrera (Hrsg.), *Approaches to Legal Ontologies* (S. vii–viii). New York: Springer.

Sowa, J. F. (1999). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove: Brooks Cole Publishing Co.

Spinoza, B. de. (1976). *Die Ethik*. (F. Bülow, Hrsg.). Stuttgart: Alfred Kröner Verlag.

Stadlhofer, B., & Salhofer, P. (2008). SeGoF: Semantic e-Government Forms. In *Proceedings of the 2008 International Conference on Digital Government Research* (S. 427–428). Digital Government Society of North America.

Steup, M. (2014). Epistemology. In (E. N. Zalta, Hrsg.)*The Stanford Encyclopedia of Philosophy*.

Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, *25*(1-2), 161–197. doi:10.1016/S0169-023X(97)00056-6

Susskind, R. E. (1987). *Expert Systems in Law*. Oxford: Clarendon Press.

Svoboda, W. R. (1975). Das Wiener System. In G. Winkler (Hrsg.), *Rechtstheorie und Rechtsinformatik* (S. 216–247). Wien: Springer.

The Gale Group. (2008). Jurimetrics. In *West's Encyclopedia of American Law*.

Thomason, R. (2013). Logic and Artificial Intelligence. In (E. N. Zalta, Hrsg.)*Stanford Encyclopedia of Philosophy*.

Tran, O. T., Ngo, B. X., Nguyen, M. Le, & Shimazu, A. (2013). Automated reference resolution in legal texts. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Law - ICAIL '13* (Bd. 22, S. 29–60). New York: ACM Press. doi:10.1007/s10506-013-9149-8

Tran, O. T., Ngo, B. X., Nguyen, M. Le, & Shimazu, A. (2014). Automated reference resolution in legal texts. *Artificial Intelligence and Law*, *22*(1), 29–60. doi:10.1007/s10506-013-9149-8

Uschold, M. (1996). Building ontologies: Towards a unified methodology. In *Proceedings of Expert Systems 1996, the 16th Annual Conference of the British Compwer Society Specialist Group on Expert Systems*. Cambridge: The University of Edinburgh.

Uschold, M., & Gruninger, M. (1996). Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, *11*(2), 93–136. doi:10.1017/S0269888900007797

Uschold, M., & King, M. (1995). Towards a Methodology for Building Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing at IJCAI-95*. Montreal.

Valente, A. (1995). *Legal Knowledge Engineering*. Amsterdam: IOS Press.

Valente, A., & Breuker, J. (1995). ON-LINE: An Architecture for Modelling Legal Information. In *Proceedings of the fifth international conference on Artificial intelligence and law - ICAIL '95* (S. 307–315). New York, New York, USA: ACM Press. doi:10.1145/222092.222265

Van de Ven, S., Breuker, J., Hoekstra, R., & Wortel, L. (2008). Automated Legal Assessment in OWL 2. In E. Francesconi, G. Sartor, & D. Tiscornia (Hrsg.), *Legal Knowledge and Information Systems - JURIX 2008: The Twenty-First Annual Conference on Legal Knowledge and Information Systems* (S. 170 – 175). Amsterdam: IOS Press. doi:10.3233/978-1-58603-952-3-170

Van de Ven, S., Hoekstra, R., Breuker, J., Wortel, L., & El Ali, A. (2008). Judging Amy: Automated Legal Assessment using OWL 2. In C. Dolbear, A. Ruttenberg, & U. Sattler (Hrsg.), *Proceedings of the Fifth OWLED Workshop on OWL: Experiences and Directions (OWLED 2008)*. Karlsruhe, Germany.

Van Engers, T. M. (2001). *Knowledge Management: The Role of Mental Models in Business Systems Design*. University of Amsterdam.

Van Engers, T. M., Gerrits, R., Boekenoogen, M., Glassée, E., & Kordelaar, P. J. M. (2001). POWER: Using UML/OCL for Modeling Legislation - an application report. In *Proceedings of the 8th international conference on Artificial intelligence and law - ICAIL '01* (S. 157–167). New York, New York, USA: ACM Press. doi:10.1145/383535.383554

Ventzke, S., Weiß, S., Wimmer, M. A., & Makolm, J. (2010). Stakeholder-Beteiligung in der Entwicklung eines Virtual Company Dossiers mithilfe von Web 2.0. In M. A. Wimmer, U. Brinkhoff, S. Kaiser, D. Lück-Schneider, E. Schweighofer, & A. Wiebe (Hrsg.), *Vernetzte IT für einen effektiven Staat - Gemeinsame Fachtagung Verwaltungsinformatik und Fachtagung Rechtsinformatik 2010* (S. 109–122). Bonn: Gesellschaft für Informatik.

Walls, C. (2011). *Spring in Action* (3. Aufl.). Shelter Island: Manning.

Walton, D. (2005). *Argumentation Methods for Artificial Intelligence in Law*. New York: Springer.

Walton, D. (2006). *Fundamentals of Critical Argumentation*. Cambridge: Cambridge University Press.

Walton, D. (2009). Argumentation Theory: A Very Short Introduction. In G. Simari & I. Rahwan (Hrsg.), *Argumentation in Artificial Intelligence* (S. 1–22). New York: Springer US. doi:10.1007/978-0-387-98197-0_1

Walton, D. (2010). Similarity, precedent and argument from analogy. *Artificial Intelligence and Law*, *18*(3), 217–246. doi:10.1007/s10506-010-9102-z

Walton, D. (2011). An Argumentation Model of Deliberative Decision-Making. In J. Yearwood & A. Stranieri (Hrsg.), *Technologies for Supporting Reasoning Communities and Collaborative Decision Making* (S. 1–17). New York: IGI Global. doi:10.4018/978-1-60960-091-4.ch001

Walton, D. (2013a). Argument from analogy in legal rhetoric. *Artificial Intelligence and Law*, *21*(3), 279–302. doi:10.1007/s10506-013-9139-x

Walton, D. (2013b). Baseballs and arguments from fairness. *Artificial Intelligence and Law*, *22*(4), 423–449. doi:10.1007/s10506-013-9151-1

Walton, D., & Gordon, T. F. (2012). The Carneades model of argument invention. *Pragmatics & Cognition*, *20*(1), 1–31. doi:10.1075/pc.20.1.01wal

Wason, P. C. (1968). Reasoning about a rule. *The Quarterly journal of experimental psychology*, *20*(3), 273–81. doi:10.1080/14640746808400161

Weinberger, O. (2006). Der handlungstheoretische Zutritt zur Normenlogik. *Philosophia Scientiæ*, (10-1), 159–176. doi:10.4000/philosophiascientiae.503

Wielinger, G. (2008). *Einführung in das österreichische Verwaltungsverfahrensrecht* (11. Aufl.). Graz: Leykam.

Wilson, G., & Shpall, S. (2012). Action. In *Stanford Encyclopedia of Philosophy*. Zalta, Edward N.

Winkels, R., & Hoekstra, R. (2012). Automatic extraction of legal concepts and definitions. In B. Schafer (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2012: The Twenty-*

Fifth Annual Conference (Bd. 250, S. 157–166). Amsterdam: IOS Press. doi:10.3233/978-1-61499-167-0-157

Wyner, A. (2008). An ontology in OWL for legal case-based reasoning. *Artificial Intelligence and Law*, *16*(4), 361–387. doi:10.1007/s10506-008-9070-8

Wyner, A., Bench-Capon, T. J. M., & Atkinson, K. (2011). Towards formalising argumentation about legal cases. In *Proceedings of the 13th International Conference on Artificial Intelligence and Law - ICAIL '11* (S. 1–10). New York: ACM Press. doi:10.1145/2018358.2018359

Wyner, A., & Hoekstra, R. (2012). A legal case OWL ontology with an instantiation of Popov v. Hayashi. *Artificial Intelligence and Law*, *20*(1), 83–107. doi:10.1007/s10506-012-9119-6

Wyner, A., & Peters, W. (2011). On rule extraction from regulations. In K. Atkinson (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2011: The Twenty-Fourth Annual Conference* (Bd. 235, S. 113–122). Amsterdam: IOS Press. doi:10.3233/978-1-60750-981-3-113

Yoshida, Y., Honda, K., Sei, Y., Nakagawa, H., Tahara, Y., & Ohsuga, A. (2013). Towards semi-automatic identification of functional requirements in legal texts for public administration. In K. D. Ashley (Hrsg.), *Legal Knowledge and Information Systems, JURIX 2013: The Twenty-Sixth Annual Conference* (Bd. 259, S. 175–184). Amsterdam: IOS Press. doi:10.3233/978-1-61499-359-9-175

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353.

Zoglauer, T. (2008). *Einführung in die formale Logik für Philosophen* (4. Aufl.). Göttingen: Vandenhoeck & Ruprecht.

Abbildungsverzeichnis

Abbildung 1: Benchmark Effective Government 21
Abbildung 2: Wissen als eine Funktion von Typ und Qualität (de Jong & Ferguson-Hessler, 1996) 118
Abbildung 3: RDF-Tripel 156
Abbildung 4: Semantic Web Stack (Berners-Lee, 2000) 158
Abbildung 5: Aktualisierter Semantic Web Stack (Bratt, 2008) 160
Abbildung 6: Legal Semantic Web Stack (Biasiotti et al., 2008) 160
Abbildung 7: Ausdrucksstärke verschiedener Sprachen (Guarino et al., 2009, S. 13) 162
Abbildung 8: Die Daten des Antragstellers 221
Abbildung 9: Ausgabe des Ergebnisses 222
Abbildung 10: Protokollauszug der Berechnung der Zusatzrente 223
Abbildung 11: Ausschnitt der Template-Methode "calculate". 234
Abbildung 12: Methoden der JavaCalculationEngine 235
Abbildung 13: Berechnung des Grundbetrags der Grundrente 237
Abbildung 14: Mapping MdE auf Grundrente in der Klasse InMemoryDataRepository 238
Abbildung 15: Erhöhung der Grundrente gemäß § 11 Abs. 2 und 3 239
Abbildung 16: Implementierung der Pflegezulage gemäß § 18 242
Abbildung 17: Berechnung der Familienzulagen ohne Kürzung 243
Abbildung 18: Berechnung der Erhöhung der Zusatzrente durch Familienzulagen 244
Abbildung 19: Berechnung des Grundbetrags der Zusatzrente gemäß § 12 Abs. 2 245
Abbildung 20: Domänenontologie des KOVG 266
Abbildung 21: Class-Individual Mirror Pattern 273
Abbildung 22: Die Ontologie in Aktion 285
Abbildung 23: Architektur Reasoning Module 290
Abbildung 24: Zusammenhänge zwischen Sprache, Modell und Inferenzmaschine 320
Abbildung 25: Beispiel einer retroaktiven Modifikation 396

Abstract

Der Einsatz von Software zur Unterstützung der Verwaltung hat in Österreich eine lange Tradition und reicht bis in die 1970er-Jahre zurück. Trotz dieses Umstands fehlt es bislang an einer einheitlichen Methode und an Modellen zur Formalisierung von Rechtsnormen. Die Implementierung und Anpassung von Expertensystemen ist auch heute aufwändig, fehleranfällig und teuer. Die Verwaltungsautomatisierung in Österreich und das Gebiet Künstliche Intelligenz und Recht entwickelten sich parallel, ohne voneinander Kenntnis zu nehmen.

Die Dissertation stellt die Hypothese auf, dass die Unzulänglichkeiten gängiger Modelle des Rechts vermieden werden können, indem die Rechtstheorie stärker berücksichtigt wird. Zum Beweis dieser Hypothese wurde das Kriegsopferversorgungsgesetz stellvertretend für das Verwaltungsrecht ausgewählt und anschließend in Java und einer in OWL 2 modellierten Ontologie formalisiert.

Aus den Modellierungsversuchen werden Erkenntnisse induktiv abgeleitet und durch theoretische Ausführungen vertieft, um die Grundlage für die Entwicklung des Konzepts einer neuartigen Rule Engine zu schaffen, genannt „rOWLer“, die auf einem ausgereiften temporalen Modell basiert und anfechtbares Schließen beherrscht. Das formale Modell basiert auf einer soliden rechtstheoretischen Grundlage und ist in der Lage, mit Novellen des Gesetzes effizient umzugehen. Ferner kann das System im Einzelfall die zu einem bestimmten Zeitpunkt anwendbaren Normen ermitteln. Die Rule Engine kombiniert die Ausdrucksstärke von OWL 2 und Regeln, bestehend aus Antezedens und Konsequenz, um eine adäquate Repräsentation der Normen und des Sachverhalts zu gewährleisten, die isomorph mit der jeweiligen Rechtsgrundlage ist.

Aus technischer Sicht besteht das System aus mehreren Modulen und bietet zahlreiche Erweiterungspunkte, um das Verhalten an spezielle Anforderungen effizient anpassen zu können. In diesem Sinne kann die Engine auch als flexibles Framework zur Implementierung von juristischen Expertensystemen verstanden werden. Diese Forschung versucht den Graben zwischen der syntaktischen Repräsentation von Normen (in XML oder anderen Formaten) und dem Bedürfnis der Verwaltung nach einer leistungstarken und einfach zu verwendenen Rule Engine zu überbrücken. Die Architektur von rOWLer ist am Semantic Web Stack ausgerichtet und kompatibel mit LegalRuleML, einem aufkommenden Standard zur

Modellierung von Rechtsnormen. Die Entwicklung von rOWLer komplementiert Anstrengungen, einen anerkannten Standard für den Rule Layer des Semantic Web Stacks zu etablieren.

Der Diskurs schließt mit einer Zusammenfassung der wichtigsten Erkenntnisse der Dissertation und gibt Ausblick auf zukünftige Forschungen.

Abstract (Englisch)

The use of software to support public administration has a long tradition in Austria which dates back to the 1970s. Despite that circumstance, there is a lack of a consistent method and models to formalize legal norms. The implementation and adaption of legal expert systems is still complex, error-prone and costly. In Austria, the automation of public administration on the one hand and research in Artificial Intelligence and Law on the other hand have developed in parallel, without taking notice of each other.

This thesis proposes the hypothesis that the drawbacks of current models of law can be avoided following legal theory. In order to prove this hypothesis, a statute has been exemplarily selected – the Kriegsopverversorgungsgesetz, which may be freely translated to War Vicitims Compensation Act – and formalized using Java and OWL 2.

Results are retrieved by deductive means from these efforts and are enriched with theoretical refelections to establish the conceptual foundation for developing a novel rule engine called “rOWLer”, which is built on a sophisticated temporal model and is capable of defeasible reasoning. The formal model is based on a solid theoretical foundation and is capable of dealing with legal change over time in an efficient manner. Moreover, the system is able to determine applicable norms in a particular case at a certain point in time. The rule engine combines the expressiveness of OWL 2 and rules, which consist of antecedent and consequent, to ensure an appropriate representation of norms and material facts, fostering isomorphism with legal sources.

From a technical perspective the system consists of several modules offering various extension points to allow for tweaking the behaviour to special reqirements in an effective way. In this sense the engine can be conceived as a flexible framework for implementing legal expert systems. This research tries to fill the gap between the syntactical representation of norms (in XML or other formats) and the need of public administration for a powerful, yet easy to use and customizable legal rule engine. The architecture of rOWLer is aligned with the Semantic Web Stack and is compatible with LegalRuleML, an upcoming standard for modelling legal rules. Hence the development of rOWLer complements efforts to acquire an acknowledged standard for the rule layer of the Semantic Web Stack.

The discourse closes with a summary of the most important findings of this thesis and gives an outlook on future research.

Curriculum Vitae

ANGABEN ZUR PERSON

Name	Ing. Mag. Johannes Sebastian Scharf
Adresse	Wiener Straße 73/1/6, 3002 Purkersdorf
Telefon	+43 664 5409421
E-Mail	johannes.scharf@gmx.at
Staatsangehörigkeit	Österreich
Geburtsdatum	13. Februar 1984

AUSBILDUNG

September 2014	Lex Summer School 2014 an der **Universität Bologna** (Campus Ravenna), Ravenna, Italien
	Erhalt eines Stipendiums der Universität Bologna
Seit Oktober 2009	Doktoratsstudium der Rechtswissenschaften an der **Universität Wien**
Oktober 2005 bis Oktober 2009	Diplomstudium der Rechtswissenschaften an der **Universität Wien** mit Schwerpunktausbildung „Computer und Recht“, Abschluss als Mag. iur.
September 1998 bis Juni 2003	**HTBLuVA**, Abteilung EDV und Organisation, St. Pölten

PUBLIKATIONEN

Februar 2015	*Scharf*, rOWLer - A Hybrid Rule Engine for Legal Reasoning, in *Schweighofer/Kummer/Hötzendorfer* (Hrsg), Kooperation – Tagungsband des 18. Internationalen Rechtsinformatik Symposions (2015) 155
Dezember 2014	*Scharf*, rOWLer - A Hybrid Rule Engine for Legal Reasoning, in *Villata/Peroni/Palmirani* (Hrsg), Proceedings of the Semantic Web for the Law and Second Jurix Doctoral Consortium Workshops (2014)
Februar 2014	*Scharf*, Die Anforderungen an Sprachen zur Formalisierung von Verwaltungsrecht, in *Schweighofer/Kummer/Hötzendorfer* (Hrsg), Transparenz – Tagungsband des 17. Internationalen Rechtsinformatik Symposions (2014) 129
Februar 2013	*Scharf*, Abstraktion als Grundlage der automatisierten Vollziehung, in *Schweighofer/Kummer/Hötzendorfer* (Hrsg), Abstraktion und Applikation – Tagungsband des 16. Internationalen Rechtsinformatik Symposions (2013) 111
Juni 2012	Seminararbeit mit dem Titel „Denial of Service aus strafrechtlicher Sicht“

Februar 2012	*Scharf*, Die Formalisierung des Kriegsopferversorgungsgesetzes, in *Schweighofer/Kummer/Hötzendorfer* (Hrsg), Transformation juristischer Sprachen – Tagungsband des 15. Internationalen Rechtsinformatik Symposions (2012) 99
Februar 2009	Seminararbeit mit dem Titel „Die Haftung des Hackers“
Juni 2008	Seminararbeit mit dem Titel „Die Haftung von WLAN-Betreibern“

VORTRÄGE

Februar 2015	Vortrag mit dem Titel „rOWLer – A hybrid rule engine for legal reasoning“ und Vorsitz der Session „Juristische Informatik-Systeme und Anwendungen“ auf dem IRIS 2015 - Das 18. Internationale Rechtsinformatik Symposion, Universität Salzburg
Dezember 2014	Vortrag mit dem Titel „rOWLer – A hybrid rule engine for legal reasoning“ auf der JURIX 2014 - The 27th International Conference on Legal Knowledge and Information Systems, Jagiellonen-Universität, Krakau, Polen
Juli 2014	Vortrag mit dem Titel „Computable Models of Public Law“ auf der SubTech2014 - Thirteenth International Conference on Substantive Technology in Legal Education and Practice, Universität Wien
Februar 2014	Vortrag mit dem Titel „Die Anforderungen an Sprachen zur Formalisierung von Verwaltungsrecht“ auf dem IRIS 2014 - Das 17. Internationale Rechtsinformatik Symposion, Universität Salzburg
Februar 2013	Vortrag mit dem Titel „Abstraktion als Grundlage der automatisierten Vollziehung“ auf dem IRIS 2013 - Das 16. Internationale Rechtsinformatik Symposion, Universität Salzburg
Februar 2012	Vortrag mit dem Titel „Die Formalisierung des KOVG“ auf dem IRIS 2012 - Das 15. Internationale Rechtsinformatik Symposion, Universität Salzburg
Dezember 2011	Vortrag mit dem Titel „The Formalisation of the War Victims Supply Act“ auf der JURIX 2011 - The 24th International Conference on Legal Knowledge and Information Systems, Universität Wien

BERUFSERFAHRUNG

Seit Oktober 2014	Jurist bei **BRZ GmbH**, Wien
Jänner 2012 bis September 2014	Software Engineer / Senior bei **BRZ GmbH**, Wien
Oktober 2011 bis Februar	Gerichtsjahr am **Bezirksgericht Purkersdorf** (Strafrecht und

2012	Familienrecht) und am **Handelsgericht Wien** (Insolvenzrecht)
Jänner 2008 bis Dezember 2011	Software Engineer / Professional bei **BRZ GmbH**, Wien
Juni 2005 bis Dezember 2007	Programmierer / Professional bei **BRZ GmbH**, Wien
Juni 2004 bis Mai 2005	Junior Programmierer bei **BRZ GmbH**, Wien
Juli 2001	Ferialpraxis bei **Scharf GmbH**, Neulengbach
Juli 1999	Ferialpraxis bei **Grabner Instruments GmbH**, Wien

Anhang

Anhang A. Implementierung des Kriegsopferversorgungsgesetzes in Java

Anhang A.1. Beschaedigtenrente.java

```
package at.js.diss.simple.domain;

import static at.js.diss.simple.service.support.Constants.ZERO;

import java.io.Serializable;
import java.util.List;

import org.joda.time.LocalDate;

import com.google.common.base.Objects;

/**
 * Klasse die eine Beschädigtenrente repräsentiert. Eine solche besteht
 * grundsätzlich aus Grundrente und Zusatzrente. Die Beschädigtenrente wird
 * monatlich berechnet und ist zum Monatsersten fällig (§ 66).
 * <p/>
 * Obwohl nach der Terminologie des Gesetzes ("zur Beschädigtenrente") die
 * einzelnen Zulagen nicht Bestandteil der Beschädigtenrente selbst sind,
 * sondern zu dieser gewährt werden, beinhaltet diese Klasse auch die Zulagen.
 * Dies dient der einfacheren Strukturierung der Daten.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class Beschaedigtenrente implements Serializable, Betrag {
	private static final long serialVersionUID = 1L;

	private LocalDate datum;
	private Grundrente grundrente;
	private Zusatzrente zusatzrente;

	// Zulagen
	private Euro schwerstbeschaedigtenzulage;
	private Euro pflegezulage;
	private Euro blindenzulage;
	private Euro blindenfuehrzulage;
	private List<Euro> kleiderpauschalen;

	public LocalDate getDatum() {
		return datum;
	}

	/**
	 * Setzt das Monat und das Jahr für das diese Beschädigtenrente berechnet
	 * wurde. Der Tag <b>muss</b> der 1. Tag des Monats sein und hat keine
	 * Relevanz.
	 *
	 * @param datum
	 *            das Datum (Monat und Jahr) der Berechnung
	 */
	public void setDatum(LocalDate datum) {
		this.datum = datum;
	}
```

```
	public Grundrente getGrundrente() {
		return grundrente;
	}

	/**
	 * Grundrente gemäß § 11 KOVG.
	 *
	 * @param grundrente
	 *            die Grundrente
	 */
	public void setGrundrente(Grundrente grundrente) {
		this.grundrente = grundrente;
	}

	public Zusatzrente getZusatzrente() {
		return zusatzrente;
	}

	/**
	 * Zusatzrente gemäß § 12 KOVG.
	 *
	 * @param zusatzrente
	 *            die Zusatzrente
	 */
	public void setZusatzrente(Zusatzrente zusatzrente) {
		this.zusatzrente = zusatzrente;
	}

	public Euro getSchwerstbeschaedigtenzulage() {
		return schwerstbeschaedigtenzulage;
	}

	/**
	 * Schwerstbeschädigtenzulage gemäß § 11a KOVG.
	 */
	public void setSchwerstbeschaedigtenzulage(Euro schwerstbeschaedigtenzulage)
{
		this.schwerstbeschaedigtenzulage = schwerstbeschaedigtenzulage;
	}

	public Euro getPflegezulage() {
		return pflegezulage;
	}

	public void setPflegezulage(Euro pflegezulage) {
		this.pflegezulage = pflegezulage;
	}

	public Euro getBlindenzulage() {
		return blindenzulage;
	}

	public void setBlindenzulage(Euro blindenzulage) {
		this.blindenzulage = blindenzulage;
	}

	public Euro getBlindenfuehrzulage() {
		return blindenfuehrzulage;
	}

}
```

```
	public void setBlindenfuehrzulage(Euro blindenfuehrzulage) {
		this.blindenfuehrzulage = blindenfuehrzulage;
	}

	public List<Euro> getKleiderpauschalen() {
		return kleiderpauschalen;
	}

	public void setKleiderpauschalen(List<Euro> kleiderpauschalen) {
		this.kleiderpauschalen = kleiderpauschalen;
	}

	/**
	 * Berechnet den Betrag der Beschädigtenrente:
	 * <code>Grundbetrag + Zusatzrente + Zulagen</code>.
	 * <p/>
	 * Die Zulagen berechnen sich wie folgt:
	 * <code>Schwerstbeschädigtenzulage + Pflegezulage/Blindenzulage + Kleider-
und Wäschepauschale</code>.
	 *
	 * @return den Betrag der Beschädigtenrente (niemals {@code null})
	 */
	@Override
	public Euro getBetrag() {
		Euro betrag = ZERO;

		// Grundrente darf nicht null sein
		betrag = betrag.plus(grundrente.getBetrag());

		if (zusatzrente != null) {
			betrag = betrag.plus(zusatzrente.getBetrag());
		}

		// Zulagen
		betrag = betrag.plus(getSummeZulagen());

		return betrag;
	}

	/**
	 * @return {@code true} wenn zumindest eine Zulage gewährt wird
	 */
	public boolean hasZulagen() {
		return schwerstbeschaedigtenzulage != null || pflegezulage != null
				|| blindenzulage != null || blindenfuehrzulage != null
				|| kleiderpauschalen != null;
	}

	/**
	 * Berechnet die Summe der Zulagen.
	 *
	 * @return die Summe
	 */
	public Euro getSummeZulagen() {
		Euro summe = ZERO;

		if (schwerstbeschaedigtenzulage != null) {
			summe = summe.plus(schwerstbeschaedigtenzulage);
		}
		if (pflegezulage != null) {
```

```
                summe = summe.plus(pflegezulage);
            }
            if (blindenzulage != null) {
                summe = summe.plus(blindenzulage);
            }
            if (blindenfuehrzulage != null) {
                summe = summe.plus(blindenfuehrzulage);
            }
            summe = summe.plus(getSummeKleiderpauschalen());

            return summe;
        }

        /**
         * Berechnet die Summe der Kleiderpauschalen.
         */
        public Euro getSummeKleiderpauschalen() {
            Euro summe = ZERO;

            if (kleiderpauschalen != null) {
                for (Euro pauschale : kleiderpauschalen) {
                    summe = summe.plus(pauschale);
                }
            }

            return summe;
        }

        @Override
        public String toString() {
            return Objects.toStringHelper(this)
                    .add("datum", datum.getYear() + "-" +
datum.getMonthOfYear())
                    .add("grundrente", grundrente).add("zusatzrente",
zusatzrente)
                    .add("betrag", getBetrag()).toString();
        }
}
```

Anhang A.2. Beschaedigter.java

```
package at.js.diss.simple.domain;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.joda.time.DateTime;

import at.js.diss.simple.service.support.Constants;

import com.google.common.base.Objects;

/**
 * Klasse die die Daten eines Beschädigten repräsentiert die für die Berechnung
 * der Beschädigtenrente relevant sind.
 *
 * @author J. Scharf
 * @since 1.0.0
```

```
 */
public class Beschaedigter implements Serializable {
	private static final long serialVersionUID = 1L;

	private String name;
	private DateTime geburtsdatum;
	private Geschlecht geschlecht;
	private int angehoerige;
	private Euro einkommen;
	private List<Dienstbeschaedigung> dienstbeschaedigungen;
	private Pflegestufe pflegestufe;
	private Blindheit blindheit;
	// Diätverpflegung
	private Diaetverpflegung diaetverpflegung;
	private List<Kleiderverbrauch> kleiderverbrauch;

	private Beschaedigter(Builder builder) {
		this.name = builder.name;
		this.geburtsdatum = builder.geburtsdatum;
		this.geschlecht = builder.geschlecht;
		this.angehoerige = builder.angehoerige;
		this.einkommen = builder.einkommen;
		this.dienstbeschaedigungen = Collections
				.unmodifiableList(builder.dienstbeschaedigungen);
		this.pflegestufe = builder.pflegestufe;
		this.blindheit = builder.blindheit;
		this.diaetverpflegung = builder.diaetverpflegung;
		this.kleiderverbrauch = Collections
				.unmodifiableList(builder.kleiderverbrauch);
	}

	public String getName() {
		return name;
	}

	public DateTime getGeburtsdatum() {
		return geburtsdatum;
	}

	public Geschlecht getGeschlecht() {
		return geschlecht;
	}

	public int getAngehoerige() {
		return angehoerige;
	}

	public Euro getEinkommen() {
		return einkommen;
	}

	public List<Dienstbeschaedigung> getDienstbeschaedigungen() {
		return dienstbeschaedigungen;
	}

	/**
	 * Ermittelt die höchste MdE der einzelnen Diensbeschädigungen.
	 */
	public int getMaxMdE() {
		int maxMde = 0;
```

```
		for (Dienstbeschaedigung dienstbeschaedigung : dienstbeschaedigungen)
{
			if (dienstbeschaedigung.getMde() > maxMde) {
				maxMde = dienstbeschaedigung.getMde();
				break;
			}
		}

		return maxMde;
	}

	/**
	 * Berechnet die Summe der MdE der einzelnen Dienstbeschädigunen.
	 * <p/>
	 * <b>Hinweis:</b> Die Summe der MdE wird zur Prüfung des Anspruchs auf
	 * Schwerstbeschädigtenzulage gemäß § 11a benötigt. Die bloße Summierung ist
	 * eine Vereinfachung. Richtigerweise sind die Bestimmungen der Absätze 2
	 * und 3 zu beachten.
	 *
	 * @return die Summe der MdE
	 */
	public int getSummeMdE() {
		int summe = 0;

		for (Dienstbeschaedigung dienstbeschaedigung : dienstbeschaedigungen)
{
			summe += dienstbeschaedigung.getMde();
		}

		return summe;
	}

	public Pflegestufe getPflegestufe() {
		return pflegestufe;
	}

	public Blindheit getBlindheit() {
		return blindheit;
	}

	/**
	 * Prüft ob {@code blindheit} != {@code null}.
	 *
	 * @return {@code true} wenn der Beschädigte blind ist, {@code false} wenn
	 *         nicht
	 */
	public boolean isBlind() {
		return this.blindheit != null;
	}

	public Diaetverpflegung getDiaetverpflegung() {
		return diaetverpflegung;
	}

	public List<Kleiderverbrauch> getKleiderverbrauch() {
		return kleiderverbrauch;
	}

	@Override
```

```
	public String toString() {
		return Objects.toStringHelper(this).add("name", name)
				.add("geburtsdatum", geburtsdatum)
				.add("geschlecht", geschlecht).add("angehoerige",
angehoerige)
				.add("einkommen", einkommen).add("pflegestufe",
pflegestufe)
				.add("blindheit", blindheit)
				.add("diaetzuschuss", diaetverpflegung)
				.add("kleiderverbrauch", kleiderverbrauch)
				.add("dienstbeschaedigungen",
dienstbeschaedigungen).toString();
	}

	public static class Builder {
		private String name;
		private DateTime geburtsdatum;
		private Geschlecht geschlecht;
		private int angehoerige = 0;
		private Euro einkommen = Constants.ZERO;
		private List<Dienstbeschaedigung> dienstbeschaedigungen = new
ArrayList<>();
		private Pflegestufe pflegestufe;
		private Blindheit blindheit;
		// Diätverpflegung
		private Diaetverpflegung diaetverpflegung;
		private List<Kleiderverbrauch> kleiderverbrauch = new ArrayList<>();

		public Builder name(String name) {
			this.name = name;
			return this;
		}

		public Builder geburtsdatum(DateTime geburtsdatum) {
			this.geburtsdatum = geburtsdatum;
			return this;
		}

		public Builder geschlecht(Geschlecht geschlecht) {
			this.geschlecht = geschlecht;
			return this;
		}

		/**
		 * Setzt die Anzahl der Angehörigen für die eine Familienzulage gemäß
§ 16 zusteht.
		 * <p/>
		 * Die ist eine Vereinfachung da ansonsten die Familienzulage für
		 * Kinder grundsätzlich nur bis zum vollendeten 18. Lebensjahr
		 * zusteht und gemäß § 41 längstens bis zur Vollendung des 27.
		 * Lebensjahres.
		 *
		 * @param anzahl
		 *            die Anzahl der Angehörigen
		 */
		public Builder angehoerige(int anzahl) {
			this.angehoerige = anzahl;
			return this;
		}
```

```
	public Builder einkommen(Euro einkommen) {
		this.einkommen = einkommen;
		return this;
	}

	/**
	 * Fügt eine Dienstbeschädigung hinzu.
	 */
	public Builder dienstbeschaedigung(int mde) {
		Dienstbeschaedigung besch = new Dienstbeschaedigung();
		besch.setMde(mde);
		this.dienstbeschaedigungen.add(besch);

		return this;
	}

	/**
	 * Setzt die Pflegestufe des Beschädigten gemäß § 18 KOVG.
	 *
	 * @param pflegestufe
	 *            die Pflegestufe
	 */
	public Builder pflegestufe(Pflegestufe pflegestufe) {
		this.pflegestufe = pflegestufe;
		return this;
	}

	/**
	 * Wird gesetzt wenn der Beschädigte blind im Sinne des § 19 KOVG
ist.
	 *
	 * @param pflegestufe
	 *            die {@link Pflegestufe} der die Blindheit entspricht
	 */
	public Builder blind(Pflegestufe pflegestufe) {
		Blindheit blindheit = new Blindheit();
		blindheit.setStufe(pflegestufe);
		this.blindheit = blindheit;
		return this;
	}

	/**
	 * Diätverpflegung zur Zusatzrente gemäß § 14 KOVG.
	 */
	public Builder diaetverpflegung(Diaetverpflegung diaetverpflegung) {
		this.diaetverpflegung = diaetverpflegung;
		return this;
	}

	/**
	 * Fügt einen außergewöhnlichen Kleider- und Wäscheverbrauch gemäß §
20a
	 * KOVG hinzu.
	 */
	public Builder kleiderverbrauch(Kleiderverbrauch verbrauch) {
		this.kleiderverbrauch.add(verbrauch);
		return this;
	}

	public Beschaedigter build() {
```

```
                return new Beschaedigter(this);
            }
        }
}
```

Anhang A.3. Betrag.java

```
package at.js.diss.simple.domain;

/**
 * Interface das die Funktionalität zur Berechnung eines Betrags repräsentiert.
 * Dieses Interfaces wird insbesondere von Klassen implementiert die eine
 * monetäre Leistung repräsentieren die sich aus mehreren Teilbeträgen
 * zusammensetzt.<br>
 * Insofern kann dieses Interface auch als Maker gesehen werden.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public interface Betrag {
	/**
	 * Berechnet den Betrag der Leistung der sich unter Umständen aus der Summe
	 * der Teilleistungen ergibt.
	 *
	 * @return den Betrag in {@link Euro} (niemals {@code null})
	 */
	public Euro getBetrag();
}
```

Anhang A.4. Blindheit.java

```
package at.js.diss.simple.domain;

import java.io.Serializable;

import com.google.common.base.Objects;

/**
 * Klasse die die "Stufe" der Blindheit repräsentiert. Siehe § 19 KOVG.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class Blindheit implements Serializable {
	private static final long serialVersionUID = 1L;

	private Pflegestufe stufe;

	public Pflegestufe getStufe() {
		return stufe;
	}

	public void setStufe(Pflegestufe stufe) {
		this.stufe = stufe;
	}

	@Override
	public String toString() {
		return Objects.toStringHelper(this).add("stufe", stufe).toString();
```

```
	}
}
```

Anhang A.5. Diaetverpflegung.java

```
package at.js.diss.simple.domain;

/**
 * Diätverpflegung gemäß § 14 KOVG. Die einzelnen Instanzen repräsentieren § 14
 * Abs. 1 Zf. 1 bis 5 KOVG.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public enum Diaetverpflegung {
	/**
	 * § 14 Abs. 1 Zf. 1.
	 */
	I,
	/**
	 * § 14 Abs. 1 Zf. 2.
	 */
	II,
	/**
	 * § 14 Abs. 1 Zf. 3.
	 */
	III,
	/**
	 * § 14 Abs. 1 Zf. 4.
	 */
	IV,
	/**
	 * § 14 Abs. 1 Zf. 5.
	 */
	V;
}
```

Anhang A.6. Dienstbeschaedigung.java

```
package at.js.diss.simple.domain;

import java.io.Serializable;

import com.google.common.base.Objects;

/**
 * Klasse die eine Dienstbeschädigung repräsentiert.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class Dienstbeschaedigung implements Serializable {
	private static final long serialVersionUID = 1L;

	private int mde;

	public int getMde() {
		return mde;
	}
```

```
	public void setMde(int mde) {
		this.mde = mde;
	}

	@Override
	public String toString() {
		return Objects.toStringHelper(this).add("mde", mde).toString();
	}
}
```

Anhang A.7. Euro.java

```
package at.js.diss.simple.domain;

import java.io.Serializable;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Currency;

import com.google.common.base.Preconditions;

/**
 * An amount of Euro with two decimal places.
 * <p/>
 * This class represents a quantity of money in Euro with two decimal places.
 * Underneath the amount is stored in a {@link BigDecimal}. The result of any
 * operation will be rounded to two decimal places.<br>
 * {@link RoundingMode#HALF_UP} is used for rounding throughout the class.
 * Therefore is not recommended to use this class for statistical calculations
 * which many rounding operations. In such cases {@link RoundingMode#HALF_EVEN}
 * ("Banker's rounding") is recommended which means this class shouldn't be
 * used.
 * <p/>
 * Although this class represents strictly Euro it cannot prevent passing in an
 * amount which in fact has another currency. So it's up to the user to ensure
 * proper exchange before passing the amount to one of the factory methods of
 * this class.
 * <p/>
 * <b>Note:</b> This class is immutable and thread-safe.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class Euro implements Serializable, Comparable<Euro> {
	private static final long serialVersionUID = 1L;

	private static final Currency currency = Currency.getInstance("EUR");
	private static final RoundingMode roundingMode = RoundingMode.HALF_UP;

	private BigDecimal amount;
	private boolean roundToTenCent = false;

	/**
	 * Obtain an instance of {@link Euro} from a {@link String}.
	 *
	 * @param amount
	 *            the amount of money in Euro
	 * @return the new instance (never {@code null})
	 * @throws NullPointerException
	 *             if amount is {@code null}
```

```
     */
    public static Euro of(String amount) {
          Preconditions.checkNotNull(amount, "The amount must not be null.");

          try {
                return of(new BigDecimal(amount));
          } catch (NumberFormatException ex) {
                throw new IllegalArgumentException("Invalid amount '" + amount
                            + "'.", ex);
          }
    }

    /**
     * Obtain an instance of {@link Euro} from a {@code long}.
     *
     * @param amount
     *            the amount of money in Euro
     * @return the new instance (never {@code null})
     * @throws NullPointerException
     *             if amount is {@code null}
     */
    public static Euro of(long amount) {
          return of(BigDecimal.valueOf(amount));
    }

    /**
     * Obtain an instance of {@link Euro} from a {@code double}.
     *
     * @param amount
     *            the amount of money in Euro
     * @return the new instance (never {@code null})
     * @throws NullPointerException
     *             if amount is {@code null}
     */
    public static Euro of(double amount) {
          return of(BigDecimal.valueOf(amount));
    }

    /**
     * Obtain an instance of {@link Euro} from a {@link String}.
     *
     * @param amount
     *            the amount of money in Euro
     * @return the new instance (never {@code null})
     * @throws NullPointerException
     *             if amount is {@code null}
     */
    public static Euro of(BigDecimal amount) {
          Preconditions.checkNotNull(amount, "The amount must not be null.");

          return new Euro(amount.setScale(currency.getDefaultFractionDigits(),
                      roundingMode));
    }

    /**
     * Private constructor for internal use only.
     */
    private Euro(BigDecimal amount) {
          this.amount = amount;
    }
```

```
/**
 * Private constructor for internal use only with additional rounding mode
 * parameter.
 */
private Euro(BigDecimal amount, boolean roundToTenCent) {
	this.amount = amount;
	this.roundToTenCent = roundToTenCent;
}

/**
 * Returns a copy of this Euro value with the amount subtracted.
 *
 * @param amount
 *            the amount to substract
 * @return the new instance with the input amount subtracted (never
 *         {@code null})
 */
public Euro minus(Euro amount) {
	Preconditions.checkNotNull(amount, "The amount must not be null.");

	return with(this.amount.subtract(amount.amount));
}

/**
 * Returns a copy of this Euro value with the amount subtracted.
 *
 * @param amount
 *            the amount to substract
 * @return the new instance with the input amount subtracted (never
 *         {@code null})
 */
public Euro minus(BigDecimal amount) {
	Preconditions.checkNotNull(amount, "The amount must not be null.");

	BigDecimal newAmount = this.amount.subtract(amount).setScale(
			currency.getDefaultFractionDigits(), roundingMode);

	return with(newAmount);
}

/**
 * Returns a copy of this Euro value with the amount subtracted.
 *
 * @param amount
 *            the amount to substract
 * @return the new instance with the input amount subtracted (never
 *         {@code null})
 */
public Euro minus(long amount) {
	return minus(BigDecimal.valueOf(amount));
}

/**
 * Returns a copy of this Euro value with the amount subtracted.
 *
 * @param amount
 *            the amount to substract
 * @return the new instance with the input amount subtracted (never
 *         {@code null})
```

```
     */
    public Euro minus(double amount) {
          return minus(BigDecimal.valueOf(amount));
    }

    /**
     * Returns a copy of this Euro value with the amount added.
     *
     * @param amount
     *            the amount to add
     * @return the new instance with the input amount added (never {@code null})
     */
    public Euro plus(Euro amount) {
          Preconditions.checkNotNull(amount, "The amount must not be null.");

          return with(this.amount.add(amount.amount));
    }

    /**
     * Returns a copy of this Euro value with the amount added.
     *
     * @param amount
     *            the amount to add
     * @return the new instance with the input amount added (never {@code null})
     */
    public Euro plus(BigDecimal amount) {
          BigDecimal newAmount = this.amount.add(amount).setScale(
                      currency.getDefaultFractionDigits(), roundingMode);

          return with(newAmount);
    }

    /**
     * Returns a copy of this Euro value with the amount added.
     *
     * @param amount
     *            the amount to add
     * @return the new instance with the input amount added (never {@code null})
     */
    public Euro plus(long amount) {
          return plus(BigDecimal.valueOf(amount));
    }

    /**
     * Returns a copy of this Euro value with the amount added.
     *
     * @param amount
     *            the amount to add
     * @return the new instance with the input amount added (never {@code null})
     */
    public Euro plus(double amount) {
          return plus(BigDecimal.valueOf(amount));
    }

    /**
     * Returns a copy of this Euro value divided by the divisor.
     *
     * @param divisor
     *            the divisor to divide by
     * @return the new divided instance (never {@code null})
```

```
         * @see #applyTenCentRounding()
         */
        public Euro divide(long divisor) {
                return divide(BigDecimal.valueOf(divisor));
        }

        /**
         * Returns a copy of this Euro value divided by the divisor.
         *
         * @param divisor
         *            the divisor to divide by
         * @return the new divided instance (never {@code null})
         * @see #applyTenCentRounding()
         */
        public Euro divide(double divisor) {
                return divide(BigDecimal.valueOf(divisor));
        }

        /**
         * Returns a copy of this Euro value divided by the divisor.
         *
         * @param divisor
         *            the divisor to divide by
         * @return the new divided instance (never {@code null})
         * @see #applyTenCentRounding()
         */
        public Euro divide(BigDecimal divisor) {
                Preconditions.checkNotNull(divisor, "The divisor must not be null.");

                if (roundToTenCent) {
                        // Round to one fractional digit
                        BigDecimal result = this.amount.divide(divisor, 1,
roundingMode);
                        // Return a result with two fractional digits
                        return
with(result.setScale(currency.getDefaultFractionDigits()));
                }

                return with(this.amount.divide(divisor, roundingMode));
        }

        /**
         * Returns a copy of this Euro value multiplied by the multiplicand.
         *
         * @param multiplicand
         *            the multiplicand to multiply by
         * @return the new multiplied instance (never {@code null})
         * @see #applyTenCentRounding()
         */
        public Euro multiply(long multiplicand) {
                return multiply(BigDecimal.valueOf(multiplicand));
        }

        /**
         * Returns a copy of this Euro value multiplied by the multiplicand.
         *
         * @param multiplicand
         *            the multiplicand to multiply by
         * @return the new multiplied instance (never {@code null})
         * @see #applyTenCentRounding()
```

```
     */
    public Euro multiply(double multiplicand) {
          return multiply(BigDecimal.valueOf(multiplicand));
    }

    /**
     * Returns a copy of this Euro value multiplied by the multiplicand.
     *
     * @param multiplicand
     *            the multiplicand to multiply by
     * @return the new multiplied instance (never {@code null})
     * @see #applyTenCentRounding()
     */
    public Euro multiply(BigDecimal multiplicand) {
          Preconditions.checkNotNull(multiplicand,
                      "The multiplicand must not be null.");

          BigDecimal result = this.amount.multiply(multiplicand);

          if (roundToTenCent) {
                // Round to one fractional digit
                result = result.setScale(1, roundingMode);
          }

          // Return a result with two fractional digits
          result = result.setScale(currency.getDefaultFractionDigits(),
                      roundingMode);
          return with(result);
    }

    /**
     * Switches internal rounding to ten-cent-mode.
     * <p/>
     * The result of the next division or multiplication will be rounded to ten
     * cent (one fractional digit). Afterwards the rounding mode will be
     * switched back to default rounding (two fractional digits),
     * <b>regardless</b> of the operation and if the result was actually rounded
     * or not. Technically the next operation will return a new instance with
     * default rounding mode.
     * <p/>
     * Rounding to ten cent is sometimes required due legal requirements, e.g.
     * as mandated by article 63 para. 3 KOVG.
     * <p/>
     * To preserve immutability a copy of the current instance will be returned
     * with rounding mode set to ten-cent-rounding.
     */
    public Euro applyTenCentRounding() {
          // To preserve immutability we have to return a new instance here
          return new Euro(this.amount, true);
    }

    /**
     * Returns a new {@code Euro}, returning {@code this} if possible.
     * <p/>
     * This instance is immutable and unaffected by this method.
     *
     * @param newInstance
     *            the new Euro to use, not null
     * @return the new instance, never null
     */
```

```
	private Euro with(BigDecimal newInstance) {
		// Only return this if default rounding mode and amount is equal
		if (!this.roundToTenCent && this.amount.equals(newInstance)) {
			return this;
		}
		return new Euro(newInstance);
	}

	/**
	 * Get the amount of Euro.
	 *
	 * @return the amount (never {@code null})
	 */
	public BigDecimal getAmount() {
		return this.amount;
	}

	/**
	 * Get the currency which is always "Euro".
	 *
	 * @return a currency representing Euro (never {@code null})
	 */
	public final Currency getCurrency() {
		return currency;
	}

	/**
	 * Round the current amount to ten cent, e.g. 12.61 will become 12.60, 12.68
	 * will become 12.70. This operation doesn't affect the original value as it
	 * is immutable.
	 *
	 * @return the amount rounded to ten cent (never {@code null})
	 */
	public Euro roundToTenCent() {
		BigDecimal roundedAmount = this.amount.setScale(1, roundingMode);

		return
with(roundedAmount.setScale(currency.getDefaultFractionDigits()));
	}

	/**
	 * Compares this amount in Euro to another.
	 *
	 * @return -1 if this is less than, 0 if equal, 1 if greater than the other
	 *         amount
	 */
	@Override
	public int compareTo(Euro amount) {
		return this.amount.compareTo(amount.amount);
	}

	/**
	 * Checks if two Euro amounts are equal.
	 *
	 * @param obj
	 *            the other Euro amount
	 * @return {@code true} if this is equal to the other amount, {@code false}
	 *         if not
	 */
	@Override
```

```
	public boolean equals(Object obj) {
		if (this == obj) {
			return true;
		}
		if (!(obj instanceof Euro)) {
			return false;
		}
		Euro other = (Euro) obj;

		// Although BigDecimal takes scale into consideration, this is no
		// problem, as scale of amount is always 2
		return this.amount.equals(other.amount);
	}

	/**
	 * Returns a hash code for this Euro amount.
	 *
	 * @return a hash code
	 */
	@Override
	public int hashCode() {
		return this.amount.hashCode() + 7;
	}

	/**
	 * Get the Euro value as string. The value is prefixed by the currency code
	 * (EUR).
	 *
	 * @return a string represenation of this Euro value
	 */
	@Override
	public String toString() {
		return new StringBuilder().append(currency.getCurrencyCode())
				.append(' ').append(amount.toPlainString()).toString();
	}
}
```

Anhang A.8. Geschlecht.java

```
package at.js.diss.simple.domain;

/**
 * Enum die das Geschlecht des Antragstellers repräsentiert.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public enum Geschlecht {
	MAENNLICH, WEIBLICH;
}
```

Anhang A.9. Grundrente.java

```
package at.js.diss.simple.domain;

import java.io.Serializable;

import at.js.diss.simple.service.support.Constants;

import com.google.common.base.Objects;
```

```java
/**
 * Klasse die eine Grundrente gemäß § 11 KOVG repräsentiert.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class Grundrente implements Serializable, Betrag {
	private static final long serialVersionUID = 1L;

	private Euro grundbetrag;
	private Euro altersabhaengigeErhoehung;

	public Euro getGrundbetrag() {
		return grundbetrag;
	}

	public void setGrundbetrag(Euro grundbetrag) {
		this.grundbetrag = grundbetrag;
	}

	public Euro getAltersabhaengigeErhoehung() {
		return altersabhaengigeErhoehung;
	}

	public void setAltersabhaengigeErhoehung(Euro altersabhaengigeErhoehung) {
		this.altersabhaengigeErhoehung = altersabhaengigeErhoehung;
	}

	/**
	 * Berechnet den Gesamtbetrag der Grundrente:
	 * <code>grundbetrag + altersabhaengigeErhoehung</code>.
	 *
	 * @return den Gesamtbetrag der Grundrente
	 */
	@Override
	public Euro getBetrag() {
		Euro betrag = Constants.ZERO;

		if (grundbetrag != null) {
			betrag = betrag.plus(grundbetrag);
		}
		if (altersabhaengigeErhoehung != null) {
			betrag = betrag.plus(altersabhaengigeErhoehung);
		}

		return betrag;
	}

	@Override
	public String toString() {
		return Objects.toStringHelper(this).add("grundbetrag", grundbetrag)
				.add("altersabhaengigeErhoehung",
altersabhaengigeErhoehung)
				.add("betrag", getBetrag()).toString();
	}
}
```

Anhang A.10. Kleiderverbrauch.java

```java
package at.js.diss.simple.domain;

/**
 * Enum die einen außergewöhnlichen Kleider- und Wäscheverbrauch gemäß § 20a
 * KOVG repräsentiert. Die einzelnen Instanzen der Enum entsprechen Zf. 1 bis 3.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public enum Kleiderverbrauch {
	/**
	 * § 20a Zf. 1.
	 */
	I,
	/**
	 * § 20a Zf. 2.
	 */
	II,
	/**
	 * § 20a Zf. 3.
	 */
	III;
}
```

Anhang A.11. Pflegestufe.java

```java
package at.js.diss.simple.domain;

/**
 * Enum die Pflegestufen gemäß § 18 Abs. 3 KOVG repräsentiert.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public enum Pflegestufe {
	/**
	 * Pflegestufe 1.
	 */
	I,
	/**
	 * Pflegestufe 2.
	 */
	II,
	/**
	 * Pflegestufe 3.
	 */
	III,
	/**
	 * Pflegestufe 4.
	 */
	IV,
	/**
	 * Pflegestufe 5.
	 */
	V,
	/**
	 * Fiktive Pflegestufe 5+ die der erhöhten Pflegestufe V gemäß § 18 Abs. 5
	 * entspricht.
	 */
	V_PLUS;
```

}

Anhang A.12. Zusatzrente.java

```
package at.js.diss.simple.domain;

import java.io.Serializable;
import java.util.List;

import at.js.diss.simple.service.support.Constants;

import com.google.common.base.Objects;

/**
 * Klasse die eine Zusatzrente gemäß § 12 KOVG repräsentiert.
 * <p/>
 * <b>Hinweis:</b> Obwohl im Gesetz die diversen Zulagen <b>kein</b> Teil der
 * Zusatzrente selbst sind sondern zu dieser gewährt werden, beinhaltet diese
 * Klasse auch die erwähnten Zulagen um diese gemeinsam mit der Zusatzrente zu
 * bündeln.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class Zusatzrente implements Serializable, Betrag {
	private static final long serialVersionUID = 1L;

	private Euro grundbetrag;
	// Keine Zulage sondern eine Erhöhung
	private Euro ausgleichszulage;

	// Zulagen
	// Familienzulagen
	private List<Euro> familienzulagen;
	// Diätzuschuss
	private Euro diaetzuschuss;

	public Euro getGrundbetrag() {
		return grundbetrag;
	}

	/**
	 * Grundbetrag der Zusatzrente gemäß § 12 Abs. 2 KOVG ohne Erhöhungen und
	 * Zulagen.
	 */
	public void setGrundbetrag(Euro grundbetrag) {
		this.grundbetrag = grundbetrag;
	}

	public Euro getAusgleichszulage() {
		return ausgleichszulage;
	}

	/**
	 * Ausgleichszulage gemäß § 12 Abs. 3 KOVG i.V.m. § 291 Abs. 1 lit. b) ASVG.
	 */
	public void setAusgleichszulage(Euro ausgleichszulage) {
		this.ausgleichszulage = ausgleichszulage;
	}
```

```
	public List<Euro> getFamilienzulagen() {
		return familienzulagen;
	}

	/**
	 * Familienzulagen zur Zusatzrente gemäß § 16 KOVG.
	 */
	public void setFamilienzulagen(List<Euro> familienzulagen) {
		this.familienzulagen = familienzulagen;
	}

	public Euro getDiaetzuschuss() {
		return diaetzuschuss;
	}

	/**
	 * Diätzuschuss zur Zusatzrente gemäß § 14 KOVG.
	 */
	public void setDiaetzuschuss(Euro diaetzuschuss) {
		this.diaetzuschuss = diaetzuschuss;
	}

	/**
	 * Berechnet den Gesamtbetrag der Zusatzrente:
	 * <code>Grundbetrag + Ausgleichszulage + Familienzulagen +
Diätzuschuss</code>.
	 *
	 * @return den Gesamtbetrag der Zusatzrente
	 */
	@Override
	public Euro getBetrag() {
		Euro betrag = Constants.ZERO;

		if (grundbetrag != null) {
			betrag = betrag.plus(grundbetrag);
		}
		if (ausgleichszulage != null) {
			betrag = betrag.plus(ausgleichszulage);
		}
		// Zulagen
		betrag = betrag.plus(getSummeZulagen());

		return betrag;
	}

	/**
	 * Berechnet die Summe des Grundbetrags § 12 Abs. 2 und der
	 * "Ausgleichszulage" gemäß Abs. 3.
	 *
	 * @return die Summe
	 */
	public Euro getSummeGrundbetragMitErhoehung() {
		Euro summe = grundbetrag;

		if (ausgleichszulage != null) {
			summe = summe.plus(ausgleichszulage);
		}

		return summe;
	}
```

```
	/**
	 * Summe der Familienzulagen.
	 *
	 * @return die Summe
	 */
	public Euro getSummeFamiliezulagen() {
		Euro summe = Constants.ZERO;

		if (familienzulagen == null) {
			return summe;
		}

		for (Euro familienzulage : familienzulagen) {
			summe = summe.plus(familienzulage);
		}

		return summe;
	}

	/**
	 * Summe des Diätzuschusses und der Familienzulage(n).
	 *
	 * @return die Summe
	 */
	public Euro getSummeZulagen() {
		Euro summe = Constants.ZERO;

		if (diaetzuschuss != null) {
			summe = summe.plus(diaetzuschuss);
		}
		summe = summe.plus(getSummeFamiliezulagen());

		return summe;
	}

	/**
	 * @return {@code true} wenn Familienzulage(n) oder ein Diätzuschuss gewährt
	 *         wird
	 */
	public boolean hasZulagen() {
		return familienzulagen != null || diaetzuschuss != null;
	}

	@Override
	public String toString() {
		return Objects.toStringHelper(this).add("grundbetrag", grundbetrag)
				.add("ausgleichszulage", ausgleichszulage)
				.add("familienzulagen", familienzulagen)
				.add("diaetzuschuss", diaetzuschuss).add("betrag",
getBetrag()).toString();
	}
}
```

Anhang A.13. Main.java

```
package at.js.diss.simple.main;

import static at.js.diss.simple.domain.Geschlecht.MAENNLICH;
import static at.js.diss.simple.support.PrintUtils.LINE_SEPARATOR;
```

```java
import static at.js.diss.simple.support.PrintUtils.print;
import static org.joda.time.DateTimeZone.UTC;

import org.joda.time.DateTime;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import at.js.diss.simple.domain.Beschaedigtenrente;
import at.js.diss.simple.domain.Beschaedigter;
import at.js.diss.simple.domain.Diaetverpflegung;
import at.js.diss.simple.domain.Euro;
import at.js.diss.simple.domain.Kleiderverbrauch;
import at.js.diss.simple.domain.Pflegestufe;
import at.js.diss.simple.service.CalculationEngine;

/**
 * Simple main program.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class Main {
	private final Logger logger = LoggerFactory.getLogger(getClass());
	private ApplicationContext ctx;
	private Beschaedigter beschaedigter;

	/**
	 * Initializes the Spring {@link ApplicationContext} and runs the example.
	 */
	public static void main(String[] args) {
		Main main = new Main();

		try {
			main.setup();
			main.runExample();
		} finally {
			main.shutdown();
		}
	}

	private void runExample() {
		CalculationEngine engine = ctx.getBean(CalculationEngine.class);

		DateTime now = new DateTime(2014, 2, 14, 0, 0, 0, UTC);

		Beschaedigtenrente rente = engine.calculate(beschaedigter, now);

		StringBuilder buffer = new StringBuilder(LINE_SEPARATOR);
		// Antragsteller
		buffer.append("Daten des Antragstellers:").append(LINE_SEPARATOR)
				.append(print(beschaedigter));
		// Rente
		buffer.append(LINE_SEPARATOR);
		buffer.append(LINE_SEPARATOR).append(print(rente));

		logger.info(buffer.toString());
	}
```

```
	private void setup() {
		this.ctx = new ClassPathXmlApplicationContext("/spring/*");

		Beschaedigter.Builder builder = new Beschaedigter.Builder();
		// Builder kann als eine Domain Specific Language zur Erstellung von
Objekten vom Typ Beschaedigter verstanden werden
		builder.name("Max Mustermann").geschlecht(MAENNLICH)
				.geburtsdatum(new DateTime(1936, 7, 26, 0, 0, 0, UTC))
				.dienstbeschaedigung(60).dienstbeschaedigung(80)
				.angehoerige(2)
				.pflegestufe(Pflegestufe.IV)
				.kleiderverbrauch(Kleiderverbrauch.I)
				.kleiderverbrauch(Kleiderverbrauch.III)
				.diaetverpflegung(Diaetverpflegung.II)
				.einkommen(Euro.of("550"));
		this.beschaedigter = builder.build();
	}

	private void shutdown() {
		if (ctx instanceof AbstractApplicationContext) {
			((AbstractApplicationContext) ctx).close();
		}
	}
}
```

Anhang A.14. DataRepository.java

```
package at.js.diss.simple.repository;

import at.js.diss.simple.domain.Diaetverpflegung;
import at.js.diss.simple.domain.Euro;
import at.js.diss.simple.domain.Kleiderverbrauch;
import at.js.diss.simple.domain.Pflegestufe;

/**
 * Zentrales Interface zum Lesen von Werten.
 * <p/>
 * Sämtliche Methoden werfen eine {@code NullPointerException} wenn ein
 * übergebener Parameter {@code null} ist.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public interface DataRepository {
	/**
	 * Liefert den Grundbetrag der Grundrente gemäß § 11 Abs. 1 KOVG zur
	 * übergebenen MdE.
	 *
	 * @param mde
	 *            die Minderung der Erwerbsfähigkeit
	 * @return den Betrag in Euro für die übergebene MdE
	 * @throws IllegalArgumentException
	 *             wenn zur übergebenen MdE kein Betrag gefunden werden kann
	 */
	public Euro getGrundbetragGrundrente(int mde);

	/**
	 * Liefert die vom Alter und der MdE abhängige Erhöhung der Grundrente gemäß
	 * § 11 Abs. 3 KOVG.
```

```
     *
     * @param alter
     *            das Alter des Beschädigten
     * @param mde
     *            die MdE des Beschädigten
     * @return den Betrag in Euro für das übergebene Alter und die MdE
     * @throws IllegalArgumentException
     *             wenn zu den übergebenen Daten kein Betrag gefunden werden
     *             kann
     */
    public Euro getErhoehungGrundrente(int alter, int mde);

    /**
     * Liefert die Schwerstbeschädigtenzulage zur übergebenen Summe der MdE
     * gemäß § 11a Abs. 4 KOVG.
     *
     * @param summeMdE
     *            die Summe der MdE der einzelnen Dienstbeschädigungen
     * @return den Betrag in Euro zur übergebenen Summe der MdE
     * @throws IllegalArgumentException
     *             wenn zu den übergebenen Daten kein Betrag gefunden werden
     *             kann
     */
    public Euro getSchwerstbeschaedigtenzulage(int summeMdE);

    /**
     * Liefert die Pflegezulage zur übergebene {@link Pflegestufe} gemäß § 18
     * Abs. 4 KOVG.
     *
     * @param pflegestufe
     *            die Pflegestufe
     * @return die Pflegezulage in Euro
     * @throws IllegalArgumentException
     *             wenn zu den übergebenen Daten kein Betrag gefunden werden
     *             kann
     */
    public Euro getPflegezulage(Pflegestufe pflegestufe);

    /**
     * Liefert die Kleider- und Wäschepauschale zum übergebenen
     * {@link Kleiderverbrauch} gemäß § 20a Abs. 1 KOVG.
     *
     * @param kleiderverbrauch
     *            der Kleiderverbrauch
     * @return die Kleider- und Wäschepauschale in Euro
     * @throws IllegalArgumentException
     *             wenn zu den übergebenen Daten kein Betrag gefunden werden
     *             kann
     */
    public Euro getKleiderpauschale(Kleiderverbrauch kleiderverbrauch);

    /**
     * Liefert den Diätzuschuss für die übergebene {@link Diaetverpflegung}
     * gemäß § 14 Abs. 1 KOVG.
     *
     * @param diaetverpflegung
     *            die Diätverpflegung
     * @return der Diätzuschuss in Euro
     * @throws IllegalArgumentException
     *             wenn zu den übergebenen Daten kein Betrag gefunden werden
```

```
         *             kann
         */
        public Euro getDiaetzuschuss(Diaetverpflegung diaetverpflegung);

        /**
         * Liefert die Blindenführzulge gemäß § 20 KOVG. Diese ist ein fixer Betrag
         * der unabhängig von anderen Daten ist.
         *
         * @return die Blindenführzulage
         */
        public Euro getBlindenfuehrzulage();
}
```

Anhang A.15. InMemoryDataRepository.java

```
package at.js.diss.simple.repository;

import static at.js.diss.simple.domain.Euro.of;
import static at.js.diss.simple.support.CalculationUtils.percentageOf;

import java.util.Collections;
import java.util.EnumMap;
import java.util.HashMap;
import java.util.Map;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Repository;

import at.js.diss.simple.domain.Diaetverpflegung;
import at.js.diss.simple.domain.Euro;
import at.js.diss.simple.domain.Kleiderverbrauch;
import at.js.diss.simple.domain.Pflegestufe;

import com.google.common.base.Preconditions;
import com.google.common.collect.ImmutableTable;
import com.google.common.collect.Table;

/**
 * Eine einfache Implementierung von {@link DataRepository} die sämtliche Daten
 * im Speicher hält.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
@Repository
public class InMemoryDataRepository implements DataRepository {
        private final Logger logger = LoggerFactory.getLogger(getClass());

        private static final Euro MAX_GRUNDRENTE = of("511.00");
        private static final Euro BLINDENFUEHRZULAGE = of("149.90");

        private final Map<Integer, Euro> mdeToGrundrente =
createMdEToGrundrenteGrundbetragMap();
        private final Table<Integer, Integer, Euro> erhoehungGrundrente =
createErhoehungGrundrenteTable();
        private final Map<Integer, Euro> mdeToSchwerstbeschaedigtenzulage =
createMdEToSchwerstbeschaedigtenzulageMap();
        private final Map<Diaetverpflegung, Euro> diaetverpflegungToDiaetzuschuss =
createDiaetverpflegungToDiaetzuschussMap();
```

```
	private final Map<Pflegestufe, Euro> pflegestufeToPflegezulage =
createPflegestufeToPflegezulageMap();
	private final Map<Kleiderverbrauch, Euro> kleiderverbrauchToKleiderpauschale
= createKleiderverbrauchToKleiderpauschaleMap();

	/**
	 * Konstruktor. Gibt Informationen aus.
	 */
	public InMemoryDataRepository() {
		logger.debug("Grundbetrag Grundrente: {}", this.mdeToGrundrente);
		logger.debug("Erhöhung Grundrente: {}", this.erhoehungGrundrente);
		logger.debug("Schwerstbeschädigtenzulage: {}",
				this.mdeToSchwerstbeschaedigtenzulage);
		logger.debug("Diätzuschuss: {}",
this.diaetverpflegungToDiaetzuschuss);
		logger.debug("Pflegezulage: {}", this.pflegestufeToPflegezulage);
		logger.debug("Kleider- und Wäschepauschale: {}",
				this.kleiderverbrauchToKleiderpauschale);
	}

	@Override
	public Euro getGrundbetragGrundrente(int mde) {
		Euro betrag = mdeToGrundrente.get(mde);

		if (betrag == null) {
			throw new IllegalArgumentException("Konnte für MdE [" + mde
					+ "] keine Grundrente ermitteln.");
		}

		return betrag;
	}

	@Override
	public Euro getErhoehungGrundrente(int alter, int mde) {
		Euro betrag = erhoehungGrundrente.get(alter, mde);

		if (betrag == null) {
			throw new IllegalArgumentException("Konnte für Alter [" + alter
					+ "] und MdE [" + mde
					+ "] keine Erhöhung der Grundrente ermitteln.");
		}

		return betrag;
	}

	@Override
	public Euro getSchwerstbeschaedigtenzulage(int summeMdE) {
		Euro zulage = mdeToSchwerstbeschaedigtenzulage.get(summeMdE);

		if (zulage == null) {
			throw new IllegalArgumentException("Konnte für Summe MdE ["
					+ summeMdE
					+ "] keine Schwerstbeschädigtenzulage
ermitteln.");
		}

		return zulage;
	}

	@Override
```

```
	public Euro getPflegezulage(Pflegestufe pflegestufe) {
		Preconditions.checkNotNull(pflegestufe,
				"Die pflegestufe darf nicht null sein.");

		Euro zulage = pflegestufeToPflegezulage.get(pflegestufe);

		if (zulage == null) {
			throw new IllegalArgumentException("Konnte für Pflegestufe ["
					+ pflegestufe + "] keine Pflegezulage
ermitteln.");
		}

		return zulage;
	}

	@Override
	public Euro getKleiderpauschale(Kleiderverbrauch kleiderverbrauch) {
		Preconditions.checkNotNull(kleiderverbrauch,
				"Die kleiderverbrauch darf nicht null sein.");

		Euro pauschale = kleiderverbrauchToKleiderpauschale
				.get(kleiderverbrauch);

		if (pauschale == null) {
			throw new IllegalArgumentException("Konnte für Kleiderverbrauch
["
					+ kleiderverbrauch + "] keine Pauschale
ermitteln.");
		}

		return pauschale;
	}

	@Override
	public Euro getDiaetzuschuss(Diaetverpflegung diaetverpflegung) {
		Preconditions.checkNotNull(diaetverpflegung,
				"Die diaetverpflegung darf nicht null sein.");

		Euro zuschuss =
diaetverpflegungToDiaetzuschuss.get(diaetverpflegung);

		if (zuschuss == null) {
			throw new IllegalArgumentException("Konnte für Diätverpflegung
["
					+ diaetverpflegung + "] keinen Zuschuss
ermitteln.");
		}

		return zuschuss;
	}

	@Override
	public Euro getBlindenfuehrzulage() {
		return BLINDENFUEHRZULAGE;
	}

	/**
	 * Erstellt eine Map mit dem Schlüssel: MdE und dem Wert: Euro.
	 *
	 * @return eine Map die die Daten gemäß § 11 Abs. 1 KOVG enthält
```

```
	 */
	private Map<Integer, Euro> createMdEToGrundrenteGrundbetragMap() {
		Map<Integer, Euro> mdeToGrundrente = new HashMap<>();
		mdeToGrundrente.put(20, percentageOf(MAX_GRUNDRENTE, 10));
		mdeToGrundrente.put(30, percentageOf(MAX_GRUNDRENTE, 20));
		mdeToGrundrente.put(40, percentageOf(MAX_GRUNDRENTE, 30));
		mdeToGrundrente.put(50, percentageOf(MAX_GRUNDRENTE, 40));
		mdeToGrundrente.put(60, percentageOf(MAX_GRUNDRENTE, 50));
		mdeToGrundrente.put(70, percentageOf(MAX_GRUNDRENTE, 60));
		mdeToGrundrente.put(80, percentageOf(MAX_GRUNDRENTE, 80));
		mdeToGrundrente.put(90, MAX_GRUNDRENTE);
		mdeToGrundrente.put(100, MAX_GRUNDRENTE);

		return Collections.unmodifiableMap(mdeToGrundrente);
	}

	/**
	 * Erstellt eine Tabelle mit folgender Struktur Zeile: Alter, Spalte: MdE,
	 * Wert: Euro.
	 * <p/>
	 * Inhalt gemäß § 11 Abs. 3 KOVG.
	 *
	 * @return eine Tabelle mit den Werten gemäß § 11 Abs. 3 KOVG
	 */
	private Table<Integer, Integer, Euro> createErhoehungGrundrenteTable() {
		return new ImmutableTable.Builder<Integer, Integer, Euro>()
				// 1. row
				.put(65, 50, of("22.90"))
				.put(65, 60, of("38.20"))
				.put(65, 70, of("46.30"))
				.put(65, 80, of("61.30"))
				.put(65, 90, of("76.60"))
				.put(65, 100, of("76.60"))
				// 2. row
				.put(70, 50, of("46.40"))
				.put(70, 60, of("76.40"))
				.put(70, 70, of("86.80"))
				.put(70, 80, of("102.30"))
				.put(70, 90, of("122.50"))
				// 3. row
				.put(75, 50, of("84.70")).put(75, 60, of("114.90"))
				.put(75, 70, of("128.00")).put(75, 80, of("143.00"))
				.put(75, 90, of("158.50"))
				.put(75, 100, of("158.50"))
				// 4. row
				.put(80, 50, of("122.50")).put(80, 60, of("153.50"))
				.put(80, 70, of("168.70")).put(80, 80, of("184.00"))
				.put(80, 90, of("199.40")).put(80, 100,
of("199.40")).build();
	}

	/**
	 * Erstellt eine Map MdE : Schwerstbeschädigtenzulage gemäß § 11a Abs. 4
	 * KOVG.
	 */
	private Map<Integer, Euro> createMdEToSchwerstbeschaedigtenzulageMap() {
		// Grundbetrag ist Grundrente für Erwerbsunfähige gemäß § 11 Abs. 1
		Euro grundrente = MAX_GRUNDRENTE;

		Map<Integer, Euro> mdeToZulage = new HashMap<>();
```

XXX

```
            mdeToZulage.put(130, percentageOf(grundrente, 30));
            mdeToZulage.put(160, percentageOf(grundrente, 40));
            mdeToZulage.put(190, percentageOf(grundrente, 50));
            mdeToZulage.put(220, percentageOf(grundrente, 60));
            mdeToZulage.put(250, percentageOf(grundrente, 70));
            mdeToZulage.put(280, percentageOf(grundrente, 80));

            return Collections.unmodifiableMap(mdeToZulage);
      }

      /**
       * Erstellt eine Map {@link Diaetverpflegung} : Diätzuschuss gemäß § 14 Abs.
       * 1 KOVG.
       */
      private Map<Diaetverpflegung, Euro>
createDiaetverpflegungToDiaetzuschussMap() {
            Map<Diaetverpflegung, Euro> zuschuss = new EnumMap<>(
                        Diaetverpflegung.class);

            zuschuss.put(Diaetverpflegung.I, of("31.90"));
            zuschuss.put(Diaetverpflegung.II, of("63.90"));
            zuschuss.put(Diaetverpflegung.III, of("96.00"));
            zuschuss.put(Diaetverpflegung.IV, of("31.90"));
            zuschuss.put(Diaetverpflegung.V, of("96.00"));

            return Collections.unmodifiableMap(zuschuss);
      }

      /**
       * Erstellt eine Map {@link Pflegestufe} : Pflegezulage gemäß § 18 Abs. 3
       * und 4 KOVG.
       */
      private Map<Pflegestufe, Euro> createPflegestufeToPflegezulageMap() {
            Map<Pflegestufe, Euro> pflegestufen = new
EnumMap<>(Pflegestufe.class);

            pflegestufen.put(Pflegestufe.I, of("671.60"));
            pflegestufen.put(Pflegestufe.II, of("1006.80"));
            pflegestufen.put(Pflegestufe.III, of("1342.90"));
            pflegestufen.put(Pflegestufe.IV, of("1678.90"));
            pflegestufen.put(Pflegestufe.V, of("2014.00"));
            // Fiktive Pflegestufe 5+ = V erhöht um 1/3
            pflegestufen.put(Pflegestufe.V_PLUS, of("2014.00").multiply(1.33));

            return Collections.unmodifiableMap(pflegestufen);
      }

      /**
       * Erstellt eine Map {@link Kleiderverbrauch} : Kleider- Wäschepauschale
       * gemäß § 20a Abs.1 KOVG.
       */
      private Map<Kleiderverbrauch, Euro>
createKleiderverbrauchToKleiderpauschaleMap() {
            Map<Kleiderverbrauch, Euro> pauschale = new EnumMap<>(
                        Kleiderverbrauch.class);

            pauschale.put(Kleiderverbrauch.I, of("22.70"));
            pauschale.put(Kleiderverbrauch.II, of("36.00"));
            pauschale.put(Kleiderverbrauch.III, of("60.20"));
```

```
            return Collections.unmodifiableMap(pauschale);
        }
}
```

Anhang A.16. CalculationEngine.java

```
package at.js.diss.simple.service;

import org.joda.time.DateTime;

import at.js.diss.simple.domain.Beschaedigtenrente;
import at.js.diss.simple.domain.Beschaedigter;

/**
 * Interface das den Kontrakt für eine Klasse zur Berechnung der
 * Beschädigtenrente definiert.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public interface CalculationEngine {
        /**
         * Berechnet die Beschädigtenrente.
         *
         * @param beschaedigter
         *            die Daten des Beschädigten
         * @param now
         *            das Datum zu dem die Berechnung durchgeführt werden soll
         * @return die Beschädigtenrente ({@code null} wenn kein Anspruch besteht)
         */
        public Beschaedigtenrente calculate(Beschaedigter beschaedigter,
                        DateTime now);
}
```

Anhang A.17. JavaCalculationEngine.java

```
package at.js.diss.simple.service;

import static at.js.diss.simple.domain.Euro.of;
import static at.js.diss.simple.domain.Geschlecht.MAENNLICH;
import static at.js.diss.simple.domain.Geschlecht.WEIBLICH;
import static at.js.diss.simple.service.support.Constants.ASVG_PENSIONSRICHTSATZ;
import static
at.js.diss.simple.service.support.Constants.ERHOEHUNG_EINKOMMENSGRENZE;
import static at.js.diss.simple.service.support.Constants.FAMILIENZULAGE;
import static at.js.diss.simple.service.support.Constants.MAX_ZUSATZRENTE;
import static at.js.diss.simple.service.support.Constants.ZERO;
import static at.js.diss.simple.support.CalculationUtils.getAge;
import static at.js.diss.simple.support.CalculationUtils.getAlterVormonat;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.joda.time.DateTime;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.util.CollectionUtils;
```

```
import at.js.diss.simple.domain.Beschaedigtenrente;
import at.js.diss.simple.domain.Beschaedigter;
import at.js.diss.simple.domain.Blindheit;
import at.js.diss.simple.domain.Diaetverpflegung;
import at.js.diss.simple.domain.Euro;
import at.js.diss.simple.domain.Grundrente;
import at.js.diss.simple.domain.Kleiderverbrauch;
import at.js.diss.simple.domain.Pflegestufe;
import at.js.diss.simple.domain.Zusatzrente;
import at.js.diss.simple.repository.DataRepository;
import at.js.diss.simple.service.support.Constants;
import at.js.diss.simple.support.CalculationUtils;

import com.google.common.base.Preconditions;

/**
 * Einfache Implementierung einer {@link CalculationEngine}. Sämtliche Werte
 * sind im Code fest codiert enthalten.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
@Service
public class JavaCalculationEngine implements CalculationEngine {
	private final Logger logger = LoggerFactory.getLogger(getClass());

	private DataRepository dataRepository;

	/**
	 * Berechnet die Beschädigtenrente gemäß §§ 11 ff. KOVG.
	 *
	 * @param beschaedigter
	 *            die Daten des Beschädigten
	 * @param now
	 *            das Datum an dem die Berechnung durchgeführt werden soll
	 * @return die Beschädigtenrente
	 */
	@Override
	public final Beschaedigtenrente calculate(Beschaedigter beschaedigter,
			DateTime now) {
		Preconditions.checkNotNull(beschaedigter,
				"The beschaedigter must not be null.");
		Preconditions.checkNotNull(now, "The now must not be null.");

		logger.debug(
				"Berechne Beschädigtenrente {}-{} für Antragsteller: {},
geboren am {}, {} Jahre...",
				new Object[] {
						now.getYear(),
						now.getMonthOfYear(),
						beschaedigter.getGeschlecht(),

	beschaedigter.getGeburtsdatum().toLocalDate(),

	getAge(beschaedigter.getGeburtsdatum().toLocalDate(),
							now.toLocalDate()) });

		long start = System.currentTimeMillis();
		Beschaedigtenrente rente = new Beschaedigtenrente();
```

```
            rente.setDatum(now.toLocalDate().withDayOfMonth(1));

            // 1 Grundrente
            Grundrente grundrente = calculateGrundrente(beschaedigter, now);
            rente.setGrundrente(grundrente);
            logger.info("§ 11: Grundrente: {}", grundrente);

            if (grundrente == null) {
                  logger.info("Kein Anspruch auf Grundrente und somit auf
Beschädigtenrente.");
                  return null;
            }

            // 2 Zusatzrente
            Zusatzrente zusatzrente = calculateZusatzrente(beschaedigter, now);
            rente.setZusatzrente(zusatzrente);
            logger.info("Zusatzrente: {}", zusatzrente);

            // 3 Zulagen zur Beschädigtenrente
            // 3.1 Schwerstbeschädigtenzulage
            Euro schwerstbeschaedigtenzulage =
calculateSchwerstbeschaedigtenzulage(beschaedigter);
            rente.setSchwerstbeschaedigtenzulage(schwerstbeschaedigtenzulage);
            logger.info("§ 11a: Schwerstbeschädigtenzulage: {}",
                        schwerstbeschaedigtenzulage);

            // 3.2 Pflegezulage
            Euro pflegezulage = calculatePflegezulage(beschaedigter);
            rente.setPflegezulage(pflegezulage);
            logger.info("§ 18: Pflegezulage: {}", pflegezulage);

            // 3.3 Blindenzulage
            Euro blindenzulage = calculateBlindenzulage(beschaedigter);
            rente.setBlindenzulage(blindenzulage);
            logger.info("§ 19: Blindenzulage: {}", blindenzulage);

            // 3.4 Blindenführzulage
            Euro blindenfuehrzulage = calculateBlindenfuehrzulage(beschaedigter);
            rente.setBlindenfuehrzulage(blindenfuehrzulage);
            logger.info("§ 20: Blindenführzulage: {}", blindenfuehrzulage);

            // 3.5 Kleider- und Wäschepauschale
            List<Euro> kleiderpauschalen =
calculateKleiderpauschalen(beschaedigter);
            rente.setKleiderpauschalen(kleiderpauschalen);
            logger.info("§ 20a: Kleider- und Wäschepauschale(n): {}",
                        kleiderpauschalen);

            // Finale Rente
            logger.debug("Beschädigtenrente in {} ms berechnet.",
                        System.currentTimeMillis() - start);
            logger.info("Beschädigtenrente: {}", rente);

            return rente;
      }

      /**
       * Berechnet die Grundrente gemäß § 11 KOVG.
       *
       * @param beschaedigter
```

```
         *            die Daten des Beschädigten
         * @param now
         *            das Datum zu dem die Berechnung durchgeführt werden soll
         * @return die Grundrente ({@code null} wenn kein Anspruch besteht)
         */
        protected Grundrente calculateGrundrente(Beschaedigter beschaedigter,
                    DateTime now) {
              logger.debug("§ 11: Berechne Grundrente...");

              int mde = beschaedigter.getMaxMdE();

              if (mde < 20) {
                    // Kein Anspruch
                    logger.debug("§ 11: MdE {} < 20 - kein Anspruch auf
Grundrente.",
                                mde);

                    return null;
              }

              Grundrente rente = new Grundrente();

              // 1 Ermittlung Grundbetrag gemäß § 11 Abs. 1 KOVG
              Euro grundbetrag = dataRepository.getGrundbetragGrundrente(mde);
              rente.setGrundbetrag(grundbetrag);
              logger.debug("§ 11 Abs. 1: Grundbetrag der Grundrente für MdE {}:
{}",
                          mde, grundbetrag);

              // Erhöhungen nur für Schwerbeschädigte
              // Prüfung wird für Abs. 2 und 3 an dieser Stelle vorweggenommen
              if (mde < 50) {
                    logger.debug("Keine Erhöhung gemäß Abs. 2 und 3 da MdE {} <
50.", mde);

                    return rente;
              }

              int alter = getAlterVormonat(beschaedigter.getGeburtsdatum()
                          .toLocalDate(), now.toLocalDate());
              logger.debug("Alter im Vormonat: {}", alter);

              Euro erhoehung = ZERO;

              // 2 Altersabhängige Erhöhung, § 11 Abs. 2 KOVG
              if ((beschaedigter.getGeschlecht() == MAENNLICH && alter >= 60)
                          || (beschaedigter.getGeschlecht() == WEIBLICH && alter
>= 55)) {
                    erhoehung = of("20.90");

                    logger.debug(
                                "§ 11 Abs. 2: Erhöhung für Geschlecht: {}, Alter:
{}, Betrag: {}",
                                beschaedigter.getGeschlecht(), alter, erhoehung);
              }

              // 3 Altersabhängige Erhöhung, § 11 Abs. 3 KOVG
              if (alter >= 65) {
                    erhoehung = dataRepository.getErhoehungGrundrente(65, mde);
```

```
                logger.debug("§ 11 Abs. 3: Alter >= 65: {}", erhoehung);
            }
            if (alter >= 70) {
                erhoehung = dataRepository.getErhoehungGrundrente(70, mde);

                logger.debug("§ 11 Abs. 3: Alter >= 70: {}", erhoehung);
            }
            if (alter >= 75) {
                erhoehung = dataRepository.getErhoehungGrundrente(75, mde);

                logger.debug("§ 11 Abs. 3: Alter >= 75: {}", erhoehung);
            }
            if (alter >= 80) {
                erhoehung = dataRepository.getErhoehungGrundrente(80, mde);

                logger.debug("§ 11 Abs. 3: Alter >= 80: {}", erhoehung);
            }

            rente.setAltersabhaengigeErhoehung(erhoehung);
            logger.debug("§ 11 Abs. 2/3: Altersabhängige Erhöhung Grundrente:
{}",
                    erhoehung);

            return rente;
        }

        /**
         * Berechnet die Zusatzrente gemäß § 12 und die allfällige Zulagen gemäß §
         * 14 und § 16.
         *
         * @param beschaedigter
         *            die Daten des Beschädigten
         * @param now
         *            das Datum zu dem die Berechnung durchgeführt werden soll
         * @return die Zusatzrente ({@code null} wenn kein Anspruch besteht)
         */
        protected Zusatzrente calculateZusatzrente(Beschaedigter beschaedigter,
                DateTime now) {
            logger.debug("Berechne Zusatzrente...");

            if (beschaedigter.getMaxMdE() < 50) {
                logger.debug("§ 12: MdE {} < 50 - kein Anspruch auf
Zusatzrente.",
                        beschaedigter.getMaxMdE());
                return null;
            }

            int alter = getAlterVormonat(beschaedigter.getGeburtsdatum()
                    .toLocalDate(), now.toLocalDate());

            if (alter < 18) {
                logger.debug(
                        "§ 12: Alter {} < 18 - kein Anspruch auf
Zusatzrente.",
                        alter);
                return null;
            }

            // 1 Zusatzrente samt allfälligen Erhöhungen gemäß § 12
            Zusatzrente zusatzrente = doCalculateZusatzrente(beschaedigter, now);
```

```
            logger.debug("§ 12: Zusatzrente: {}", zusatzrente);

            // 2 Diätzuschuss
            Euro diaetzuschuss = calculateDiaetzuschuss(beschaedigter);
            zusatzrente.setDiaetzuschuss(diaetzuschuss);
            logger.info("§ 14: Diätzuschuss: {}", diaetzuschuss);

            // 3 Familienzulage
            List<Euro> familienzulagen = calculateFamilienzulagen(beschaedigter,
                        zusatzrente);
            zusatzrente.setFamilienzulagen(familienzulagen);
            logger.info("§ 16: Familienzulagen: {}", familienzulagen);

            return zusatzrente;
      }

      /**
       * Berechnet die Zusatzrente gemäß § 12 KOVG inklusive Erhöhung durch
       * Familienzulagen (Abs. 2) und Ausgleichszulage (Abs. 3).
       *
       * @param beschaedigter
       *            die Daten des Beschädigten
       * @param now
       *            das Datum zu dem die Berechnung durchgeführt werden soll
       * @return die Zusatzrente
       */
      private Zusatzrente doCalculateZusatzrente(Beschaedigter beschaedigter,
                  DateTime now) {
            logger.debug("§ 12: Berechne Zusatzrente (samt Erhöhungen)...");

            Zusatzrente rente = new Zusatzrente();

            // 1. Berechnung Einkommensgrenze ohne Familienzulagen
            Euro einkommensgrenze = calculateStandardEinkommensgrenze();
            logger.debug("§ 12 Abs. 2: Einkommensgrenze ohne Familienzulagen:
{}",
                        einkommensgrenze);

            // 2. Erhöhung Einkommensgrenze aufgrund von Familienzulagen
            int angehoerige = beschaedigter.getAngehoerige();
            if (angehoerige > 0) {
                  Euro erhoehungEinkommensgrenze = ERHOEHUNG_EINKOMMENSGRENZE
                              .multiply(angehoerige);
                  logger.debug("§ 12 Abs. 2: {} * {} = Erhöhung {}",
                              ERHOEHUNG_EINKOMMENSGRENZE, angehoerige,
                              erhoehungEinkommensgrenze);

                  Euro egAlt = einkommensgrenze;
                  einkommensgrenze =
einkommensgrenze.plus(erhoehungEinkommensgrenze);
                  logger.debug("§ 12 Abs. 2: {} + Erhöhung {} = Einkommensgrenze
{}",
                              egAlt, erhoehungEinkommensgrenze,
einkommensgrenze);
            }

            // 3. Berechnung der Zusatzrente
            Euro zusatzrente = calculateGrundbetragZusatzrente(einkommensgrenze,
                        beschaedigter.getEinkommen());
            rente.setGrundbetrag(zusatzrente);
```

```
            logger.debug("§ 12 Abs. 2: Grundbetrag Zusatzrente: {}",
zusatzrente);

            // 4. Ausgleichszulage
            Euro ausgleichszulage = calculateAusgleichszulage(beschaedigter
                        .getEinkommen());
            rente.setAusgleichszulage(ausgleichszulage);

            logger.debug("§ 12 Abs. 3: Ausgleichszulage zur Zusatzrente: {}",
                        ausgleichszulage);

            return rente;
      }

      /**
       * Berechnet den Grundbetrag Zusatzrente gemäß § 12 Abs. 2 KOVG in
       * Abhängigkeit zur übergebenen Einkommensgrenze.
       * <p/>
       * <code>Zusatzrente = min(266,9, max(Einkommensgrenze - Einkommen,
0))</code>
       *
       * @param einkommensgrenze
       *            die Einkommensgrenze für die Berechnung
       * @param einkommen
       *            das Einkommen des Beschädigten
       */
      private Euro calculateGrundbetragZusatzrente(Euro einkommensgrenze,
                  Euro einkommen) {
            Euro zusatzrente = einkommensgrenze.minus(einkommen);

            // Wenn Zusatzrente negativ wird sie auf € 0.00 gesetzt
            zusatzrente = CalculationUtils.max(ZERO, zusatzrente);
            logger.debug(
                        "§ 12 Abs. 2: max(0, Einkommensgrenze {} - Einkommen {})
= {}",
                        einkommensgrenze, einkommen, zusatzrente);

            // Zusatzrente ist mit Höchstbetrag begrenzt
            Euro maxZusatzrente = CalculationUtils
                        .min(zusatzrente, MAX_ZUSATZRENTE);
            logger.debug("§ 12 Abs. 2: Grundbetrag Zusatzrente: min({}, {}) =
{}",
                        zusatzrente, MAX_ZUSATZRENTE, maxZusatzrente);

            return maxZusatzrente;
      }

      /**
       * Berechnet die Ausgleichszulage zur Zusatzrente gemäß § 12 Abs. 3 KOVG.
       */
      private Euro calculateAusgleichszulage(Euro einkommen) {
            Euro ausgleichszulage = ASVG_PENSIONSRICHTSATZ.minus(einkommen);

            // Wenn Ausgleichszulage negativ wird sie auf € 0.00 gesetzt
            ausgleichszulage = CalculationUtils.max(ZERO, ausgleichszulage);

            logger.debug(
                        "§ 12 Abs. 3: ASVG Richtsatz {} - Einkommen {} =
Ausgleichszulage {}",
```

```
                        ASVG_PENSIONSRICHTSATZ, einkommen, ausgleichszulage);

                return ausgleichszulage;
        }

        /**
         * Berechnet den Diätzuschuss zur Zusatzrente gemäß § 14 KOVG.
         */
        protected Euro calculateDiaetzuschuss(Beschaedigter beschaedigter) {
                logger.debug("§ 14: Berechne Diätzuschuss...");

                Diaetverpflegung verpflegung = beschaedigter.getDiaetverpflegung();

                if (verpflegung == null) {
                        logger.debug("§ 14: Keine Diätverpflegung - kein Anspruch auf
Diätzuschuss.");

                        return null;
                }

                Euro zuschuss = dataRepository.getDiaetzuschuss(verpflegung);
                logger.debug("Diätverpflegung {}: {}", verpflegung, zuschuss);

                return zuschuss;
        }

        /**
         * Berechnet die Familienzulagen zur Zusatzrente gemäß § 16 KOVG.
         *
         * @return eine Liste die die Familienzulagen enthält ({@code null} wenn
         *         kein Anspruch besteht)
         */
        protected List<Euro> calculateFamilienzulagen(Beschaedigter beschaedigter,
                        Zusatzrente zusatzrente) {
                logger.debug("§ 16: Berechne Familienzulagen...");

                int mde = beschaedigter.getMaxMdE();
                if (mde < 50) {
                        logger.debug("§ 12: MdE {} < 50 - kein Anspruch auf
Zusatzrente.",
                                        mde);
                        return null;
                }

                int angehoerige = beschaedigter.getAngehoerige();
                if (angehoerige < 1) {
                        logger.debug(
                                        "§ 16: Anzahl der Angehörigen {} < 1 - kein
Anspruch auf Familienzulagen.",
                                        angehoerige);
                        return null;
                }

                List<Euro> familienzulagen = new ArrayList<>(angehoerige);

                // 1. Berechnung Familienzulagen ohne Kürzung
                for (int i = 0; i < angehoerige; i++) {
                        familienzulagen.add(FAMILIENZULAGE);
                }
```

```
            logger.debug("§ 16: Familienzulagen ohne Kürzung: {}",
familienzulagen);

            // 2. Berechnung der Erhöhung der Zusatzrente
            // 2.1 Berechnung Zusatzrente ohne Erhöhung der Einkommensgrenze
durch
            // Familienzulagen
            Euro einkommensgrenze = calculateStandardEinkommensgrenze();
            Euro einkommen = beschaedigter.getEinkommen();
            Euro zrOhneErhoehung = calculateGrundbetragZusatzrente(
                        einkommensgrenze, einkommen);

            logger.debug(
                        "§ 16: Zusatzrente ohne Erhöhung der Einkommensgrenze:
{}",
                        zrOhneErhoehung);

            // 2.2 Berechnung Erhöhung Zusatzrente wegen Erhöhung der
            // Einkommensgrenze
            // aufgrund von Familienzulagen
            Euro zrMitErhoehung = zusatzrente.getGrundbetrag();
            Euro erhoehungZusatzrente = zrMitErhoehung.minus(zrOhneErhoehung);

            logger.debug("§ 16: Zusatzrente bei Erhöhung der Einkommensgrenze:
{}",
                        zrMitErhoehung);
            logger.debug("§ 16: Erhöhte ZR {} - ZR {} = Erhöhung ZR {}",
                        zrMitErhoehung, zrOhneErhoehung, erhoehungZusatzrente);

            // 3. Wenn keine Erhöhung durch Familienzulagen dann keine Kürzung
            if (erhoehungZusatzrente.equals(ZERO)) {
                  logger.debug("§ 16: Keine Erhöhung der Zusatzrente durch
Familienzulagen, daher keine Kürzung.");

                  return familienzulagen;
            }

            // 4. Sonderfall § 16 Abs. 1 letzter Satz: Wenn ohne Erhöhung der
            // Einkommensgrenze keine Zusatzrente dann beträgt die Familienzulage
€ 40,60
            if (zrOhneErhoehung.equals(ZERO)) {
                  logger.debug("§ 16 Abs. 1 letzter Satz: Familienzulage € 40,60
da ohne Erhöhung der Einkommensgrenze keine Zusatzrente.");

                  return Collections.singletonList(of("40.60"));
            }

            // 5. Berechnung der Kürzung gemäß § 16 Abs. 1: Familienzulagen sind
zu gleichen Teilen zu kürzen
            Euro kuerzung = erhoehungZusatzrente.divide(beschaedigter
                        .getAngehoerige());
            logger.debug(
                        "§ 16: Erhöhung ZR {} / Anzahl Angehörige {} = Kürzung
pro Familienzulage {}",
                        erhoehungZusatzrente, beschaedigter.getAngehoerige(),
kuerzung);

            familienzulagen = new ArrayList<>();
            for (int i = 0; i < beschaedigter.getAngehoerige(); i++) {
                  familienzulagen.add(FAMILIENZULAGE.minus(kuerzung));
```

```
            }
            logger.debug("§ 16: Familienzulagen nach Kürzung: {}",
familienzulagen);
            // Bei ungerader Anzahl kann ein Cent verloren gehen

            return Collections.unmodifiableList(familienzulagen);
      }

      /**
       * Berechnet die Schwerstbeschädigtenzulage zur Beschädigtenrente gemäß §
       * 11a.
       *
       * @return die Schwerstbeschädigtenzulage
       */
      protected Euro calculateSchwerstbeschaedigtenzulage(
                  Beschaedigter beschaedigter) {
            logger.debug("§ 11a: Berechne Schwerstbeschädigtenzulage...");

            if (beschaedigter.getMaxMdE() < 50) {
                  logger.debug(
                              "§ 11a: MdE {} < 50 - kein Anspruch auf
Schwerstbeschädigtenzulage.",
                              beschaedigter.getMaxMdE());
                  return null;
            }

            int mde = beschaedigter.getSummeMdE();

            if (mde < 130) {
                  logger.debug(
                              "§ 11a: Summe MdE {} < 130 - kein Anspruch auf
Schwerstbeschädigtenzulage.",
                              mde);
                  return null;
            }

            Euro zulage = ZERO;
            // Abs. 4
            if (mde >= 130) {
                  zulage = dataRepository.getSchwerstbeschaedigtenzulage(130);

                  logger.debug("§ 11a Abs. 4 lit. a) MdE: {}, Zulage: {}", mde,
                              zulage);
            }
            if (mde >= 160) {
                  zulage = dataRepository.getSchwerstbeschaedigtenzulage(160);

                  logger.debug("§ 11a Abs. 4 lit. b) MdE: {}, Zulage: {}", mde,
                              zulage);
            }
            if (mde >= 190) {
                  zulage = dataRepository.getSchwerstbeschaedigtenzulage(190);

                  logger.debug("§ 11a Abs. 4 lit. c) MdE: {}, Zulage: {}", mde,
                              zulage);
            }
            if (mde >= 220) {
                  zulage = dataRepository.getSchwerstbeschaedigtenzulage(220);

                  logger.debug("§ 11a Abs. 4 lit. d) MdE: {}, Zulage: {}", mde,
```

```
                        zulage);
            }
            if (mde >= 250) {
                  zulage = dataRepository.getSchwerstbeschaedigtenzulage(250);

                  logger.debug("§ 11a Abs. 4 lit. e) MdE: {}, Zulage: {}", mde,
                        zulage);
            }
            if (mde >= 280) {
                  zulage = dataRepository.getSchwerstbeschaedigtenzulage(280);

                  logger.debug("§ 11a Abs. 4 lit. f) MdE: {}, Zulage: {}", mde,
                        zulage);
            }

            // Abs. 5: Erhöhung für Empfänger einer Pflege- oder Blindenzulage
            Pflegestufe stufe = null;

            if (beschaedigter.isBlind()) {
                  stufe = beschaedigter.getBlindheit().getStufe();
            } else {
                  stufe = beschaedigter.getPflegestufe();
            }

            if (stufe == null) {
                  logger.debug("§ 11a Abs. 5: Beschädiger erhält keine
Pflegezulage und ist nicht blind - "
                        + "kein Anspruch auf erhöhte
Schwerstbeschädigtenzulage.");
                  return zulage;
            }

            Euro zulagePflegestufe = ZERO;

            if (stufe == Pflegestufe.I && beschaedigter.getPflegestufe() != null)
{
                  // Stufe I und Pflegezulage
                  zulagePflegestufe = dataRepository
                        .getSchwerstbeschaedigtenzulage(130);

            } else if (stufe == Pflegestufe.II
                        && beschaedigter.getPflegestufe() != null) {
                  // Stufe II und Pflegezulage
                  zulagePflegestufe = dataRepository
                        .getSchwerstbeschaedigtenzulage(160);
            } else if (stufe == Pflegestufe.III) {
                  // Stufe III
                  zulagePflegestufe = dataRepository
                        .getSchwerstbeschaedigtenzulage(190);
            } else if (stufe == Pflegestufe.IV) {
                  // Stufe IV
                  zulagePflegestufe = dataRepository
                        .getSchwerstbeschaedigtenzulage(220);
            } else if (stufe == Pflegestufe.V) {
                  // Stufe V
                  zulagePflegestufe = dataRepository
                        .getSchwerstbeschaedigtenzulage(250);
            } else if (stufe == Pflegestufe.V_PLUS) {
                  // Erhöhte Stufe V
                  zulagePflegestufe = dataRepository
```

```
                                .getSchwerstbeschaedigtenzulage(280);
            }

            logger.debug(
                        "§ 11a Abs. 5: Pflegestufe: {}, Blind: {} - erhöhte
Schwerstbeschädigtenzulage: {}.",
                        stufe, beschaedigter.isBlind(), zulagePflegestufe);

            // Abs. 5, 1. Satz: Zulage nach Abs. 5 nur wenn sich nicht aus Abs. 4
            // ein höherer Betrag ergibt
            if (zulagePflegestufe.compareTo(zulage) == 1) {
                  // Wenn Zulage nach Pflegestufe größer
                  logger.debug(
                              "§ 11a Abs. 5: Erhöhte Schwerstbeschädigtenzulage
{} > {}",
                              zulagePflegestufe, zulage);
                  zulage = zulagePflegestufe;
            }

            return zulage;
      }

      /**
       * Berechnet die Pflegezulage gemäß § 18.
       *
       * @return die Pflegezulage
       */
      protected Euro calculatePflegezulage(Beschaedigter beschaedigter) {
            logger.debug("§ 18: Berechne Pflegezulage...");

            if (beschaedigter.isBlind()) {
                  logger.debug("§ 19: Beschädigter ist blind - kein Anspruch auf
Pflegezulage.");
                  return null;
            }

            Pflegestufe stufe = beschaedigter.getPflegestufe();

            if (stufe == null) {
                  logger.debug("§ 18: Kein Pflegebedarf - kein Anspruch auf
Pflegezulage.");
                  return null;
            }

            Euro zulage = dataRepository.getPflegezulage(stufe);
            logger.debug("§ 18: Pflegezulage Stufe {}: {}", stufe, zulage);

            return zulage;
      }

      /**
       * Berechnet die Blindenzulage gemäß § 19.
       *
       * @return die Blindenzulage
       */
      protected Euro calculateBlindenzulage(Beschaedigter beschaedigter) {
            logger.debug("§ 19: Berechne Blindenzulage...");

            Blindheit blindheit = beschaedigter.getBlindheit();
```

```
            if (blindheit == null) {
                  logger.debug("§ 19: Beschädigter ist nicht blind - kein
Anspruch auf Blindenzulage.");

                  return null;
            }

            // Blindheit wird nach Pflegestufe entschädigt
            Euro zulage = dataRepository.getPflegezulage(blindheit.getStufe());
            logger.debug("Blindenzulage Stufe {}: {}", blindheit.getStufe(),
zulage);

            return zulage;
      }

      /**
       * Berechnet die Blindenführzulage gemäß § 20.
       *
       * @return die Blindenführzulage
       */
      protected Euro calculateBlindenfuehrzulage(Beschaedigter beschaedigter) {
            logger.debug("§ 20: Berechne Blindenführzulage...");

            if (beschaedigter.getBlindheit() == null) {
                  logger.debug("§ 20: Beschädigter ist nicht blind - kein
Anspruch auf Blindenführzulage.");

                  return null;
            }

            return dataRepository.getBlindenfuehrzulage();
      }

      /**
       * Berechnet die Kleider- und Wäschepauschale(n) gemäß § 20a. Mehere
       * Ansprüche sind gemäß § 20a Abs. 2 KOVG kumulativ zu gewähren.
       *
       * @return eine Liste der Kleiderpauschalen ({@code null} wenn kein
       *         Anspruch)
       */
      protected List<Euro> calculateKleiderpauschalen(Beschaedigter beschaedigter)
{
            logger.debug("§ 20a: Berechne Kleider- und Wäschepauschale(n)...");

            List<Kleiderverbrauch> verbrauchList = beschaedigter
                        .getKleiderverbrauch();

            if (CollectionUtils.isEmpty(verbrauchList)) {
                  logger.debug("§ 20a: Kein außergewöhnlicher Kleider- und
Wäscheverbrauch - "
                              + "kein Anspruch auf Kleider- und
Wäschepauschale.");

                  return null;
            }

            List<Euro> pauschalen = new ArrayList<>();
            for (Kleiderverbrauch verbrauch : verbrauchList) {
                  Euro pauschale = dataRepository.getKleiderpauschale(verbrauch);
                  logger.debug("Kleiderverbrauch {}: {}", verbrauch, pauschale);
```

```
                        pauschalen.add(pauschale);
                }

                return Collections.unmodifiableList(pauschalen);
        }

        /**
         * Berechnet die Einkommensgrenze gemäß § 12 Abs. 2 <b>ohne</b> eine
         * Erhöhung durch Familienzulagen.
         *
         * @return die Einkommensgrenze (Grundrente für erwerbsunfähige
         *         Schwerbeschädigte ohne Erhöhungen + max. Zusatzrente ohne
         *         Erhöhungen)
         */
        private Euro calculateStandardEinkommensgrenze() {
                // Berechnung Einkommensgrenze gemäß § 12 Abs. 2:
                // Grundrente für Erwerbsunfähige (MdE >= 90) + Zusatzrente ohne
                // Familienzulagen
                Euro grundrente = dataRepository

        .getGrundbetragGrundrente(Constants.MDE_ERWERBSUNFAEHIGE);
                Euro einkommensgrenze = grundrente.plus(MAX_ZUSATZRENTE);
                logger.debug(
                                "Einkommensgrenze gemäß § 12 Abs. 2 ohne
Familienzulagen: "
                                                + "Grundrente {} + Zusatzrente {} = {}",
grundrente,
                                MAX_ZUSATZRENTE, einkommensgrenze);

                return einkommensgrenze;
        }

        public DataRepository getDataRepository() {
                return dataRepository;
        }

        /**
         * Setzt das zu verwendende {@link DataRepository}.
         *
         * @param dataRepository
         *            das Repository
         */
        @Autowired
        public void setDataRepository(DataRepository dataRepository) {
                this.dataRepository = dataRepository;
        }
}
```

Anhang A.18. Constants.java

```
package at.js.diss.simple.service.support;

import static at.js.diss.simple.domain.Euro.of;
import at.js.diss.simple.domain.Euro;

/**
 * Class containing constants used throughout the application.
 *
 * @author J. Scharf
```

```
 * @version 1.0.0
 */
public class Constants {
	private Constants() {
	}

	/**
	 * Amount in Euro representing 0.
	 */
	public static final Euro ZERO = of(0);

	/**
	 * Maximale Zusatzrente gemäß § 12 Abs. 2 KOVG.
	 */
	public static final Euro MAX_ZUSATZRENTE = of("266.90");

	/**
	 * 2/3 des Richtsatzes für Pensionsberechtigte auf Witwen(Witwer)pension
	 * oder Pension nach § 259 ASVG (§ 291 Abs. 1 lit. b ASVG).
	 */
	// Rundung auf 10 Cent gemäß § 63 Abs. 3 KOVG
	public static final Euro ASVG_PENSIONSRICHTSATZ = of("837.63")
			.applyTenCentRounding().multiply(0.66);

	/**
	 * Erhöhung der Einkommensgrenze gemäß § 12 Abs. 2 KOVG wenn Familienzulagen
	 * gebühren.
	 */
	public static final Euro ERHOEHUNG_EINKOMMENSGRENZE = of("40.60");

	/**
	 * Familienzulage zur Zusatzrente gemäß § 16 KOVG. Diese ist das doppelte
	 * des Betrags gemäß § 12 Abs. 2 KOVG.
	 */
	public static final Euro FAMILIENZULAGE = ERHOEHUNG_EINKOMMENSGRENZE
			.multiply(2);

	/**
	 * MdE für erwerbsunfähige Schwerbeschädigte.
	 */
	public static final int MDE_ERWERBSUNFAEHIGE = 90;
}
```

Anhang A.19. CalculationUtils.java

```
package at.js.diss.simple.support;

import org.joda.time.LocalDate;
import org.joda.time.Years;

import at.js.diss.simple.domain.Euro;

import com.google.common.base.Preconditions;

/**
 * Klasse mit Hilfsmethoden für die Berechnung.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
```

```java
public class CalculationUtils {
	private CalculationUtils() {
	}

	/**
	 * Berechnet den absoluten Prozentwert des gegebenen Prozentsatzes v.H. vom
	 * Grundwert {@code amount}.
	 *
	 * @param amount
	 *            der Grundwert in Euro
	 * @param percentage
	 *            der Prozentsatz v.H.
	 * @return den absolute Prozentwert des Prozentsatzes vom Grundwert
	 */
	public static Euro percentageOf(Euro amount, int percentage) {
		Preconditions.checkNotNull(amount, "The amount must not be null.");

		return amount.multiply((double) percentage / 100);
	}

	/**
	 * Ermittelt den größeren der beiden Beträge.
	 *
	 * @param a
	 *            Betrag a
	 * @param b
	 *            Betrag b
	 * @return den größeren Betrag
	 */
	public static Euro max(Euro a, Euro b) {
		Preconditions.checkNotNull(a, "The amount a must not be null.");
		Preconditions.checkNotNull(b, "The amount b must not be null.");

		if (a.getAmount().compareTo(b.getAmount()) >= 0) {
			// Return a if >= b
			return a;
		}
		return b;
	}

	/**
	 * Ermittelt den kleineren der beiden Beträge.
	 *
	 * @param a
	 *            Betrag a
	 * @param b
	 *            Betrag b
	 * @return den kleineren Betrag
	 */
	public static Euro min(Euro a, Euro b) {
		Preconditions.checkNotNull(a, "The amount a must not be null.");
		Preconditions.checkNotNull(b, "The amount b must not be null.");

		if (a.getAmount().compareTo(b.getAmount()) <= 0) {
			// Return a if <= b
			return a;
		}
		return b;
	}
```

```
	/**
	 * Ermittelt das Alter zum gegebenen Datum {@code now}.
	 *
	 * @param birthday
	 *            der Geburtstag
	 * @param now
	 *            das Datum an dem das Alter ermittelt werden soll
	 * @return das Alter am gegebenen Datum {@code now}
	 */
	public static int getAge(LocalDate birthday, LocalDate now) {
		Preconditions.checkNotNull(birthday, "The birthday must not be
null.");
		Preconditions.checkNotNull(now, "The now must not be null.");

		return Years.yearsBetween(birthday, now).getYears();
	}

	/**
	 * Ermittelt das Alter des Beschädigten im Vormonat. Dies ist wichtig, da
	 * diverse Erhöhungen erst ab dem 1.1. des nachfolgenden Monats gewährt
	 * werden. Vgl. § 11 Abs. 2 und 3 KOVG.
	 * <p/>
	 * Der Tag des Geburtstags wird ignoriert - d.h. die Methode liefert das
	 * Alter des Beschädigten am Ende des Vormonats.
	 * <p/>
	 * Beispiel: <br>
	 * 1) Aktuelles Datum: 1.3.2014, Geburtsdatum: 13.2.1984, Ergebnis: 30. <br>
	 * 2) Aktuelles Datum: 14.2.2014, Geburtsdatum: 13.2.1984, Ergebnis: 29,
	 * obwohl der Beschädigte am 13. Geburstag hatte. Es wird das Alter im
	 * Vormonat ermittelt.
	 *
	 * @param birthday
	 *            der Geburtstag des Beschädigten
	 * @param now
	 *            das aktuelle Datum
	 * @return das Alter des Beschädigten im Vormonat
	 */
	public static int getAlterVormonat(LocalDate birthday, LocalDate now) {
		Preconditions.checkNotNull(birthday, "The birthday must not be
null.");
		Preconditions.checkNotNull(now, "The now must not be null.");

		LocalDate localNow = now.minusMonths(1).withDayOfMonth(1);
		LocalDate localBirthday = birthday.withDayOfMonth(1);

		return getAge(localBirthday, localNow);
	}
}
```

Anhang A.20. PrintUtils.java

```
package at.js.diss.simple.support;

import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.List;

import org.apache.commons.lang3.StringUtils;
import org.joda.time.format.DateTimeFormat;
import org.joda.time.format.DateTimeFormatter;
```

```java
import org.springframework.util.CollectionUtils;

import at.js.diss.simple.domain.Beschaedigtenrente;
import at.js.diss.simple.domain.Beschaedigter;
import at.js.diss.simple.domain.Blindheit;
import at.js.diss.simple.domain.Diaetverpflegung;
import at.js.diss.simple.domain.Dienstbeschaedigung;
import at.js.diss.simple.domain.Euro;
import at.js.diss.simple.domain.Grundrente;
import at.js.diss.simple.domain.Kleiderverbrauch;
import at.js.diss.simple.domain.Pflegestufe;
import at.js.diss.simple.domain.Zusatzrente;

import com.google.common.base.Preconditions;

/**
 * Klasse die Methoden zur Ausgabe diverser Objekte enthält.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class PrintUtils {
	/**
	 * The new line character.
	 */
	public static final String LINE_SEPARATOR = System.getProperty(
			"line.separator", "\n");
	private static final DateTimeFormatter formatter = DateTimeFormat
			.forPattern("yyyy-MM-dd");
	private static final DateTimeFormatter shortFormatter = DateTimeFormat
			.forPattern("yyyy-MM");
	private static final NumberFormat currencyFormatter = new DecimalFormat(
			"#,##0.00");

	private PrintUtils() {
	}

	/**
	 * Erzeugt eine formatierte Ausgabe eines Objekts vom Typ
	 * {@link Beschaedigter}.
	 *
	 * @param beschaedigter
	 *            das zu formatierende Objekt
	 * @return eine textuelle Repräsentation des Objekts
	 */
	public static String print(Beschaedigter beschaedigter) {
		Preconditions.checkNotNull(beschaedigter,
				"The beschaedigter must not be null.");

		TablePrinter printer = new TablePrinter();
		int width = 30;

		printer.column("Name:", width).column(beschaedigter.getName());
		printer.row().column("Geschlecht:", width)
				.column(beschaedigter.getGeschlecht());
		printer.row().column("Geb. am:", width)

	.column(formatter.print(beschaedigter.getGeburtsdatum()));
		printer.row().column("Einkommen:", width)
				.column(beschaedigter.getEinkommen().toString());
```

```
		printer.row().column("Angehörige:", width)
				.column(beschaedigter.getAngehoerige());

		Pflegestufe pflegestufe = beschaedigter.getPflegestufe();
		if (pflegestufe != null) {
			printer.row().column("Pflegestufe:", width)
					.column(beschaedigter.getPflegestufe());
		}

		Blindheit blindheit = beschaedigter.getBlindheit();
		if (blindheit != null) {
			printer.row().column("Blind - Stufe:", width)
					.column(blindheit.getStufe());
		}

		Diaetverpflegung verpflegung = beschaedigter.getDiaetverpflegung();
		if (verpflegung != null) {
			printer.row().column("Diätverpflegung - Zf.: ", width)
					.column(verpflegung);
		}

		List<Kleiderverbrauch> kleiderverbraeuche = beschaedigter
				.getKleiderverbrauch();
		if (!CollectionUtils.isEmpty(kleiderverbraeuche)) {
			int index = 1;
			printer.row().column("Außergewöhnlicher Kleiderverbrauch:");

			for (Kleiderverbrauch verbrauch : kleiderverbraeuche) {
				printer.row().column("  [" + index + "] Zf.: ", width)
						.column(verbrauch);
				index++;
			}
		}

		printer.row().column("Dienstbeschädigungen:");
		int index = 1;
		for (Dienstbeschaedigung beschaedigung : beschaedigter
				.getDienstbeschaedigungen()) {
			printer.row().column("  [" + index + "] MdE: ", width)
					.column(beschaedigung.getMde());
			index++;
		}

		return printer.toString();
	}

	/**
	 * Erzeugt eine formatierte Ausgabe eines Objekts vom Typ
	 * {@link Beschaedigtenrente}.
	 *
	 * @param rente
	 *            das zu formatierende Objekt
	 * @return eine textuelle Repräsentation des Objekts
	 */
	public static String print(Beschaedigtenrente rente) {
		Preconditions.checkNotNull(rente, "The rente must not be null.");

		ResultTablePrinter printer = new ResultTablePrinter();
		printer.first(
```

L

```
                        "Berechnung der Beschädigtenrente gemäß §§ 7 ff. KOVG
für "
                                        + shortFormatter.print(rente.getDatum()) +
":").row();

                // Grundrente
                Grundrente grundrente = rente.getGrundrente();
                print(grundrente, printer);

                // Zusatzrente
                if (rente.getZusatzrente() != null) {
                        print(rente.getZusatzrente(), printer);
                }

                // Zulagen zur Beschädigtenrente
                printZulagen(rente, printer);

                // Gesamtbetrag
                printer.row().separator().row()
                                .first("Gesamtbetrag der Beschädigtenrente").second("")
                                .third(print(rente.getBetrag()));

                return printer.toString();
        }

        private static void print(Grundrente gr, ResultTablePrinter printer) {
                printer.row().first("Berechnung der Grundrente gemäß § 11:");
                printer.row().first("  Grundbetrag (Abs. 1)")
                                .second(print(gr.getGrundbetrag()));
                if (gr.getAltersabhaengigeErhoehung() != null) {
                        printer.row().first("  Erhöhung (Abs. 2 und 3)")
                                        .second(print(gr.getAltersabhaengigeErhoehung()));
                }
                printer.separator();
                printer.row().first("Gesamtbetrag Grundrente").second("")
                                .third(print(gr.getBetrag()));
        }

        private static void print(Zusatzrente zr, ResultTablePrinter printer) {
                printer.row().row().first("Berechnung der Zusatzrente gemäß § 12:");
                printer.row().first("  Grundbetrag (Abs. 2)")
                                .second(print(zr.getGrundbetrag()));

                if (zr.getAusgleichszulage() != null) {
                        printer.row().first("  Ausgleichszulage (Abs. 3)")
                                        .second(print(zr.getAusgleichszulage()));
                }
                printer.separator();
                printer.row().first("Gesamtbetrag Zusatzrente").second("")
                                .third(print(zr.getSummeGrundbetragMitErhoehung()));

                // Zulagen
                if (!zr.hasZulagen()) {
                        return;
                }
                printer.row().row().first("Berechnung Zulagen zur Zusatzrente:");

                if (zr.getDiaetzuschuss() != null) {
                        printer.row().first("  Diätzuschuss (§ 14)")
                                        .second(print(zr.getDiaetzuschuss()));
```

```
            }
            if (zr.getFamilienzulagen() != null) {
                  printer.row()
                              .first("  " + zr.getFamilienzulagen().size()
                                          + "  Familienzulage(n) (§ 16)")
                              .second(print(zr.getSummeFamiliezulagen()));
            }
            printer.separator();
            printer.row().first("Gesamtbetrag der Zulagen zur Zusatzrente")
                        .second("").third(print(zr.getSummeZulagen()));
      }

      private static void printZulagen(Beschaedigtenrente rente,
                  ResultTablePrinter printer) {
            if (!rente.hasZulagen()) {
                  return;
            }

            printer.row().row().first("Berechnung Zulagen zur
Beschädigtenrente:");

            if (rente.getSchwerstbeschaedigtenzulage() != null) {
                  printer.row().first("  Schwerstbeschädigtenzulage (§ 11a)")

      .second(print(rente.getSchwerstbeschaedigtenzulage()));
            }
            if (rente.getPflegezulage() != null) {
                  printer.row().first("  Pflegezulage (§ 18)")
                              .second(print(rente.getPflegezulage()));
            }
            if (rente.getBlindenzulage() != null) {
                  printer.row().first("  Blindenzulage (§ 19")
                              .second(print(rente.getBlindenzulage()));
            }
            if (rente.getBlindenfuehrzulage() != null) {
                  printer.row().first("  Blindenführzulage (§ 20)")
                              .second(print(rente.getBlindenfuehrzulage()));
            }
            if (rente.getKleiderpauschalen() != null) {
                  printer.row()
                              .first("  " + rente.getKleiderpauschalen().size()
                                          + " Kleiderpauschale(n) (§ 20a)")
                              .second(print(rente.getSummeKleiderpauschalen()));
            }
            printer.separator();
            printer.row().first("Gesamtbetrag der Zulagen zur Beschädigtenrente")
                        .second("").third(print(rente.getSummeZulagen()));
      }

      /**
       * Formatiert den übergebenen Betrag. Die erzeugte Ausgabe ist inklusive
       * Währungszeichen zehn Zeichen breit.
       *
       * @param amount
       *            der zu formatierende Betrag
       * @return eine textuelle Repräsentation des übergebenen Betrags
       */
      public static String print(Euro amount) {
            Preconditions.checkNotNull(amount, "The amount must not be null.");
```

```
            int width = 9;
            StringBuilder buffer = new StringBuilder("€");
            buffer.append(StringUtils.leftPad(
                        currencyFormatter.format(amount.getAmount()), width));

            return buffer.toString();
      }
}
```

Anhang A.21. ResultTablePrinter.java

```
package at.js.diss.simple.support;

import org.apache.commons.lang3.StringUtils;

/**
 * Utility class for printing a calculation resulat as a table. Columns two and
 * three are supposed to hold a currency.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class ResultTablePrinter {
      private final TablePrinter printer = new TablePrinter();
      private final int firstWidth = 50;
      private final int secondWidth = 16;
      private final int thirdWidth = 16;

      /**
       * First column.
       */
      public ResultTablePrinter first(String text) {
            printer.column(text, firstWidth);
            return this;
      }

      /**
       * Second column.
       */
      public ResultTablePrinter second(String text) {
            // Add whitespace on left side
            text = StringUtils.leftPad(text, secondWidth - 1);
            printer.column(text, secondWidth);
            return this;
      }

      /**
       * Third column.
       */
      public ResultTablePrinter third(String text) {
            // Add whitespace on left side
            text = StringUtils.leftPad(text, thirdWidth - 1);
            printer.column(text, thirdWidth);
            return this;
      }

      /**
       * Starts a new row.
       */
      public ResultTablePrinter row() {
```

```
			printer.row();
			return this;
		}

		/**
		 * Starts a new row and prints a horizontal separator across columns two and
		 * three.
		 */
		public ResultTablePrinter separator() {
			printer.row();
			first("");
			second(StringUtils.rightPad("", secondWidth, "-"));
			third(StringUtils.rightPad("", thirdWidth, "-"));
			return this;
		}

		/**
		 * Returns the appended text aligned as table.
		 */
		@Override
		public String toString() {
			return printer.toString();
		}
}
```

Anhang A.22. TablePrinter.java

```
package at.js.diss.simple.support;

import static org.apache.commons.lang3.StringUtils.rightPad;

import com.google.common.base.Preconditions;

/**
 * Utility class for printing values aligned as a table.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class TablePrinter {
		/**
		 * The new line character.
		 */
		private static final String LINE_SEPARATOR = System.getProperty(
				"line.separator", "\n");

		private final String padStr;
		private final StringBuilder buffer = new StringBuilder();

		/**
		 * Constructs a new {@link TablePrinter}. A whitespace character will be
		 * used for padding.
		 */
		public TablePrinter() {
			this.padStr = "";
		}

		/**
		 * Constructs a new {@link TablePrinter}.
		 *
```

```
	 * @param padStr
	 *            the text used for padding (may not be {@code null})
	 */
	public TablePrinter(String padStr) {
		this.padStr = Preconditions.checkNotNull(padStr,
				"The padStr must not be null.");
	}

	/**
	 * Appends the given text to the internal buffer.
	 */
	public TablePrinter column(Object text) {
		buffer.append(text);

		return this;
	}

	/**
	 * Applies right-sided padding to the given text and appends the result to
	 * the internal buffer.
	 *
	 * @param text
	 *            the text to append
	 * @param size
	 *            the desired width of the column
	 */
	public TablePrinter column(Object text, int width) {
		if (text == null) {
			text = "null";
		}
		buffer.append(rightPad(text.toString(), width, padStr));

		return this;
	}

	/**
	 * Starts a new row by writing a line separator.
	 */
	public TablePrinter row() {
		buffer.append(LINE_SEPARATOR);

		return this;
	}

	/**
	 * Returns the content of the internal buffer.
	 */
	@Override
	public String toString() {
		return buffer.toString();
	}
}
```

Anhang A.23. LoggingTestWatcher.java

```
package at.js.diss.simple.test;

import org.junit.rules.TestWatcher;
import org.junit.runner.Description;
import org.slf4j.Logger;
```

```java
import org.slf4j.LoggerFactory;

import com.google.common.base.Preconditions;

/**
 * {@link TestWatcher} which prints out start and result of each unit test.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public final class LoggingTestWatcher extends TestWatcher {
	private Logger logger = LoggerFactory.getLogger(getClass());

	/**
	 * Empty constructor. Default logger using the class' name is used.
	 */
	public LoggingTestWatcher() {
	}

	/**
	 * Constructor which allows for specifying the logger to use.
	 *
	 * @param logger
	 *            the logger to use
	 */
	public LoggingTestWatcher(Logger logger) {
		this.logger = Preconditions.checkNotNull(logger,
				"The logger must not be null.");
	}

	/**
	 * Called when test starts.
	 */
	@Override
	protected void starting(Description description) {
		logger.info("Starting {}...", description.getMethodName());
	}

	/**
	 * Called when test finished.
	 */
	@Override
	protected void finished(Description description) {
		logger.info("Finished {}.", description.getMethodName());

	}

	/**
	 * Called when test suceeded.
	 */
	@Override
	protected void succeeded(Description description) {
		logger.info("Succeeded {}.", description.getMethodName());
	}

	/**
	 * Called when test failed
	 */
	@Override
	protected void failed(Throwable e, Description description) {
```

```
            logger.info("Failed {}", description.getMethodName(), e);
            super.failed(e, description);
        }
}
```

Anhang A.24. EuroTest.java

```
package at.js.diss.simple.test.domain;

import java.math.BigDecimal;

import org.junit.Assert;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TestRule;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import at.js.diss.simple.domain.Euro;
import at.js.diss.simple.test.LoggingTestWatcher;

/**
 * Unittest for the {@link Euro} class.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class EuroTest {
        private final Logger logger = LoggerFactory.getLogger(getClass());
        @Rule
        public final TestRule loggingRule = new LoggingTestWatcher(logger);

        @Test
        public void testPlus() {
                Euro amount = Euro.of("12.40");
                long amountToAdd = 7;
                BigDecimal expected = new BigDecimal("19.40");

                Euro result = amount.plus(amountToAdd);
                logger.info("{} + {} = {}", amount, amountToAdd, result);
                Assert.assertEquals(expected, result.getAmount());
        }

        @Test
        public void testMinus() {
                Euro amount = Euro.of("12.40");
                long amountToSubstract = 7;
                BigDecimal expected = new BigDecimal("5.40");

                Euro result = amount.minus(amountToSubstract);
                logger.info("{} - {} = {}", amount, amountToSubstract, result);
                Assert.assertEquals(expected, result.getAmount());
        }

        @Test
        public void testMultiply() {
                Euro amount = Euro.of("12.40");
                long multiplicand = 7;
                BigDecimal expected = new BigDecimal("86.80");
```

```
        Euro result = amount.multiply(multiplicand);
        logger.info("{} * {} = {}", amount, multiplicand, result);
        Assert.assertEquals(expected, result.getAmount());
    }

    @Test
    public void testDivide() {
        Euro amount = Euro.of("12.40");
        long divisor = 7;
        BigDecimal expected = new BigDecimal("1.77");

        Euro result = amount.divide(divisor);
        logger.info("{} / {} = {}", amount, divisor, result);
        Assert.assertEquals(expected, result.getAmount());
    }

    @Test
    public void testCalculatePercentage() {
        Euro amount = Euro.of("478.30");
        long percentage = 30;
        BigDecimal expected = new BigDecimal("143.49");

        Euro result = amount.multiply(percentage).divide(100);
        logger.info("{} percent of {} = {}", percentage, amount, result);
        Assert.assertEquals(expected, result.getAmount());
    }

    @Test
    public void testRoundToTenCentUp() {
        Euro amount = Euro.of("143.45");
        BigDecimal expected = new BigDecimal("143.50");

        Euro result = amount.roundToTenCent();
        logger.info("{} rounded to ten cent: {}", amount, result);
        Assert.assertEquals(expected, result.getAmount());
    }

    @Test
    public void testRoundToTenCentDown() {
        Euro amount = Euro.of("143.44");
        BigDecimal expected = new BigDecimal("143.40");

        Euro result = amount.roundToTenCent();
        logger.info("{} rounded to ten cent: {}", amount, result);
        Assert.assertEquals(expected, result.getAmount());
    }

    /**
     * Test rounding to ten cent directly.
     */
    @Test
    public void testDivideWithRoundingToTenCent() {
        // The result of the following operation is the tricky
        // value 143.145
        Euro amount = Euro.of("400");
        double divisor = 2.794369;
        BigDecimal expected = new BigDecimal("143.10");

        // Uncomment the following to show what happens if rounding to two
        // fractional digits first
```

```
            /*
             * logger.info("Rounded to two fractional digits first: {}", amount
             * .divide(divisor).roundToTenCent());
             */

            Euro result = amount.applyTenCentRounding().divide(2.794369);
            logger.info("{} divided by {} rounded to ten cent directly: {}",
                        amount, divisor, result);
            Assert.assertEquals(expected, result.getAmount());
      }
}
```

Anhang A.25. JavaCalculationEngineTest.java

```
package at.js.diss.simple.test.service;

import static at.js.diss.simple.domain.Geschlecht.MAENNLICH;
import static org.joda.time.DateTimeZone.UTC;

import org.joda.time.DateTime;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TestRule;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import at.js.diss.simple.domain.Beschaedigtenrente;
import at.js.diss.simple.domain.Beschaedigter;
import at.js.diss.simple.service.JavaCalculationEngine;
import at.js.diss.simple.test.LoggingTestWatcher;

/**
 * Tests für die {@link JavaCalculationEngine}.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(value = "/spring/*")
public class JavaCalculationEngineTest {
      private final Logger logger = LoggerFactory.getLogger(getClass());
      @Rule
      public final TestRule loggingRule = new LoggingTestWatcher(logger);

      @Autowired
      private JavaCalculationEngine calculationEngine;

      @Test
      public void testCalculateGrundrente() {
            Beschaedigter beschaedigter = createBeschaedigter();
            DateTime now = new DateTime(2014, 2, 14, 0, 0, 0, UTC);

            Beschaedigtenrente rente = calculationEngine.calculate(beschaedigter,
                        now);

            logger.info("Rente: {}", rente);
```

```
	}

	private Beschaedigter createBeschaedigter() {
		Beschaedigter.Builder builder = new Beschaedigter.Builder();

		builder.name("Max Mustermann").geschlecht(MAENNLICH)
				.geburtsdatum(new DateTime(1936, 7, 26, 0, 0, 0, UTC))
				.dienstbeschaedigung(60).angehoerige(2);
		return builder.build();
	}
}
```

Anhang A.26. CalculationUtilsTest.java

```
package at.js.diss.simple.test.support;

import static at.js.diss.simple.domain.Euro.of;

import org.joda.time.LocalDate;
import org.junit.Assert;
import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.TestRule;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import at.js.diss.simple.domain.Euro;
import at.js.diss.simple.support.CalculationUtils;
import at.js.diss.simple.test.LoggingTestWatcher;

/**
 * Tests für {@link CalculationUtils}.
 *
 * @author J. Scharf
 * @since 1.0.0
 */
public class CalculationUtilsTest {
	private final Logger logger = LoggerFactory.getLogger(getClass());
	@Rule
	public final TestRule loggingRule = new LoggingTestWatcher(logger);

	@Test
	public void testGetAge() {
		LocalDate now = new LocalDate(2014, 2, 14);
		LocalDate birthday = new LocalDate(1984, 2, 13);

		int result = CalculationUtils.getAge(birthday, now);
		logger.info("Now: {}, Birthday: {}, Age: {}", now, birthday, result);

		Assert.assertEquals(30, result);
	}

	@Test
	public void testGetAlterVormonat() {
		LocalDate now = new LocalDate(2014, 2, 14);
		LocalDate birthday = new LocalDate(1984, 2, 13);

		int result = CalculationUtils.getAlterVormonat(birthday, now);
		logger.info("Now: {}, Birthday: {}, Alter im Vormonat: {}", now,
				birthday, result);
```

```
		Assert.assertEquals(29, result);

		now = new LocalDate(2014, 3, 1);
		result = CalculationUtils.getAlterVormonat(birthday, now);
		logger.info("Now: {}, Birthday: {}, Alter im Vormonat: {}", now,
				birthday, result);
		Assert.assertEquals(30, result);
	}

	@Test
	public void testMax() {
		Euro a = of("12.0");
		Euro b = of("15.0");
		Euro max = CalculationUtils.max(a, b);

		logger.info("max({}, {}) = {}", a, b, max);
		Assert.assertEquals(b, max);
	}

	@Test
	public void testMin() {
		Euro a = of("12.0");
		Euro b = of("15.0");
		Euro max = CalculationUtils.min(a, b);

		logger.info("min({}, {}) = {}", a, b, max);
		Assert.assertEquals(a, max);
	}
}
```

Anhang B. Ontologie des Kriegsopferversorgungsgesetzes

Anhang B.1. kovg.owl

```
<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#" xml:base="http://js.at/diss/kovg"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
      ontologyIRI="http://js.at/diss/kovg">
      <Prefix name="" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
      <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
      <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
      <Declaration>
            <Class IRI="#Angehöriger" />
      </Declaration>
      <Declaration>
            <Class IRI="#Antragsteller" />
      </Declaration>
      <Declaration>
            <Class IRI="#BeruflicheMaßnahme" />
      </Declaration>
      <Declaration>
            <Class IRI="#Beschädigtenrente" />
      </Declaration>
      <Declaration>
            <Class IRI="#Beschädigter" />
      </Declaration>
      <Declaration>
            <Class IRI="#Blindenzulage" />
      </Declaration>
      <Declaration>
            <Class IRI="#Ehegatte" />
      </Declaration>
      <Declaration>
            <Class IRI="#Erwerbsunfähiger" />
      </Declaration>
      <Declaration>
            <Class IRI="#Familienzulage" />
      </Declaration>
      <Declaration>
            <Class IRI="#Geschlecht" />
      </Declaration>
      <Declaration>
            <Class IRI="#Heilfürsorge" />
      </Declaration>
      <Declaration>
            <Class IRI="#Kind" />
      </Declaration>
      <Declaration>
```

```
        <Class IRI="#MonetäreVersorgungsleistung" />
</Declaration>
<Declaration>
        <Class IRI="#OrthopädischeVersorgung" />
</Declaration>
<Declaration>
        <Class IRI="#Pflegestufe" />
</Declaration>
<Declaration>
        <Class IRI="#Pflegezulage" />
</Declaration>
<Declaration>
        <Class IRI="#Rente" />
</Declaration>
<Declaration>
        <Class IRI="#Schwerbeschädigter" />
</Declaration>
<Declaration>
        <Class IRI="#Schwerstbeschädigtenzulage" />
</Declaration>
<Declaration>
        <Class IRI="#SozialeMaßnahme" />
</Declaration>
<Declaration>
        <Class IRI="#Versorgungsleistung" />
</Declaration>
<Declaration>
        <Class IRI="#Zulage" />
</Declaration>
<Declaration>
        <ObjectProperty IRI="#hatAngehörigen" />
</Declaration>
<Declaration>
        <ObjectProperty IRI="#hatEhegatten" />
</Declaration>
<Declaration>
        <ObjectProperty IRI="#hatGeschlecht" />
</Declaration>
<Declaration>
        <ObjectProperty IRI="#hatKind" />
</Declaration>
<Declaration>
        <ObjectProperty IRI="#hatPflegestufe" />
</Declaration>
<Declaration>
        <ObjectProperty IRI="#istAngehörigerVon" />
</Declaration>
<Declaration>
        <ObjectProperty IRI="#istEhegatteVon" />
</Declaration>
<Declaration>
        <ObjectProperty IRI="#istKindVon" />
</Declaration>
<Declaration>
        <DataProperty IRI="#hatAlter" />
</Declaration>
<Declaration>
        <DataProperty IRI="#hatMdE" />
</Declaration>
<Declaration>
```

```
        <DataProperty IRI="#istBlind" />
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#Männlich" />
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#Pflegestufe_I" />
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#Pflegestufe_II" />
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#Pflegestufe_III" />
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#Pflegestufe_IV" />
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#Pflegestufe_V" />
    </Declaration>
    <Declaration>
        <NamedIndividual IRI="#Weiblich" />
    </Declaration>
    <EquivalentClasses>
        <Class IRI="#Geschlecht" />
        <ObjectOneOf>
            <NamedIndividual IRI="#Weiblich" />
            <NamedIndividual IRI="#Männlich" />
        </ObjectOneOf>
    </EquivalentClasses>
    <EquivalentClasses>
        <Class IRI="#Pflegestufe" />
        <ObjectOneOf>
            <NamedIndividual IRI="#Pflegestufe_IV" />
            <NamedIndividual IRI="#Pflegestufe_III" />
            <NamedIndividual IRI="#Pflegestufe_V" />
            <NamedIndividual IRI="#Pflegestufe_II" />
            <NamedIndividual IRI="#Pflegestufe_I" />
        </ObjectOneOf>
    </EquivalentClasses>
    <SubClassOf>
        <Class IRI="#BeruflicheMaßnahme" />
        <Class IRI="#Versorgungsleistung" />
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Beschädigtenrente" />
        <Class IRI="#Rente" />
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Beschädigter" />
        <Class IRI="#Antragsteller" />
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Blindenzulage" />
        <Class IRI="#Zulage" />
    </SubClassOf>
    <SubClassOf>
        <Class IRI="#Ehegatte" />
        <Class IRI="#Angehöriger" />
    </SubClassOf>
```

```
<SubClassOf>
      <Class IRI="#Erwerbsunfähiger" />
      <Class IRI="#Schwerbeschädigter" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Familienzulage" />
      <Class IRI="#Zulage" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Heilfürsorge" />
      <Class IRI="#Versorgungsleistung" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Kind" />
      <Class IRI="#Angehöriger" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#MonetäreVersorgungsleistung" />
      <Class IRI="#Versorgungsleistung" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#OrthopädischeVersorgung" />
      <Class IRI="#Versorgungsleistung" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Pflegezulage" />
      <Class IRI="#Zulage" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Rente" />
      <Class IRI="#MonetäreVersorgungsleistung" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Schwerbeschädigter" />
      <Class IRI="#Beschädigter" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Schwerstbeschädigtenzulage" />
      <Class IRI="#Zulage" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#SozialeMaßnahme" />
      <Class IRI="#Versorgungsleistung" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Versorgungsleistung" />
      <Class abbreviatedIRI="owl:Thing" />
</SubClassOf>
<SubClassOf>
      <Class IRI="#Zulage" />
      <Class IRI="#MonetäreVersorgungsleistung" />
</SubClassOf>
<DisjointClasses>
      <Class IRI="#Angehöriger" />
      <Class IRI="#Antragsteller" />
      <Class IRI="#Geschlecht" />
      <Class IRI="#Pflegestufe" />
      <Class IRI="#Versorgungsleistung" />
</DisjointClasses>
<DisjointClasses>
```

```
        <Class IRI="#BeruflicheMaßnahme" />
        <Class IRI="#Heilfürsorge" />
        <Class IRI="#MonetäreVersorgungsleistung" />
        <Class IRI="#OrthopädischeVersorgung" />
        <Class IRI="#SozialeMaßnahme" />
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#Blindenzulage" />
        <Class IRI="#Familienzulage" />
        <Class IRI="#Pflegezulage" />
        <Class IRI="#Schwerstbeschädigtenzulage" />
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#Ehegatte" />
        <Class IRI="#Kind" />
    </DisjointClasses>
    <DisjointClasses>
        <Class IRI="#Rente" />
        <Class IRI="#Zulage" />
    </DisjointClasses>
    <ClassAssertion>
        <Class IRI="#Geschlecht" />
        <NamedIndividual IRI="#Männlich" />
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#Pflegestufe" />
        <NamedIndividual IRI="#Pflegestufe_I" />
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#Pflegestufe" />
        <NamedIndividual IRI="#Pflegestufe_II" />
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#Pflegestufe" />
        <NamedIndividual IRI="#Pflegestufe_III" />
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#Pflegestufe" />
        <NamedIndividual IRI="#Pflegestufe_IV" />
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#Pflegestufe" />
        <NamedIndividual IRI="#Pflegestufe_V" />
    </ClassAssertion>
    <ClassAssertion>
        <Class IRI="#Geschlecht" />
        <NamedIndividual IRI="#Weiblich" />
    </ClassAssertion>
    <DifferentIndividuals>
        <NamedIndividual IRI="#Männlich" />
        <NamedIndividual IRI="#Weiblich" />
    </DifferentIndividuals>
    <DifferentIndividuals>
        <NamedIndividual IRI="#Pflegestufe_I" />
        <NamedIndividual IRI="#Pflegestufe_II" />
        <NamedIndividual IRI="#Pflegestufe_III" />
        <NamedIndividual IRI="#Pflegestufe_IV" />
        <NamedIndividual IRI="#Pflegestufe_V" />
    </DifferentIndividuals>
    <SubObjectPropertyOf>
```

```
        <ObjectProperty IRI="#hatEhegatten" />
        <ObjectProperty IRI="#hatAngehörigen" />
</SubObjectPropertyOf>
<SubObjectPropertyOf>
        <ObjectProperty IRI="#hatKind" />
        <ObjectProperty IRI="#hatAngehörigen" />
</SubObjectPropertyOf>
<SubObjectPropertyOf>
        <ObjectProperty IRI="#istEhegatteVon" />
        <ObjectProperty IRI="#istAngehörigerVon" />
</SubObjectPropertyOf>
<SubObjectPropertyOf>
        <ObjectProperty IRI="#istKindVon" />
        <ObjectProperty IRI="#istAngehörigerVon" />
</SubObjectPropertyOf>
<InverseObjectProperties>
        <ObjectProperty IRI="#istAngehörigerVon" />
        <ObjectProperty IRI="#hatAngehörigen" />
</InverseObjectProperties>
<InverseObjectProperties>
        <ObjectProperty IRI="#istEhegatteVon" />
        <ObjectProperty IRI="#hatEhegatten" />
</InverseObjectProperties>
<InverseObjectProperties>
        <ObjectProperty IRI="#istKindVon" />
        <ObjectProperty IRI="#hatKind" />
</InverseObjectProperties>
<AsymmetricObjectProperty>
        <ObjectProperty IRI="#hatPflegestufe" />
</AsymmetricObjectProperty>
<IrreflexiveObjectProperty>
        <ObjectProperty IRI="#hatAngehörigen" />
</IrreflexiveObjectProperty>
<IrreflexiveObjectProperty>
        <ObjectProperty IRI="#hatEhegatten" />
</IrreflexiveObjectProperty>
<IrreflexiveObjectProperty>
        <ObjectProperty IRI="#hatKind" />
</IrreflexiveObjectProperty>
<IrreflexiveObjectProperty>
        <ObjectProperty IRI="#hatPflegestufe" />
</IrreflexiveObjectProperty>
<ObjectPropertyDomain>
        <ObjectProperty IRI="#hatAngehörigen" />
        <Class IRI="#Antragsteller" />
</ObjectPropertyDomain>
<ObjectPropertyDomain>
        <ObjectProperty IRI="#hatEhegatten" />
        <Class IRI="#Antragsteller" />
</ObjectPropertyDomain>
<ObjectPropertyDomain>
        <ObjectProperty IRI="#hatKind" />
        <Class IRI="#Antragsteller" />
</ObjectPropertyDomain>
<ObjectPropertyDomain>
        <ObjectProperty IRI="#hatPflegestufe" />
        <Class IRI="#Antragsteller" />
</ObjectPropertyDomain>
<ObjectPropertyRange>
        <ObjectProperty IRI="#hatAngehörigen" />
```

```
            <Class IRI="#Angehöriger" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty IRI="#hatEhegatten" />
            <Class IRI="#Ehegatte" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty IRI="#hatGeschlecht" />
            <Class IRI="#Geschlecht" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty IRI="#hatKind" />
            <Class IRI="#Kind" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty IRI="#hatPflegestufe" />
            <Class IRI="#Pflegestufe" />
      </ObjectPropertyRange>
      <DataPropertyDomain>
            <DataProperty IRI="#hatMdE" />
            <Class IRI="#Antragsteller" />
      </DataPropertyDomain>
      <DataPropertyRange>
            <DataProperty IRI="#hatAlter" />
            <DatatypeRestriction>
                  <Datatype abbreviatedIRI="xsd:short" />
                  <FacetRestriction facet="&xsd;minExclusive">
                        <Literal datatypeIRI="&xsd;integer">0</Literal>
                  </FacetRestriction>
            </DatatypeRestriction>
      </DataPropertyRange>
      <DataPropertyRange>
            <DataProperty IRI="#hatMdE" />
            <DatatypeRestriction>
                  <Datatype abbreviatedIRI="xsd:short" />
                  <FacetRestriction facet="&xsd;minExclusive">
                        <Literal datatypeIRI="&xsd;integer">0</Literal>
                  </FacetRestriction>
            </DatatypeRestriction>
      </DataPropertyRange>
      <DataPropertyRange>
            <DataProperty IRI="#istBlind" />
            <Datatype abbreviatedIRI="xsd:boolean" />
      </DataPropertyRange>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>#Angehöriger</IRI>
            <Literal datatypeIRI="&rdf;PlainLiteral">Repräsentiert einen
Familienangehörigen gemäß § 16 Abs. 2 KOVG.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>#MonetäreVersorgungsleistung</IRI>
            <Literal datatypeIRI="&rdf;PlainLiteral">Monetäre Versorgungsleistung
die in in Euro geleistet wird.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>#Pflegestufe</IRI>
```

```
            <Literal datatypeIRI="&rdf;PlainLiteral">Pflegestufe gemäß §
18.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>#Rente</IRI>
            <Literal datatypeIRI="&rdf;PlainLiteral">Kategorie für selbstständige
monetäre Versorgungsleistungen. Zu diesen gehört beispielsweise die
Beschädigtenrente.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>#Versorgungsleistung</IRI>
            <Literal datatypeIRI="&rdf;PlainLiteral">Superklasse die eine
Versorgungsleistung nach dem KOVG repräsentiert.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>#Zulage</IRI>
            <Literal datatypeIRI="&rdf;PlainLiteral">Kategorie für
unselbstständige monetäre Versorgungsleistungen die nur zusätzlich zu einer Rente
zugesprochen werden dürfen. </Literal>
      </AnnotationAssertion>
</Ontology>
```

Anhang B.2. kovg-norms.owl

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#" xml:base="http://js.at/diss/kovg-
norms"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
      ontologyIRI="http://js.at/diss/kovg-norms">
      <Prefix name="" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
      <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
      <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
      <Import>http://js.at/diss/kovg</Import>
      <Import>http://js.at/diss/lkif-core-light</Import>
      <Declaration>
            <Class IRI="#Art11_1_GC" />
      </Declaration>
      <Declaration>
            <Class IRI="#Art11_1_Permission" />
      </Declaration>
      <Declaration>
            <Class IRI="#Art11_2_GC" />
```

```
	</Declaration>
	<Declaration>
		<Class IRI="#Art11_2_Permission" />
	</Declaration>
	<Declaration>
		<Class IRI="#Art11_3_GC" />
	</Declaration>
	<Declaration>
		<Class IRI="#Art11_3_Permission" />
	</Declaration>
	<Declaration>
		<Class IRI="#Art16_GC" />
	</Declaration>
	<Declaration>
		<Class IRI="#Art16_Permission" />
	</Declaration>
	<Declaration>
		<Class IRI="#Art18_GC" />
	</Declaration>
	<Declaration>
		<Class IRI="#Art18_Permission" />
	</Declaration>
	<Declaration>
		<Class IRI="#Art19_GC" />
	</Declaration>
	<Declaration>
		<Class IRI="#Art19_Permission" />
	</Declaration>
	<Declaration>
		<Class IRI="#Defined" />
	</Declaration>
	<Declaration>
		<Class IRI="#Definition" />
	</Declaration>
	<Declaration>
		<Class IRI="#Sachverhaltsbeschreibung" />
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#defined_by" />
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#defines" />
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#hatAnspruchAuf" />
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#rechtsgrundlage" />
	</Declaration>
	<Declaration>
		<ObjectProperty IRI="#rechtsgrundlageVon" />
	</Declaration>
	<Declaration>
		<NamedIndividual IRI="http://js.at/diss/kovg#Art11_1_Grundrente" />
	</Declaration>
	<Declaration>
		<NamedIndividual
IRI="http://js.at/diss/kovg#Art11_2_ErhöhungGrundrente" />
	</Declaration>
	<Declaration>
```

```
            <NamedIndividual IRI="http://js.at/diss/kovg#Art16_Familienzulage" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="http://js.at/diss/kovg#Art18_Pflegezulage" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="http://js.at/diss/kovg#Art19_Blindenzulage" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Art11_1" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Art11_2" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Art11_3" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Art16" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Art18" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Art19" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Art9" />
      </Declaration>
      <Declaration>
            <NamedIndividual
                  IRI="http://js.at/diss/kovg/usecase-
1#Art11_3_ErhöhungGrundrente" />
      </Declaration>
      <EquivalentClasses>
            <Class IRI="http://js.at/diss/kovg#Beschädigter" />
            <Class IRI="#Art11_1_GC" />
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://js.at/diss/kovg#Beschädigter" />
            <DataSomeValuesFrom>
                  <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
                  <DatatypeRestriction>
                        <Datatype abbreviatedIRI="xsd:short" />
                        <FacetRestriction facet="&xsd;minInclusive">
                              <Literal datatypeIRI="&xsd;integer">20</Literal>
                        </FacetRestriction>
                  </DatatypeRestriction>
            </DataSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://js.at/diss/kovg#Erwerbsunfähiger" />
            <DataSomeValuesFrom>
                  <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
                  <DatatypeRestriction>
                        <Datatype abbreviatedIRI="xsd:short" />
                        <FacetRestriction facet="&xsd;minInclusive">
                              <Literal datatypeIRI="&xsd;integer">90</Literal>
                        </FacetRestriction>
                  </DatatypeRestriction>
```

```
            </DataSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://js.at/diss/kovg#Schwerbeschädigter" />
            <DataSomeValuesFrom>
                  <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
                  <DatatypeRestriction>
                        <Datatype abbreviatedIRI="xsd:short" />
                        <FacetRestriction facet="&xsd;minInclusive">
                              <Literal datatypeIRI="&xsd;integer">50</Literal>
                        </FacetRestriction>
                  </DatatypeRestriction>
            </DataSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art11_1_Permission" />
            <ObjectOneOf>
                  <NamedIndividual IRI="#Art11_1" />
            </ObjectOneOf>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art11_1_Permission" />
            <ObjectHasValue>
                  <ObjectProperty IRI="#rechtsgrundlageVon" />
                  <NamedIndividual
IRI="http://js.at/diss/kovg#Art11_1_Grundrente" />
            </ObjectHasValue>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art11_2_GC" />
            <ObjectIntersectionOf>
                  <Class IRI="http://js.at/diss/kovg#Schwerbeschädigter" />
                  <ObjectUnionOf>
                        <ObjectIntersectionOf>
                              <ObjectHasValue>
                                    <ObjectProperty
IRI="http://js.at/diss/kovg#hatGeschlecht" />
                                    <NamedIndividual
IRI="http://js.at/diss/kovg#Männlich" />
                              </ObjectHasValue>
                              <DataSomeValuesFrom>
                                    <DataProperty
IRI="http://js.at/diss/kovg#hatAlter" />
                                    <DatatypeRestriction>
                                          <Datatype abbreviatedIRI="xsd:short"
/>
                                          <FacetRestriction
facet="&xsd;minInclusive">
                                                <Literal
datatypeIRI="&xsd;integer">60</Literal>
                                          </FacetRestriction>
                                    </DatatypeRestriction>
                              </DataSomeValuesFrom>
                        </ObjectIntersectionOf>
                        <ObjectIntersectionOf>
                              <ObjectHasValue>
                                    <ObjectProperty
IRI="http://js.at/diss/kovg#hatGeschlecht" />
                                    <NamedIndividual
IRI="http://js.at/diss/kovg#Weiblich" />
```

```
                                    </ObjectHasValue>
                                    <DataSomeValuesFrom>
                                          <DataProperty
IRI="http://js.at/diss/kovg#hatAlter" />
                                          <DatatypeRestriction>
                                                <Datatype abbreviatedIRI="xsd:short"
/>
                                                <FacetRestriction
facet="&xsd;minInclusive">
                                                      <Literal
datatypeIRI="&xsd;integer">55</Literal>
                                                </FacetRestriction>
                                          </DatatypeRestriction>
                                    </DataSomeValuesFrom>
                              </ObjectIntersectionOf>
                        </ObjectUnionOf>
                  </ObjectIntersectionOf>
            </EquivalentClasses>
            <EquivalentClasses>
                  <Class IRI="#Art11_2_Permission" />
                  <ObjectOneOf>
                        <NamedIndividual IRI="#Art11_2" />
                  </ObjectOneOf>
            </EquivalentClasses>
            <EquivalentClasses>
                  <Class IRI="#Art11_2_Permission" />
                  <ObjectHasValue>
                        <ObjectProperty IRI="#rechtsgrundlageVon" />
                        <NamedIndividual
IRI="http://js.at/diss/kovg#Art11_2_ErhöhungGrundrente" />
                  </ObjectHasValue>
            </EquivalentClasses>
            <EquivalentClasses>
                  <Class IRI="#Art11_3_GC" />
                  <ObjectIntersectionOf>
                        <Class IRI="http://js.at/diss/kovg#Schwerbeschädigter" />
                        <DataSomeValuesFrom>
                              <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
                              <DatatypeRestriction>
                                    <Datatype abbreviatedIRI="xsd:short" />
                                    <FacetRestriction facet="&xsd;minInclusive">
                                          <Literal
datatypeIRI="&xsd;integer">65</Literal>
                                    </FacetRestriction>
                              </DatatypeRestriction>
                        </DataSomeValuesFrom>
                  </ObjectIntersectionOf>
            </EquivalentClasses>
            <EquivalentClasses>
                  <Class IRI="#Art11_3_Permission" />
                  <ObjectOneOf>
                        <NamedIndividual IRI="#Art11_3" />
                  </ObjectOneOf>
            </EquivalentClasses>
            <EquivalentClasses>
                  <Class IRI="#Art11_3_Permission" />
                  <ObjectHasValue>
                        <ObjectProperty IRI="#rechtsgrundlageVon" />
                        <NamedIndividual
```

```
                        IRI="http://js.at/diss/kovg/usecase-
1#Art11_3_ErhöhungGrundrente" />
            </ObjectHasValue>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art16_GC" />
            <ObjectIntersectionOf>
                  <Class IRI="http://js.at/diss/kovg#Schwerbeschädigter" />
                  <ObjectMinCardinality cardinality="1">
                        <ObjectProperty
IRI="http://js.at/diss/kovg#hatAngehörigen" />
                        <Class IRI="http://js.at/diss/kovg#Angehöriger" />
                  </ObjectMinCardinality>
            </ObjectIntersectionOf>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art16_Permission" />
            <ObjectOneOf>
                  <NamedIndividual IRI="#Art16" />
            </ObjectOneOf>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art16_Permission" />
            <ObjectHasValue>
                  <ObjectProperty IRI="#rechtsgrundlageVon" />
                  <NamedIndividual
IRI="http://js.at/diss/kovg#Art16_Familienzulage" />
            </ObjectHasValue>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art18_GC" />
            <ObjectIntersectionOf>
                  <ObjectSomeValuesFrom>
                        <ObjectProperty
IRI="http://js.at/diss/kovg#hatPflegestufe" />
                        <Class IRI="http://js.at/diss/kovg#Pflegestufe" />
                  </ObjectSomeValuesFrom>
                  <ObjectHasValue>
                        <ObjectProperty IRI="#hatAnspruchAuf" />
                        <NamedIndividual
IRI="http://js.at/diss/kovg#Art11_1_Grundrente" />
                  </ObjectHasValue>
            </ObjectIntersectionOf>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art18_Permission" />
            <ObjectOneOf>
                  <NamedIndividual IRI="#Art18" />
            </ObjectOneOf>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art18_Permission" />
            <ObjectHasValue>
                  <ObjectProperty IRI="#rechtsgrundlageVon" />
                  <NamedIndividual
IRI="http://js.at/diss/kovg#Art18_Pflegezulage" />
            </ObjectHasValue>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art19_GC" />
```

```
            <ObjectIntersectionOf>
                  <ObjectSomeValuesFrom>
                        <ObjectProperty
IRI="http://js.at/diss/kovg#hatPflegestufe" />
                        <Class IRI="http://js.at/diss/kovg#Pflegestufe" />
                  </ObjectSomeValuesFrom>
                  <ObjectHasValue>
                        <ObjectProperty IRI="#hatAnspruchAuf" />
                        <NamedIndividual
IRI="http://js.at/diss/kovg#Art11_1_Grundrente" />
                  </ObjectHasValue>
                  <DataHasValue>
                        <DataProperty IRI="http://js.at/diss/kovg#istBlind" />
                        <Literal datatypeIRI="&xsd;boolean">true</Literal>
                  </DataHasValue>
            </ObjectIntersectionOf>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art19_Permission" />
            <ObjectOneOf>
                  <NamedIndividual IRI="#Art19" />
            </ObjectOneOf>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Art19_Permission" />
            <ObjectHasValue>
                  <ObjectProperty IRI="#rechtsgrundlageVon" />
                  <NamedIndividual
IRI="http://js.at/diss/kovg#Art19_Blindenzulage" />
            </ObjectHasValue>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Defined" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty IRI="#defined_by" />
                  <Class IRI="#Definition" />
            </ObjectSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="#Definition" />
            <ObjectIntersectionOf>
                  <ObjectSomeValuesFrom>
                        <ObjectProperty IRI="#defines" />
                        <Class IRI="#Defined" />
                  </ObjectSomeValuesFrom>
                  <ObjectAllValuesFrom>
                        <ObjectProperty IRI="#defines" />
                        <Class IRI="#Defined" />
                  </ObjectAllValuesFrom>
            </ObjectIntersectionOf>
      </EquivalentClasses>
      <SubClassOf>
            <Class IRI="http://js.at/diss/kovg#Beschädigter" />
            <ObjectHasValue>
                  <ObjectProperty IRI="#defined_by" />
                  <NamedIndividual IRI="#Art9" />
            </ObjectHasValue>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://js.at/diss/kovg#Erwerbsunfähiger" />
```

```
            <ObjectHasValue>
                  <ObjectProperty IRI="#defined_by" />
                  <NamedIndividual IRI="#Art9" />
            </ObjectHasValue>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://js.at/diss/kovg#Schwerbeschädigter" />
            <ObjectHasValue>
                  <ObjectProperty IRI="#defined_by" />
                  <NamedIndividual IRI="#Art9" />
            </ObjectHasValue>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_1_GC" />
            <Class IRI="#Sachverhaltsbeschreibung" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_1_GC" />
            <ObjectHasValue>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
                  <NamedIndividual IRI="#Art11_1" />
            </ObjectHasValue>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_1_Permission" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_1_Permission" />
            <ObjectAllValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
                  <Class IRI="#Art11_1_GC" />
            </ObjectAllValuesFrom>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_2_GC" />
            <Class IRI="#Sachverhaltsbeschreibung" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_2_GC" />
            <ObjectHasValue>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
                  <NamedIndividual IRI="#Art11_2" />
            </ObjectHasValue>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_2_Permission" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_2_Permission" />
            <ObjectAllValuesFrom>
```

```
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
                  <Class IRI="#Art11_2_GC" />
            </ObjectAllValuesFrom>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_3_GC" />
            <Class IRI="#Sachverhaltsbeschreibung" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_3_GC" />
            <ObjectHasValue>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
                  <NamedIndividual IRI="#Art11_3" />
            </ObjectHasValue>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_3_Permission" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art11_3_Permission" />
            <ObjectAllValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
                  <Class IRI="#Art11_3_GC" />
            </ObjectAllValuesFrom>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art16_GC" />
            <Class IRI="#Sachverhaltsbeschreibung" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art16_GC" />
            <ObjectHasValue>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
                  <NamedIndividual IRI="#Art16" />
            </ObjectHasValue>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art16_Permission" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="#Art16_Permission" />
            <ObjectAllValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
                  <Class IRI="#Art16_GC" />
            </ObjectAllValuesFrom>
      </SubClassOf>
```

```
	<SubClassOf>
		<Class IRI="#Art18_GC" />
		<Class IRI="#Sachverhaltsbeschreibung" />
	</SubClassOf>
	<SubClassOf>
		<Class IRI="#Art18_GC" />
		<ObjectHasValue>
			<ObjectProperty
				IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
			<NamedIndividual IRI="#Art18" />
		</ObjectHasValue>
	</SubClassOf>
	<SubClassOf>
		<Class IRI="#Art18_Permission" />
		<Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
	</SubClassOf>
	<SubClassOf>
		<Class IRI="#Art18_Permission" />
		<ObjectAllValuesFrom>
			<ObjectProperty
				IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
			<Class IRI="#Art18_GC" />
		</ObjectAllValuesFrom>
	</SubClassOf>
	<SubClassOf>
		<Class IRI="#Art19_GC" />
		<Class IRI="#Sachverhaltsbeschreibung" />
	</SubClassOf>
	<SubClassOf>
		<Class IRI="#Art19_GC" />
		<ObjectHasValue>
			<ObjectProperty
				IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
			<NamedIndividual IRI="#Art19" />
		</ObjectHasValue>
	</SubClassOf>
	<SubClassOf>
		<Class IRI="#Art19_Permission" />
		<Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
	</SubClassOf>
	<SubClassOf>
		<Class IRI="#Art19_Permission" />
		<ObjectAllValuesFrom>
			<ObjectProperty
				IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
			<Class IRI="#Art19_GC" />
		</ObjectAllValuesFrom>
	</SubClassOf>
	<SubClassOf>
		<Class IRI="#Defined" />
		<Class
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
	</SubClassOf>
```

```
	<SubClassOf>
		<Class IRI="#Definition" />
		<Class IRI="http://www.estrellaproject.org/lkif-core/norm.owl#Norm"
/>
	</SubClassOf>
	<DisjointClasses>
		<Class IRI="http://js.at/diss/kovg#Angehöriger" />
		<Class IRI="http://js.at/diss/kovg#Geschlecht" />
		<Class IRI="http://js.at/diss/kovg#Pflegestufe" />
		<Class IRI="http://js.at/diss/kovg#Versorgungsleistung" />
		<Class IRI="#Sachverhaltsbeschreibung" />
	</DisjointClasses>
	<ClassAssertion>
		<Class IRI="http://js.at/diss/kovg#Beschädigtenrente" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Art11_1_Grundrente" />
	</ClassAssertion>
	<ClassAssertion>
		<Class IRI="http://js.at/diss/kovg#Beschädigtenrente" />
		<NamedIndividual
IRI="http://js.at/diss/kovg#Art11_2_ErhöhungGrundrente" />
	</ClassAssertion>
	<ClassAssertion>
		<Class IRI="http://js.at/diss/kovg#Familienzulage" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Art16_Familienzulage" />
	</ClassAssertion>
	<ClassAssertion>
		<Class IRI="http://js.at/diss/kovg#Pflegezulage" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Art18_Pflegezulage" />
	</ClassAssertion>
	<ClassAssertion>
		<Class IRI="http://js.at/diss/kovg#Blindenzulage" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Art19_Blindenzulage" />
	</ClassAssertion>
	<ClassAssertion>
		<Class IRI="http://js.at/diss/kovg#Beschädigtenrente" />
		<NamedIndividual
			IRI="http://js.at/diss/kovg/usecase-
1#Art11_3_ErhöhungGrundrente" />
	</ClassAssertion>
	<DifferentIndividuals>
		<NamedIndividual IRI="#Art11_1" />
		<NamedIndividual IRI="#Art11_2" />
		<NamedIndividual IRI="#Art11_3" />
		<NamedIndividual IRI="#Art16" />
		<NamedIndividual IRI="#Art18" />
		<NamedIndividual IRI="#Art19" />
		<NamedIndividual IRI="#Art9" />
	</DifferentIndividuals>
	<SubObjectPropertyOf>
		<ObjectProperty IRI="#defined_by" />
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualified_by" />
	</SubObjectPropertyOf>
	<SubObjectPropertyOf>
		<ObjectProperty IRI="#defines" />
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualifies" />
	</SubObjectPropertyOf>
```

```
	<InverseObjectProperties>
		<ObjectProperty IRI="#defined_by" />
		<ObjectProperty IRI="#defines" />
	</InverseObjectProperties>
	<InverseObjectProperties>
		<ObjectProperty IRI="#rechtsgrundlage" />
		<ObjectProperty IRI="#rechtsgrundlageVon" />
	</InverseObjectProperties>
	<ObjectPropertyDomain>
		<ObjectProperty IRI="#hatAnspruchAuf" />
		<Class IRI="http://js.at/diss/kovg#Antragsteller" />
	</ObjectPropertyDomain>
	<ObjectPropertyDomain>
		<ObjectProperty IRI="#rechtsgrundlageVon" />
		<Class IRI="http://www.estrellaproject.org/lkif-core/norm.owl#Norm"
/>
	</ObjectPropertyDomain>
	<ObjectPropertyRange>
		<ObjectProperty IRI="#defines" />
		<Class IRI="#Defined" />
	</ObjectPropertyRange>
	<ObjectPropertyRange>
		<ObjectProperty IRI="#hatAnspruchAuf" />
		<Class IRI="http://js.at/diss/kovg#Versorgungsleistung" />
	</ObjectPropertyRange>
	<ObjectPropertyRange>
		<ObjectProperty IRI="#rechtsgrundlageVon" />
		<Class IRI="http://js.at/diss/kovg#Versorgungsleistung" />
	</ObjectPropertyRange>
	<SubObjectPropertyOf>
		<ObjectPropertyChain>
			<ObjectProperty
				IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
			<ObjectProperty IRI="#rechtsgrundlageVon" />
		</ObjectPropertyChain>
		<ObjectProperty IRI="#hatAnspruchAuf" />
	</SubObjectPropertyOf>
	<AnnotationAssertion>
		<AnnotationProperty abbreviatedIRI="rdfs:comment" />
		<IRI>http://js.at/diss/kovg#Art11_1_Grundrente</IRI>
		<Literal datatypeIRI="&rdf;PlainLiteral">§ 11 Abs. 1</Literal>
	</AnnotationAssertion>
	<AnnotationAssertion>
		<AnnotationProperty abbreviatedIRI="rdfs:comment" />
		<IRI>#Art11_1_GC</IRI>
		<Literal datatypeIRI="&rdf;PlainLiteral">Sachverhalt für Gewährung
der Grundrente.
		</Literal>
	</AnnotationAssertion>
	<AnnotationAssertion>
		<AnnotationProperty abbreviatedIRI="rdfs:comment" />
		<IRI>#Art18</IRI>
		<Literal datatypeIRI="&rdf;PlainLiteral">Pflegezulage gemäß §
18.</Literal>
	</AnnotationAssertion>
	<AnnotationAssertion>
		<AnnotationProperty abbreviatedIRI="rdfs:comment" />
		<IRI>#Art9</IRI>
		<Literal datatypeIRI="&rdf;PlainLiteral">Definiert Beschädigten,
LXXX
```

```
                  Schwerbeschädigten usw. Beschädigter ist im Gesetz in Wahrheit
an einer anderen Stelle definiert.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>#Sachverhaltsbeschreibung</IRI>
            <Literal datatypeIRI="&rdf;PlainLiteral">Klasse die eine abstrakte
Beschreibung des Sachverhalts repräsentiert. Analog Generic Case von Valente (GC).
            </Literal>
      </AnnotationAssertion>
</Ontology>
```

Anhang B.3. lkif-core-light.owl

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#" xml:base="http://js.at/diss/lkif-
core-light"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
      ontologyIRI="http://js.at/diss/lkif-core-light">
      <Prefix name="" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
      <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
      <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
      <Declaration>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification" />
      </Declaration>
      <Declaration>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
      </Declaration>
      <Declaration>
            <Class IRI="http://www.estrellaproject.org/lkif-core/norm.owl#Norm"
/>
      </Declaration>
      <Declaration>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
      </Declaration>
      <Declaration>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obligation" />
      </Declaration>
```

```
	<Declaration>
		<Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
	</Declaration>
	<Declaration>
		<Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
	</Declaration>
	<Declaration>
		<Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Prohibition" />
	</Declaration>
	<Declaration>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
	</Declaration>
	<Declaration>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-core/norm.owl#allows"
/>
	</Declaration>
	<Declaration>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallowed_by" />
	</Declaration>
	<Declaration>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallows" />
	</Declaration>
	<EquivalentClasses>
		<Class
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification" />
		<ObjectSomeValuesFrom>
			<ObjectProperty
				IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualifies" />
			<Class
				IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
		</ObjectSomeValuesFrom>
	</EquivalentClasses>
	<EquivalentClasses>
		<Class
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
		<ObjectSomeValuesFrom>
			<ObjectProperty
				IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualified_by" />
			<Class
				IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification" />
		</ObjectSomeValuesFrom>
	</EquivalentClasses>
	<EquivalentClasses>
```

```
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Allowed" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
                  <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
            </ObjectSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Allowed" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#normatively_equivalent_or_worse" />
                  <Class
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
            </ObjectSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Disallowed" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallowed_by" />
                  <Class
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Prohibition" />
            </ObjectSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://www.estrellaproject.org/lkif-core/norm.owl#Norm"
/>
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualifies" />
                  <Class
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
            </ObjectSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualified_by" />
                  <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Norm" />
            </ObjectSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
```

```
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obligation" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Prohibition" />
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
                  <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obligation" />
            </ObjectSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#normatively_strictly_worse" />
                  <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Disallowed" />
            </ObjectSomeValuesFrom>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
            <ObjectIntersectionOf>
                  <ObjectSomeValuesFrom>
                        <ObjectProperty
                              IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
                        <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Allowed" />
                  </ObjectSomeValuesFrom>
                  <ObjectAllValuesFrom>
                        <ObjectProperty
                              IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
                        <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Allowed" />
                  </ObjectAllValuesFrom>
            </ObjectIntersectionOf>
      </EquivalentClasses>
      <EquivalentClasses>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Prohibition" />
            <ObjectIntersectionOf>
                  <ObjectSomeValuesFrom>
                        <ObjectProperty
                              IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
                        <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
                  </ObjectSomeValuesFrom>
                  <ObjectSomeValuesFrom>
                        <ObjectProperty
```

```
                              IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallows" />
                        <Class
                              IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Disallowed" />
                  </ObjectSomeValuesFrom>
                  <ObjectAllValuesFrom>
                        <ObjectProperty
                              IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allows" />
                        <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
                  </ObjectAllValuesFrom>
                  <ObjectAllValuesFrom>
                        <ObjectProperty
                              IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallows" />
                        <Class
                              IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Disallowed" />
                  </ObjectAllValuesFrom>
            </ObjectIntersectionOf>
      </EquivalentClasses>
      <SubClassOf>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-core/lkif-
top.owl#Mental_Object" />
      </SubClassOf>
      <SubClassOf>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification" />
            <ObjectAllValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualifies" />
                  <Class
                        IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
            </ObjectAllValuesFrom>
      </SubClassOf>
      <SubClassOf>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
            <Class abbreviatedIRI="owl:Thing" />
      </SubClassOf>
      <SubClassOf>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualitatively_comparable" />
                  <Class
```

```
                        IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
            </ObjectSomeValuesFrom>
      </SubClassOf>
      <SubClassOf>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
            <ObjectAllValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualitatively_comparable" />
                  <Class
                        IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
            </ObjectAllValuesFrom>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Allowed" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Disallowed" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://www.estrellaproject.org/lkif-core/norm.owl#Norm"
/>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification" />
      </SubClassOf>
      <SubClassOf>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
      </SubClassOf>
      <SubClassOf>
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
            <ObjectSomeValuesFrom>
                  <ObjectProperty
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#normatively_comparable" />
                  <Class
                        IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
            </ObjectSomeValuesFrom>
      </SubClassOf>
      <SubClassOf>
```

```
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obligation" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Allowed" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
            <Class IRI="http://www.estrellaproject.org/lkif-core/norm.owl#Norm"
/>
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Prohibition" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Prohibition" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Permission" />
      </SubClassOf>
      <SubClassOf>
            <Class IRI="http://www.estrellaproject.org/lkif-core/norm.owl#Right"
/>
            <Class IRI="http://www.estrellaproject.org/lkif-core/norm.owl#Norm"
/>
      </SubClassOf>
      <SubObjectPropertyOf>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualified_by" />
      </SubObjectPropertyOf>
      <SubObjectPropertyOf>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-core/norm.owl#allows"
/>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualifies" />
      </SubObjectPropertyOf>
      <SubObjectPropertyOf>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#commanded_by" />
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
      </SubObjectPropertyOf>
```

```
	<SubObjectPropertyOf>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#commands" />
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-core/norm.owl#allows"
/>
	</SubObjectPropertyOf>
	<SubObjectPropertyOf>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallowed_by" />
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualified_by" />
	</SubObjectPropertyOf>
	<SubObjectPropertyOf>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallows" />
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualifies" />
	</SubObjectPropertyOf>
	<InverseObjectProperties>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-core/norm.owl#allows"
/>
	</InverseObjectProperties>
	<InverseObjectProperties>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallowed_by" />
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallows" />
	</InverseObjectProperties>
	<ObjectPropertyDomain>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualified_by" />
		<Class
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
	</ObjectPropertyDomain>
	<ObjectPropertyDomain>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualifies" />
		<Class
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification" />
	</ObjectPropertyDomain>
	<ObjectPropertyDomain>
		<ObjectProperty
			IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualitatively_comparable" />
```

```
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
      </ObjectPropertyDomain>
      <ObjectPropertyDomain>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#allowed_by" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Allowed" />
      </ObjectPropertyDomain>
      <ObjectPropertyDomain>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#commanded_by" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
      </ObjectPropertyDomain>
      <ObjectPropertyDomain>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallowed_by" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Disallowed" />
      </ObjectPropertyDomain>
      <ObjectPropertyDomain>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#normatively_comparable" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
      </ObjectPropertyDomain>
      <ObjectPropertyDomain>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#normatively_strictly_worse" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
      </ObjectPropertyDomain>
      <ObjectPropertyRange>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualified_by" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualifies" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#qualitatively_comparable" />
```

```
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-core/norm.owl#allows"
/>
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Allowed" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#commands" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#disallows" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Disallowed" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#normatively_comparable" />
            <Class
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified" />
      </ObjectPropertyRange>
      <ObjectPropertyRange>
            <ObjectProperty
                  IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#normatively_strictly_better" />
            <Class IRI="http://www.estrellaproject.org/lkif-
core/norm.owl#Obliged" />
      </ObjectPropertyRange>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-
core/expression.owl#Qualification
            </IRI>
            <Literal datatypeIRI="&xsd;string">A qualification expresses e.g. a
                  judgment. The thing qualified by the qualification is
comparable to something else.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-
core/expression.owl#Qualified
            </IRI>
            <Literal datatypeIRI="&xsd;string">Something that is qualified by
some qualification</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-core/norm.owl#Norm</IRI>
```

```
            <Literal datatypeIRI="&xsd;string">A norm is a kind of Qualification.
A qualification which normatively qualifies some thing (i.e. some normatively
qualified): i.e. a qualification which allows or disallows some thing.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-
core/norm.owl#Normatively_Qualified
            </IRI>
            <Literal datatypeIRI="&xsd;string">Some thing which is qualified
(allowed, disallowed) by a norm, i.e. a norm applies to the thing. Taking the
principle of deontic choice to mean that the utterer of a normative statement
intends to influence choices made by the addressee of the statement, the qualified
thing should is comparable to some alternative. Note that Qualified does not
partition into Allowed and Disallowed. Firstly, things can be Allowed by one Norm
and Disallowed by another one. Secondly, the logical complement of the thing
allowed by a permission is qualified (as worse or equal than the thing allowed),
but neither allowed nor disallowed.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-core/norm.owl#Obligation
            </IRI>
            <Literal datatypeIRI="&xsd;string">See prohibition</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-core/norm.owl#Obliged</IRI>
            <Literal datatypeIRI="&xsd;string">Is allowed by an Obligation, i.e.
the alternative is strictly_worse according to the Obligation. Alternative labels:
directed, commanded.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-core/norm.owl#Permission
            </IRI>
            <Literal datatypeIRI="&xsd;string">Permission allows propositions, to
which therefore the predication Allowed applies. The thing(s) allowed is/are a
subset of the thing(s) qualified by the permission. If S then P(A) means that S is
qualified and S and A is allowed.</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-core/norm.owl#Prohibition
            </IRI>
            <Literal datatypeIRI="&xsd;string">Prohibition obliges/allows
thing(s), to which therefore the predication Obliged applies, and disallows
thing(s), to which therefore the predication Disallowed applies. The union of the
thing(s) allowed and the thing(s) disallowed is qualified by the prohibition. The
things allowed and disallowed by the prohibition are disjoint (and completely
partition the space of things qualified). If S then O(A) means that S is
qualified, S and A is obliged/allowed, and S and not A is disallowed. If S then
F(A) means that S is qualified, S and A is disallowed, and S and not A is
obliged/allowed. The difference between Obligation and Prohibition is in which
part of the partition of the qualified space is explicitly described by the
normative statement. Alternative label: directive
            </Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
```

```
            <IRI>http://www.estrellaproject.org/lkif-core/norm.owl#allowed_by
            </IRI>
            <Literal datatypeIRI="&xsd;string">Relates a qualified Allowed to the
norm allowing it</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-core/norm.owl#allows</IRI>
            <Literal datatypeIRI="&xsd;string">Relates a norm to the thing it
allows
            </Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-core/norm.owl#disallowed_by
            </IRI>
            <Literal datatypeIRI="&xsd;string">Relates a qualified Disallowed to
the norm disallowing it</Literal>
      </AnnotationAssertion>
      <AnnotationAssertion>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <IRI>http://www.estrellaproject.org/lkif-
core/norm.owl#disallows</IRI>
            <Literal datatypeIRI="&xsd;string">Relates a norm to the thing it
disallows
            </Literal>
      </AnnotationAssertion>
</Ontology>
```

Anhang B.4. usecase-1.owl

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
xml:base="http://js.at/diss/kovg/usecase-1"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
      ontologyIRI="http://js.at/diss/kovg/usecase-1">
      <Prefix name="" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
      <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
      <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
      <Import>http://js.at/diss/kovg-norms</Import>
      <Annotation>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <Literal datatypeIRI="&rdf;PlainLiteral">Ontologie mit Instanzen für
den Test der grundlegenden Klassifizierung und dem Anspruch auf Grundrente.
            </Literal>
```

```
</Annotation>
<Declaration>
	<NamedIndividual IRI="#Adam" />
</Declaration>
<Declaration>
	<NamedIndividual IRI="#Hans" />
</Declaration>
<Declaration>
	<NamedIndividual IRI="#Johannes" />
</Declaration>
<Declaration>
	<NamedIndividual IRI="#Martin" />
</Declaration>
<DifferentIndividuals>
	<NamedIndividual IRI="#Adam" />
	<NamedIndividual IRI="#Hans" />
	<NamedIndividual IRI="#Johannes" />
	<NamedIndividual IRI="#Martin" />
</DifferentIndividuals>
<ObjectPropertyAssertion>
	<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
	<NamedIndividual IRI="#Hans" />
	<NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
	<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
	<NamedIndividual IRI="#Johannes" />
	<NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
	<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
	<NamedIndividual IRI="#Martin" />
	<NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
</ObjectPropertyAssertion>
<DataPropertyAssertion>
	<DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
	<NamedIndividual IRI="#Adam" />
	<Literal datatypeIRI="&xsd;short">10</Literal>
</DataPropertyAssertion>
<DataPropertyAssertion>
	<DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
	<NamedIndividual IRI="#Hans" />
	<Literal datatypeIRI="&xsd;short">30</Literal>
</DataPropertyAssertion>
<DataPropertyAssertion>
	<DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
	<NamedIndividual IRI="#Hans" />
	<Literal datatypeIRI="&xsd;short">30</Literal>
</DataPropertyAssertion>
<DataPropertyAssertion>
	<DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
	<NamedIndividual IRI="#Johannes" />
	<Literal datatypeIRI="&xsd;short">20</Literal>
</DataPropertyAssertion>
<DataPropertyAssertion>
	<DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
	<NamedIndividual IRI="#Johannes" />
	<Literal datatypeIRI="&xsd;short">90</Literal>
</DataPropertyAssertion>
<DataPropertyAssertion>
```

```
            <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
            <NamedIndividual IRI="#Martin" />
            <Literal datatypeIRI="&xsd;short">65</Literal>
      </DataPropertyAssertion>
      <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
            <NamedIndividual IRI="#Martin" />
            <Literal datatypeIRI="&xsd;short">60</Literal>
      </DataPropertyAssertion>
</Ontology>
```

Anhang B.5. usecase-2.owl

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
xml:base="http://js.at/diss/kovg/usecase-2"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
      ontologyIRI="http://js.at/diss/kovg/usecase-2">
      <Prefix name="" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
      <Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
      <Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
      <Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
      <Import>http://js.at/diss/kovg-norms</Import>
      <Annotation>
            <AnnotationProperty abbreviatedIRI="rdfs:comment" />
            <Literal datatypeIRI="&rdf;PlainLiteral">Ontologie für die
Demonstration der Klassifizierung von Schwerbeschädigten für den Anspruch auf
Erhöhung der Grundrente gemäß § 11 Abs. 2 und 3.</Literal>
      </Annotation>
      <Declaration>
            <NamedIndividual IRI="#Christina" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Johanna" />
      </Declaration>
      <Declaration>
            <NamedIndividual IRI="#Johannes" />
      </Declaration>
      <DifferentIndividuals>
            <NamedIndividual IRI="#Christina" />
            <NamedIndividual IRI="#Johanna" />
            <NamedIndividual IRI="#Johannes" />
      </DifferentIndividuals>
      <ObjectPropertyAssertion>
            <ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
            <NamedIndividual IRI="#Christina" />
```

```
            <NamedIndividual IRI="http://js.at/diss/kovg#Weiblich" />
        </ObjectPropertyAssertion>
        <ObjectPropertyAssertion>
            <ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
            <NamedIndividual IRI="#Johanna" />
            <NamedIndividual IRI="http://js.at/diss/kovg#Weiblich" />
        </ObjectPropertyAssertion>
        <ObjectPropertyAssertion>
            <ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
            <NamedIndividual IRI="#Johannes" />
            <NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
        </ObjectPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
            <NamedIndividual IRI="#Christina" />
            <Literal datatypeIRI="&xsd;short">70</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
            <NamedIndividual IRI="#Christina" />
            <Literal datatypeIRI="&xsd;short">60</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
            <NamedIndividual IRI="#Johanna" />
            <Literal datatypeIRI="&xsd;short">56</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
            <NamedIndividual IRI="#Johanna" />
            <Literal datatypeIRI="&xsd;short">60</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
            <NamedIndividual IRI="#Johannes" />
            <Literal datatypeIRI="&xsd;short">56</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
            <NamedIndividual IRI="#Johannes" />
            <Literal datatypeIRI="&xsd;short">60</Literal>
        </DataPropertyAssertion>
</Ontology>
```

Anhang B.6. usecase-3.owl

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
xml:base="http://js.at/diss/kovg/usecase-3"
```

```
	xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
	xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
	ontologyIRI="http://js.at/diss/kovg/usecase-3">
	<Prefix name="" IRI="http://www.w3.org/2002/07/owl#" />
	<Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
	<Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
	<Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
	<Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
	<Import>http://js.at/diss/kovg-norms</Import>
	<Annotation>
		<AnnotationProperty abbreviatedIRI="rdfs:comment" />
		<Literal datatypeIRI="&rdf;PlainLiteral">Demonstriert den Anspruch
auf Familienzulage sofern der Antragsteller zumindest einen Familienangehörigen
hat.</Literal>
	</Annotation>
	<Declaration>
		<NamedIndividual IRI="#Christina" />
	</Declaration>
	<Declaration>
		<NamedIndividual IRI="#Johannes" />
	</Declaration>
	<Declaration>
		<NamedIndividual IRI="#Robert" />
	</Declaration>
	<Declaration>
		<NamedIndividual IRI="http://js.at/diss/kovg/usecse-3#Lukas" />
	</Declaration>
	<DifferentIndividuals>
		<NamedIndividual IRI="#Christina" />
		<NamedIndividual IRI="#Johannes" />
		<NamedIndividual IRI="#Robert" />
		<NamedIndividual IRI="http://js.at/diss/kovg/usecse-3#Lukas" />
	</DifferentIndividuals>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
		<NamedIndividual IRI="#Christina" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Weiblich" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatEhegatten" />
		<NamedIndividual IRI="#Johannes" />
		<NamedIndividual IRI="#Christina" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
		<NamedIndividual IRI="#Johannes" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatKind" />
		<NamedIndividual IRI="#Johannes" />
		<NamedIndividual IRI="http://js.at/diss/kovg/usecse-3#Lukas" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
		<NamedIndividual IRI="#Robert" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
	</ObjectPropertyAssertion>
```

```
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
		<NamedIndividual IRI="http://js.at/diss/kovg/usecse-3#Lukas" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
	</ObjectPropertyAssertion>
	<DataPropertyAssertion>
		<DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
		<NamedIndividual IRI="#Johannes" />
		<Literal datatypeIRI="&xsd;short">40</Literal>
	</DataPropertyAssertion>
	<DataPropertyAssertion>
		<DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
		<NamedIndividual IRI="#Johannes" />
		<Literal datatypeIRI="&xsd;short">50</Literal>
	</DataPropertyAssertion>
</Ontology>
```

Anhang B.7. usecase-4.owl

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>


<Ontology xmlns="http://www.w3.org/2002/07/owl#"
xml:base="http://js.at/diss/kovg/usecase-4"
	xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
	xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
	ontologyIRI="http://js.at/diss/kovg/usecase-4">
	<Prefix name="" IRI="http://www.w3.org/2002/07/owl#" />
	<Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
	<Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
	<Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
	<Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
	<Import>http://js.at/diss/kovg-norms</Import>
	<Annotation>
		<AnnotationProperty abbreviatedIRI="rdfs:comment" />
		<Literal datatypeIRI="&rdf;PlainLiteral">Ontologie die einen
Anwendungsfall der lex specialis anhand § 18 und § 19 zeigt.</Literal>
	</Annotation>
	<Declaration>
		<NamedIndividual IRI="#Larissa" />
	</Declaration>
	<Declaration>
		<NamedIndividual IRI="#Lars" />
	</Declaration>
	<DifferentIndividuals>
		<NamedIndividual IRI="#Larissa" />
		<NamedIndividual IRI="#Lars" />
	</DifferentIndividuals>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
```

```
            <NamedIndividual IRI="#Larissa" />
            <NamedIndividual IRI="http://js.at/diss/kovg#Weiblich" />
        </ObjectPropertyAssertion>
        <ObjectPropertyAssertion>
            <ObjectProperty IRI="http://js.at/diss/kovg#hatPflegestufe" />
            <NamedIndividual IRI="#Larissa" />
            <NamedIndividual IRI="http://js.at/diss/kovg#Pflegestufe_II" />
        </ObjectPropertyAssertion>
        <ObjectPropertyAssertion>
            <ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
            <NamedIndividual IRI="#Lars" />
            <NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
        </ObjectPropertyAssertion>
        <ObjectPropertyAssertion>
            <ObjectProperty IRI="http://js.at/diss/kovg#hatPflegestufe" />
            <NamedIndividual IRI="#Lars" />
            <NamedIndividual IRI="http://js.at/diss/kovg#Pflegestufe_III" />
        </ObjectPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
            <NamedIndividual IRI="#Larissa" />
            <Literal datatypeIRI="&xsd;short">20</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
            <NamedIndividual IRI="#Larissa" />
            <Literal datatypeIRI="&xsd;short">40</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#istBlind" />
            <NamedIndividual IRI="#Larissa" />
            <Literal datatypeIRI="&xsd;boolean">true</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
            <NamedIndividual IRI="#Lars" />
            <Literal datatypeIRI="&xsd;short">40</Literal>
        </DataPropertyAssertion>
        <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
            <NamedIndividual IRI="#Lars" />
            <Literal datatypeIRI="&xsd;short">30</Literal>
        </DataPropertyAssertion>
</Ontology>
```

Anhang B.8. usecase-5.owl

```
<?xml version="1.0"?>


<!DOCTYPE Ontology [
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY xml "http://www.w3.org/XML/1998/namespace" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>

<Ontology xmlns="http://www.w3.org/2002/07/owl#"
xml:base="http://js.at/diss/kovg/usecase-5"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
```

```
	xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:xml="http://www.w3.org/XML/1998/namespace"
	ontologyIRI="http://js.at/diss/kovg/usecase-5">
	<Prefix name="" IRI="http://www.w3.org/2002/07/owl#" />
	<Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#" />
	<Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#" />
	<Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#" />
	<Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#" />
	<Import>http://js.at/diss/kovg-norms</Import>
	<Annotation>
		<AnnotationProperty abbreviatedIRI="rdfs:comment" />
		<Literal datatypeIRI="&rdf;PlainLiteral">Ontologie die einen
vollständigen Anwendungsfall enthält. Der Antragsteller erhält Beschädigtenrente
mit Erhöhungen und zusätzliche Leistungen.</Literal>
	</Annotation>
	<Declaration>
		<NamedIndividual IRI="#Christina" />
	</Declaration>
	<Declaration>
		<NamedIndividual IRI="#Johanna" />
	</Declaration>
	<Declaration>
		<NamedIndividual IRI="#Johannes" />
	</Declaration>
	<DifferentIndividuals>
		<NamedIndividual IRI="#Christina" />
		<NamedIndividual IRI="#Johanna" />
		<NamedIndividual IRI="#Johannes" />
	</DifferentIndividuals>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
		<NamedIndividual IRI="#Christina" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Weiblich" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
		<NamedIndividual IRI="#Johanna" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Weiblich" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatEhegatten" />
		<NamedIndividual IRI="#Johannes" />
		<NamedIndividual IRI="#Christina" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatGeschlecht" />
		<NamedIndividual IRI="#Johannes" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Männlich" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatKind" />
		<NamedIndividual IRI="#Johannes" />
		<NamedIndividual IRI="#Johanna" />
	</ObjectPropertyAssertion>
	<ObjectPropertyAssertion>
		<ObjectProperty IRI="http://js.at/diss/kovg#hatPflegestufe" />
		<NamedIndividual IRI="#Johannes" />
		<NamedIndividual IRI="http://js.at/diss/kovg#Pflegestufe_III" />
	</ObjectPropertyAssertion>
	<DataPropertyAssertion>
```

```
            <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
            <NamedIndividual IRI="#Christina" />
            <Literal datatypeIRI="&xsd;short">55</Literal>
      </DataPropertyAssertion>
      <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatAlter" />
            <NamedIndividual IRI="#Johannes" />
            <Literal datatypeIRI="&xsd;short">62</Literal>
      </DataPropertyAssertion>
      <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#hatMdE" />
            <NamedIndividual IRI="#Johannes" />
            <Literal datatypeIRI="&xsd;short">60</Literal>
      </DataPropertyAssertion>
      <DataPropertyAssertion>
            <DataProperty IRI="http://js.at/diss/kovg#istBlind" />
            <NamedIndividual IRI="#Johannes" />
            <Literal datatypeIRI="&xsd;boolean">true</Literal>
      </DataPropertyAssertion>
</Ontology>
```

C