universität wien

MASTERARBEIT

Titel der Masterarbeit

„Konzeption und Umsetzung erweiterter Versionsverwaltungsverfahren für prozessorientierte Informationssysteme“

verfasst von

Ing. Kristof Böhmer, MSc

angestrebter akademischer Grad

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2013

Studienkennzahl lt. Studienblatt:	A 066 926
Studienrichtung lt. Studienblatt:	Wirtschaftsinformatik
Betreuerin:	Univ.-Prof. Dipl.-Math. Dr. Stefanie Rinderle-Ma

Diese Seite wurde absichtlich freigelassen.

Danksagung

Ich möchte meinen Eltern und Freunden sowie meiner Freundin für die mir entgegengebrachte Geduld und Unterstützung danken, ohne die die Realisierung dieser Arbeit nicht möglich gewesen wäre.

Außerdem möchte ich mich bei meiner Betreuerin, Frau Univ.-Prof. Dipl.-Math. Dr. Stefanie Rinderle-Ma, bedanken, welche mich bei der Erstellung dieser Arbeit mit interessanten Diskussionen sowie hilfreicher Kritik unterstützt hat.

Anmerkungen

Aus Gründen der leichteren Lesbarkeit wurde auf eine geschlechtsneutrale Schreibweise verzichtet. Unter jeder als männlich bezeichneten Person ist daher auch die weibliche Form zu verstehen.

Im Rahmen dieser Arbeit wurde versucht, die Verwendung von Anglizismen auf ein Minimum zu beschränken. Da jedoch viele Begriffe aus der Informatik stark an die englische Sprache angelehnt sind und es von manchen Fachbegriffen nur wenig geläufige oder nicht eindeutige Übersetzungen in der deutschen Sprache gibt, lässt sich die Verwendung von Anglizismen nicht völlig vermeiden, ohne die Lesbarkeit dieser Arbeit zu beeinträchtigen.

Inhaltsverzeichnis

1 Einleitung

In den letzten Jahrzehnten hat die Bedeutung von Prozessen in Unternehmen und in anderen Bereichen wie der Forschung stark zugenommen. So ist es für Firmen zum Erhalt der Wettbewerbsfähigkeit momentan wichtiger, die eigenen Geschäftsprozesse zu optimieren, als Verbesserungen im Bereich der Produktionsmethoden vorzunehmen. (Vgl. Harrington, 1991, S. Xff)
Zusätzlich wird die aktuelle Situation durch eine sich oft stark und schnell ändernde Unternehmenslandschaft beeinflusst. Beispiele für Veränderung finden sich in Fusionen und Kooperationen, bei denen zahlreiche Prozesse zusammengeführt und überarbeitet werden müssen, oder auch in sich ändernden gesetzlichen Bestimmungen.
Als weitere Aspekte spielen die aktuell rasanten technologischen Umwälzungen hinein. Bestehende Prozesse müssen immer an die aktuellen technischen Gegebenheiten und Anforderungen angepasst und mit diesen abgestimmt werden. Diese entwickeln sich gezwungenermaßen weiter, um den Anschluss an die Konkurrenz nicht zu verlieren, was zu einem ständigen Bedarf an Veränderungen und Anpassungen führt. (Vgl. Davenport & Short, 1995, S. 1ff)
Sich wiederholende Anpassungszyklen und die steigende Bedeutung von Geschäftsprozessen führen dazu, dass die Anzahl der sich im Einsatz befindlichen Prozesse zunimmt. Als Beispiel lässt sich hier der SAP-Referenzprozesskatalog nennen, welcher bereits mehr als 600 Prozesse beinhaltet. (Vgl. Curran & Keller, 1998, S. 1ff) Zusätzlich ist es schwierig, bestehende Prozesse schnell aus dem laufenden Betrieb zu entfernen. So kann ein Prozess sehr lange benötigen, um abgeschlossen zu werden, weil er etwa die Behandlung langwieriger Krankheiten beschreibt. Alternativ kann ein laufender Prozess auch durch kurzfristig durchgeführte Änderungen bestehender Prozesse entstanden sein. Da diese Änderungen selten gut dokumentiert sind und zumeist nicht häufig auftretende Spezialfälle behandeln, ist eine spätere Wiedereinbindung in gewöhnliche Prozesse schwierig. (Vgl. Zahran, 1998, S. 393)

1.1 Problemstellung

Die oben beschriebenen aktuellen Szenarien machen es notwendig, neue Methoden zu finden, um insbesondere die Zahl der Prozesse und damit den hierdurch entstehenden Erstellungs- und Wartungsaufwand zu reduzieren.
Möglich ist dies vor allem durch zwei Maßnahmen: Die Zahl der neu zu erstellenden Prozesse kann reduziert werden, indem versucht wird, bereits bestehende Prozesse wiederzuverwenden bzw. an die neuen Anforderungen anzupassen. Um diese Methode sinnvoll einsetzen zu können, muss den Verantwortlichen eine Möglichkeit geboten werden, Ähnlichkeitsmessungen zwischen Prozessen durchzuführen. Die so ermittelten Informationen erlauben es anschließend, vorhandene Prozesse miteinander zu vergleichen und vergleichbare bzw. verwandte Prozesse zu finden. Ein solcher Prozess kann danach als Basis für einen neuen Prozess verwendet werden, sodass redundante Prozessdefinitionen vermieden werden.

Alternativ kann versucht werden, bestehende Prozesse aufzulassen. Hierbei besteht das Problem, dass zumeist für die aktuell im Einsatz befindlichen Prozesse ein adäquater Ersatz gefunden werden muss. Als Lösungsansatz kann dazu das Vereinen bestehender Prozesse gesehen werden. Wenn beispielsweise mit dem zuvor beschriebenen Ansatz ähnliche Prozesse gefunden wurden, können diese anschließend in einen einzigen Prozess zusammengeführt werden. Dies ermöglicht eine deutliche Reduktion der sich unternehmensweit im Einsatz befindlichen Prozesse.

Die Lösungen für die beiden oben vorgestellten Ansätze sollen außerdem die folgenden Kriterien erfüllen. Dies ermöglicht eine Maximierung des Nutzens, welcher aus dieser Master Thesis gewonnen werden kann.

Korrektheit
Falls korrekte Prozesse verarbeitet werden, soll diese Korrektheit auch im Ergebnis der Algorithmen erhalten bleiben. Zusätzlich stellt sich die Herausforderung, dass es auch möglich sein soll, teilweise inkorrekte Prozessdefinitionen oder weiche Korrektheitskriterien zu bearbeiten.

Unabhängigkeit
Die ermittelten Lösungen sollen außerdem möglichst unabhängig von den Modellierungssprachen und Modellierungsplattformen sein, welche zur Erstellung der Prozessdefinitionen verwendet wurden.

Automatisierbarkeit
Die durchzuführenden Operationen sollen eine möglichst hohe Automatisierbarkeit haben. Die Operationen sollen, soweit dies sinnvoll und möglich ist, ohne Benutzerinteraktion auskommen, um die Anwender zu entlasten.

1.2 Forschungsfrage und Forschungsgegenstand

Kernbestandteil dieser Arbeit wird es sein, Lösungen für die zuvor vorgestellten Problemstellungen zu entwickeln. Diese sollen die angeführten Anforderungen bezüglich Korrektheit, Unabhängigkeit und Automatisierbarkeit einhalten.
Hierbei gilt es also, passende Algorithmen zu entwickeln, welche die Ähnlichkeit zweier Prozesse messen können. Zusätzlich bedarf es einer geeigneten Darstellung der ermittelten Ergebnisse. Diese Aufbereitung soll für den Anwender leicht zu verstehen und einfach weiterzuverarbeiten sein.
Weiters muss ein Algorithmus gefunden werden, welcher es erlaubt, verschiedene Prozesse miteinander zu vereinen. Es stellt sich dabei die Herausforderung, unter verschiedenen grundlegenden Konzepten wie die Drei- oder Zwei-Wege-Vereinigung (Vgl. Mens, 2002, S. 450f) das passende auszuwählen. Auch die am besten geeignete Methode zum "gleichzeitigen" Vereinen von mehreren Prozessen muss gefunden werden.

Basierend auf den vorangegangenen Informationen lassen sich folgende zwei Forschungsfragen finden:

- Welches Vorgehen ist notwendig, um die Ähnlichkeit zwischen Prozessen zu messen?
- Welcher Konzepte bedarf es, um mehrere Prozesse miteinander zu vereinen?

Weiters ist es auch noch möglich, mehrere Unterfragen zu definieren, welche zusätzlich geklärt werden müssen:

- Wie kann die Unabhängigkeit zwischen Modellierungsplattformen und Modellierungssprachen und den ermittelten Lösungen erreicht werden?
- Wie können Nutzerentscheidungen in die Berechnungen mit eingebunden werden?
- Wie kann die Ähnlichkeit zwischen Prozessen dargestellt werden?
- Wie kann die Korrektheit der ermittelten Lösungen in einem sinnvollen Umfang erreicht werden?
- Wie ist eine Evaluierung der erreichten Lösungen möglich?
- Wie kann eine beliebige Anzahl von Prozessen vereint werden?
- Welche Vereinigungsstrategie eignet sich für Prozessinformationen?

1.3 Aufbau der Arbeit

Die ersten beiden Kapitel dieser Arbeit ermöglichen es, ein theoretisches Fundament für die eigentlichen Kernaspekte der Arbeit (Vereinigung/Ähnlichkeitsmessung von Prozessen) zu legen. So

wird in Kapitel 2 grundlegend dargestellt, wie sich Prozesse aufbauen und auf welchen Ebenen diese betrachtet werden können. Hierbei werden auch verschiedene mögliche Problemszenarien wie die Strukturkorrektheit (Vgl. Abschnitt 2.3.3) behandelt. In diesem theoretischen Kapitel wird als Vorarbeit für die folgenden Kapitel auch bereits ein modellierungssprachenunabhängiges bzw. übergreifendes Metamodell für Prozessmodelle erstellt, welches im späteren Verlauf zur Definition eines eigenen Datenformats zur Transformation und Übertragung/Verarbeitung von Prozessen verwendet wird.

In Kapitel 3 wiederum wird behandelt, wie Unterschiede bzw. Distanzen zwischen Prozessen beurteilt und ermittelt werden können. Verschiedene Methoden zur Ermittlung der Unterschiede finden genauso Erwähnung wie die unterschiedlichen Möglichkeiten, einen Prozess zu modifizieren. Basierend auf den hier beschriebenen Methoden wurde ein mengenoperationsbasierter Ansatz gewählt, welcher später in Kapitel 6 als Basiskonzept zur Vereinigung bestehender Prozesse herangezogen wird.

Das zuvor beschriebene Kapitel über mögliche Unterschiede in Prozessen führt anschließend zum Kapitel 5, in welchem Methoden beschrieben werden, welche zur Ermittlung eines Ähnlichkeitsmaßes verwendet werden können. Insbesondere wird hier eine Methode ausgearbeitet, welche die textuellen und strukturellen Informationen der Prozesse vergleicht. Bezüglich der strukturellen Information erfolgt in dieser Arbeit außerdem die Entwicklung eines Ansatzes, welcher es ermöglich, die Nachbarschaftsbeziehungen der Prozesselemente zu berücksichtigen. Dies erlaubt es, die Aussagefähigkeit der Ergebnisse im Vergleich mit den Ergebnissen ohne Erweiterung zu erhöhen. Eine Analyse und Betrachtung der Vorteile erfolgt in der Diskussion unter 8.1.

Die nun gefundenen Prozesse können in einem weiteren Schritt vereinigt werden, um das zuvor angesprochene Ziel der Reduzierung der sich im Einsatz befindlichen Prozesse zu erreichen. Zu diesem Zweck wird in Kapitel 6 ein mengenbasierter Ansatz beschrieben, welcher es erlaubt, unterstützt durch verschiedene Mechanismen zur Konfliktauflösung wie eine automatische Kontrollflusswiederherstellung, Prozesse mittels einer Drei-Wege-Vereinigung zu vereinen. Vorteile zeigt dieser Ansatz während der Evaluierung in 8.2 und im Vergleich mit bereits bekannten Ansätzen vor allem hinsichtlich der Wartbarkeit und Praxistauglichkeit.

Die anderen Bereiche der Arbeit dienen vor allem zur Unterstützung und Vervollständigung. So wird in Kapitel 7 kurz die erfolgte Implementierung der Algorithmen der beiden Kernthemen vorgestellt. In Kapitel 4 wird ein Intermediate Format besprochen und definiert, welches es erlaubt, die erstellen Algorithmen modellierungssprachenunabhängig zu gestalten. Das erste der beiden abschließenden Kapitel (Kapitel 8) beinhaltet eine Evaluierung der Ergebnisse dieser Arbeit und zeigt die Leistungsfähigkeit der erstellten Lösungen auf. Das Kapitel 9 beinhaltet die Konklusion und bietet einen Ausblick und Anregungen für zukünftige Forschungsmöglichkeiten.

1.4 Forschungsmethodik

Zur Durchführung dieser Arbeit wurde zu Beginn eine umfassende Literaturrecherche durchgeführt, deren Ergebnisse sich unter anderem auch in den theoretischen Einleitungskapiteln (Kapitel 2 und 3) wiederfinden. Anschließend wurden die Aufgabenstellung konkretisiert und zusätzlich nichtfunktionale Anforderungen hinsichtlich der Automatisierbarkeit, Unabhängigkeit und Korrektheit (siehe 1.1) abgeleitet. Danach wurden neuartige Erweiterungen zu bestehenden Ansätzen zur Ähnlichkeitsmessung von Prozessen entwickelt und hier beschrieben. Zur Vereinigung von

Prozessen wurden neuartige Konzepte entworfen und umgesetzt. Nach der Vorstellung dieser Algorithmen und Lösungen erfolgt abschließend eine Evaluierung der Ergebnisse in theoretischer und praktischer Form (alle Algorithmen wurden auch implementiert).

2 Prozessmodellierung

Dieses Kapitel beschreibt die grundlegenden Elemente verschiedener Prozessmodellierungstechniken bzw. Modellierungsmodelle. Eingegangen wird hierbei vor allem auf die verschiedenen Komponenten der Modellierungssprachen, was insbesondere im Hinblick auf das Kapitel 4 (Intermediate Format) von Bedeutung ist. Zusätzlich zu den grundlegenden Modellierungsaspekten wird im Abschnitt 2.3 auch darauf eingegangen, wie die Korrektheit von Modellen geprüft werden kann bzw. welche Korrektheitskriterien für die Überprüfung von Modellen herangezogen werden können. Hierdurch ist eine bessere Bewertung der später erzeugten Modelle möglich. Zusätzliche Informationen, welche während der Modellierung anfallen, z.B. sogenannte Änderungslogs, werden ebenfalls betrachtet.

2.1 Grundlegende Eigenschaften von Prozessmodellen

Grundsätzlich bestehen Prozessdiagramme aus verschiedenen miteinander verbundenen Elementen wie z.B. Aktivitäten. Die Modellierung der Aktivitäten, bei welchen es sich z.B. um das Anfordern eines Angebotes handeln kann, sowie der zusätzlich zur Verfügung stehenden Beschreibungselemente können sich hierbei stark bezüglich der verschiedenen Modellierungssprachen unterscheiden. Deshalb werden in den späteren Abschnitten dieses Kapitels Modellierungssprachen nicht nur theoretisch untersucht, sondern es findet auch eine Betrachtung bestehender Modellierungssprachen statt.

2.1.1 Datenfluss

Zusätzlich zu der Abarbeitungsreihenfolge benötigt ein Prozess zumeist auch noch weitere Daten. Diese Daten und deren Fluss durch das System werden mit dem Schlagwort Datenfluss beschrieben. Es können auch Eingang und Ausgang der Daten als Teil des Datenflusses verstanden werden. Neben der Möglichkeit, dass es sich beim Datenfluss um einen in das Modell integrierten Bestandteil handelt, gibt es auch verschiedene Modellierungssprachen, welche sich hauptsächlich mit der Abbildung von Datenflüssen beschäftigen z.B. das Datenflussdiagramm. (Vgl. Bruza & Weide, 1993, S. 1ff)

2.1.2 Attribute

Weiters ist es auch möglich, während der Modellierung zusätzliche Attribute einfließen zu lassen, seien es Informationen zu zeitlichen Abläufen oder zu notwendigen Ressourcen. (Vgl. Object Management Group, 2011, S. 3)

In der folgenden Grafik werden die zuvor beschriebenen Prozessmodellteile unter Verwendung der Modellierungssprache Business Process Modeling Notation abgebildet und beschrieben.

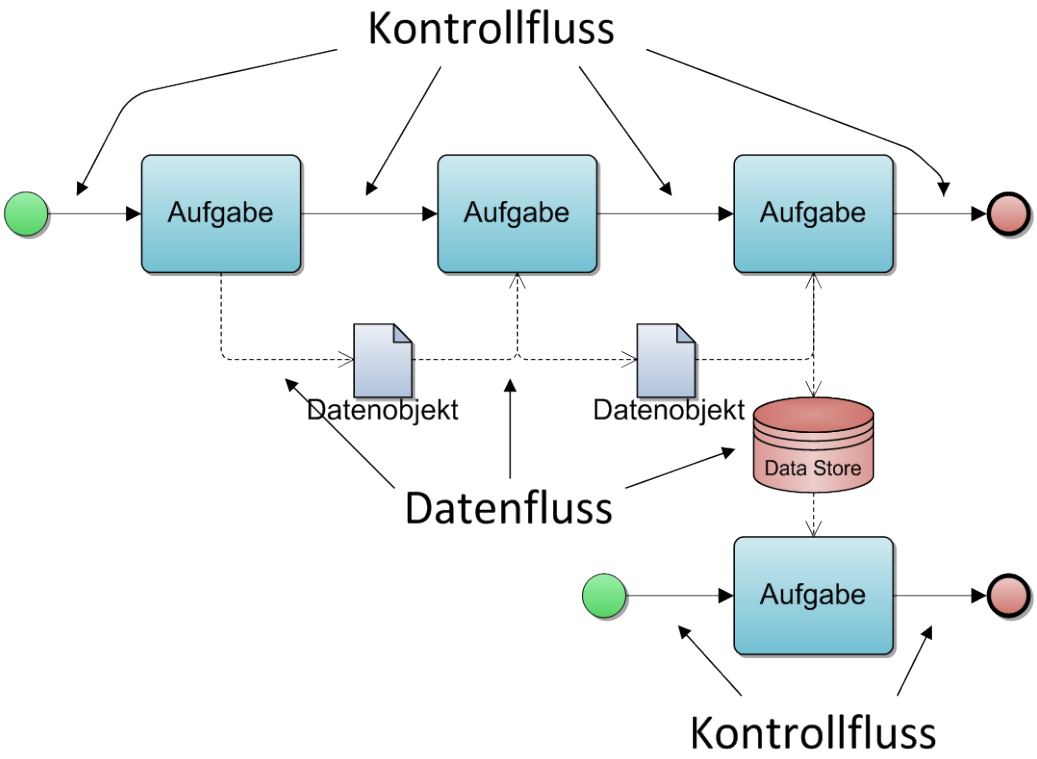

Abbildung 1: Beispiel für Kontroll- und Datenfluss anhand von BPMN

Quelle: Eigene Erstellung

2.2 Modellierungssprachen

Um eine einfache und gut zu überblickende Darstellung der oben beschriebenen Prozessmodelleigenschaften zu erhalten, werden Modellierungssprachen eingesetzt. Diese arbeiten zumeist mit grafischen Elementen und zeigen mit diesen beispielsweise an, zwischen welchen Aktivitäten ein Zusammenhang besteht. Da die Anforderungen an die Modelle umfangreich sind und sich auch die Modellierungsansätze über mehrere Jahre und Jahrzehnte weiterentwickelt haben, sind zahlreiche unterschiedliche Modellierungssprachen verfügbar, z.B. die ereignisgesteuerte Prozesskette (EPK) (Vgl. Nüttgens & Rump, 2002, S. 1ff), die Business Process Modeling Notation (BPMN) (Vgl. Allweyer, 2009, S. 8ff) oder die UML Activity Diagrams (UML AD) (Vgl. Object Management Group, 2010, S. 1ff).

Es gilt nun, aus dieser schwer überschaubaren Menge von Modellierungssprache, von denen zuvor nur einige sehr bekannte angeführt wurden, diejenigen auszuwählen, welche in dieser Arbeit weiter betrachtet werden sollen. Die Wahl fiel hierbei auf BPMN und EPK. Die Entscheidung wurde vor allem aufgrund der weiten Verbreitung dieser beiden Modellierungssprachen gefällt. Die folgende Beschreibung der gewählten Modellierungssprache beinhaltet einige Vereinfachungen, so wird z.B. der Datenfluss ignoriert. Diese Vereinfachung wurde gewählt, da in den späteren Kapiteln vor allem Bezug auf den Kontrollfluss genommen wird. Die Darstellung der Modellierungssprache selbst dient der Vorbereitung auf das Kapitel 4 (Datenformat), in welchem eine textbasierende generische Abbildungsform für unterschiedliche Modellierungssprachen, angepasst an die Anforderungen dieser Arbeit, entwickelt wird.

2.2.1 BPMN

Die Business Process Modeling Notation ist eine Modellierungssprache für Geschäftsprozesse und beinhaltet zahlreiche grafische Notationselemente. Eine erste Version der BPMN wurde im Jahre 2001 erstellt. Derzeit ist die Version 2.0 verfügbar, welche von der Object Management Group (OMG) vorangetrieben wird. (Vgl. Object Management Group, 2011, S. 1ff)

Die für diese Arbeit wichtigsten BPMN-Elemente sind die sogenannten Aktivitäten (Aufgaben oder Subprozesse, wobei eine Aktivität eine atomare Aufgabe darstellt, welche sich nicht weiter aufbrechen lässt, z.B. das Versenden eines Auftrags) (Vgl. Object Management Group, 2011, S. 151-203), Konnektoren (diese verbinden die verschiedenen Elemente des Modells miteinander und machen so z.B. den Übergang von einer Aufgabe zur nächsten kenntlich) (Vgl. Object Management Group, 2011, S. 42ff) und Gateways (diese steuern den Fluss bzw. die Abarbeitung der Aufgaben, so kann z.B. ein Gateway genützt werden, um für Aufgaben eine parallele Abarbeitung zu ermöglichen) (Vgl. Object Management Group, 2011, S. 287-302). Abschließend sind auch noch Events zu erwähnen, welche in den verschiedensten Spielarten innerhalb von BPMN definiert wurden. Für diese Arbeit sind vor allem die Start- und Endevents ohne die weiterführenden Spezialisierungen von Bedeutung.

Neben den oben beschriebenen Strukturen gibt es noch zahlreiche weitere Elemente wie Swimlines (partitionieren und organisieren von z.B. Aktivitäten) oder Events (Elemente, die auf Ereignisse reagieren z.B. auf einen gewissen Zeitpunkt), welche weitere Informationen abbilden. (Vgl. Object Management Group, 2011, S. 305-309)

Die folgende Abbildung zeigt einen exemplarischen Prozess, der in BPMN-Notation modelliert ist, in welchem auftragsspezifische Daten verarbeitet bzw. geprüft werden.

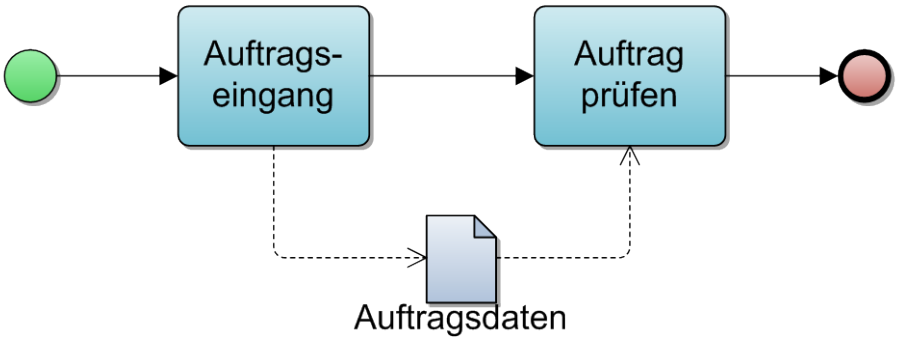

Abbildung 2: Beispiel Modellierung mit BPMN

Quelle: Eigene Erstellung

2.2.2 EPK

Auch die ereignisgesteuerten Prozessketten stellen eine Modellierungssprache für Geschäftsprozesse dar. Veröffentlicht wurde dieser Ansatz erstmals 1992; auch hier gab es, ähnlich wie bei der fortschreitenden Evolution von BPMN, seit dem Erscheinen neue Entwicklungen und Erweiterungen, wie z.B. das Konzept der erweiterten ereignisgesteuerten Prozesskette (eEPK). (Vgl. Keller, Nüttgens, & Scheer, 1992, S. 1ff)

Die für diese Arbeit wichtigsten EPK-Elemente stellen die Ereignisse (ein Zustand, welcher vor/nach einer Funktion auftritt wie z.B. der Eingang eines Auftrags) (Vgl. Scheer, ARIS – Modellierungsmethoden, Metamodelle, Anwendungen, 2001, S. 21ff), die Funktionen (welche

einer Aufgabe im zuvor besprochenen BPMN entsprechen) (Vgl. Scheer, ARIS – Modellierungsmethoden, Metamodelle, Anwendungen, 2001, S. 21ff) sowie die Konnektoren (diese dienen dem Verbinden der verschiedenen Elemente) (Vgl. Scheer, ARIS – Modellierungsmethoden, Metamodelle, Anwendungen, 2001, S. 102-146) dar. Auch die sogenannten Prozesswegweiser, welche Hinweise auf andere Prozesse geben, fließen in die folgende Betrachtung mit ein, da aus den oben angeführten Teilen ein „Subprozess“ gebildet werden kann. Da spezielle Start- bzw. Endelemente fehlen, besteht die Möglichkeit, diese zu emulieren, indem beispielsweise passende Funktionen dynamisch für den Mergevorgang erzeugt werden. Diese Emulation verfolgt das Ziel, stabile Ankerpunkte für die verwendeten Algorithmen anzulegen. Zur Verwendung der Ankerpunkte siehe Abschnitt 6.4.3.

Zusätzlich zu den oben beschriebenen EPK-Elementen sind, ähnlich wie bei dem zuvor angeführten BPMN, noch zusätzliche Ausprägungen definiert wie z.B. spezifische Ressourcenquellen. Darunter kann man sich z.B. eine Datenbank oder eine Zuordnung von Aufgaben zu bestimmten Organisationseinheiten vorstellen. (Vgl. Scheer, ARIS – Modellierungsmethoden, Metamodelle, Anwendungen, 2001, S. 170-175)

Die folgende Grafik zeigt einen beispielhaften Prozess, welcher unter Nutzung der EPK-Notation modelliert wurde. Bei diesem Prozess wird auf das Ereignis Auftragseingang reagiert. Der Auftrag wird anschließend von einem Vertriebsmitarbeiter unter Zuhilfenahme der Auftragsdatenbank bearbeitet.

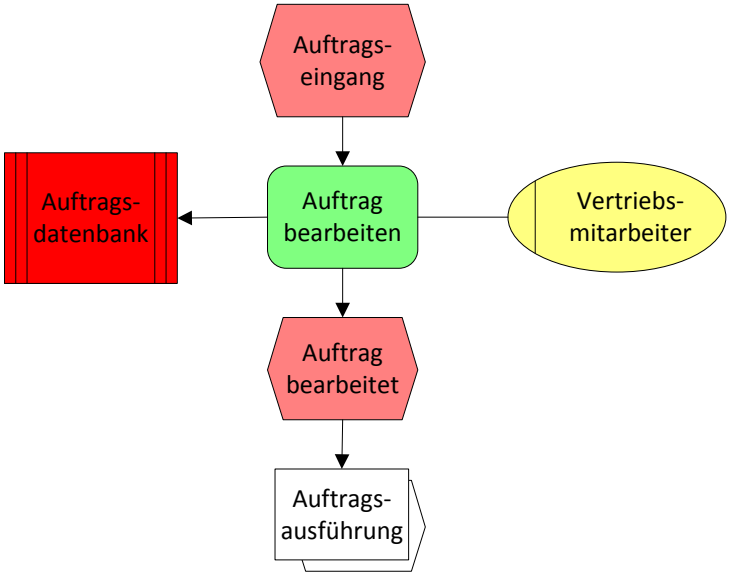

Abbildung 3: Beispiel Modellierung mit EPK

Quelle: Eigene Erstellung

Betrachtet man nun die weiter oben beschriebenen Grundelemente, aus denen EPK und BPMN aufgebaut sind, ist es möglich, diese abzugleichen und auf einen gemeinsamen Nenner zu bringen. Zur Verdeutlichung dieses Ansatzes dient die Abbildung 4: Gegenüberstellung verschiedener Modellierungssprachen. Die dort gezeigte Gegenüberstellung kann als Vorschau auf das Kapitel Datenformat gesehen werden, da es zeigt, in welcher Art und Weise eine generische Zusammenfassung unterschiedlicher Sprachen erfolgen kann. Da nicht alle Elemente in allen Modellierungs-

sprachen vorhanden sind, werden nur die für diese Arbeit wichtigsten Elemente des Kontrollflusses betrachtet.

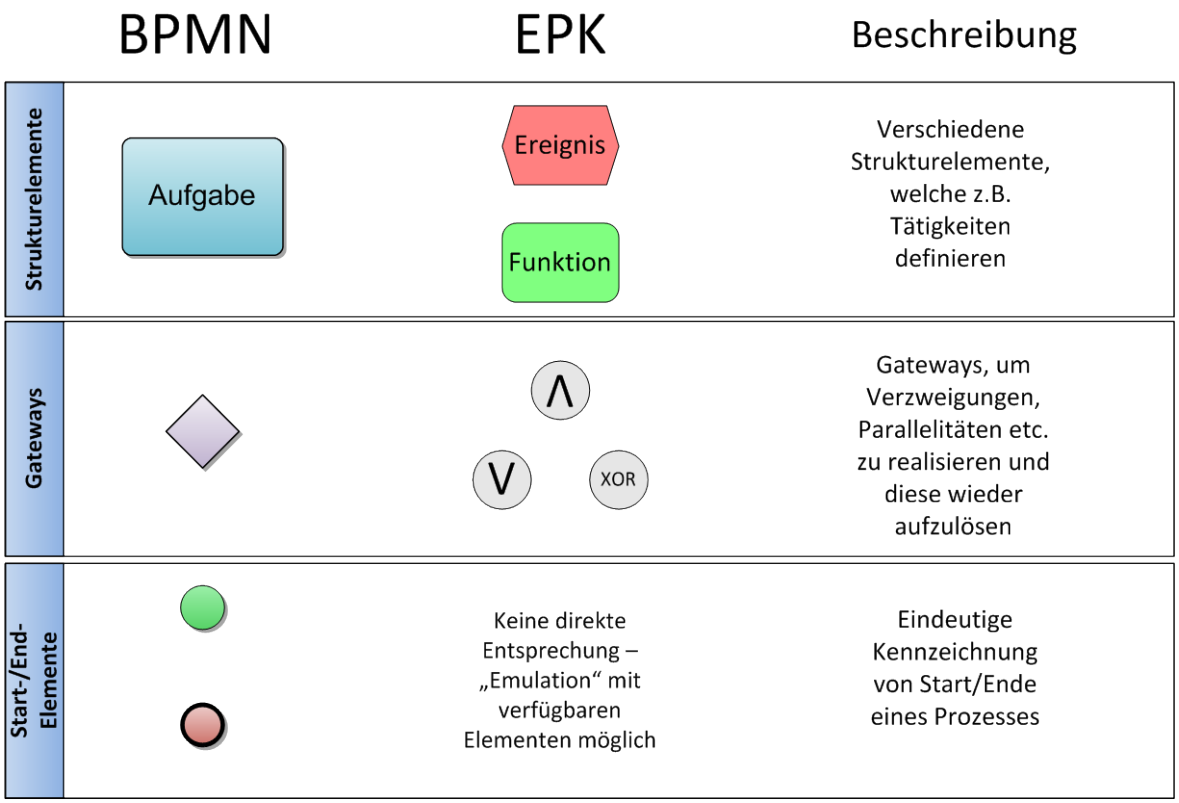

Abbildung 4: Gegenüberstellung verschiedener Modellierungssprachen

Quelle: Eigene Erstellung

Ausgehend von der in der vorhergehenden Grafik dargestellten tabellarischen Übersicht wurde das folgende Metamodell abgeleitet, welches das UML-Klassendiagramm als Metametamodell verwendet. Es zeigt die verschiedenen Elemente und deren Abhängigkeiten voneinander. Als Grundelement für alle Elemente ist hierbei das Prozesselement definiert. Davon abgeleitet werden die Elemente Konnektor (dient dem Verbinden von Prozesselementen), Steuerelement (dient dem Treffen von Entscheidungen über den Ausführungsablauf, wobei hier unterschiedliche Ausprägungen wie z.B. ein Join oder ein Parallel Gateway vorhanden sind) und Strukturelement (welches Aktivitäten und Subprozesse beinhaltet).

Einschränkungen sind insofern vorhanden, dass ein Konnektor nicht mit einem anderen Konnektor verbunden sein darf, sondern nur mit anderen Prozesselementen, welche keine Konnektoren sind. Weitere Einschränkungen bestehen nicht. So lassen sich z.B. Parallelitäten mit einem Parallel Gateway modellieren oder auch durch den Einsatz mehrerer von einem Element ausgehender Kanten. Die Darstellung komplexerer Elemente wie z.B. Schleifen wird nicht extra modelliert, denn diese können durch eine Kombination der bestehenden Elemente erstellt werden.

Eine verfeinerte Darstellung mit z.B. zusätzlichen Attributen ist im Kapitel Datenformat (Kapitel 4) ersichtlich, in welchem selbiges Modell in angepasster Form für die Generierung eines generischen Datenformats herangezogen wird. Durch die Verwendung dieses Datenformats sollen die entwickelten Algorithmen (Merge und Ähnlichkeitsanalyse) von den zahlreichen verfügbaren Modellierungssprachen möglichst unabhängig werden.

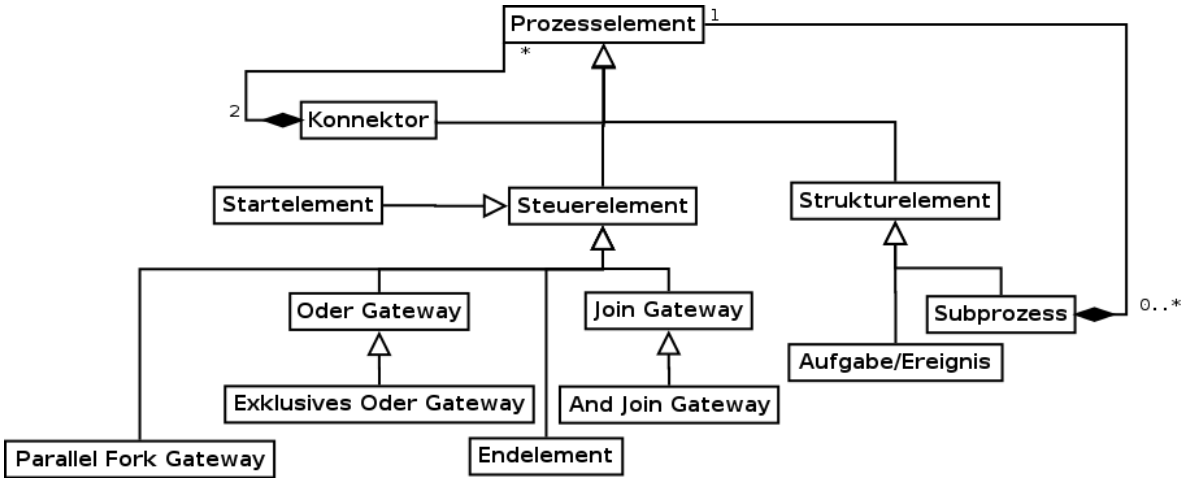

Abbildung 5: Modellierungssprache übergreifendes Metamodell

Quelle: Eigene Erstellung

2.3 Korrektheit von Modellen

Für die Prozessmodellierung ist nicht nur die Modellierungssprache wichtig, es ist auch wesentlich, dass die erzeugten Modelle verschiedenen Korrektheitskriterien genügen. Durch die Sicherstellung dieser Kriterien können bei der späteren Ausführung der modellierten Prozesse Probleme präventiv vermieden werden. Die ersten beiden Punkte dieser Auflistung lassen sich unter dem Punkt semantische Korrektheit zusammenfassen, der letzte Punkt befasst sich jedoch mehr mit dem Zusammenhang der Elemente und deren strukturellen Eigenschaften.

2.3.1 Schleifen

Schleifen können die Korrektheitskriterien verletzen, wenn diese ungewollt sind oder nicht korrekt implementiert wurden und so eine sogenannte Endlosschleife ergeben. Eine Schleife, deren Abbruchbedingung nie erfüllt werden kann, macht im schlechtesten Fall eine vollständige Abarbeitung des modellierten Prozesses unmöglich. (Vgl. Weske, 2007, S. 278ff)

2.3.2 Deadlocks

Auch Deadlocks (zu Deutsch Verklemmungen) stellen bezüglich der Korrektheit von Prozessmodellen ein Problem dar. Bei Deadlocks handelt es sich um einen Zustand, bei welchem die vollständige Ausführung des Prozesses nicht möglich ist, da auf ein Ereignis gewartet wird, das aufgrund der Modellgestaltung prinzipbedingt nicht eintreten kann. Als Beispiel wäre hier unter anderem die Verbindung eines XOR Gateways mit einem AND Join zu nennen. Ein Beispiel hierzu ist in Abbildung 6 zu finden. (Vgl. Dongen, Mendling, & Aalst, Structural Patterns for Soundness of Business Process Models, 2006, S. 6f)

2.3.3 Strukturkorrektheit

Unter Strukturkorrektheit können Modellierungsprobleme wie z.B. Aktivitäten, welche nicht korrekt in die Ausführung eingebunden sind und daher keine passende Verbindung zu anderen Elementen des Modells besitzen, verstanden werden. Dies bezeichnet die Pfadeigenschaft der Elemente. Hierbei ist es möglich, dass das Prozesselement über gar keine oder eine nur unzureichende Verbindung zu anderen Elementen verfügt. Daher bestehen z.B. eingehende Verbindungen; allerdings fehlen die ausgehenden Verbindungen. Ob diese Einschränkung zutrifft, hängt davon ab, um welches Element es sich handelt. So existiert z.B. das Endelement, welches keine ausgehenden Verbindungen benötigt. Ähnliche inverse Überlegungen können z.B. auch für das

Startelement angestellt werden. (Vgl. Dongen, Mendling, & Aalst, Structural Patterns for Soundness of Business Process Models, 2006, S. 6ff)

Wesentlich hierfür ist auch die Bedeutung der Erreichbarkeit. Diese ist unabhängig von den eigentlichen Verbindungen zu betrachten, da beispielsweise fehlerhafte Bedingungen in Gateways dazu führen können, dass Verbindungen nie genützt und daher Aktivitäten isoliert werden. Eine Prüfung von Erreichbarkeitsbedingungen ist mit einer sogenannten Erreichbarkeitsanalyse möglich. (Vgl. Dongen, Mendling, & Aalst, Structural Patterns for Soundness of Business Process Models, 2006, S. 3-5)

Weitere Probleme bei den Elementen können in den Elementtypen bzw. den Beschränkungen der Modellierungssprache begründet sein. Falls also in einer Modellierungssprache zwingend gefordert wird, dass eine parallele Ausführung von Aktivitäten mit einem Parallel Gateway eingeleitet und mit einem Join Gateway abgeschlossen wird und im Modell der Join fehlt, wird hierdurch dieses Korrektheitskriterium verletzt. (Vgl. Dongen, Mendling, & Aalst, Structural Patterns for Soundness of Business Process Models, 2006, S. 6)

Das Einhalten dieser Regeln ist nicht nur vor der Durchführung einer Vereinigung wichtig. So wird im Kapitel 6 vorausgesetzt, dass nicht nur die zu verarbeitenden Modelle, sondern auch das jeweilige Ergebnis korrekt sind. Das Ergebnis der entwickelten Algorithmen soll die verschiedenen Korrektheitskriterien möglichst nicht verletzen. Je nach Einsatzzweck kann es z.B. auch sinnvoll sein, die Korrektheitskriterien aufzuweichen, um beispielsweise die Bearbeitung von Prozessen zu vereinfachen. (Vgl. Dehnert & Rittgen, Relaxed Soundness of Business Processes, 2001, S. 1ff)

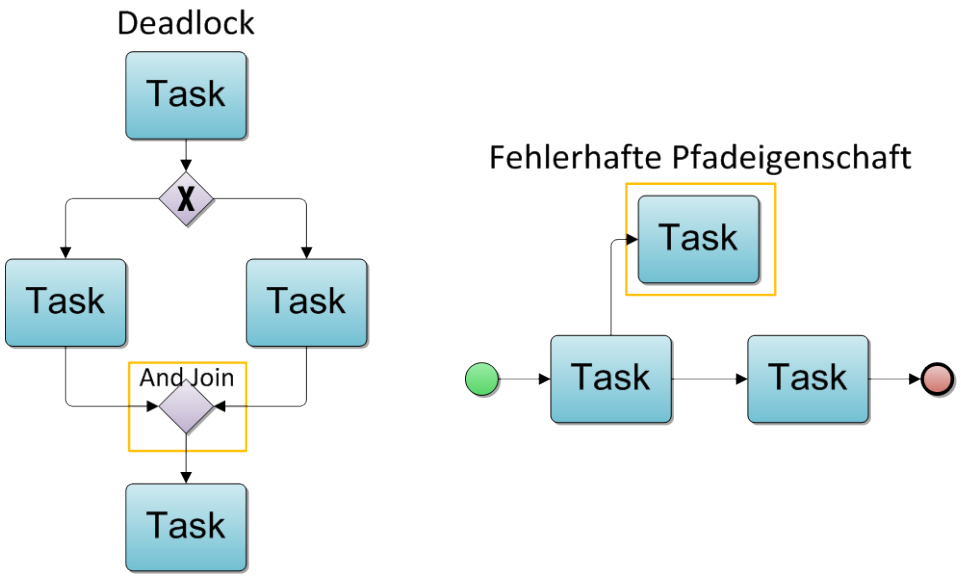

Abbildung 6: Beispiele für Prozessmodellkorrektheit

Quelle: Eigene Erstellung

2.3.4 Fehler in Syntax und Semantik der Prozesse

Bei syntaktischen Fehlern handelt es sich um Fehler, die von der eigentlichen Aufgabe des Modells unabhängig sind. Modellierungsfehler sind beispielsweise Deadlocks, welche im vorhergehenden Abschnitt bereits behandelt worden sind. Diese Fehler werden teilweise bereits von den Modellierungsvorschriften ausgeschlossen und/oder lassen sich automatisiert erkennen. Komplexer gestaltet sich die Erkennung von semantischen Fehlern. Ob ein solcher Fehler vorliegt, lässt

sich nur ermitteln, wenn der aktuelle Prozesskontext mit der Prozessmodellierung abgeglichen wird. Hierzu ist zumeist ein entsprechendes Domänenwissen vonnöten. Die beiden Fehlertypen können voneinander unabhängig auftreten. (Vgl. Karnath & Ramamritham, 1998, S. 334-341)

2.3.5 Fehlerebenen

Wie bereits in den ersten Abschnitten dieses Kapitels festgehalten wurde, besteht ein Modell aus mehreren Ebenen, einer Organisationsebene, dem Kontroll- und dem Datenfluss. (Vgl. Galler, 1995, S. 21-32) Die verschiedenen Fehler können sich hierbei auf eine einzelne Ebene beschränken, wie bei einem Deadlock, der durch ein Modellierungsproblem im Kontrollfluss bedingt ist. Es ist allerdings auch möglich, dass ein Fehler mehrere Ebenen gleichzeitig umfasst. Als Beispiel wäre hier z.B. ein Gateway zu nennen, welcher für eine Verzweigung genützt wird und hierbei Daten auswertet, die im Datenfluss nicht korrekt bedacht wurden. Hierbei tritt das Problem im Kontrollfluss zu Tage, da die Reihenfolge des Prozesses nicht wie vorgesehen abgearbeitet wird. Die Ursache liegt jedoch im Datenfluss, welcher so definiert ist, dass der Gateway die notwendigen Daten nicht oder in nicht korrekter Form erhält. (Vgl. Aalst & Jablonski, Dealing with workflow change: identification of issues and solutions, 2000, S. 6)

2.3.6 Dauerhafte und vorübergehende Fehler

Die Unterscheidung zwischen dauerhaften und vorübergehenden Fehlern bezieht sich vor allem auf Änderungen, welche, ausgehend von einem Vaterprozess, auch von allen momentanen, von diesem Vater abgeleiteten Prozessinstanzen übernommen werden sollen. Ein dauerhafter Fehler wäre hierbei ein Problem, welches immer auftritt wie z.B. ein Deadlock, von welchem auch alle nach der Änderung erstellten Prozessinstanzen betroffen wären. Vorübergehende Fehler treten nur dann auf, wenn die Änderungen auf bestehende Prozessinstanzen übertragen würden. Neue Instanzen, die nach der Änderung erzeugt werden, würden daher keine Probleme aufweisen. (Vgl. Aalst & Jablonski, Dealing with workflow change: identification of issues and solutions, 2000, S. 6)

Weitere Informationen zum Umgang mit Prozessinstanzen sowie Beispiele für mögliche Probleme sind im Kapitel 3 zu finden.

Werden die verschiedenen vorhergehenden Punkte eingehalten, ergibt sich daraus die sogenannte Zuverlässigkeit (Soundness) des Prozessmodells. (Vgl. Dongen, Mendling, & Aalst, Structural Patterns for Soundness of Business Process Models, 2006, S. fff)

2.4 Ausführungs- und Änderungsprotokolle

Außer dem eigentlichen Prozessmodell stehen durch die Generierung des Modells sowie durch dessen Ausführung zusätzliche Informationen zur Verfügung. Hierbei sind die sogenannten Ausführungs- und Änderungslogs zu nennen.

Die Ausführungslogs entstehen während der Ausführung der modellierten Prozesse und speichern die Reihenfolge, in welcher die verschiedenen Prozesselemente durchlaufen werden. Probleme ergeben sich hierbei dadurch, dass Ausführungslogs standardmäßig nur Start und Ende einer Aktivität vermerken, wodurch sich eine parallele Abarbeitung aus den Logs nachträglich schlecht auslesen lässt. (Vgl. Agrawal, Gunopulos, & Leymann, 1998, S. 1ff)

Änderungslogs speichern die während der Modellierung getätigten Modifikationen. Dies umfasst nicht nur die während der initialen Modellgenerierung durchgeführten Änderungen, sondern auch die später notwendig gewordenen Anpassungen. Bei den Änderungslogs muss dabei zwi-

schen primitiven und höherwertigen Operationen unterschieden werden. Primitive Operationen sind die einfachsten Aktionen, welche zur Modellmodifikation ausgeführt werden können. Hierunter ist beispielsweise das Löschen eines Elements zu verstehen. Höherwertige Operationen fassen für eine leichtere Lesbarkeit sowie Fehlerprävention mehrere primitive Operationen in einer atomaren höherwertigen Operation zusammen. Da sich diese Operationen, z.B. das Einfügen von Element X und ein späteres Löschen von Element X, gegenseitig aufheben können, ist es auch sinnvoll, vor Verwendung dieser Logart das Logfile so weit wie möglich zu bereinigen. Näher eingegangen wird auf diese Operationsarten und das Bereinigen von Ereignisprotokollen im Kapitel 3. (Vgl. Rinderle, Reichert, Jurisch, & Kreher, 2006, S. 2-14)

Die folgende Tabelle zeigt ein Beispiel für Ausführungslogs (linke Seite) sowie für Änderungslogs (rechte Seite). Das Ausführungslog zeigt den Start und das Ende der Aktivitäten „lies Bestellung“ und „prüfe Bestellung“. Die Änderungslogs zeigen das Hinzufügen der Aktivität „prüfe Bestellung“ zwischen „lies Bestellung“ und „führe Bestellung aus“. Weiters werden die Aktivitäten „liefere Bestellung“ und die zugehörige Verbindung gelöscht.

Ausführungslog	**Änderungslog**
start(lies Bestellung, 11:30) end(lies Bestellung, 11:45) start(prüfe Bestellung; 12:00) end(prüfe Bestellung; 12:20)	insertActivity(S, lies Bestellung, führe Bestellung aus, prüfe Bestellung) deleteConnection(S, führe Bestellung aus, liefere Bestellung) deleteActivity(S, liefere Bestellung)

Tabelle 1: Gegenüberstellung von Ausführungs- und Änderungslogs

Quelle: Eigene Erstellung

2.5 Weiterführende Literatur

In diesem Abschnitt werden einige weiterführende Arbeiten angeführt, welche für dieses Kapitel als Grundlage dienten oder dazu inspiriert haben.

Die Idee, für diese Arbeit eine Modellierungssprache in einer abstrakten Form zu erfassen, wurde durch eigene Überlegungen gewonnen und durch die Arbeit „Metamodellierung als Ansatz zur Lösung der Modellaustauschproblematik im Umfeld objektorientierter Modellierungssprachen“ bestärkt. In dieser werden Aspekte der Metamodellierung und Modelltransformation in Hinsicht auf die von der OMG eingeführten Standards betrachtet. (Vgl. Jeckle, 2000)

Als Vorbereitung für eine abstrakte Modellierungssprachendefinition werden in dieser Arbeit verschiedene Modellierungssprachen erfasst. Es wurde hierzu auf zahlreiche Informationen aus den jeweiligen Standards, insbesondere aus denen der Object Management Group und Architektur integrierter Informationssysteme (ARIS) (Vgl. Scheer, Architektur integrierter Informationssysteme. Grundlagen der Unternehmensmodellierung, 1992) zurückgegriffen. In diesen erfolgt eine genaue Definition der jeweiligen Modellierungssprachen sowie deren Konzepte und Elemente.

Daraus abgeleitet wurde die Notwendigkeit, zusätzliche theoretische Informationen bezüglich der Korrektheit von Modellen und der verfügbaren Prozessprotokolle darzulegen. Dieses Thema wurde hier nur einleitend dargelegt. Autoren wie Rinderle et al. (Vgl. Ly, Rinderle, & Dadam, 2006) liefern hier eine zusätzliche Spezialisierungen bezüglich semantischer Korrektheit. Dehnert et al. wiederum sehen die Anwendung von strengen Korrektheitskriterien als suboptimal an und empfehlen eine Aufweichung dieser im Zusammenspiel mit einer leichter durchzuführenden Strukturüberprüfung. (Vgl. Dehnert & Zimmermann, On the suitability of correctness criteria for business process models, 2005) Ein genereller Vorschlag bezüglich der Aufweichung von Korrektheitskriterien, angepasst an die jeweiligen Anforderungen, erfolgt in der Arbeit „Relaxed Soundness of Business Processes“. (Vgl. Dehnert & Rittgen, Relaxed Soundness of Business Processes, 2001)

3 Unterschiede zwischen Prozessen

Die beiden Hauptthemen dieser Arbeit beschäftigen sich mit der Ähnlichkeitsmessung zwischen Prozessen und der Vereinigung unterschiedlicher Prozessdefinitionen. Für diese beiden Gebiete ist es notwendig, die Unterschiede zwischen den verarbeiteten Prozessen zu erkennen, um entsprechend reagieren zu können.

Der Umgang mit Prozessdefinitionsmodifikationen ist durchaus relevant, da sich Änderungen in bestehenden Prozessen auf Dauer nicht vermeiden lassen. So waren in den letzten Jahren bestehende Unternehmensprozesse zahlreichen unterschiedlichen Faktoren ausgesetzt. Hierbei sind z.B. Globalisierung, Wartungsaufgaben (logische oder technische Fehler/Probleme im Prozess), Gesetzesänderungen sowie aufkommende Technologieumwälzungen zu nennen. Deshalb ist es notwendig, bestehende Geschäftsprozesse regelmäßig anzupassen, wobei die durchzuführenden Änderungen je nach Optimierungspotential in unterschiedlicher Tiefe und Ausprägung erfolgen können. Werden diese Anpassung nicht vorgenommen, verlieren Prozesse im Laufe der Zeit mehr und mehr an Sinnhaftigkeit, Qualität und Relevanz. (Vgl. Kettinger, Teng, & Guha, 1997, S. 1ff), (Vgl. Küster, Gerth, Fürster, & Engels, 2008, S. 1), (Vgl. Georgakopoulos, Hornick, & Sheth, 1995, S. 1)

3.1 Prozessmodifikation

Änderungen an Prozessen können auf allen Ebenen (Kontrollfluss, Datenfluss sowie bei den Attributen) erfolgen. So können sowohl der Kontroll- und Datenfluss als auch die zusätzlichen Prozessattribute von Änderungen betroffen sein. Da das Hauptaugenmerk dieser Arbeit auf dem Kontrollfluss liegt, wird dieser hier auch bevorzugt behandelt.

3.1.1 Operationen zur Modifikation

Die Operationen, welche zur Erzeugung der Änderungen genützt werden, lassen sich, wie bereits im Kapitel 2 beschrieben, in höherwertige und primitive Operationen einteilen, wobei, wie bereits angedeutet, höherwertige Operationen aus mehreren niederwertigen Operationen bestehen.

Die Tabelle 2 zeigt die verschiedenen möglichen Operationen. Eingeteilt werden diese, von links nach rechts, in primitive Operationen, höherwertige Operationen und höherwertige Fragmentoperationen. Bei Fragmentoperationen werden mehrere höherwertige Operationen, ähnlich dem Schritt von primitiven zu höherwertigen Operationen, zusammengefasst. (Vgl. Rinderle, Reichert, Jurisch, & Kreher, 2006, S. 4-7) Hierdurch ist es möglich, mehrere Prozesselemente mit einer ein-

zelnen Operation in den Prozess zu integrieren. Weitere Vorteile durch die höherwertigen Operationen sowie durch Fragmentoperationen ergeben sich bei der Lesbarkeit, weil mehrere kleine Schritte zu einem großen zusammengefasst werden. Ein weiterer Vorteil ist eine erleichterte Korrektheitsprüfung, da bei einer solchen atomaren Operation alle hinzugehörigen Schritte gemeinsam überprüft werden können. Weitere Informationen bezüglich der Korrektheit von Prozessen finden sich im Abschnitt 2.3. (Vgl. Cardoso & Aalst, 2009, S. 184f)

Primitive Operationen	**Höherwertige Operationen**	**Fragmentoperationen**
Einfügen	Einfügen	Fragment einfügen
Löschen	Löschen	Fragment löschen
	Verändern	Fragment verändern
	Verschieben	Fragment verschieben

Tabelle 2: Darstellung unterschiedliche Operationsarten

Quelle: Eigene Erstellung

Außerdem wurde zur Erhöhung der Flexibilität im Umgang mit Prozessen und deren Adaption, basierend auf den zuvor beschriebenen Operationen, das Konzept der „Change Patterns“ erdacht. Bei diesen handelt es sich um Abläufe, welche von Prozessbearbeitungssystemen unterstützt werden können, um Prozesse zu modifizieren. Diese Konzentration auf strukturierte und definierte Abläufe erleichtert die Vergleichbarkeit und Analyse von Änderungen sowie die Bewertung von Prozessmanagementsystemen. (Vgl. Weber, Reichert, & Rinderle-Ma, Change Patterns and Change Support Features – Enhancing Flexibility in Process-Aware Information Systems, 2006, S. 1-43)

In Tabelle 3 wird dargestellt, wie sehr die verschiedenen niederwertigen Operationen durch höherwertige Operationen vereinfacht werden können. Links sind primitive Operationen angeführt, welche zum Einfügen zweier neuer Tasks verwendet werden, mittig wurden diese zu zwei höherwertigen Operationen vereinfacht. In der als „Fragmentoperationen“ bezeichneten Spalte ist ersichtlich, dass für die gleiche Aufgabe nur mehr eine einzige Fragmentoperation notwendig ist. Erkennbar ist ferner, dass bei den primitiven Operationen neben den Einfügeoperationen für die Tasks auch die Herstellung der korrekten Verbindungen zwischen diesen schrittweise angegeben werden muss. Dieser Aufwand ist bei den mittigen höherwertigen Operationen nicht mehr notwendig bzw. schon implizit in den höherwertigen Einfügeoperationen deklariert.

Primitive Operationen	Höherwertige Operationen	Fragmentoperationen
deleteConnector(A,D) insertTask(B) insertTask(C) createConnector(A,B) createConnector(B,C) createConnector(C,D)	insertTask(B, A, D) insertTask(C, B, D)	insertFragment(Task B, Task C, A, D)

Tabelle 3: Zusammenfassung von Operationen

Quelle: Eigene Erstellung

Die folgende Abbildung 7 visualisiert die in der Tabelle 3 gezeigten Operationen. Die Prozessdarstellung mit der Bezeichnung „Version vor den Änderungen“ zeigt den Prozess in seinem Originalzustand zu sehen. Der mit „Version nach den Änderungen“ bezeichnete Prozess zeigt den Zustand des Prozesses nach den Änderungen.

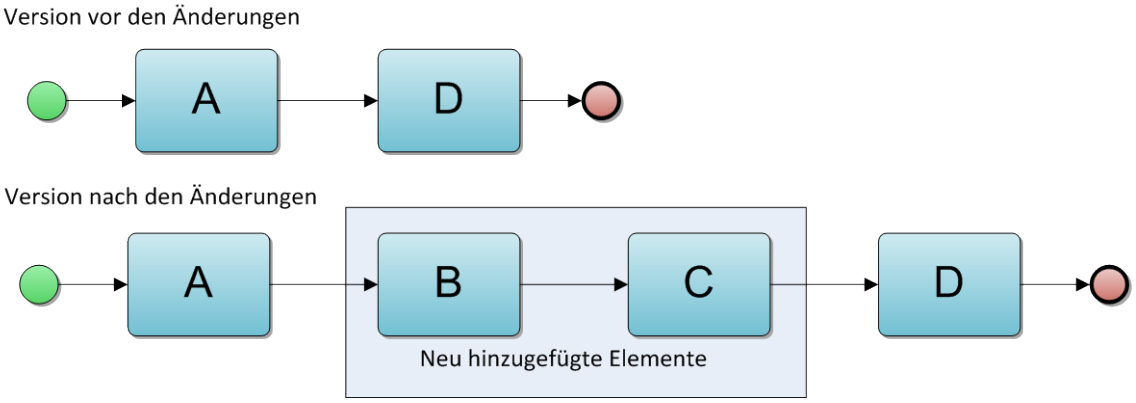

Abbildung 7: Visualisierung der Auswirkung von Modifikationsoperationen

Quelle: Eigene Erstellung

Die zuvor angeführten Operationen können auf Prozesse in unterschiedlichen Stadien angewendet werden. Daher kann eine Modifikation eines Prozesses Auswirkungen nur auf das Prozessmodell oder auf die jeweils vom Prozessmodell ausgehenden und aktuell aktiven Prozessinstanzen haben. Werden die Änderungen nur auf den zugehörigen Prozesstyp angewandt und nicht auf davon abgeleitete Instanzen übertragen, ist die Überprüfung der Änderungen einfacher, da dabei der aktuelle Ausführungsstatus der Prozessinstanz nicht in die Evaluierung mit einbezogen werden muss. Informationen hierzu finden sich im Abschnitt Korrektheit von Modellen 2.3.

Das Verhältnis von Prozesstypen und Instanzen gestaltet sich folgendermaßen: Die verschiedenen zu unterstützenden Prozesse wie das Kaufen eines Produktes oder Bearbeiten eines Urlaubsantrages werden in Prozesstypen definiert. Prozessschemata wiederum basieren auf diesen Typen und beinhalten die Abläufe, welche zu durchlaufen sind, um den Prozess abzuschließen. Instanzen

wiederum werden, basierend auf einem dieser Schemata, erzeugt und beinhalten den jeweils aktuellen Status der Ausführung des Prozesses. (Vgl. Weber, Wild, Reichert, & Dadam, 2007, S. 2)

3.1.2 Änderung von Prozessinstanzen

Problematischer wird es, wenn Änderungen an einzelnen laufenden Instanzen bzw. ausgehend von einem "Prozesstyp" an allen laufenden Instanzen vollzogen werden sollen. Ob sich eine konkrete Prozessinstanz basierend auf den im Prozesstype vollzogenen Änderungen modifizieren lässt, ist immer vom jeweiligen Ausführungszustand der Instanz abhängig. Deshalb ist es notwendig, jede Prozessinstanz hinsichtlich der Anwendbarkeit der gewünschten Modifikation einzeln zu überprüfen. Fällt eine der Überprüfungen negativ auf, gilt es zu entscheiden, ob die Änderung hier "erzwungen" werden soll oder ob Instanzen, bei welchen die Änderungen nicht problemlos möglich sind, ignoriert werden sollen. Kurzfristig erscheint der Weg des Ignorierens als der einfachere. Längerfristig bedeutet dies jedoch, dass bei zukünftigen Änderungen des Vaterprozessmodells die Änderungen bei zuvor ignorierten Modellen erschwert oder im schlechtesten Fall nicht übernommen werden können. Weiters erhöht sich hierdurch auch die Anzahl der unterschiedlichen sich im Einsatz befindlichen Prozessmodelle, was Auswirkungen auf die Wartungskosten des Systems hat. Teilweise ist es auch nicht möglich bzw. sinnvoll, problematische Instanzen zu ignorieren. Als Beispiel seien hier medizinische Prozesse zu nennen. Dauert eine Behandlung bzw. die Prozessdurchlaufzeit mehrere Monate bzw. Jahre, wäre es ohne Instanzänderungen möglich, dass Patienten unnötig lange auf neue Behandlungsmethoden warten müssen (die neue Behandlungsmethode, die Instanzänderung, würde dann nicht übernommen werden), was eine negative Auswirkung auf die Heilungsaussichten mit sich bringen würde. (Vgl. Rinderle, Reichert, & Dadam, On Dealing with Structural Conflicts between Process Type and Instance Changes, 2004, S. 274-289)

Neben dem zuvor beschriebenen Beispiel des Ignorierens und Überführens bestehender Instanzen in das neue Prozessschema bieten sich noch weitere Möglichkeiten, sodass sich insgesamt sechs verschiedene Ansätze für den Umgang mit der Thematik finden lassen. Die Möglichkeiten eins bis fünf eignen sich hierbei besonders für den Umgang mit Prozessinstanzen, deren Transformation Probleme bereitet.

1. *Vorwärtsgerichtete Kompensation*: Hierbei werden bereits bestehende Instanzen innerhalb des Workflowsystems abgebrochen und mit ad hoc erstellten individuell ausgearbeiteten Lösungen fertiggestellt. Die hierzu nötigen Aktivitäten werden außerhalb der Kontrolle der bestehenden Workflow-Management-Systeme (WfMS) ausgeführt und gesteuert. (Vgl. Rinderle, Bassil, & Reichert, A Framework for Semantic Recovery Strategies in Case of Process Activity Failures, 2006, S. 1f)
2. *Rückwärtsgerichtete Kompensation*: Hierbei werden ebenfalls bestehende Instanzen abgebrochen und entsprechende Kompensationsaktionen ausgeführt, um die von der Instanz durchgeführten Änderungen zurückrollen zu können. Ähnlichkeiten finden sich hier mit den Kompensationsalgorithmen zahlreicher Datenbanksysteme. (Vgl. Rinderle, Bassil, & Reichert, A Framework for Semantic Recovery Strategies in Case of Process Activity Failures, 2006, S. 1f)
3. *Abbruch ohne Kompensation*: Diese Methode stellt das einfachste Vorgehen dar, da der aktuelle Status der Ausführung etc. vollständig ignoriert wird. Diese Methode ist vor allem für einfache Prozessinstanzen geeignet, bei welchen eine Kompensation wenig Nutzen

mit sich bringt bzw. das Risiko eines einfachen Abbruchs überschaubar ist. (Vgl. Greenfield, Fekete, Jang, & Kuo, 2003, S. 4)

4. *Fortführung*: Hierbei werden bereits bestehende Instanzen unverändert fortgeführt. Neue Instanzen werden unter Verwendung der bereits angepassten Prozessdefinition erstellt. (Vgl. Casati, Ceri, Pernici, & Pozzi, 1996, S. 8)
5. *Umgehung*: Diese eignet sich besonders dann, wenn es während der Adaption bestehender Instanzen zu Problemen kommt. Hierbei wird die Transformation aufgrund dieser Probleme kurzfristig ausgesetzt, um eine weitere Ausführung bestehender Instanzen zu ermöglichen. Sobald eine Lösung gefunden wurde, wird die Transformation fortgeführt.
6. *Transformation*: Diese stellt den optimalen Fall dar, in welchem alle alten Instanzen vollständig auf die neue Prozessdefinition überführt werden. (Vgl. Aalst & Jablonski, Dealing with workflow change: identification of issues and solutions, 2000, S. 3f)

Ein genauer beschriebenes Beispiel für die auftretende Problematik, Änderungen bei bereits laufenden Prozessinstanzen zu realisieren, ist in Abbildung 8 zu sehen. Zu erkennen sind zwei Prozessmodelle, welche jeweils eine mögliche Interpretation eines Versandprozesses darstellen, wobei die Möglichkeit besteht, das linke Modell in das rechte zu überführen, um z.B. die Bestellungen schneller abarbeiten zu können, oder umgekehrt das rechte Modell in das linke, um z.B. den Prozess zu vereinfachen. Abhängig von der Transformationsrichtung und dem jeweiligen Ausführungszustand der Modelle zeigen sich nun unterschiedliche Problemstellungen. Befinden sich die Instanzen in einem unkritischen Ausführungszustand, wurde also beispielweise die Abarbeitung der Aktivität "Versand vorbereiten" noch nicht abgeschlossen, ist die Transformation einfacher. Komplizierter wird es, wenn sich der Ausführungszustand innerhalb des Parallel Segments (rechtes Modell) befindet, und zwar dann, wenn z.B. die Aktivität "Versende Güter" bereits abgeschlossen ist, die Aktivität "Versende Rechnung" aber noch nicht begonnen hat. Bei der Transformation des linken Modells ist es dann notwendig, dass die Abarbeitung der "Versende Rechnung"-Aktivitäten korrekt begonnen wird, da es sonst zu einer Ausführungsverklemmung kommt. Bei der Transformation des rechten Modells auf das linke Modell, angenommen "Versende Rechnung" ist bereits abgeschlossen und "Versende Güter" wurde noch nicht beendet, wie es ja bei einer parallelen Ausführung beider Zweige möglich ist, bereitet die sequenzielle Abarbeitung der rechten Seite Probleme. Wird die Abarbeitung von "Versende Güter" nach der Transformation nicht gestartet, kommt es zu einer fehlerhaften Abarbeitung des Prozesses und der Kunde wird keine Warenlieferung erhalten. Wird die Transaktion als notwendig erkannt und gestartet, beginnt im späteren Verlauf, basierend auf den im Kontrollfluss beschriebenen Abarbeitungsregeln, auch die eigentlich bereits ausgeführte Aktivität "Sende Rechnung" erneut. Der Kunde erhält hierdurch mehr Rechnungen zugesandt, als zur korrekten Abarbeitung des Geschäftsfalls notwendig wären. (Vgl. Aalst & Basten, Inheritance of Workflows – An approach to tackling problems related to change, 2002, S. 2-4)

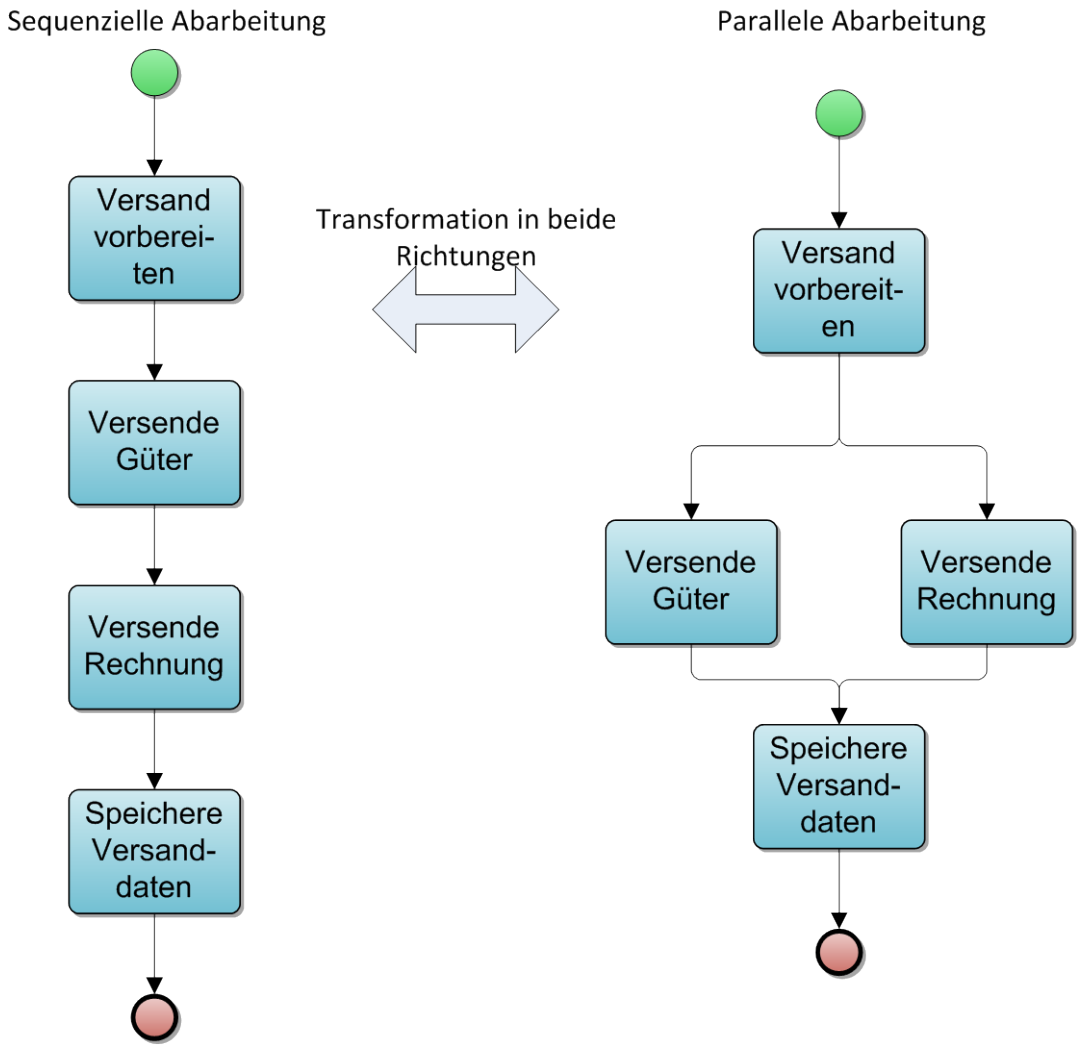

Abbildung 8: Beispiel für eine problematische Prozesstransformation

Quelle: (Vgl. Aalst & Basten, Inheritance of Workflows – An approach to tackling problems related to change, 2002, S. 2-4)

Der vorhergehende Teil dieses Kapitels zeigt verschiedene Problemfelder auf, welche bei der Veränderung bestehender Prozessinstanzen auftreten.

Diese Probleme werden in diesem Abschnitt zusammengefasst, sodass sich fünf Problemgruppen ergeben:

1. *Vergangenheitsänderung*: Diese Gruppe beschäftigt sich mit der Problematik, dass Änderungen dann nicht übernommen werden dürfen, wenn diese einen Teil der Prozessinstanz betreffen, welcher bereits ausgeführt wurde. Wird diese Regel nicht beachtet, kann dies zu Verklemmungen führen, welche die weitere Ausführung verhindern. Auch die Datenflussebene kann davon negativ beeinflusst werden, weil Vergangenheitsänderung zu inkonsistenten Daten innerhalb der Instanz führen können, wenn Aktivitäten mit Informationen arbeiten, welche durch geänderte Aktivitäten innerhalb des bereits ausgeführten Prozessteils erstellt werden sollten. Die notwendigen Daten stehen dann nicht mehr zur Verfügung, weil die entsprechend angepassten Aktivitäten nicht mehr ausgeführt werden.

(Vgl. Rinderle, Reichert, & Dadam, Correctness criteria for dynamic changes in workflow systems – a survey, 2004, S. 14)

2. *Schleifentoleranz*: Dieser Punkt beschäftigt sich mit der Anwendung von Änderungen in Schleifen. Er ermöglicht je nach Prozess eine Aufweichung des vorherigen Problems, da bei Schleifen die gleichen Aktivitäten mehrmals durchlaufen werden, wodurch Aktivitäten in bereits ausgeführten Bereichen korrekt mit eingebunden werden können. (Vgl. Rinderle, Reichert, & Dadam, Correctness criteria for dynamic changes in workflow systems – a survey, 2004, S. 14f)
3. *Unsicherer Status*: Dieses Problem beschreibt den Fall, bei dem nicht eindeutig feststellbar ist, ob eine Aktivität bereits ausgeführt wird oder nur die Möglichkeit besteht, die Aktivität auszuführen, diese aber noch nicht begonnen hat. Ob dieses Problem auftreten kann, hängt maßgeblich von den verwendeten Ausführungssystemen ab. (Vgl. Rinderle, Reichert, & Dadam, Correctness criteria for dynamic changes in workflow systems – a survey, 2004, S. 15)
4. *Reihenfolgenänderung*: Hier wird die Reihenfolge der auszuführenden Aktivitäten verändert. Dies ermöglicht die Erzeugung einer parallelen Aktivitätsabarbeitung, das Vertauschen der Reihenfolge von Aktivitäten oder die Umwandlung einer parallelen in eine sequenzielle Ausführung. (Vgl. Rinderle, Reichert, & Dadam, Correctness criteria for dynamic changes in workflow systems – a survey, 2004, S. 15)
5. *Parallelisierung*: Hierunter werden speziell Probleme zusammengefasst, welche auftreten können, wenn durch das Hinzufügen neuer Elemente ein neuer Parallelisierungszweig entsteht. Dies kann zu Verklemmungen führen, wenn die Abarbeitung des neuen Parallelzweiges nicht korrekt gestartet wird. (Vgl. Aalst & Basten, Inheritance of Workflows – An approach to tackling problems related to change, 2002, S. 3f)

3.2 Unterschiedliche Änderungsarten

Basierend auf den in Abschnitt 3.1.1 angegebenen Modifikationsoperationen können verschiedene Änderungsmöglichkeiten an der Kontrollflussperspektive eines Prozesses ermittelt werden. Diese Änderungen lassen sich in drei Gruppen einteilen: Änderungen an den Eigenschaften eines Prozessmodells, Änderungen an der Prozesselementmenge und Änderungen im Kontrollfluss, wobei hier insbesondere Änderungen in der Reihenfolge und Ordnung der Elemente besprochen werden.

3.2.1 Kontrollfluss

Beim Kontrollfluss handelt es sich in der Informatik um eine vorgegebene Ordnung, in welcher Aufgaben, Codeaufrufe etc. abgearbeitet werden sollen. Diese Abarbeitungsvorgaben finden sich auch bei der Prozessmodellierung und beeinflussen daher auch diese Arbeit. Um jedoch den Rahmen dieser Arbeit nicht zu sprengen, wird bei den Algorithmen und bei den Betrachtungen in den Kapiteln, welche sich mit Ähnlichkeitsmessung (Kapitel 5) und der Vereinigung von Prozessen (Kapitel 6) beschäftigen, vor allem der Umgang mit dem Kontrollfluss untersucht. (Vgl. Dubray, S. 3ff)

3.2.2 Änderungen an den Eigenschaften

Dieser Fall tritt ein, wenn die Attribute von Prozesselementen geändert werden. Daher gehören Änderungen der Beschreibung einer Aktivität oder das Modifizieren der Verzweigungsbedingungen eines Gateways zu dieser Gruppe. Modifikationen dieser Gruppen sind einfach zu erkennen,

indem Elemente der Kindprozesse mit denen des Vaters eins zu eins verglichen werden. Dargestellt wird diese Möglichkeit in Abbildung 9, welche die Modifikation eines Prozesselements zeigt.

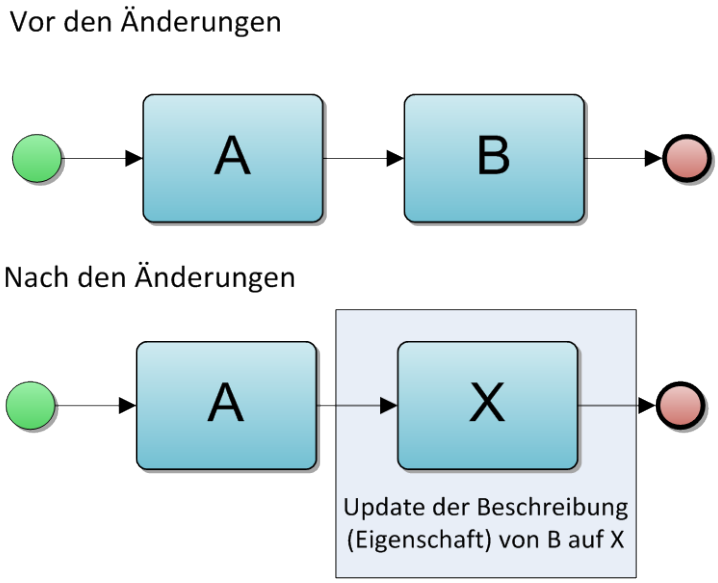

Abbildung 9: Beispiel für die Änderung von Prozesselementeigenschaften

Quelle: Eigene Erstellung

3.2.3 Änderungen an der Elementmenge

Bei Änderungen in dieser Gruppe fand eine Änderung der Menge der definierten Prozesselemente statt; es wurden neue Elemente hinzugefügt oder entfernt. Für jedes Prozesselement gilt hierbei eine von folgenden drei Möglichkeiten:

1. Das Element ist sowohl im Vater- wie auch im Kindprozess vorhanden; es wurde also weder gelöscht noch neu hinzugefügt.
2. Das Element ist im Vaterprozess, aber nicht im Kindprozess vorhanden; es wurde also während der Erzeugung des Kindes entfernt.
3. Das Element ist im Kindprozess, aber nicht im Vaterprozess vorhanden; es wurde also während der Erzeugung des Kindprozesses neu hinzugefügt.

Stehen die Änderungen nicht als Ereignisprotokoll zur Verfügung (siehe Abschnitt 3.4), dann ist es schwierig, eine Änderung der Elementmenge von Änderungen an Elementeigenschaften zu unterscheiden, da in beiden Fällen ein Element im Vater vorhanden, in den Kindern jedoch nicht mehr gegeben ist. Eine Möglichkeit, dieses Problem zu lösen, ist es, eindeutige Identifikationsnummern an alle Elemente zu vergeben, welche bei einer Änderung an den Elementeigenschaften unverändert übernommen werden. Dieser Ansatz wird bei der in dieser Arbeit beschriebenen mengenbasierten Prozessvereinigung verwendet (siehe Abschnitt 6.4.2).

In Abbildung 10 wird die oben beschriebene dritte Möglichkeit gezeigt, bei welcher ein neues Element hinzugefügt wird. Die Grafik zeigt das Hinzufügen einer Aktivität mit dem Beschreibungstext „C“.

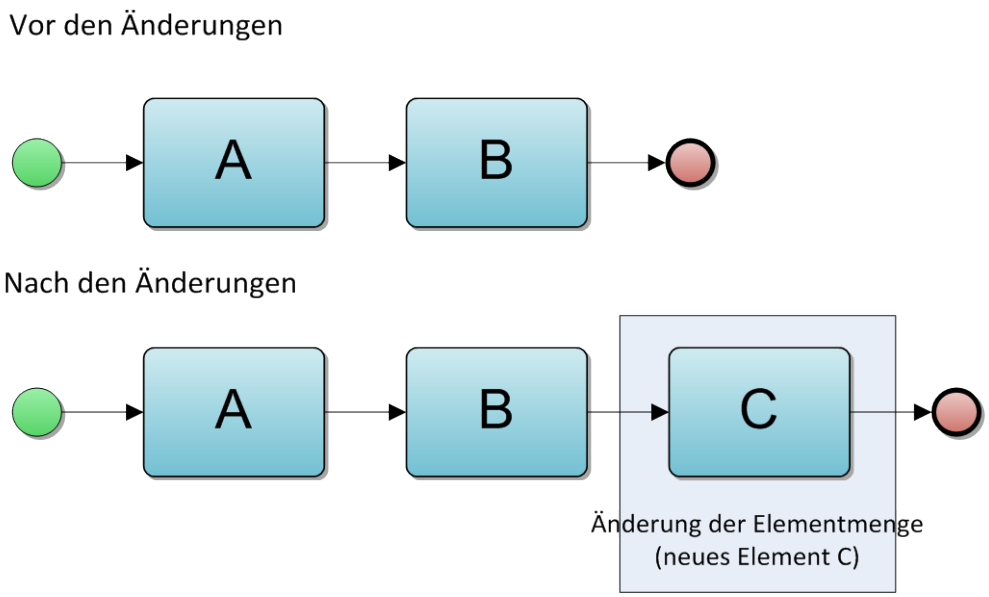

Abbildung 10: Beispiel für die Änderung der Prozesselementmenge

Quelle: Eigene Erstellung

3.2.4 Änderungen in der Reihenfolge

Außer den zwei zuvor angesprochenen Möglichkeiten kann auch die Position eines Prozesselements innerhalb des Prozesses geändert werden. Dies ist unabhängig davon zu betrachten, welche Elemente aus dem Prozess gelöscht bzw. ob neue Elemente eingefügt wurden. Eine Erkennung, ob ein Element verschoben wurde, ist möglich, wenn die Vor- bzw. Nachfolgerbeziehung des jeweiligen Elements untersucht wird. Dabei werden die Elemente, welche vor bzw. nach einem zu untersuchenden Hauptelement abgearbeitet werden würden, betrachtet. Der Kontrollfluss eines der Prozesse kann hierbei genützt werden, um die notwendigen Informationen zu erhalten. Es gilt danach anhand der erkannten Unterschiede zu ermitteln, inwiefern es zu einer Reihenfolgenänderung gekommen ist.

Die Abbildung 11 zeigt eine Verschiebung der Aktivität C an eine neue Position und macht damit die im vorherigen Absatz beschriebene Änderungsart deutlich.

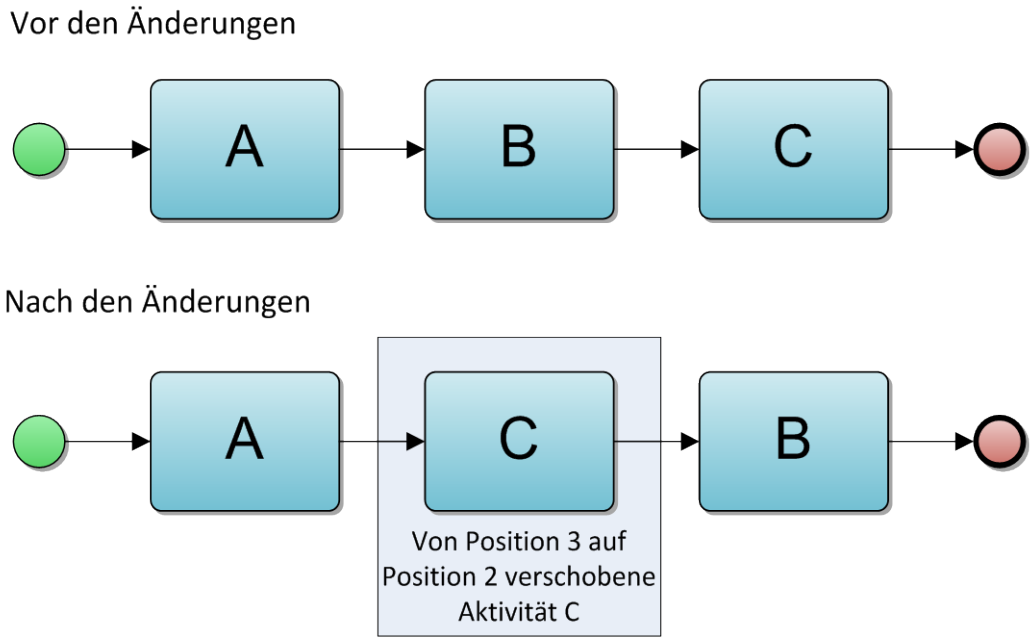

Abbildung 11: Beispiel für die Verschiebung von Prozesselementen

Quelle: Eigene Erstellung

3.2.5 Überschneidende und unabhängige Änderungsarten

Unterschiede in Prozessen, relativ zu einem Vaterprozess (welcher die grundlegende Basis für die Kindprozesse bildet), lassen sich in verschiedene Arten und Ausprägungen einteilen. Zur Verdeutlichung wird hier eine beispielhafte Situation herangezogen, bei welcher von einem Vaterprozess X zwei unterschiedliche Versionen abgeleitet wurden, die Kindprozesse A und B, welche, relativ zum Vater, jeweils unterschiedliche Änderungen beinhalten. Sollen nun die beiden Kindprozesse wieder vereint werden, ist es unter anderem wichtig zu betrachten, ob sich die durchgeführten Änderungen überlappen oder voneinander unabhängig sind. Unabhängige Änderungen können hierbei leicht zusammengeführt werden, da diese auf den Vaterprozess vollkommen unterschiedliche Auswirkungen haben, welche einander nicht beinträchtigen. Dies ist beispielsweise dann möglich, wenn in A und B jeweils an entgegengesetzten Punkten, z.B. in A am Prozessstart und in B am Prozessende, neue Aktivitäten eingeführt worden wären. (Vgl. Rinderle, Reichert, & Dadam, Disjoint and Overlapping Process Changes: Challenges, Solutions, Applications, 2004, S. 101-120)

Die folgende Abbildung 12 zeigt ein Szenario mit unabhängigen Änderungen; diese (hinzufügen der Aktivitäten A und B) finden in voneinander unabhängigen Bereichen statt.

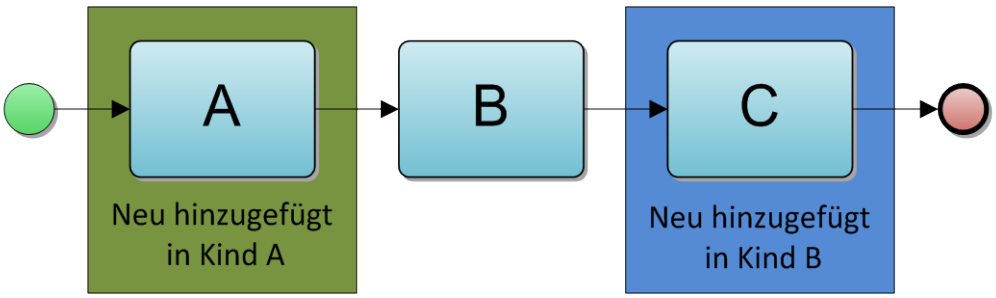

Abbildung 12: Unabhängige Änderungen an zwei Kindprozessen

Quelle: Eigene Erstellung

Anders ist es bei sich überlappenden Änderungen. Hier lassen sich diese nicht eindeutig trennen und es besteht die Möglichkeit, dass Änderungen nicht bzw. nur durch eine endgültige Anwenderentscheidung übernommen werden können. Dies ist beispielsweise dann der Fall, wenn sowohl bei A als auch bei B neue Aktivitäten an exakt der gleichen Stelle eingefügt wurden. Es stellt sich hier die Frage, in welcher Reihenfolge diese übernommen werden sollen. Eine mögliche Lösung würde sich z.B. durch eine Betrachtung des Datenflusses bieten, aus welcher erkannt werden kann, dass die beiden Aktivitäten unterschiedliche Informationen bearbeiten und deshalb voneinander unabhängig sind. Stehen allerdings nur Informationen zum Kontrollfluss zur Verfügung, hat ein menschlicher Benutzer diese Entscheidung zu treffen, welcher über ein entsprechendes Domänenwissen verfügen muss. (Vgl. Rinderle, Reichert, & Dadam, Disjoint and Overlapping Process Changes: Challenges, Solutions, Applications, 2004, S. 108ff)

Die folgende Abbildung 13 zeigt zwei unterschiedliche Änderungen in zwei Kindprozessen. Steht nur die dargestellte Kontrollperspektive zur Verfügung, lässt sich bei einem Zusammenfassen beider Prozesse nicht eindeutig sagen, ob B vor C oder C vor B eingefügt werden muss.

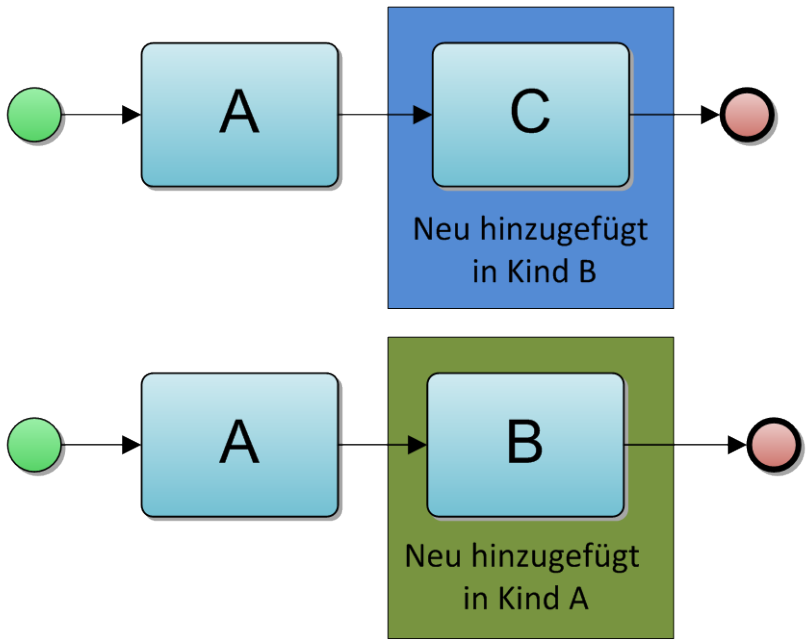

Abbildung 13: Überlappende Änderungen an zwei Kindprozessen

Quelle: Eigene Erstellung

Ein vergleichbares Problem findet sich, wenn das gleiche Prozesselement, vom Vaterprozess ausgehend, in A und in B jeweils unterschiedlich adaptiert wurde. Die im Vater vorhandene Aktivität X ist zwar in beiden Kindprozessen enthalten, jedoch wurden die Attribute des Prozesselements in A und B unterschiedlich verändert. Auch bei diesem Fall muss eine endgültige Entscheidung von einem Menschen gefällt werden. Als schnelle Lösung dieses Problems kann eine der beiden Möglichkeiten für die weitere Verwendung gewählt werden – das betreffende Prozesselement komplett ignorieren oder das Originalement aus dem Vaterprozess entnehmen. Aufwändiger ist es, wenn ein Benutzer beide Möglichkeiten analysiert und händisch zusammenführt. Dieser Lösungsweg verspricht bessere Ergebnisse als die zuvor vorgestellten, da es hier dem Anwender möglich ist, beide Änderungen zusammenzufassen, sodass z.B. Fehlerbehebungen nicht verloren gehen.

Auch wenn das Zusammenführen der Änderungen selbst leicht möglich ist, weil z.B. nur unabhängige Änderungen vorhanden sind, garantiert dies nach dem Zusammenfügen trotzdem nicht die Einhaltung der gewünschten Korrektheitskriterien. So kann aus dem Vereinigen zweier unabhängiger Änderungen leicht eine Verklemmung entstehen, wie es in Abbildung 14 dargestellt wird. Gezeigt wird hier das Ergebnis der Zusammenfassung zweier Kindprozesse: Kindprozess A hat hierbei die grün gekennzeichnete Kante hinzugefügt; vom Kindprozess B stammt die blau gekennzeichnete Kante. Jeder Prozess (Vater- und Kindprozess) für sich selbst gesehen ist korrekt, auch die Änderungen sind nicht überlappend ausgeführt. Beim Zusammenführen der Prozesse entsteht jedoch durch die markierten Kanten eine Verklemmung. (Vgl. Rinderle, Reichert, & Dadam, On Dealing with Structural Conflicts between Process Type and Instance Changes, 2004, S. 285ff)

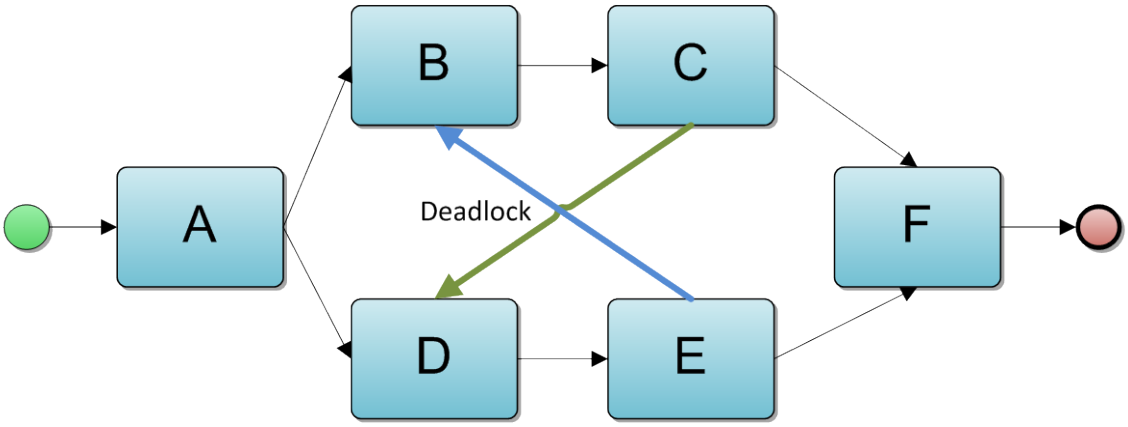

Abbildung 14: Unabhängige Änderungen führen zu einer Verklemmung

Quelle: Eigene Erstellung in Anlehnung an (Rinderle, Reichert, & Dadam, On Dealing with Structural Conflicts between Process Type and Instance Changes, 2004, S. 283)

3.3 Bestimmen von Unterschieden zwischen Prozessmodellen

Die bis jetzt besprochenen Probleme und Lösungen setzten alle voraus, dass bereits bekannt ist, welche Änderungen an Prozessen im Vergleich mit einem anderen Prozess durchgeführt wurden. Ist dies nicht bekannt, wäre nicht zu erkennen, dass Änderungen sich überschneiden oder auch welche Änderungen durchgeführt werden müssen, um alte Instanzen auf eine neue Prozessdefinition zu überführen. Um dieses Problem anzugehen, werden nun kurz zwei Herangehensweisen vorgestellt, von denen eine auf dem Longest Common Subsequence (LCS) Algorithmus basiert, während es sich bei der anderen um eine mengenorientierte Lösung handelt. Insbesondere die zweite Methode, welche mengenorientiert arbeitet, wird für die im Rahmen dieser Arbeit erstellten Algorithmen in einer adaptierten Form Verwendung finden. Neben diesen beiden Methoden, welche direkt mit den Prozessdefinitionen arbeiten, wird als dritter Lösungsansatz auch noch ein Weg vorgestellt, der mit Änderungsereignissen, also den gespeicherten Aktionen, die zur Änderung der originalen Prozessdefinition verwendet wurden, arbeitet.

3.3.1 Bestimmung der Differenzen mittels Longest Common Subsequence

Beim Longest Common Subsequence Problem soll die längstmögliche Sequenz, die zumindest auf zwei anderen Sequenzen basiert, ermittelt werden. (Vgl. Cormen, Leiserson, Rivest, & Stein, 2009, S. 390ff) Welche Informationen genau eine Subsequenz bilden, hängt hierbei vom konkreten Problem ab. Eine Subsequenz kann aus einer Folge von Ribonukleinsäuresträngen (RNA) (Vgl. Bereg, Kubica, Wale, & Zhu, 2005, S. 1ff), Textzeilen in einer Datei (dies wird z.B. oft in Quellcode-

verwaltungssystemen wie Subversion – SVN – verwendet) (Vgl. Kim & Notkin, 2006, S. 2) oder auch – passend zu dieser Arbeit – aus einer Reihe von Prozesselementen bestehen.

Um ein besseres Verständnis für das Problem zu erhalten, eignet sich folgendes Beispiel, in welchem es gilt, aus der Sequenz A, die aus den Elementen a b c d e besteht, und aus der Sequenz B, welche aus den Zeichen a c f gebildet wird, die längstmögliche Subsequenz zu bilden. Bei den beiden Sequenzen A und B entspricht die LCS den Elementen a und c. Beim Umgang mit Prozessen lassen sich mit dieser Methode beispielsweise diejenigen Elemente herauslesen, welche während der Erzeugung einer neuen Prozessversion aus dem Vaterprozess entfernt worden sind.

Ein Vorteil dieses Ansatzes ist die weite Verbreitung dieser Technologie. So sind verschiedenste Applikationen vorhanden, welche LCS erzeugen und auch so genannte Differenzinformationen (Diffs) erstellen können. Bei Diffs handelt es sich um eine Sammlung von Werten, welche die Unterschiede zwischen verglichenen Daten, hier z.B. Prozessdefinitionen, angibt. Damit allerdings bereits vorhandene Tools wie z.B. Diff oder diff3 praktikabel eingesetzt werden können, ist besonders auf den Aufbau des für die Speicherung der Prozessdefinitionen verwendeten Dateiformats zu achten. Probleme bereitet hier insbesondere eine Reihenfolgenveränderung der eingetragenen Datensätze. Dies kann dazu führen, dass Elemente als neu bzw. gelöscht erkannt werden, obwohl diese "nur" eine neue Position im Prozess erhalten haben. Dieses Problem wird in der Tabelle 4 verdeutlicht. Die dargestellte Auswertung in Form eines Diffs wurde hierbei mit den GNU Diffutils in Version 3.2-2 unter Verwendung der Standardeinstellung erstellt. Es zeigt sich das weiter oben angesprochene Problem, dass verschobene, aber inhaltlich unveränderte Prozesselemente als gelöscht und neu eingefügt erkannt werden, was die Anzahl der ermittelten Änderungsoperationen erhöht. In der Ergebnisspalte stehen die Buchstaben zwischen den Zahlen für hinzugefügt „a“, gelöscht „d“ und verändert „c“. Die Zahl vor dem Buchstaben ist die Zeilennummer im Originaldokument, während die Zahl danach die Zeilennummer im modifizierten Dokument darstellt. Danach folgende links gerichtete Spitzklammer kennzeichnen vom modifizierten Dokument kommende Änderungen, während rechts gerichtete Spitzklammern vom Originaldokument kommende Informationen bezeichnen. Drei Bindestriche markieren unveränderte Bereiche zwischen den erkannten Unterschieden.

Problematisch ist auch, dass die Subsequenzen nicht unbedingt eindeutig sein müssen; daher kann es mehrere Sequenzen derselben Länge, allerdings mit unterschiedlichem Aufbau und Zusammensetzung geben, die alle ein gültiges LCS für die gleichen Daten darstellen.

Das folgende Beispiel zeigt einen Nachteil eines auf LCS basierenden Ansatzes. So werden die beiden Testdatensätze 1 und 2 miteinander verglichen. Beide Datensätze unterscheiden sich vom Inhalt kaum, da dieser sich bei beiden nur in der Reihenfolge der Einträge variiert. Mittels LCS ist die Erkennung dieser Verschiebungen allerdings nicht möglich, sodass die Unterschiede nicht als Verschiebung, sondern als unabhängige Lösch- und Einfügeoperationen erkannt werden. Es müsste hier also zusätzlicher Nachbearbeitungsaufwand investiert werden, um die Änderung der Reihenfolge zu erkennen.

Testdatensatz 1	Testdatensatz 2	Ergebnis-Diff
Task 1	Task 1	2c2
Task 2	Task 7	< Task 2
Task 4	Task 4	---
Task 5	Task 5	> Task 7
Task 7	Task 2	5c5
		< Task 7

		> Task 2

Tabelle 4: Differenz ermittelt durch einen LCS-Algorithmus

Quelle: Eigene Erstellung

3.3.2 Bestimmung der Differenzen mittels Mengenorientierung

Bei einem mengenorientierten Ansatz werden die einzelnen Elemente eines Prozesses in Mengen zusammengefasst und hierauf dann typische Mengenoperationen, Differenzmenge, Durchschnittsmenge etc. angewandt. Exemplarisch werden hier zwei Beispielprozesse, bestehend aus Knoten N und Kanten E, P:=(N, E) verwendet, welche folgende Mengen bilden N1 ={7; 8; 9; 10} und N2 ={4; 5; 6; 7}. Es gilt, diejenigen Elemente zu finden, welche in beiden Prozessversionen unverändert vorhanden sind. Dies kann durch die Bildung einer Durchschnittsmenge von N1 und N2 erreicht werden. In diesem Beispiel würde das eine Menge N1∩N2 = {7} ergeben, woraus abgeleitet werden kann, dass das Prozesselement 7 beispielsweise bei einer Vereinigung der Beispielprozesse ohne weitere Verarbeitungsschritte in den Ergebnisprozess übernommen werden kann. Die Abbildung 15 verdeutlicht diesen Vorgang. Ein zusätzlicher Vorteil dieses Ansatzes ist, dass sich die unterschiedlichen Mengenelemente gut als Objekt in einer objektorientierten Programmiersprache (OOP) darstellen und verarbeiten lassen. Viele Programmiersprachen wie z.B. Java stellen bereits vordefinierte Operationen zum Vereinigen einer Menge (union) oder auch für die Bildung einer Differenzmenge (removeAll) in ihren vorgefertigten Implementierungen einer Elementliste zur Verfügung.

Mengenoperationen wie Durchschnitt, Vereinigung oder Differenz bilden nur die grundlegenden Voraussetzungen für die verschiedenen in dieser Arbeit erstellten Algorithmen. (Vgl. Loskiewicz-Buczak & Uhrig, 1995, S. 1ff) Komplexe Algorithmen, welche beispielweise dafür verwendet werden können, verschiedene Ausführungspfade zu bilden oder zusammenhängende Elemente zu identifizieren, können nicht allein mit Mengenoperationen realisiert werden.

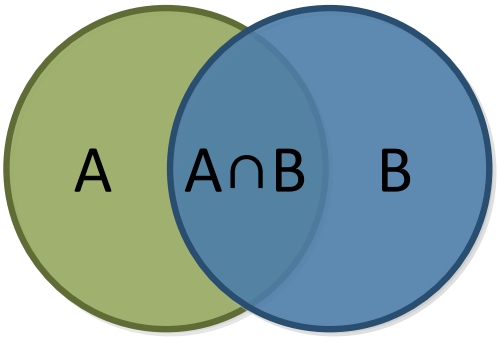

Abbildung 15: Beispiel für eine Mengenoperation zur Bestimmung unveränderter Elemente

Quelle: Eigene Erstellung

3.3.3 Bestimmung der Differenzen mittels Ereignisprotokollen/Änderungslogs

Sollen Unterschiede zwischen Vater- und Kindprozessen ausgewertet und analysiert werden, können neben den zuvor genützten Ansätzen auch die während der Modifikation erzeugten Änderungsprotokolle genützt werden. Da diese bereits alle Änderungen beinhalten, welche vorgenommen wurden, müssen sie nicht erst aus den bereits geänderten Prozessen extrahiert werden. Dieser Ansatz eignet sich jedoch nicht für diese Arbeit, da hier davon ausgegangen wird, dass Ereignisprotokolle nicht zur Verfügung stehen.

Vor der Verwendung dieser Ereignisprotokolle sollten diese bereinigt, also unnötige Einträge entfernt werden. Dies erleichtert und beschleunigt die spätere Weiterverarbeitung und dient zur Erzeugung von so genannten Nettoprotokollen. (Vgl. Gottschalk, Aalst, & Jansen-Vullers, Mining Reference Process Models and their Configurations, 2008, S. 267) Wie in Abschnitt 17 angesprochen, ist es hierbei z.B. möglich, einander aufhebende Operationen aus der Menge der Ereignisse zu entfernen. Hier ist beispielsweise das Hinzufügen und anschließende Löschen eines Elements zu nennen. Diese beiden Operationen würden einen neutralen Nettoeffekt ergeben, da die Einfügeoperation von der späteren Löschoperation aufgehoben wird. Allerdings bestehen auch noch andere Möglichkeiten der Bereinigung, bei welcher Ereignisse nur teilweise übernommen werden. Wenn z.B. nach mehreren Änderungsereignissen ein abschließendes Löschereignis auf das gleiche Element auftritt, können die vorhergehenden Änderungsereignisse entfernt werden. (Vgl. Aalst & Weijters, Process mining: a research agenda, 2003, S. 7-12)

Für eine genauere Beschreibung der Bereinigungsmöglichkeiten wird die folgende Tabelle verwendet. Die Beispiele basieren hierbei auf den in Abschnitt 3.1.1 vorgestellten Operationen.

Änderung	Effekt	Beispiel
Einfügen(A);Löschen(A)	Das Element wird eingefügt und anschließend gelöscht. Der Nettoeffekt ist deshalb neutral.	InsertTask(A) DeleteTask(A)
Modifizieren(A);Löschen(A)	Ein existierendes Element wird modifiziert und später gelöscht. Der Nettoeffekt kann daher auf die Löschung redu-	UpdateTask(A) DeleteTask(A)

	ziert werden.	
Löschen(A);Einfügen(A)	Ein existierendes Element wird entfernt und später wieder eingefügt. Wird exakt das gleiche Element wieder eingefügt, ist die Auswirkung neutral. Wird ein Element mit veränderten Attributen eingefügt, können die Operationen in eine Modifikation zusammengefasst werden.	DeleteTask(A) InsertTask(A)
Modifizieren(A);Modifizieren(A)	Auf ein existierendes Element werden mehrere Modifikationen angewandt. Falls das zweite Update den Zustand vor der ersten Modifikation wiederherstellt, ist das Ergebnis neutral. Falls die zweite Modifikation die Änderungen der ersten überschreibt, kann die erste wegfallen. Falls beide Modifikationen unterschiedliche Bereiche ändern, können diese in eine einzelne Operation zusammengefasst werden.	UpdateTask(A) UpdateTask(A)
Einfügen(A);Modifizieren(A)	Ein neues Element wird eingefügt und später modifiziert. Hier ist es möglich, die Einfügeoperation mit der Modifikation zusammenzufassen, sodass eine einzelne Einfügung ausreicht.	InsertTask(A) UpdateTask(A)

Tabelle 5: Bereinigung von Ereignisprotokollen bei unterschiedlichen Ereignissen

Quelle: (Vgl. Rinderle, Jurisch, & Reichert, On Deriving Net Change Information From Change Logs – The DELTALAYER-Algorithm –, 2007, S. 4-6)

Nach der Bereinigung, welche zur Reduzierung des Rauschens in den Protokollen dient, werden die verfügbaren Änderungsprotokolle zusammengefasst. Prinzipiell wäre dies durch ein einfaches Zusammenfügen der beiden Protokolle realisierbar. Dies wird jedoch durch die Möglichkeit von überlappenden Änderungen erschwert. Auch können konkurrierende Änderungen (siehe Abschnitt 3.2.5) auftreten, welche während der Bereinigung nicht aufgelöst werden konnten. Dieses Problem betrifft nicht nur die Verarbeitung der Ereignisprotokolle, sondern alle drei vorgestellten Verfahren.

3.4 Abbildung von Ereignissen mittels MXML und XES

Die Art und Weise, in welcher Ereignisse bezüglich der Änderung bzw. Ausführung von Prozessen, wie sie im zuvor vorgestellten Ansatz benötigt werden, von verschiedenen Systemen gespeichert werden, unterscheidet sich in verschiedenen Softwareplattformen deutlich. Um dieses Problem zu lösen bzw. eine Vereinheitlichung zu ermöglichen, wurde von der Forschungsgruppe „Architecture of Information Systems“ an der Universität Eindhoven das sogenannte Mining XML (MXML)-Format entworfen. Dieses XML-basierte Format dient der vereinheitlichten Speicherung von Ereignissen, welche bei der Prozessausführung aufgezeichnet wurden. (Vgl. Dongen & Aalst, A Meta Model for Process Mining Data, 2005, S. 1-12)

MXML wurde erstmals im Jahre 2005 von Van Dongen and Van der Aalst in einem Paper namens "A Meta Model for Process Mining Data" beschrieben.

Ein Beispiel für ein in MXML definiertes Ereignisprotokoll ist in der folgenden Programmauflistung zu sehen. Es zeigt die Definition eines einzelnen Prozesses, bei welchem nur ein einzelnes Event aufgezeichnet wurde. Der abgebildete Prozess selbst bzw. die dargestellte Prozessinstanz ist in diesem Beispiel eine Bestellung. Es ist zu beachten, dass jede Prozessinstanz eine eindeutige ID erhalten muss. In diesem Fall ist es die ID "Bestellung1". Die „description“ wiederum dient nur als zusätzliche Hilfe und kann dazu herangezogen werden, die Prozessinstanz genauer zu beschreiben. Die während der Prozessausführung aufgezeichneten Ereignisse werden in "AuditTrailEntry"-Elementen abgelegt. Jedes dieser Elemente beinhaltet ein "WorkflowModelElement", welches dazu dient zu spezifizieren, welche Aktivität das hier aufgezeichnete Event ausgelöst hat. Der darunter ersichtliche "EventType" zeigt den Status der Aktivität. In diesem Beispiel ist der Status als "complete" festgelegt, was besagt, dass die Aktivität abgeschlossen ist. Andere mögliche Werte für dieses Feld wären suspend oder resume, welche angeben, dass die Aktivität pausiert bzw. später fortgesetzt worden ist. Zusätzlich wird hier auch noch ein optionales Feld namens „Originator“ verwendet, welches denjenigen Nutzer beschreibt, der die Aktivität ausgeführt hat. Hierbei muss es sich nicht unbedingt um einen menschlichen Nutzer handeln, sondern es kann auch ein rechnergestütztes System benannt werden. Mit dem darunterliegenden Feld, welches ebenfalls optional ist und die Bezeichnung Timestamp trägt, wird der Zeitpunkt der Eventerzeugung beschrieben. Hier handelt es sich um den 24. März 2012 um 11 Uhr 26 und 12 Sekunden, wobei Greenwich Mean Time (GMT) +2 verwendet wurde. Ähnlich dem Datensegment im Bereich "ProzessInstance" kann auch in dem durch "AuditTrailEntry" beschränkten XML-Bereich ein Datensegment vorhanden sein.

```
<Process>
<ProcessInstance id="Bestellung1" description=”InstanzFürBestellung”>
<Data>
      <Attribute name=”Bestellungsgesamtsumme”>1337</Attribute>
</Data>
<AuditTrailEntry>
      <WorkflowModelElement>Bestellung aufneh-
      men</WorkflowModelElement>
      <EventType>complete</EventType>
      <Originator>Kristof</Originator>
      <Timestamp>2012-03-24T11:26:12.000+02:00</Timestamp>
      <Data>
      <Attribute name=”Produktanzahl”>12</Attribute>
      <Attribute name=”Besteller”>Thombas</Attribute>
      <Attribute name=”Anbieter”>CalendarMobile Inc.</Attribute>
      <Attribute na-me=”GenwünschtLieferungAm”>2012-04-
      24T11:05:10.000+02:00</Attribute>
      </Data>
</AuditTrailEntry>
...
</ProcessInstance>
...
</Process>
```

Programmausdruck 1: Beschreibung von Ereignissen mittels MXML

Quelle: Eigene Erstellung

Ein Problem vom MXML ist, dass alle in diesem Format gespeicherten Informationen als Strings behandelt werden. Es ist also nicht möglich, Werte so zu speichern, dass Informationen zum Verwendeten Datentyp erhalten bleiben, wie z.B., dass ein Wert ein Datum in einem definierten Format darstellt; hierdurch gehen zahlreiche semantische Informationen verloren. Dieses Problem wird von der IEEE Task Force Process Mining angegangen, in dem derzeit ein neues Format namens eXtensible Event Stream (XES) entwickelt wird. (Vgl. Günther, 2009, S. 1-22)

Ein Ziel dieses Formats ist eine vereinfachte Darstellung der Information, um eine leichtere Verarbeitung und Generierung des Formats zu erreichen. Weitere Ziele finden sich in der gewünschten Flexibilität und Erweiterbarkeit, welche darauf abzielen, den Standard konstant weiterzuentwickeln und als Allzweckformat für die Speicherung von Ereignisinformationen zu etablieren. Darin inkludiert ist der Wunsch, Erweiterungen für spezifische Domänen zu integrieren. Ein abschließendes Ziel dieses neuen Formats ist es, die Ausdrucksfähigkeit gegenüber dem älteren MXML zu erhöhen. Dies wird durch die strenge Typisierung der Felder, siehe oberer Absatz, erreicht.

Zu Verdeutlichung der Unterschiede zwischen MXML und XES dient folgendes Beispiel, welches die gleichen Informationen wie beim MXML-Beispiel (siehe Programmausdruck 1: Beschreibung von Ereignissen mittels MXML) im XES-Stil darstellt.

```
<trace>
  <string key="concept:name" value="Bestellung1"/>
  <float key="order:Bestellungsgesamtsumme" value="1337"/>
  <event>
    <string key="concept:name" value="Bestellung aufnehmen"/>
    <string key="lifecycle:transition" value="Complete"/>
    <string key="org:resource" value="Kristof"/>
    <date key="time:timestamp" value="2012-03-24T11:26:12.000+02:00"/>
    <float key="order:currentValue" value="1337"/>
    <string key="details" value="BestellungDetails">
      <int key="Produktanzahl" value="12"/>
      <string key="Besteller" value="Eric"/>
      <string key="Anbieter" value="FluxiInc."/>
      <date key="GenwünschtLieferungAm" valu-
      e="2012-04-24T11:05:10.000+02:00"/>
    </string>
  </event>
</trace>
```

Programmausdruck 2: Beschreibung von Ereignissen mittels XES

Quelle: Eigene Erstellung

Eine Realisierung von Business Process Mining unter Verwendung der oben beschriebenen Formate lässt sich z.B. mit der Software ProM 6.1 der Process Mining Group erreichen. (Vgl. Aalst, Reijersa, Weijtersa, & Dongena, 2006, S. 5ff) Eine praktische Darstellung eines möglichen Miningansatzes, basierend auf Prozesseventlogs, ist in der Arbeit „ProM 4.0: Comprehensive Support for Real Process Analysis“ von Aalst et al. ersichtlich. (Vgl. Aalst, et al., 2007)

3.5 Diskussion der unterschiedlichen Ansätze

Es folgt nun eine abschließende Betrachtung der drei Ansätze, welche mit einer kurzen Übersichtstabelle abgeschlossen wird. Der auf den LCS basierende Ansatz besticht durch seine Einfachheit. Die Toolunterstützung für verschiedenste Datenquellen ist hierbei groß und der Algorithmus selbst bereits sehr ausgereift, weit verbreitet und durch seine große Bedeutung schon umfassend analysiert und optimiert. Verschiedene Programmiersprachen z.B. PHP: Hypertext Preprocessor (PHP) (Vgl. The PHP Group, 2012) bringen außerdem Implementierungen dieses Ansatzes mit. Als Nachteil ist zu bewerten, dass dieser Ansatz stark von dem für die Abbildung der Prozesse verwendeten Dateiformat abhängt und aufgrund der Arbeitsweise der Diffprogramme die Reihenfolge der dortigen Elemente wichtig ist. (Vergleiche hierzu Abschnitt 3.3.1) Dies führt zu dem in diesem Kapitel beschriebenen Problem, dass Änderungen wie die Verschiebung von Elementen „falsch“ erkannt werden.

Der mengenorientiere Ansatz hat vergleichbare Vorteile wie der LCS-Ansatz. Die Unterstützung bei den Programmiersprachen ist ebenfalls gut. So sind mengenorientierte Datenstrukturen in Java (Vgl. Oracle Corporation, 2011) oder .NET (Vgl. Microsoft Corporation, 2012) standardmäßig vorhanden. Gegenüber LCS zeichnet er sich dadurch aus, dass die Elemente in willkürlicher Anordnung verarbeitet werden können.

Mittels Ereignisprotokollen kann leicht eine umfassende Datenbasis erhoben werden. Diese umfassende Basis führt, verglichen mit den anderen beiden Lösungen, zu einem erhöhten Berech-

nungsaufwand. Weiters müssen diese Protokolle auch erhoben werden\zur Verfügung stehen bzw. Lösungen entwickelt werden, um diese zu erheben. Beide Eigenschaften sind problematisch und können deshalb als Nachteile gewertet werden. Gleichzeitig liegt darin allerdings auch eine Stärke des Ansatzes, können hierdurch doch viele Informationen gewonnen werden, welche zur Verbesserung der ermittelten Ergebnisse verwendet werden können.

Der wesentlichste Nachteil liegt jedoch darin, dass Ereignisprotokolle nicht eindeutig sein müssen, da diese nur die jeweiligen Ausführungshistorie verschiedener Prozessinstanzen beinhalten. Hierbei sind aufgrund der Möglichkeit verschiedene Ausführungspfade zu wählen. Es ist nicht mit endgültiger Sicherheit möglich, Aussagen über Prozessmodifikationen zu treffen. Weiters sind Ausführungspfade auch nicht absolut eindeutig. So kann eine parallele Ausführung von Prozessschritten zu den gleichen Logeinträgen wie eine sequenzielle Ausführung führen. Daher ist es immer notwendig, als zusätzliche Informationsquelle auch Änderungslogs (Vgl. hierzu Abschnitt 3.3.3) heranzuziehen. (Vgl. Li, Reichert, & Wombacher, On Measuring Process Model Similarity based on High-level Change Operations, 2008, S. 253f)

Ansatz	Vorteile	Nachteile
LCS	einfach, wissenschaftlich umfangreich behandelt	stark abhängig von der Darstellung der Informationen
Mengenorientierung	einfach, von der Informationsdarstellung unabhängig	
Event- und Änderungslogs	zahlreiche Daten erfassbar, hohe Lösungsqualität	aufwändig

Tabelle 6: Gegenüberstellung verschiedener Ansätze zur Differenzermittlung

Quelle: Eigene Erstellung

Allen Ansätzen gemein ist, dass eine reine Implementierung einer der Methoden – also z.B. nur der Einsatz der von der Mengenorientierung bereitgestellten Möglichkeiten – keine optimalen Ergebnisse liefern kann. So ist es z.B. durch reine Mengenoperationen nicht möglich, die Verbindungen von Prozesselementen über mehrere aufeinander folgende Elemente hinweg zu verfolgen und zu analysieren.

3.6 Weiterführende Literatur

Als ausführlichere Vorbereitung der den Algorithmen gewidmeten Kapitel empfiehlt sich eine genauere Betrachtung der hier vorgestellten Analyseansätze. So stellt Walter F. Tichy einen optimierten Algorithmus zur Ermittlung der LCS vor, welche die Effizienz hinsichtlich der Minimierung der benötigten Rechenzeit verbessert (im Vergleich zum originalen Hunt-McIlroy Algorithmus (Vgl. Hunt & McIlroy, 1976)). (Vgl. Tichy, 1983)

Weiters ermöglicht die Arbeit von Li et al. einen guten Einblick in verschiedene Ansätze. Hierbei werden diverse Szenarien und Lösungsmöglichkeiten vorgestellt, wobei bei allen ein Bezug auf dem Umgang mit Geschäftsprozessen genommen wird. Interessant ist hierbei die Analyse von Algorithmen, welche dazu dienen, basierend auf mehreren Kindprozessen einen unbekannten Vaterprozess zu ermitteln. Dies stellt eine Umkehrung der in Kapitel 6 behandelten Anforderungen dar. (Vgl. Li, Reichert, & Wombacher, Mining Business Process Variants: Challenges, Scenarios, Algorithms, 2011)

4 Intermediate Format

Bereits am Beginn dieser Arbeit bestand der Wunsch, die hier entwickelten Algorithmen von bestimmten Modellierungssprachen wie EPK oder BPMN sowie herstellerspezifischen Modellierungsplattformen unabhängig zu gestalten. Um dies zu ermöglichen, wird in diesem Kapitel ein eigens für diese Arbeit entwickeltes Datenformat vorgestellt. Dieses hat den Vorteil, von bestehenden Formaten abstrahiert arbeiten zu können.

Um die Anforderungen der wissenschaftlichen Fragestellung zu erfüllen, muss das Datenformat maschinen- und menschenlesbar sein. Es soll mit einfachen Mitteln gelesen und erzeugt werden können. Die Darstellung aller im Kapitel 2 beschriebenen Elemente des Metamodells muss von dem Format unterstützt werden. Zusätzlich soll für jedes Element die Möglichkeit gegeben sein, Metainformationen verlustfrei abzubilden.

Dies wird durch einen mehrstufigen Prozess ermöglicht. In diesem werden zuerst die vorhandenen Prozessdaten in das in diesem Kapitel beschriebene Datenformat umgewandelt. Je nach Datenquelle muss hierbei ein eigener Wandler erzeugt werden. Danach werden die Daten im hier beschriebenen Format an die Implementierung der Algorithmen übergeben und bearbeitet. Das Resultat, insbesondere das des in Kapitel 6 beschriebenen Vereinigungsalgorithmus, wird dann ebenfalls in diesem generischen Datenformat zurückgeliefert. Basierend auf diesen Informationen kann dann wieder eine Transformation zurück in die ursprüngliche Prozessbeschreibungsnotation durchgeführt werden.

Dieser Ansatz erlaubt es, dass die Algorithmen vollständig unabhängig von den Datenquellen implementiert werden können.

Eine grafische Aufbereitung der Abläufe ist in der folgenden Grafik ersichtlich. Gezeigt wird die Möglichkeit, verschiedene Formate mittels unterschiedlicher Transformatoren aufzubereiten und umzuwandeln, sodass die eigentlichen Algorithmen unverändert weiterbenutzt werden können.

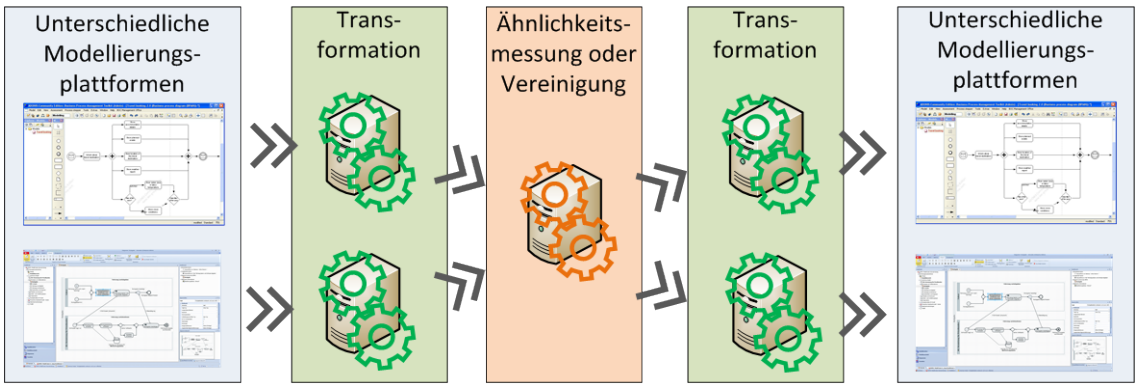

Abbildung 16: Unabhängigkeit durch intermediate Format

Quelle: Eigene Erstellung

4.1 Aufbau des Formats

Das Format basiert in seinen grundlegenden Eigenschaften auf dem Comma-Separated Values (CSV) Format, welches im Request for Comments (RFC) 4180 im Jahr 2005 genauer definiert wurde (Vgl. Shafranovich, 2005). Das Format dient dem Spezifizieren der verschiedenen in einem Modell vorhandenen Elemente, wobei hierbei keine spezielle Ordnung oder Reihenfolge der Elemente eingehalten werden muss.
Zur Beschreibung des Formats findet hier zuerst eine Bearbeitung der grundlegenden Probleme wie der Darstellung spezieller Zeichen statt. Anschließend wird dargelegt, wie unterschiedliche Modellelemente, wie eine Aktivität, innerhalb des Formats abgebildet werden können.

Die grundlegende Struktur der Daten besteht immer aus einem Datum und einem darauffolgenden Trennzeichen. Als Trennzeichen wird der Strichpunkt (;) verwendet (hexadezimale Zeichencodierung 3B). Die Daten werden zeilenweise organisiert, wobei Werte einer Zeile immer als Einheit betrachtet werden. Vor dem ersten und nach dem letzten Datum einer Zeile wird kein Trennzeichen eingefügt.

Das folgende Beispiel visualisiert das Prinzip, wobei aaa, bbb und ccc als einzelne Werte zu interpretieren sind.

```
aaa;bbb;ccc
```

Als Zeichensatz wird UTF-8 (Universal Character Set Transformation Format) verwendet. Falls das Trennzeichen in den Daten eingefügt werden soll, muss dieses mit einem Maskierungszeichen versehen werden. Dies geschieht durch einen Schrägstrich (/, hexadezimaler Zeichencode 2F) vor dem eigentlichen Strichpunkt. Das Trennzeichen verliert hierdurch seine Steuerfunktion. Ein einzelner Schrägstrich, welcher nicht als Maskierungszeichen genützt wird, kann durch das Einfügen von zwei aufeinanderfolgenden Schrägstrichen (//) dargestellt werden. Ein Zeilenumbruch erfolgt durch ein einzelnes „new line“ Symbol (/n, hexadezimale Zeichenkodierung A).

4.1.1 Umsetzung der Modellelemente

Basierend auf den in Kapitel 2 ermittelten modellierungssprachenübergreifenden Basiselementen wurden die folgenden Darstellungsformen definiert. Diese dienen dazu, diese Basiselemente in textueller maschinen- und menschenlesbarer Form abzubilden. Um Datenverluste während der Transformation zu vermeiden, erhalten alle Elemente ein spezielles Datenfeld angehängt. Dieses dient dazu, beliebige Informationen zu speichern, welche der Algorithmus zwar nicht benötigt, die allerdings auch nicht verloren gehen sollen. Hiermit können unter anderem von der erzeugenden Plattform generierte Metainformationen gespeichert und verlustfrei übertragen werden.

Ein Datensatz, der ein Element beschreibt, besteht immer aus einer einzelnen Zeile innerhalb der Datei. Zu Beginn der Zeile wird immer der Typ des Datensatzes angegeben. Danach folgt ein eindeutiger Name bzw. ein eindeutiger Universally Unique Identifier (UUID) des Elements. Darauf aufbauend können weitere elementspezifische Informationen angehängt werden. Abschließend wird der Tag angehängt, welcher die zuvor beschriebenen, nicht zur Verarbeitung vorgesehenen Zusatzinformationen abbilden kann.

```
Type;UUID;[Datensatzspezifisch];Tag
```

Die folgenden Elemente werden von dem Datenformat und damit auch von den in dieser Arbeit entwickelten Algorithmen unterstützt.

4.1.2 Aktivität

Eine Aktivität bekommt den Type „TASK“ zugeordnet. Als zusätzliche Information ist eine Beschreibung der Aktivität enthalten, welche genützt werden kann, um die während der Ausführung durchzuführenden Operationen genauer zu definieren. Das Beispiel stellt hier eine Aktivität mit der Identifikationsbezeichnung (ID) Task1 und der Beschreibung "Patient untersuchen" dar.

Aufbau

```
TASK;UUID;Beschreibung;Tag
```

Beispiel

```
TASK;Task1;Patient untersuchen;
```

4.1.3 Kante

Eine Kante hat den Type „ARC“. Zusätzlich werden auch noch die eindeutigen ID des Start- und des Zielelements der gerichteten Kante gespeichert. Das Beispiel mit der UUID „Arc1“ zeigt eine Kante zwischen „Task1“ und „Task2“.

Aufbau

```
ARC;UUID;UUID Startelement;UUID Endelement;Tag
```

Beispiel

```
ARC;Arc1;Task1;Task2;
```

4.1.4 Gateway

Ein Gateway bekommt immer den Type „GATEWAY“ zugewiesen. Da von Gateways verschiedene Arten wie XOR, OR oder AND existieren, kann diese Unterart ebenfalls angegeben werden. Im Bereich der Untertypen wird einer der folgenden entsprechenden Typen eingesetzt: XOR Split, OR Split, AND Split, Parallel oder Join.

Aufbau

```
GATEWAY;UUID;[Gateway Type];Tag
```

Beispiel

```
GATEWAY;Gateway1;AndSplit;
```

4.1.5 Start

Das Start-Element hat die Typedefinition „START“. Ansonsten sind nur die weiter oben beschriebenen Standardelemente vorhanden.

Aufbau

```
START;UUID;Tag
```

Beispiel

```
START;Start1;
```

4.1.6 Ende

Das Ende-Element hat den gleichen Aufbau wie das Start-Element. Der einzige Unterschied ist, dass als Type „END“ verwendet wird.

Aufbau

```
END;UUID;Tag
```

Beispiel

```
END;End1;
```

Zum besseren Verständnis für das Format wird das folgende Beispiel herangezogen. Die Abbildung 17 zeigt hierzu eine in BPMN erfolgte Abbildung eines Prozesses. In diesem Prozess wird die Erstellung einer Quartalsübersicht für ein Unternehmen verkürzt dargestellt. Hierzu werden zuerst die dazu notwendigen Daten erhoben und diese mittels verschiedener Berechnungen weiterverarbeitet. Die so ermittelten Ergebnisse werden überprüft. Fällt die Prüfung positiv aus, werden Berichte erstellt und präsentiert. Ansonsten wird die Berechnung mit veränderten Parametern erneut durchgeführt.

Anschließend erfolgt die Darstellung des gleichen Prozesses mittels des hier beschriebenen Formates.

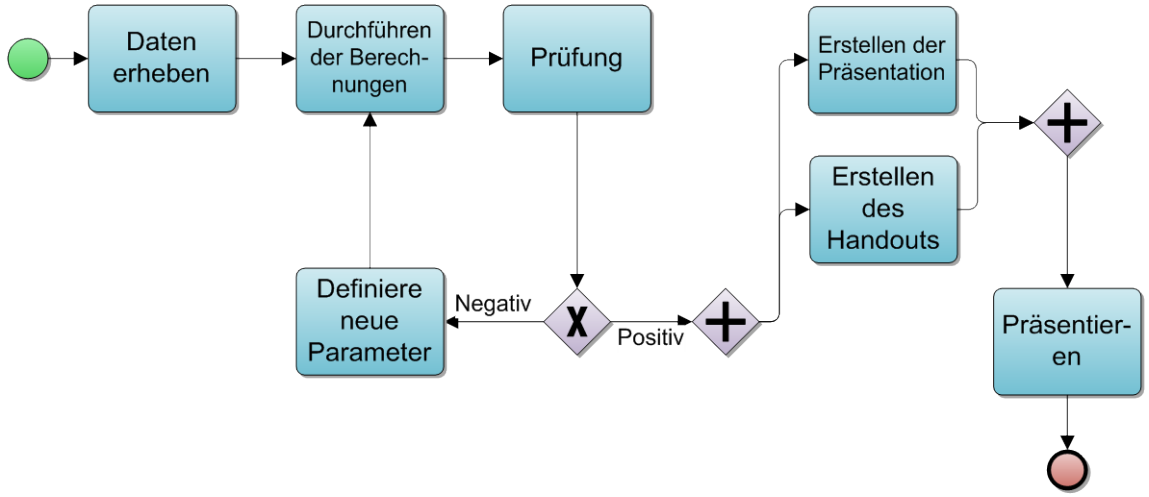

Abbildung 17: BPMN-Beispiel für eine Transformation ins intermediate Format

Quelle: Eigene Erstellung

Im Folgenden ist der obige Prozess mittels des hier beschriebenen intermediate Formats dargestellt. Zu beachten ist, dass die Reihenfolge der Elemente nicht definiert ist und daher beliebig gewählt werden kann.

```
START;Start1;
TASK;Task1;Daten erheben;
ARC;Arc1;Start1;Task1;
TASK;Task2;Durchführung der Berechnungen;
ARC;Arc2;Task1;Task2;
TASK;Task3;Prüfung;
ARC;Arc3;Task2;Task3;
GATEWAY;Gateway1;XORSplit;
ARC;Arc4;Task3;Gateway1;
TASK;Task4;Definiere neue Parameter;
ARC;Arc5;Gateway1;Task4;
ARC;Arc6;Task4;Task2;
GATEWAY;Gateway2;Parralel;
ARC;Arc7;Gateway1;Gateway2;
TASK;Task5;Erstellen der Präsentation;
TASK;Task6;Erstellen des Handouts;
ARC;Arc8;Gateway2;Task5;
ARC;Arc9;Gateway2;Task6;
GATEWAY;Gateway3;Join;
ARC;Arc10;Task5;Gateway3;
ARC;Arc11;Task6;Gateway3;
TASK;Task7;Präsentieren;
ARC;Arc12;Gateway3;Task7;
END;Ende1;
ARC;Arc13;Task7;Ende1;
```

4.2 Diskussion

Die Vorteile der hier vorgestellten Herangehensweise liegen darin, dass man von verschiedenen Modellierungssprachen und Datenformaten unabhängig ist. Möglich wird dies durch das hier definierte intermediate Format. Dieses erlaubt es, neue, herstellerspezifische Formate zu unterstützen, ohne die Algorithmen selbst verändern zu müssen. Stattdessen muss nur ein neuer Trans-

formator für die Übersetzung eines herstellerspezifischen Formats zum intermediate Format und zurück implementiert werden.

Durch die Möglichkeit, beliebige Informationen anzuhängen, wird außerdem ein sonst bei der Transformation möglicher Datenverlust vermieden.

Das Datenformat selbst ist maschinen- sowie menschenlesbar und ermöglicht es aufgrund seiner standardisierten UTF-8 Basis des verwendeten Zeichensatzes und seiner Einfachheit, dass Veränderungen manuell vorgenommen werden können.

5 Ermittlung ähnlicher Prozesse

Basierend auf der in der Einführung dargelegten Situation (ständig wachsende Anzahl an Prozessen in Unternehmen, aus denen ähnliche Prozesse identifiziert werden müssen, um Duplikate zu vermeiden und bestehende Prozesse vereinigen zu können) und den vorgestellten Forschungsfragen soll in dieser Arbeit auch ein Weg aufgezeigt werden, um die Ähnlichkeit von Prozessen zu messen. Die grundlegende Problemstellung hierbei ist, aus einer Vielzahl von Prozessen einige wenige herauszufiltern, die mit besonderen Merkmalen versehen sind. Die wissenschaftliche Fragestellung behandelt daher das Problem, aus einer gegebenen Menge P von Prozessen diejenigen Prozesse zu identifizieren, welche einander ähneln. Die Auswahl kann hierbei auf zweierlei Arten erfolgen.

1. Auswahl anhand bestimmter Kriterien wie einer vorgegebenen Anzahl an Prozessschritten oder verschiedener Aufgaben, welche behandelt werden müssen. Die Ähnlichkeit der Prozesse wird hierbei durch die zuvor gewählten Kriterien bestimmt bzw. davon, wie gut ein Prozess diese erfüllt.

2. Auswahl anhand eines vergleichbaren Prozessmodells. Es wird ein Prozessmodell erstellt oder ein vorhandenes gewählt, welches dann mit anderen Prozessen abgeglichen wird. Die Ähnlichkeit wird hierbei daran gemessen, wie sehr das zur Suche verwendete Modell und das jeweils analysierte Modell einander gleichen. Da es verschiedene Ansätze gibt, „Gleichheit“ zu definieren, werden in der Folge verschiedene Methoden zur Messung vorgestellt.

Da die hier beschriebene Ähnlichkeitsmessung in weiterer Folge eingesetzt werden soll, um Prozesse zu finden, welche eine vergleichbare Funktionalität aufweisen und sich deswegen für eine Verschmelzung eignen (siehe Kapitel 6), wird hier der zweite beschriebene Fall behandelt.

Für die Messung der Ähnlichkeit gibt es verschiedene Ähnlichkeitskriterien. Je nach gewähltem Kriterium müssen anschließend unterschiedliche Methoden für die weitere Analyse gewählt werden. Zu unterscheiden ist zwischen:

- *Äquivalenzanalyse*: Untersuchung der Prozesse danach, welche Elemente in den analysierten Prozessen in gleicher Form (Inhalt, Art) vorkommen.
    - Dieses Verfahren ist sehr schnell durchzuführen. Problematisch ist jedoch, dass bereits leichte Unterschiede in den einzelnen Elementen große Auswirkung auf das Endergebnis haben. Der Grund hierfür ist, dass nur vollständig übereinstimmende Elemente als gleich bzw. ähnlich erkannt werden.
      Dieses Verfahren eignet sich daher schlecht als alleinige Methode für eine Ähnlichkeitsmessung. Allerdings kann es aufgrund der hohen Geschwindigkeit des Ansatzes zur schnellen Vorfilterung der Prozesse vor einer genaueren Untersuchung verwendet werden. (Vgl. Yan, Dijkman, & Grefen, 2010, S. 60ff)

- *Inhaltsanalyse*: Verarbeitet die Inhalte bzw. deren Ähnlichkeit der verschiedenen Elemente, welche zur Bildung des Prozesses verwendet werden, wie die beschreibenden textuellen Informationen der Aufgaben (engl. Tasks). Hierdurch kann auch festgestellt werden, welche Elemente in den untersuchten Prozessen vorhanden sind bzw. fehlen. (Vgl. Wombacher, 2006, S. 257-263)
    - Der Vorteile dieser Methode ist, dass sie sehr schnell durchgeführt werden kann. So sind für die Messung der textuellen Ähnlichkeit viele optimierte Algorithmen wie die Lewenshtein-Distanz bekannt.
    - Nachteilig ist, dass keine Informationen darüber mit einbezogen werden, wie die verschiedenen Elemente im Prozess auftreten. Prozesse, welche völlig unterschiedliche Aufgaben bearbeiten (ähnliche Elemente werden hierbei in wechselnder Reihenfolge kombiniert um verschiedene Ziele zu erreichen), können hierdurch als sehr ähnlich erkannt werden.
- *Strukturanalyse*: Vergleicht den Kontrollfluss und damit die Strukturierung der verschiedenen Prozesse miteinander. (Vgl. Mino, Tartakovski, & Bergmann, 2007, S. 228-230)
    - Dieser Ansatz ermöglicht es erstmals, die Beziehungen der Prozesselemente zueinander in die Analyse mit einfließen zu lassen. Eine ganz neue Informationsebene, der Kontrollfluss, wird hierdurch der Analyse zugeführt. Die Ergebnisqualität wird hierdurch verbessert.
    - Im Vergleich mit dem vorherigen Ansatz steigt der Berechnungsaufwand an.
- *Verhaltensanalyse*: Analysiert das Verhalten von Prozessen und ist durch die mögliche Ausführungsreihenfolge der Elemente und durch die im Prozess abgebildeten Abläufe definiert. Bei der Verhaltensanalyse werden diese verglichen. Hierbei werden fehlende (nicht realisierbare) und überzählige (mögliche, aber eigentlich nicht notwendige) Abläufe entsprechend negativ bewertet. (Vgl. Nejati, Sabetzadeh, Chechik, Easterbrook, & Zave, 2007)
    - Kann als Weiterführung des obigen Ansatzes gesehen werden und verbessert dessen Leistungsfähigkeit.
    - Nachteilig ist, dass die zur Speicherung, Berechnung und Analyse notwendige Leistung bei diesem Ansatz deutlich über den vorangehend beschriebenen Methoden liegt.
- *Änderungsoperationsanalyse*: Hierbei wird versucht, die zu analysierenden Prozesse so anzupassen, dass diese einander gleichen. Die hierzu verwendete minimale mögliche Menge von Operationen (siehe 3.1.2) wird ermittelt, bewertet und anschließend für die Berechnung der Ähnlichkeit verwendet. (Vgl. Li, Reichert, & Wombacher, On Measuring Process Model Similarity based on High-level Change Operations, 2008, S. 248-264)
    - Dies ermöglicht eine genaue Analyse des Aufwandes (Anzahl der notwendigen Operationen, multipliziert mit aufwandsabhängigen Kosten), welcher für eine Anpassung der verschiedenen Prozesse aneinander zu betreiben ist.
    - Nachteilig ist auch hier der hohe Bedarf an Rechenleistung zur Umsetzung dieser Methode.
- *Hybride Ansätze*: Hierbei wird eine Kombination verschiedener Ansätze verwendet, um bessere Ergebnisse zu erreichen, als dies mit einem einzelnen Verfahren möglich wäre.

In dieser Arbeit fiel die Wahl auf eine hybride Methode. Hierbei werden der Inhalt der einzelnen Elemente zur Ermittlung des Ergebnisses und deren strukturelle Beziehung zueinander vereinigt. Den Ausschlag hierzu gab vor allem der Vorteil der Geschwindigkeit der kombinierten Ansätze im Zusammenhang damit, dass die entwickelten Algorithmen zur gleichzeitigen Analyse hunderter umfangreicher Prozesse eingesetzt werden. Aufwändigere Algorithmen wie die Verhaltensanalyse sind hierdurch praktisch nicht einsetzbar.

Zur Umsetzung der gewählten Verfahren wird folgendes Vorgehen umgesetzt:

- Zunächst werden die zu verarbeitenden Informationen durch Reduktion und Abstraktion der Daten (siehe Kapitel 4) aufbereitet. Hierdurch soll eine Verringerung des Rechenaufwandes und eine Verbesserung der Lösungsqualität erreicht werden.
- Anschließend werden die nun zur Verfügung stehenden textuellen Informationen analysiert. Hierbei wird auch versucht, Möglichkeiten aufzuzeigen, welche zur Optimierung dieser textuellen Analyse verwendet werden können. Basierend auf den hierbei gewonnenen Daten ist bereits eine erste Messung der Ähnlichkeit möglich.
- Zur weiteren Optimierung der Ergebnisse wird versucht, zusätzliche Informationen in die Berechnung mit einfließen zu lassen. Verschiedene Möglichkeiten werden hierzu im Abschnitt 5.6 diskutiert. Für diese Arbeit wurde anschließend ein strukturbasierter Ansatz gewählt.

Die folgende Grafik zeigt exemplarisch eine Suchanfrage an eine Prozessdatenbank (engl. „Prozess-Repository“). Die Anfrage geschieht anhand eines gewählten Vergleichsprozesses. Alle Prozesse der Datenbank werden anschließend analysiert und aufgrund ihrer Ähnlichkeit zum Vergleichsprozess gereiht.

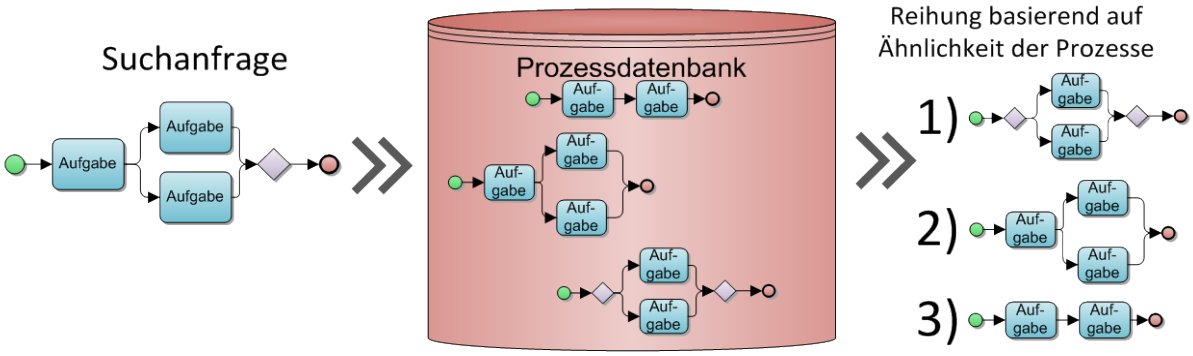

Abbildung 18: Reihung von Prozessen anhand eines Vergleichsprozesses

Quelle: Eigene Erstellung

Um die Qualität der Lösung zu garantieren, müssen die Bedingungen der Symmetrie und der Reflexivität eingehalten werden. Daher muss für alle Elemente einer Prozessmenge gelten, dass der Vergleich von Prozess x mit Prozess y die gleichen Ergebnisse liefert wie der Vergleich von y mit x. Gilt dies, dann ist die Symmetrie erfüllt. Die Reflexivität wiederum ist erfüllt, wenn der Vergleich eines Prozesses x mit sich selbst eine vollständige Übereinstimmung ergibt. (Vgl. Richter, 1999, S. 1ff)

5.1 Aufbereitung der Prozessdaten

Bevor die eigentliche Analyse der Prozesse beginnen kann, müssen zuvor die Daten (welche die Prozesselemente und deren Verbindungen beschreiben (vergleiche Kapitel 4), aufbereitet werden. Hierbei wird in zwei Phasen vorgegangen: Zuerst werden alle Gateway-Elemente, also die verschiedenen XOR, OR oder Parallel Split Gateways, und die zugehörigen Joins entfernt. Mittels neu hinzugefügter, dynamisch erzeugter Kanten werden die so aufgebrochenen Verbindungen wieder hergestellt. Dies ist notwendig, weil das zugrundeliegende Metamodell (vergleiche Kapitel 2 und 4) nicht eindeutig vorschreibt, dass beispielsweise eine Vereinigung mehrerer Ausführungspfade mittels eines Join Gateway Elements erzeugt werden muss. Diese Flexibilität trägt den unterschiedlichen Modellierungstechniken der Anwender Rechnung. Der Vorteil dieses Vorgehens liegt darin, dass eine einheitliche Datenbasis bezüglich der Verwendung der Gateways hergestellt

wird. Dies verbessert die Qualität der Messungen (durch die zuvor erwähnte Vereinheitlichung) und reduziert gleichzeitig die Rechenlast, da weniger Elemente untersucht werden müssen.

Problematisch ist hierbei, dass Unterschiede an den Gateways hierdurch nicht ausreichend berücksichtig werden. So kann eine exemplarische Verzweigung des Kontrollflusses mittels eines XOR- oder eines OR-Gateways durchgeführt werden. Je nach Lösungsart ergeben sich hierdurch unterschiedliche mögliche Ausführungspfade. Würden die Gateways entfernt, ist es allerdings nicht möglich, diese Information in die Ähnlichkeitsmessung mit einfließen zu lassen. Die Lösung wäre eine vor der Fertigstellung der Ergebnisse abschließende Nachverarbeitung (engl. „Post Processing“), welche speziell die Gateways und deren Ähnlichkeit untersucht.

Eine Reduktion des Rechenaufwandes lässt sich auch dadurch erreichen, dass „nicht aussagekräftige“ Elemente entfernt werden. Hierbei sind vor allem die Elemente betroffen, welche keine aussagekräftigen Beschreibungen erhalten können. In diese Kategorie fallen zusätzlich Elemente, welche wenig Auswirkungen auf die einzigartige Struktur eines Prozesses haben, also Elemente, welche sehr oft an den gleichen Stellen vorkommen. Beide Bedingungen treffen auf die Start/End-Elemente der Prozesse zu. Daher würde der Versuch, Unterschiede in der Struktur der Prozesse anhand dieser Elemente zu ermitteln, nicht zu verwertbaren Ergebnissen führen, da sich diese bei allen Prozessen an den gleichen Positionen (Anfang bzw. Ende des Prozesses) befinden. Gleiche Bedingungen treffen auf die Beschreibungen zu, da Start- und End-Element immer die gleichen Beschreibungsinhalte enthalten. Eine Messung von Unterschieden ist deshalb nicht möglich bzw. nicht sinnvoll. Die entsprechenden Elemente und die mit diesen Elementen verknüpften Kanten werden deshalb ebenfalls entfernt.

Zusätzlich zu der Entfernung dieser für den Kontrollfluss wichtigen Elemente sind je nach Modellierungstechnik auch Hilfselemente vorhanden, welche der Strukturierung des Prozesses dienen. Bei der Ausführung haben diese sogenannten stillen Prozesselemente keinen Einfluss und können deshalb entfernt werden. Peschler et al. zeigen hierbei in “The Projected TAR and its Application to Conformance Checking” eine Reduktion der zu verarbeitenden Elemente von 50 Prozent. Da hierbei allerdings ein spezieller Prozess zur Verdeutlichung der Vorteile verwendet wurde, ist bei realen Prozessen mit geringeren Vorteilen zu rechnen. (Vgl. Preschler, Weidlich, & Mendling, 2012, S. 159)

Neben der Entfernung von Elementen kann gleichzeitig auch ein verlustloses Vorgehen unter Verwendung verschiedener Abstraktionsschichten verwendet werden. Hierbei können Teile des Prozesses als Subprozesse zusammengefasst und in dieser aggregierten Form weiterverarbeitet werden. (Vgl. Gubała, Bubak, Malawski, & Rycerz, 2006, S. 663) Eine weitere Möglichkeit wird in „Computation of Behavioural Profiles of Processes Models“ von Weidlich et al. vorgestellt. Diese befasst sich damit, die charakteristischesten Elemente der Prozesse zu identifizieren und auf diese eine vom Originalprozess abgeleitete Sicht (vergleichbar zu Views in Datenbankumfeld) anzubieten. Hierdurch gelingt es ebenfalls, die Anzahl der zu untersuchenden Elemente und hierdurch auch den notwendigen Rechenaufwand zu reduzieren. (Vgl. Weidlich, Mendling, & Weske, 2010, S. 1-8)

Diese hier vorgestellten Reduktionsschritte (Entfernen von Elementen und Rekonstruktion des Kontrollflusses) werden automatisiert, daher ohne Anwenderinteraktion von der im Rahmen dieser Arbeit entwickelten Lösung durchgeführt. Die Algorithmen sind hierbei so gestaltet, dass diese in ihrer Funktionalität von der Größe der Prozesse (Anzahl der Elemente) unabhängig sind.

Die folgende Grafik zeigt, basierend auf beispielhaft verkürzten Bestellprozessen, das Vorgehen zur Vereinfachung der Prozesse. Die in der Realität auftretenden und zumeist deutlich komplexeren Prozesse stellen aber, wie oben beschrieben, ebenfalls kein „Problem“ für den hier beschrie-

benen Ansatz dar. Im linken Teil der Grafik sind die beiden Prozesse im Original zu sehen; in der Mitte sind die Prozesse nach der Anwendung des ersten Schritts abgebildet. Die Gateways wurden entfernt und die korrekten Verbindungen (Rekonstruktion des Kontrollflusses zwischen dem Vorgänger- und Nachfolgerelementen des entfernten Gateways) wieder hergestellt. Am rechten Rand der Grafik ist der letzten Schritt zu erkennen, in welchem die Start/End-Elemente sowie die zugehörigen Verbindungen entfernt wurden. Die beiden hier vorgestellten Prozesse dienen im weiteren Verlauf dieses Kapitels der praktischen/beispielhaften Präsentation der hier erarbeiteten Algorithmen.

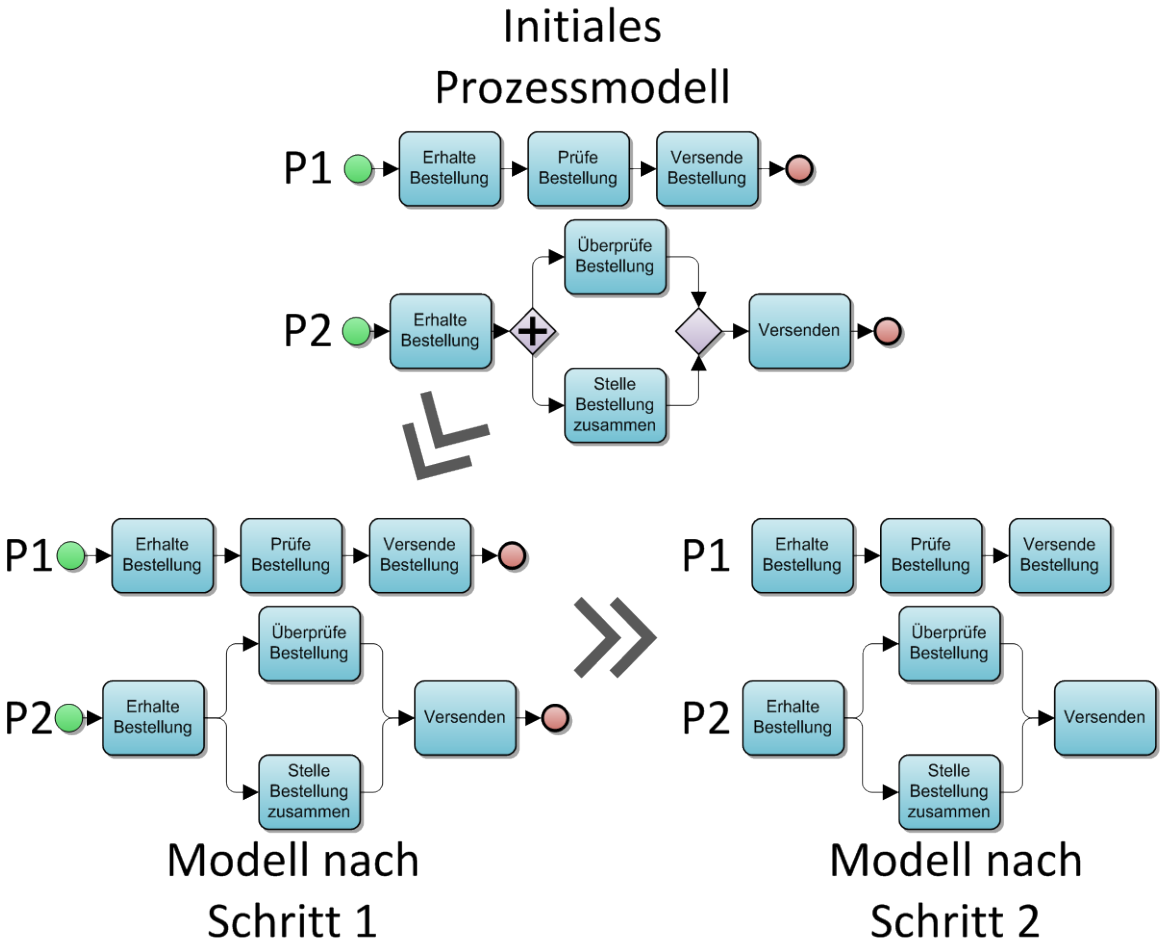

Abbildung 19: Beispiel für die Vereinfachung von Prozessdefinitionen

Quelle: Eigene Erstellung

5.2 Analyse der textuellen Informationen

Im Anschluss an die Datenaufbereitung wird, wie in der Einleitung besprochen, zuerst eine Analyse der einzelnen Elemente gestartet. Die Beziehung dieser Elemente zueinander wird erst in einem folgenden Schritt betrachtet.

Für den Vergleich verschiedener Elemente, insbesondere der Aufgaben (engl. „Tasks“), die diese ausführlichen Beschreibungen und Bezeichnungen (engl. „Labels“) erhalten, eignen sich Algorithmen, welche zur Ähnlichkeitsmessung von Texten herangezogen werden können. Basierend auf der mit der vergleichsweise primitiveren Hammingdistanz gelegten Basis (Vgl. Hamming, 1950, S. 147-160), welche die Unterschiede zwischen gleich langen Zeichenketten ermittelt, wurde die Texteditierdistanz entwickelt. Diese wird nach dem Wissenschaftler Wladimir Lewenstein, welcher sie 1965 erstmals beschrieb, auch Lewenshtein-Distanz genannt. Diese versucht, die minima-

le Anzahl an Änderungsoperationen bzw. Änderungskosten zu ermitteln, um eine Zeichenkette in eine andere Zeichenkette umzuwandeln. (Vgl. Levenshtein, 1965, S. 845-848)

Betrachtet werden hierbei die folgenden vier Fälle, um ein Wort eins in ein Wort zwei umzuwandeln:

1. *Einfügeoperationen*: Eine Einfügeoperation ist notwendig, wenn das Wort zwei länger als das Wort eins ist.
2. *Löschoperationen*: Sollte eines der beiden Wörter länger als das andere sein, ist es notwendig, die überschüssigen Zeichen zu entfernen.
3. *Substitutionsoperationen*: Hierbei werden Zeichen des einen Wortes so ersetzt, dass diese mit Zeichen des anderen Wortes übereinstimmen.
4. *Kopieroperationen*: Bei der Kopieroperation können Zeichen unverändert übernommen werden, da diese mit den betreffenden Stellen der zu vergleichenden Zeichenketten übereinstimmen.

Um die zuvor erwähnten Änderungskosten zu ermittelt, werden den verschiedenen Operationen unterschiedliche Kosten zugeteilt. So kann eine Kopieroperation beispielsweise kostenlos sein, also Kosten in der Höhe null verursachen, während Einfügeoperationen/Löschoperationen deutlich höhere Kosten als Kosten von zwei zugeordnet bekommen. Die Wahl der Kosten muss an die Anforderungen des Messergebnisses und der zu verarbeitenden Daten angepasst sein.

Die Berechnung selbst findet mittels dynamischer Programmierung statt, bei welcher, basierend auf der einfachen Berechnung von Teilproblemen, deutlich komplexere Gesamtprobleme gelöst werden. (Vgl. Bellman & Dreyfus, 2010, S. 81-90) Das hier vorgestellte Vorgehen ermöglicht es, die minimalen Editierkosten zu berechnen. Ist es zusätzlich auch gewünscht, die genaue Reihenfolge der Operationen zu ermitteln, welche für die Editierung durchgeführt werden müssen, sind eine Anreicherung der Daten mit rückwärtsgerichteten Zeigern und eine abschließende Rückwärtsverfolgung notwendig. (Vgl. Priyam, Karan, Sahoo, & Anwar, 2009, S. 6-8)

Der Algorithmus für die Berechnung ist im folgenden Pseudocode dargelegt. Eine ausführlichere textuelle Erklärung findet im Anschluss statt.

Algorithmus: EdDistanz(source, target, instCost, subCost, delCost)

Eingabe:
source und target als Zeichenketten, instCost als Einfügekosten, subCost als Substitutionskosten, delCost als Kosten für Zeichenentfernung

Ausgabe:
Minimale Editierkosten

Variablen:
n Länge der originalen Zeichenkette, m Länge der zu vergleichenden Zeichenkette, distmat Matrix zum Speichern der gelösten Teilprobleme (Editierdistanzen), cost Kosten für Editierung, costIns ermittelte Kosten für eine Einfügeoperation, costReplace ermittelte Kosten für Substitution, costDel ermittelte Kosten für Löschoperation

```
1   n = Length(source);
2   m = Length(target);
3
4   distmat = new matrix[n + 1, m + 1] ;
5
6
```

```
// Initialisierung
distmat[0,0] = 0;
für i=1 ... n
{
      distmat[i,0] = distmat[i - 1, 0] + instCost;
}
für j=1 ... m
{
      distmat[0,j] = distmat[0, j - 1] + delCost;
}

für i = 1 ... n
{
      für j = 1 ... m
      {
            int cost = distmat(i - 1, j - 1);

            charn = source[i];
            charm = target[j];

            // Kosten ermitteln und günstiges Vorgehen wählen
            falls charn != charm dann
            {
                  costIns = distmat(i - 1, j) + instCost;
                  costReplace = distmat(i - 1, j - 1) + subCost;
                  costDel = distmat(i, j - 1) + delCost;

                  cost = Min(costIns,costDel,costReplace)
            }

            distmat[i , j] = cost
      }
}

returniert distmat[n - 1, m - 1]
```

Programmausdruck 3: Pseudocode für die Ermittlung der minimalen Lewenshtein-Distanz

Quelle: Eigene Erstellung in Anlehnung an (Levenshtein, 1965, S. 845-848)

Der beschriebene Algorithmus verwendet den zuvor angesprochenen Ansatz der dynamischen Programmierung. Hierzu wird eine Matrix erzeugt, in welcher die Ergebnisse der gelösten Teilprobleme hinterlegt werden. Diese wird in der ersten Spalte/Zeile so initialisiert, als wäre jeweils eine der beiden Zeichenketten leer und hierdurch weder Lösch- noch Einfügeoperationen möglich. Die Zelle an der Stelle 0,0 wird mit 0 initialisiert.

Anschließend werden die beiden Zeichenketten Zeichen für Zeichen miteinander verglichen. Dabei werden bei jedem Vergleich die minimalen Kosten für alle Möglichkeiten (Löschen, Einfügen, Ersetzen) ermittelt und die optimale Lösung des Teilproblems (die mit den geringsten Kosten) übernommen. Dies geschieht nach der folgenden Berechnungsformel. Bei dieser werden die Substituierungskosten als 0 angenommen, wenn die verglichenen Zeichen gleich sind.

$$distmat[i,j] = \min \begin{cases} distmat[i-1,j] + instCost \\ distmat[i-1,j-1] + subCost(source, taget) \\ distmat[i,j-1] + delCost \end{cases}$$

Formel 1: Berechnung eines Teilproblems der Editierdistanz

Quelle: (Vgl. Jurafsky & Martin, 2008, S. 57-90)

Ein Beispiel für den Aufbau der Matrix ist in der Folge zu sehen. Die Ermittlung der Editierdistanz findet für die Wörter "Berechnung" und "Bestellung" statt. Für die Erzeugung wurden folgende Kosten angenommen: Eine Substitution erhöht die Gesamtkosten um zwei; eine Lösch- bzw. Einfügeoperation um eins.

Zur Verdeutlichung der Berechnung wurden verschiedene Zahlen farbig markiert. Die Kosten von 1 (grün) entstanden durch den Vergleich der Buchstaben e und b. Das Ergebnis war, dass die Zeichen nicht gleich waren. Daran werden anhand der in Formel 1 festgelegten Möglichkeiten die Werte für eine Einfügung (Kosten 3), Substitution (Kosten 3) oder eine Entfernung (Kosten 1) berechnet. Die günstigsten Kosten von 1 wurden danach übernommen.

Als Endergebnis werden bei Beendigung der Ausführung des Algorithmus die (rot markierten) Editierkosten von 8 bestimmt.

	#	b	e	s	t	e	l	l	u	n	g
#	0	1	2	3	4	5	6	7	8	9	10
b	1	0	1	2	3	4	5	6	7	8	9
e	2	**1**	0	1	2	3	4	5	6	7	8
r	3	2	1	2	3	4	5	6	7	8	9
e	4	3	2	3	4	3	4	5	6	7	8
c	5	4	3	4	5	4	5	6	7	8	9
h	6	5	4	5	6	5	6	7	8	9	10
n	7	6	5	6	7	6	7	8	9	8	9
u	8	7	6	7	8	7	8	9	8	9	10
n	9	8	7	8	9	8	9	10	9	8	9
g	10	9	8	9	10	9	10	11	10	9	**8**

Tabelle 7: Beispiel für eine Distanzmatrix bei der Berechnung der minimalen Editierdistanz

Quelle: Eigene Erstellung

Der oben beschriebene Algorithmus berechnet eine beliebig hohe Zahl als Editierkosten. Für die weitere Verarbeitung der ermittelten Daten ist allerdings eine Normierung auf einen Zahlenbereich zwischen 0 und 1 sinnvoll. Dies ermöglicht gleichzeitig auch eine Umwandlung in Prozentwerte, welche sich für die Darstellung besser eignen als abstrakte, nach oben offene Zahlenbereiche. Hierzu werden die Kosten weg von Ganzzahlen hin zu normierten Werten zwischen 0 und 1 angepasst. Eine vergleichbare Kostenstruktur zum vorhergehenden Beispiel würde dann folgende Kosten verwenden: Einfügen und Löschen würde bei 0,5 und Substituieren bei 1 liegen. Das Ergebnis wird dann durch die Länge der verglichenen Wörter dividiert. Da das Maximum der notwendigen Operationen nie länger sein kann als die Länge der längsten Zeichenkette, führt dies zu der gewünschten Normierung.

Das Ergebnis dieser Berechnung ist der Anteil der Zeichenkettenteile, welche modifiziert werden müssten, um Gleichheit zu erreichen. Zieht man diesen von 1 ab, erhält man das Gegenteil, also den Anteil, welcher nicht modifiziert werden müsste. Dieser Ansatz wird in der folgenden Formel verwendet.

$$symilarity(source, target) = 1 - \frac{ed(source, target)}{\max(source, target)}$$

Formel 2: Normierung der minimalen textuellen Editierdistanz

Quelle: Eigene Erstellung

Für andere Beispiele ergeben sich analog die folgenden Editierdistanzen, welche, basierend auf der Formel 2, berechnet wurden:

Erstes Wort	Zweites Wort	Berechnete Ähnlichkeit
Test	Test	1
Achtung	Verpackung	0,55
Paketversand	Briefversand	0,67

Tabelle 8: Verschiedene Editierdistanzen für unterschiedliche Zeichenketten

Quelle: Eigene Erstellung

5.2.1 Verbesserung der Eingangsdaten

Der Vergleich der Texte lässt sich anschließend noch dadurch verbessern, indem die mit dem Algorithmus zu verarbeitenden Daten vor Analysebeginn angepasst werden.

Der erste Schritt ist hierbei, dass alle Textdaten in Klein- bzw. Großbuchstaben umgewandelt werden, um die zumeist nicht essenziellen Unterschiede bzw. mögliche Schreibfehler auszugleichen. In der endgültigen Implementierung fiel bei der Vereinheitlichung die Wahl auf Kleinbuchstaben, weil hierdurch der Rechenaufwand für die Umwandlungen verringert werden kann. Die Ursache hierfür liegt darin, dass die zu verarbeitenden Wörter zu einem überwiegenden Teil bereits aus Kleinbuchstaben bestehen.

Problematisch ist dieser Ansatz bei den Wörtern, welche sowohl in Groß- als auch in Kleinschreibung existieren. Insbesondere Abkürzungen wie "SOAP" (Simple Object Access Protocol), ein Netzwerkprotokoll zum Datenaustausch (Vgl. MIT, ERCIM, & Keio, 2007) und das englischen Wort für Seife (in englischer Sprache: "soap") könnten hierbei Probleme bereiten. Umgehen ließe sich das hier beschriebene Problem durch eine Umwandlung in Kleinbuchstaben, bei welcher Wörter, welche vollständig in Großbuchstaben geschrieben wurden, ausgenommen sind.

Zusätzlich lässt sich auch eine Liste von „Stoppwörter“ führen. Dies sind Wörter, welche bei der Indexierung von Informationen ignoriert werden (Vgl. Wilbur & Sirotkin, 1992, S. 45), welche dann dazu bestimmt wären, um Wörter zu identifizieren, welche nicht bearbeitet werden sollen.

Eine weitere Vereinheitlichung der Daten lässt sich durch das Ausnützen von Synonymen erreichen. Synonyme sind Wortgruppen, welche trotz unterschiedlicher Schreibung die gleiche Bedeutung besitzen. (Vgl. Schmidt, 2008, S. 68-74) Übergibt man dem zuvor vorgestellten Algorithmus die beiden Wörter "liefern" und "bringen" würde er große Unterschiede zwischen diesen beiden messen. Zieht man allerdings zuvor die Synonyme des Wortes "liefern" in die Auswertung mit ein, zeigt sich, dass beide Wörter die gleiche Bedeutung besitzen. Deshalb ist es möglich, im zum Vergleich verwendeten Zieltext das Wort "bringen" durch "liefern" zu ersetzen. Die ermittelten Ergebnisse gewinnen hierdurch an Relevanz.

Beispiele für die zuvor erwähnten Synonyme für das Wort "liefern" sind im Folgenden angeführt:

- anliefern, beliefern, bringen, [ins Haus] schaffen, mit Waren versorgen, schicken, senden, spedieren, tragen, übergeben, versenden, zukommen lassen, zuliefern; (gehoben) überbringen; (umgangssprachlich) [he]rausrücken; (Amtssprache) zustellen; (Kaufmannssprache) ausliefern
- auswerfen, bilden, erzeugen, hervorbringen, produzieren; (bildungssprachlich) generieren; (umgangssprachlich, oft abwertend) fabrizieren; (Wirtschaft) ausstoßen
- beschaffen, besorgen, bringen, geben, heranschaffen, [her]beibringen, sorgen für, verschaffen, vorlegen; (umgangssprachlich) auftreiben, organisieren

Quelle: (Vgl. Bibliographisches Institut GmbH, 2012)

Eine Automatisierung dieses Vorgangs ist durch eine Lösung wie „WordNet“ möglich. Diese erlaubt es, aus Datenbanken automatisiert Synonyme abzufragen. Diese lassen sich dann entsprechend dem vorher beschriebenen Ansatz verwerten. (Vgl. Fellbaum, 1998, S. 1ff)

Ein weiterer Sonderfall sind Homonyme. Bei diesen handelt es sich um Wörter, welche unterschiedliche Bedeutung, aber gleiche Schreibung besitzen. (Vgl. Schmidt, 2008, S. 223-227) Dieses Problem lässt sich durch die Anreicherung der Daten mit semantischen Informationen, beispielsweise unter Verwendung der Web Ontology Language (OWL), lösen. Der Ansatz mittels Ontologien ermöglicht es auch, dem Problem der Synonyme zu begegnen. (Vgl. Ehrig, Koschmider, & Oberweis, 2007, S. 1ff) Problematisch hierbei ist allerdings, dass alle Prozesselemente mit den verfügbaren bzw. erst zu erzeugenden Ontologien verknüpft werden müssen. Eine solche Verknüpfung ist mit einigem Aufwand verbunden und steht bei den geplanten Testdaten nicht zur Verfügung, sodass hierauf nicht weiter eingegangen wird.

Die hier beschriebene Problematik, dass Gleichheit der textuellen Beschreibungen der Elemente nicht unbedingt bedeutet, dass auch die Elemente selbst die gleichen Aktionen ausführen (siehe oben beschriebene Problematik mit Synonymen/Homonymen) wurde auch von Rinderle-Ma et al. in „On Utilizing Web Service Equivalence for Supporting the Composition Life Cycle“ erkannt.
(Vgl. Rinderle-Ma, Reichert, & Jurisch, 2011)

Das Problem, dass Wörter in verschiedenen Ausprägungen unterschiedliche Schreibungen aufweisen, lässt sich mit einer Wortstammreduktion lösen. Die Auswirkung kann am folgenden Beispiel betrachtet werden. So werden die Wörter gehe, ging, gegangen durch eine Wortstammreduktion auf gehe zurückgeführt und hierdurch vereinheitlicht. Dies ermöglicht es, die Qualität und Aussagekraft der ermittelten Editierdistanz weiter zu verbessern. Problematisch hierbei ist, dass Begriffe mit unterschiedlicher Bedeutung auf das gleiche Stammwort abgebildet werden. Die Abbildung muss deshalb injektiv sein. (Vgl. Paice, 1994, S. 2-9) Für die praktische Umsetzung der Wortstammreduktion sind entsprechende Algorithmen sowie zugehörige Datenbanken und Wörterbücher vorhanden. Zu erwähnen sind hierbei unter anderem Algorithmen von Jörg Caumanns für deutschsprachige Wörter. (Vgl. Caumanns, 1999, S. 1-10)

Anschließend findet eine Filterung der zu verarbeitenden Daten statt. So werden Wörter, welche in den beiden zu vergleichenden Datensätzen gleichzeitig vorkommen, von der Berechnung ausgenommen. Dies geschieht durch ein Aufteilen der Texte an den die Wörter trennenden Leerzeichen. Die Aufgabenbeschreibungen "Versende Werbung" und "Versende Rechnung" werden hierdurch zu ((Versende), (Werbung)) und ((Versende), (Rechnung)). Anschließend werden die so entstandenen Einzelteile miteinander verglichen und gleichlautende Textteile, welche sich an der gleichen Position befinden, entfernt. Dies führt beim vorherigen Beispiel dazu, dass nur "Werbung" und "Rechnung" miteinander verglichen wird. Vor der Filterung können auch die einzelnen Teile der zuvor beschriebenen Abläufe herangezogen werden. Techniken wie eine Wortstammreduktion oder die Beachtung von Synonymen erhöhen die Wahrscheinlichkeit, dass Wörter gefiltert werden können. Der Nutzen dieses Ansatzes liegt in einer Verringerung des Rechenaufwandes und in einer Verbesserung der Ergebnisse.

In der folgenden Tabelle findet eine Zusammenfassung der zuvor beschriebenen Methoden zur Datenqualitätsverbesserung statt.

Methode	Vorteile	Nachteile
Vereinheitlichung der Groß- und Kleinschreibung	Vereinheitlichung der Daten, erhöhte Toleranz gegenüber Tippfehlern	Falscherkennung beispielsweise von Abkürzungen
Erkennung von Synonymen	Vereinheitlichung der Datenbasis	Starke Erhöhung der zu verarbeitenden Daten
Erkennung von Homonymen	Vermeidung einer fehlerhaften Erkennung von ähnlichen Wörtern	Optimale Erkennung bzw. Unterscheidung nur durch aufwändig erstellte Ontologien möglich
Wortstammreduktion	Vereinheitlichung der Datenbasis	Reduktionsergebnis muss injektiv sein, um Fehlerkennungen zu vermeiden
Datenfilterung	Verringerung der zu verarbeitenden Datenbasis	

Tabelle 9: Zusammenfassung verschiedener Methoden zur Optimierung textueller Daten für eine Ähnlichkeitsmessung

Die hier vorgestellten Ansätze (ohne Verwendung von semantischen Informationen, da diese nicht zur Verfügung stehen) werden innerhalb der für diese Arbeit entwickelten Prototypen automatisiert angewendet. Ein zusätzlicher Aufwand für die Anwender besteht daher nicht.

5.3 Analyse der Modellstruktur

Grundsätzlich wäre es mit der weiter oben beschriebenen Methode bereits möglich, die Ähnlichkeit zwischen unterschiedlichen Prozessen zu messen. Allerdings fehlen, wie in der Einleitung bereits besprochen, dann Informationen zum Verhalten bzw. strukturellen Komponente der Prozesse. Da dies ebenfalls ein wichtiges Unterscheidungsmerkmal darstellt und die reinen textuellen Untersuchungen nicht die benötigte Lösungsqualität garantieren können (Vgl. Rinderle-Ma, Reichert, & Jurisch, 2011, S. 5f), wird nun in diesem Abschnitt versucht, einen Weg zu finden, dies in die Ergebnisse mit einfließen zu lassen.

Neben der Betrachtung einzelner Prozesselemente können weitere Informationen auch durch die Analyse der Modellstruktur gewonnen werden. Hierzu eignet sich beispielweise eine Verhaltensanalyse der Modelle, welche von Alves de Medeiros et al. in "Quantifying process equivalence based on observed behavior" beschrieben werden. Bei dieser werden die möglichen Abläufe, abhängig davon, in welcher Reihenfolge die verschiedenen Aufgaben des Prozesses abgearbeitet werden, analysiert und deren Wiederherstellbarkeit in den zu vergleichenden Prozessen untersucht. (Vgl. Medeiros, Aalst, & Weijters, 2008, S. 1-20) Als weitere Alternative ist auch die Bestimmung der Grapheneditierdistanz möglich. Bei dieser handelt es sich um einen ähnlichen Ansatz wie bei der zuvor genannten linguistischen Editierdistanz.

Auch hier werden verschiedene atomare Operationen untersucht, insbesondere das Hinzufügen/Entfernen von Knoten bzw. Kanten sowie die Substitution von Knoten. Zur Bestimmung dieser Operationen werden zwei Graphen miteinander verglichen und diese so lange mit Einfüge- und Löschoperationen sowie der Substitution von Knoten behandelt, bis der eine Graph an den anderen angepasst wurde. (Vgl. Zhang & Shasha, 1989, S. 1245-1262)

Jede diese Operationen kann nun auch noch mit Kosten versehen werden. Dies ermöglicht es, die Operationen zu bewerten und festzustellen, wie viel Aufwand notwendig ist, um die Graphen anzupassen. Je weniger Kosten hier gemessen werden, desto ähnlicher sind einander die Graphen. Hierdurch definiert sich, dass die optimale Editierdistanz die Operationskombination mit den geringsten Kosten ist. (Vgl. Sanfeliu & Fu, 1983, S. 353–363)

Die Wahl der Methode fiel auf die Ermittlung der Grapheneditierdistanz. Der Hintergrund hierzu ist, dass hierdurch nicht nur die Struktur für die Ähnlichkeitsmessung herangezogen werden kann, sondern auch noch eine Verbesserung beim Vergleichen einzelner Elemente der Graphen möglich ist. Auf diese Analyse des Elementkontextes wird später in diesem Kapitel im Abschnitt 5.5 noch eingegangen. Alternativen hierzu wären die in der Kapiteleinleitung beschriebenen Verhaltensanalysen. Eine Realisierung wäre hierbei über eine Erreichbarkeitsanalyse der unterschiedlichen Elemente in den Prozessen oder über einen Vergleich der unterschiedlichen Ausführungspfade möglich. Aufgrund des hohen Rechenaufwandes sowie der großen Anzahl (voraussichtlich mehrere hundert) zu vergleichender Prozesse wurde bei diesem Verfahren im Laufe dieser Arbeit erkannt, dass sich diese hier nicht sinnvoll einsetzen lassen.

Der Basisalgorithmus für die Bestimmung der Grapheneditierdistanz, grundlegend basierend auf der Arbeit „Graph Matching Algorithms for Business Process Model Similarity Search“ von Dijkman et al., wird im Folgenden präsentiert. Zu Beginn wird versucht, eine Datenbasis zu erzeugen, welche alle Möglichkeiten enthält, die beiden zu vergleichenden Graphen aneinander anzupassen. In einem ersten Schritt werden hierzu beliebige Kombinationen zweier Knoten (Knoten eins von Graph 1 und Knoten zwei von Graph 2) erzeugt. Zur Unterstützung dieses Vorgehens wird eine Liste aller Knoten des Graphen G1 und G2 geführt. Nachdem ein Pärchen erstellt worden ist, werden dessen Teilelemente aus den Listen der verfügbaren Knoten entfernt. Basierend auf dieser Datenbasis wird eine neue Verzweigung erzeugt; auf dieser nun veränderten Basis werden wieder alle noch möglichen Kombinationen generiert. Damit auch die zuvor erwähnte Möglichkeit der Entfernung eines Knotens überprüft werden kann, wird in einem eigenen Schritt immer auch ein möglicher Knoten aus einem der beiden Knotenlisten entfernt und, basierend auf diesen Informationen, ebenfalls eine neue Verzweigung erstellt. Die Generierung dieser Datenbasis geschieht in diesem initialen Schritt unabhängig von den Kosten der ermittelten möglichen Lösungen, da deren Bewertung erst später erfolgt. Das Erzeugen der Lösungen wird so lange wie möglich durchgeführt, als die Liste der verfügbaren Knoten von Graph 1 bzw. Graph 2 noch Elemente enthält.

Zur Verdeutlichung des Ansatzes wird die folgende Grafik verwendet, welche die entstehenden Verzweigungen visuell aufbereitet. Die beiden verwendeten Prozesse basieren hierbei auf den in Abbildung 19 verwendeten Beispielen P1 und P2. Um den verfügbaren Platz besser ausnützen zu können, werden nur die Anfangsbuchstaben der in den Prozessen definierten Aufgaben dargestellt. Der Prozess P1 besteht dementsprechend aus (E, P, V); der zweite Prozess P2 besitzt die Elemente (E, Ü, S, V). Die einzelnen Datenfelder teilen sich auf in Pärchen, verfügbare Elemente Graph 1 und verfügbare Elemente Graph 2. Zweige, welche durch das Löschen eines Elemente entstanden sind, wurden rot markiert.

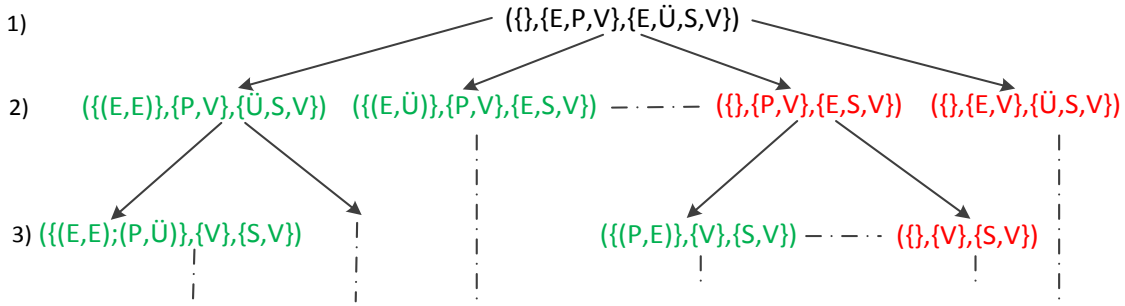

Abbildung 20: Verdeutlichung der Baumstruktur während der Datengenerierung

Quelle: Eigene Erstellung

Der Algorithmus für die Bestimmung der Datenbasis ist im folgenden Pseudocode ersichtlich. Er setzt die zuvor beschriebenen Abläufe um (Erstellen von Pärchen und Entfernen von Elementen). Sobald eine Lösung abgeschlossen ist und nicht mehr weiter verfeinert werden kann, wird diese gespeichert.

`Algorithmus: GraphEdDistanzBase(B, finalBranches)`

Eingabe:
Branch Element B, welches Folgendes beinhaltet: Liste N1 mit allen verfügbaren Knoten des Graphen eins, Liste N2 mit allen verfügbaren Knoten des Graphen G2, Liste M von erstellten Pärchen, removedElements Anzahl der gelöschten Elemente; finalBranches Liste der generierten möglichen Lösungen

Ausgabe:
Ermittelte mögliche Lösungen

Variablen:
mapping neu generiertes Pärchen mit zwei Knoten N1 und N2, branch neu generierte Lösung bzw. Zweig mit möglichen Lösungen

```
1   // speichern fertiger Lösungen
2   falls Count(B.N1) == 0 und Count(B.N2) == 0 dann
3   {
4         finalBranches = finalBranches ∪ B;
5   }
6
7   // erzeugen neuer Pärchen
8   für i=0 ... B.N1
9   {
10        für j=0 ... B.N2
11        {
12
13              mapping = NeuesMapping(B.N1[i],B.N2[j]);
14
15              //Branch basierend auf den übergebenen Daten, Erzeugung
16              //inkludiert Aktualisierung der Knotenlisten
17              branch = NeuerBranch(B, mapping);
18
19              GraphEdDistanzBase(branch, finalBranches);
20        }
21  }
22
23
```

```
24  //erzeugen von Lösungen mit entfernten Elementen
25  für i=0 ... B.N1
26  {
27        branch = NeuerBranch(B);
28        branch.removedElements = branch.removedElements++;
29        branch.N1 = branch.N1 \ B.N1[i];
30        GraphEdDistanzBase(branch, finalBranches);
31  }
32
33  für i=0 ... B.N2
34  {
35        branch = NeuerBranch(B);
36        branch.removedElements = branch.removedElements++;
37        branch.N1 = branch.N2 \ B.N2[i];
38        GraphEdDistanzBase(branch, finalBranches);
39  }
```

Programmausdruck 4: Berechnung der Datenbasis zur Ermittlung der Grapheneditierdistanz

Quelle: Eigene Erstellung in Anlehnung an (Vgl. Dijkman, Dumas, & Garcia-Banuelos, Graph Matching Algorithms for Business Process Model Similarity Search, 2009, S. 6-11)

Basierend auf den ermittelten Pärchen kann anschließend der Substitutionsaufwand ermittelt werden. Hierzu wird die textuelle Editierdistanz der beiden Elemente, welche ein Pärchen bilden, berechnet. Diese Distanzen werden anschließend für alle Pärchen aufsummiert. Elemente, welche in keinem Pärchen vorkommen, können als hinzugefügte bzw. gelöschte Elemente betrachtet werden.

Zusätzlich zu den Knoten können als weitere Informationsquelle auch noch die Kanten der Graphen analysiert werden. Eine Kante gilt nur dann als substituiert, wenn sich diese in dem zu vergleichenden Prozess wieder bilden lässt. Für eine Kante E1 im Graphen G1 mit dem Startknoten N1 und dem Endknoten N2 muss Folgendes zutreffen, um die Kante "wiederzufinden":

1. Es existiert ein Pärchen P1, welches den Knoten N1 (erster Graph) und einen Knoten M1 des zweiten Graphen G2 enthält.
2. Es existiert ein Pärchen P2, welches den Knoten N2 (erster Graph) und einen Knoten M2 des zweiten Graphen G2 enthält.
3. Knoten M1 und M2 waren im Graphen G2 mit einer Kante verbunden.

Ein vergleichbarer Ansatz für die Methode der Kantenvergleiche wurde bereits früher in der Arbeit "Graph Matching Algorithms for Business Process Model Similarity Search" von Dijkman et al. vorgestellt. (Vgl. Dijkman, Dumas, & Garcia-Banuelos, Graph Matching Algorithms for Business Process Model Similarity Search, 2009, S. 4f)

Diese beiden zuvor genannten Ansätze werden in der folgenden Grafik visuell aufbereitet. Gezeigt wird ein möglicher Weg, die beiden dargestellten Graphen aneinander anzupassen. Hierzu ist in der Grafik das Bilden von Pärchen angedeutet. Ein Pärchen stellen hierbei zwei (je eines von Graph eins und eines von Graph zwei) mit zwischen den Prozessen verlaufenden Pfeilen verbundene Prozesselemente dar. Zusätzlich werden entfernte Knoten rot und hinzugefügte Knoten grün markiert. Analog dazu werden entfernte bzw. hinzugefügte Kanten gekennzeichnet. Sollte eine Kante direkt übernommen werden können, wurde diese schwarz eingezeichnet.

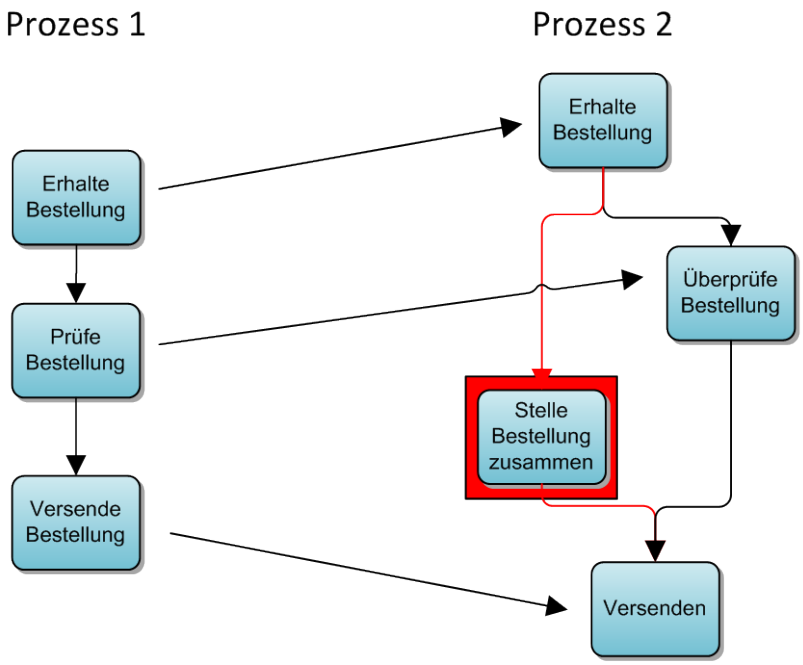

Abbildung 21: Darstellung möglicher Anpassungsoperationen zur Vereinigung zweier Prozesse

Quelle: Eigene Erstellung

Der folgende Pseudocode zeigt das für die Analyse der Kantenverbindungen genutzte Verfahren. Dazu werden die erzeugten Pärchen einer möglichen Lösung mit den Kanten der Graphen abgeglichen. Hierzu wird jede Kante des Graphen 1 durchgegangen und versucht, die Start/End-Elemente in den Pärchen der untersuchten Lösungen zu finden. Gelingt dies, wird anschließend untersucht, ob die beiden entsprechend zugeordneten Elemente in Graph zwei mittels einer Kante verbunden sind. Basierend darauf, wie viele Kanten hierdurch verifiziert werden konnten, werden anschließend die gelöschten/neu zu erstellenden Kanten berechnet.

Algorithmus: KantenEd(M, E1, E2)

Eingabe:
E1 Liste aller Kanten des ersten Graphen, E2 Liste aller Kanten des zweiten Graphen, M Menge von Pärchen mit zwei Nodes N1,N2, costRemove Kosten zu entfernende Kanten, costAdded Kosten für hinzuzufügende Kanten

Ausgabe:
Ermittelte Anzahl der hinzugefügten/entfernten Kanten

Variablen:
foundEdges Anzahl der wiedergefundenen Kanten, mappedStart Gefundenes Start Element einer Kante, mappedEnd Gefundenes End Element einer Kante

```
1   foundEdges = 0;
2
3   für i=0 ... E1-1
4   {
5         edge1 = E1[i];
6
7         mappedStart = NULL;
8         mappedEnd = NULL;
9         für m=1 ... M
10        {
```

```
11              mapping = M[m];
12
13              falls mapping.N1 == edge1.start dann
14              {
15                    mappedStart = mapping.N2;
16              }
17              falls mapping.N1 == edge1.end dann
18              {
19                    mappedEnd = mapping.N2;
20              }
21        }
22
23        falls mappedStart != NULL V mappedEnd != NULL dann
24        {
25              für j=0 ... E2-1;
26
27              edge2 = E2[j];
28
29              falls edge2.start == mappedStart V edge2.end == mappedEnd dann
30              {
31                    foundEdges++;
32              }
33        }
34  }
35
36  sumAdded = (Count(E1)- foundEdges);
37  sumRemoved = (Count(E2)- foundEdges);
38
39  retuniert sumAdded+sumRemoved;
```

Programmausdruck 5: Ermittlung der Kanteneditierdistanz

Quelle: Eigene Erstellung in Anlehnung an (Vgl. Dijkman, Dumas, & Garcia-Banuelos, Graph Matching Algorithms for Business Process Model Similarity Search, 2009, S. 4f)

Da bei den Vergleichen alle Knoten des einen Graphen mit allen Knoten des anderen Graphen verglichen werden müssen, ist dieses Verfahren als nichtdeterministische Polynomialzeit-Schwer (NP-Schwer) zu bezeichnen. (Vgl. Dijkman, Dumas, & Garcia-Banuelos, Graph Matching Algorithms for Business Process Model Similarity Search, 2009, S. 1) Mittels einer Heuristik ist auch eine schnellere Implementierung möglich, welche allerdings, je nach gewünschter Verkürzung der Suchdauer, schlechtere Ergebnisse liefern kann. (Vgl. Neuhaus, Riesen, & Bunke, 2006, S. 1-10) Implementierungen eines möglichen Vorgehens sind hierbei mittels des A* Algorithmus, eines mittels heuristischem Verfahren arbeitenden Suchverfahrens für kürzeste Wege, beschrieben worden. (Vgl. Stentz, 1994, S. 1-8)

5.4 Berechnung der Ähnlichkeit

In den zuvor vorgestellten Abläufen wurden alle notwendigen Informationen erhoben, welche zur Beurteilung der Ähnlichkeit benötigt werden.

Basierend auf diesen wird nun die eigentliche Ähnlichkeit, normiert auf einem Wert zwischen 0 und 1, bestimmt. Dabei stehen pro ermittelter möglicher Lösung (vergleiche hierzu Abschnitt 5.3) folgende Informationen zur Verfügung:

1. *sumEdges*: Anzahl der hinzugefügten/gelöschten Kanten
2. *sumNodes*: Anzahl der hinzugefügten/gelöschten Knoten
3. *sumSub*: Summe des Substitutionsaufwandes für die erzeugten Pärchen

Zuerst ist eine weitere Normierung der ermittelten Werte notwendig. Hierzu wird sumEdges durch die Gesamtanzahl der Kanten von Graph 1 (E1) und Graph 2 (E2) dividiert. Bei sumNodes wird ähnlich vorgegangen, wobei hier die Gesamtanzahl der Knoten von Graph 1 (N1) und Graph 2 (N2) genommen wird. Der Substitutionsaufwand (sumSub) wird durch die Division des Wertes durch die Gesamtanzahl der Pärchen normiert.

$$sumEdgesNor = \frac{sumEdges}{E1 + E2}$$

$$sumNodesNor = \frac{sumNodes}{N1 + N2}$$

$$sumSubNor = \frac{sumSub}{Pärchenanzahl}$$

Formel 3: Normierung der ermittelten Werte

Quelle: Eigene Erstellung

Die endgültige Grapheneditierdistanz wird anschließend durch die folgende Formel berechnet. Diese ermöglicht es außerdem, die verschiedenen Werte zu gewichten. Hierdurch können beispielsweise Unterschiede bei den Pärchen (sumSubNor) einen höheren Einfluss auf das Gesamtergebnis eingeräumt bekommen als die anderen beiden Faktoren. Um die Normierung der Werte zu garantieren, müssen diese im Bereich zwischen 0 und 1 liegen. Folgende Faktoren werden verwendet:

- *wEdges*: Faktor für die Kantenoperationen
- *wNodes*: Faktor für die Knotenoperationen
- *wSub*: Faktor für den ermittelten Substitutionsaufwand

Die normierte Grapheneditierdistanz (bzw. Graphenähnlichkeit durch die Invertierung mittels des vorangestellten " 1 –") ermittelt sich wie folgt:

$$GSim = 1 - \frac{wEdges * sumEdgesNor + wNodes * sumNodesNor + wSub * sumSubNor}{wEdges + wNodes + wSub}$$

Formel 4: Berechnung der Graphenähnlichkeit

Quelle: (Vgl. Dijkman, Dumas, & Garcia-Banuelos, Graph Matching Algorithms for Business Process Model Similarity Search, 2009, S. 5)

Ein mögliche Lösung, welche auf den in Abbildung 19 vorgestellten Graphen basiert, wird nun im Folgenden zur Verdeutlichung des oben beschriebenen Vorgehens berechnet. In dieser existieren Pärchen zwischen "Erhalte Bestellung" und "Erhalte Bestellung" sowie zwischen "Prüfe Bestellung" und "Überprüfung“ und zwischen "Versende Bestellung" und "Versenden". Die Variable sumEdges beträgt hierbei 2 (zwei gelöschte Kanten, $sumEdgesNor = 0{,}33\ [\frac{2}{6}]$), sumNodes ist 1 (ein gelöschter Knoten, $sumNodesNor = 0{,}14\ [\frac{1}{7}]$) und ist damit vergleichbar mit den in Abbildung 21 gezeigten Anpassungsoperationen. Die Summe der auf zwei Stellen gerundeten Substitutionskosten der Pärchen beträgt entsprechend 0,72 (0.0 + 0,46 + 0,26). Für die Gewichtung der Werte wurde Folgendes festgelegt: wEdges = 0,2; wNodesd = 0,3; wSub = 0,5

Hieraus ergibt sich eine ermittelte Grapheneditierdistanz von:

$$GSim = 1 - \frac{0{,}2 * 0{,}33 + 0{,}3 * 0{,}14 + 0{,}5 * 0{,}72}{0{,}2 + 0{,}3 + 0{,}5} = 1 - \frac{0{,}468}{1} = 0{,}532$$

5.5 Erweiterung der klassischen Substitutionskostenberechnung

Die zuvor beschriebene Berechnung der Substitutionskosten (vergleiche Abschnitt 5.1) betrachtet ausschließlich das zu substituierende Element an sich. Als Erweiterung dieses Ansatzes wird im Folgenden vom Autor dieser Arbeit vorgeschlagen, auch die Nachbarschaftsbeziehungen der jeweils untersuchten Elemente mit einzubeziehen. Hierdurch soll die Qualität der gewonnenen strukturellen Ähnlichkeitsmessung verbessert werden. Dies ist möglich, da die normale Strukturanalyse nur eine Aussage darüber trifft, ob sich die verglichenen Prozesselemente an den gleichen oder an abweichenden Positionen im Prozess befinden. Der hier vorgestellte Ansatz kann dem gegenüber ermitteln, wie sehr die Positionen voneinander abweichen und hierdurch eine differenziertere Analyse der strukturellen Unterschiede erstellen, was die Ergebnisqualität verbessert.

Hierzu wird beim Vergleich zweier Graphen G1 und G2 analysiert, welche direkten Vorgänger bzw. Nachfolger das Element besitzt und versucht, diese Beziehungen in den Graphen G2 nachzuvollziehen. Ein Vorgänger bzw. Nachfolger kann hierbei einen von vier verschiedenen Zuständen annehmen.

- *Zustand 1:* Er kann nicht gefunden werden.
- *Zustand 2:* Er kann gefunden werden, ist aber weiter entfernt als im Vergleichsgraphen.
- *Zustand 3:* Er kann gefunden werden und ist gleich weit entfernt.
- *Zustand 4:* Er kann gefunden werden, hat aber die Position getauscht. Ein Vorgänger wurde daher zu einem Nachfolger und umgekehrt oder er befindet sich in einem neuen parallelen Ausführungszweig. In diesem Fall wird er auch als nicht gefunden angenommen.

Die Erweiterung der Substitutionskosten funktioniert wie folgt: Zu Beginn wird ermittelt, wie viele Vorgänger/Nachfolger das zu untersuchende Element im originalen Graphen hatte. Anschließend werden die erzeugten Pärchen hinsichtlich dieser Vorgänger/Nachfolger untersucht. Dort, wo ein Vorgänger/Nachfolger von keinem Pärchen repräsentiert wird, kann er als nicht gefunden und daher im Zustand eins angenommen werden. Ein nicht gefundenes Element bekommt den Wert 0 zugewiesen. Wurde ein Pärchen gefunden, muss es noch hinsichtlich der anderen Zustände untersucht werden. Hierzu wird, beginnend mit dem Pärchen, welches das initiale Element repräsentiert, versucht, das jeweilige Vorgänger- bzw. Nachfolgerpärchen zu finden. Gelingt dies, werden die Entfernung bestimmt und die erweiterte Ähnlichkeit berechnet, indem die Ähnlichkeit der beiden Teile des Pärchens durch die Entfernung dividiert wird. Gelingt dies nicht, wird angenommen, dass der Zustand 4 zutrifft und der Wert auf 0 gesetzt. Abschließend werden die ermittelten Werte aufaddiert und durch die Gesamtanzahl der Nachfolger/Vorgänger dividiert, um eine Normierung des Endergebnisses auf einen Wert zwischen 0 und 1 zu ermöglichen.

$$simpleSimilarity(n) = \begin{cases} \frac{similarity(n)}{distance(n)} & wenn\ gefunden \\ 0 & sonst \end{cases}$$

$$similarityEnhancedNeighbor = \frac{\left(\sum_{(n \in N)} simpleSimilarity(n)\right)}{sumPostNodes + sumPreNodes}$$

Formel 5: Berechnung der Nachbarschaftseditierdistanz

Quelle: Eigene Erstellung

Die vorhergehende Formel errechnet die normierte Editierdistanz der Nachbarn abhängig von deren Entfernung. Die Berücksichtigung des Ausgangsknotens geschieht mit der folgenden Formel. In dieser kann mittels zweier Gewichtungselemente (wOriginal und wNeighbor), welche zusammen 1,0 ergeben müssen, der Einfluss der Nachbarschaftsbeziehungen im Vergleich zur Ähnlichkeit des Ausgangselementes konfiguriert werden.

$$EDEnhanced = SimilarityEnhancedNeighbor(n) * wNeighbor + similarity(n) * wOriginal$$

Formel 6: Berechnung der erweiterten Editierdistanz

Quelle: Eigene Erstellung

Die Abbildung 22 verdeutlicht die oben beschriebenen Prinzipien. Gezeigt werden zwei Ausschnitte von Prozessen, wobei die Nachbarschaftsbeziehungen der orange markierten Elemente untersucht werden. Grün markierte Elemente konnten hierbei in einem Pärchen wiedergefunden werden; bei rot markierten Elementen gelang dies nicht. Als Gewichte für die Berechnung wurde 0,3 für die Nachbarschaftsbeziehungen und 0,7 für die Ähnlichkeit der Ausgangselemente verwendet. Unter Verwendung dieser Gewichte des zu Beginn dieses Absatzes genannten Beispiels ergibt sich die folgende Berechnung für die erweiterte Ähnlichkeit.

$$similarity = \frac{(0{,}45 \ + \ 1 \ + \ 0)}{3} * 0{,}3 \ \ + \ 0{,}73 \ * \ 0{,}7 = 0{,}656$$

Eine Ähnlichkeitsmessung ohne die Einbeziehung der Nachbarschaft würde einen deutlich höheren Wert von 0,73 (textuelle Ähnlichkeit von Verpackung zu Waren verpacken) ergeben, in welche nur die in Abbildung 22 orange markierten Elemente einfließen würden.

Die hier beschriebene Erweiterung bietet prinzipbedingt nur die Möglichkeit, die ermittelte Ähnlichkeit auf dem gleichen oder niedrigeren Niveau wie ohne Erweiterung zu belassen. Die Evaluierung in Abschnitt 8.1 zeigt jedoch, dass dies von Vorteil ist, da hierdurch strukturell und inhaltlich ähnliche Prozesse stärker aus der Masse der durchschnittlich ähnlichen Prozesse herausstechen.

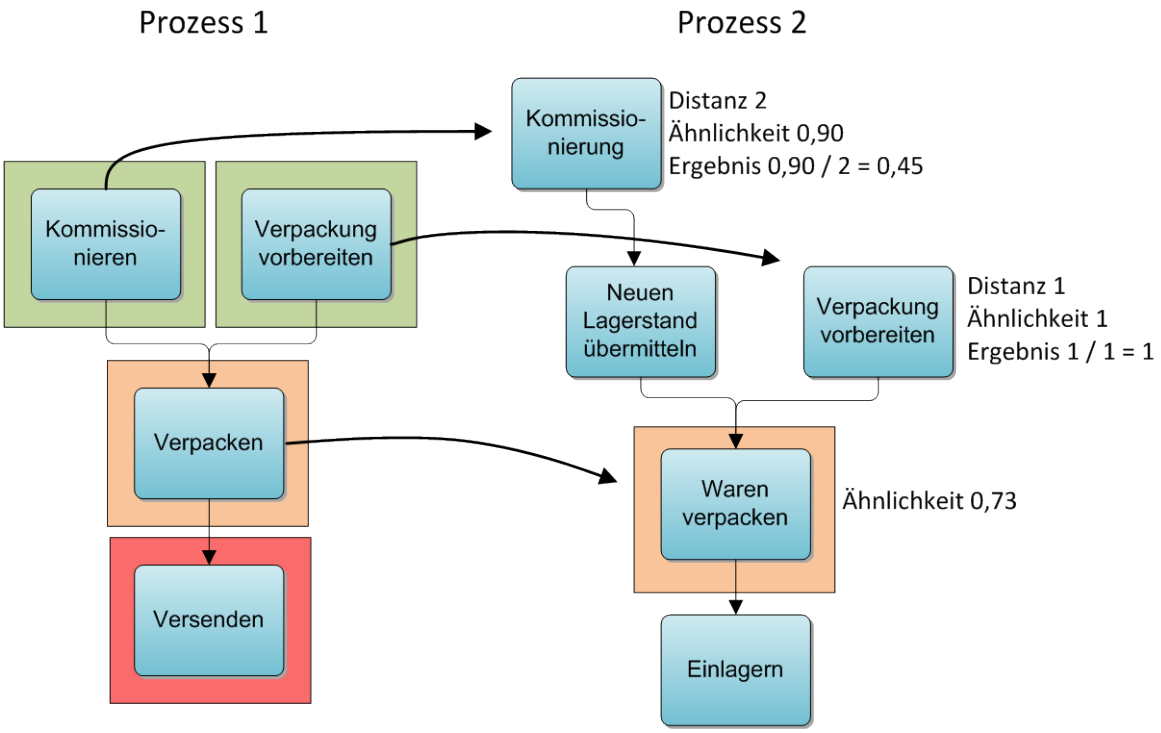

Abbildung 22: Beispiel für die erweiterte Ähnlichkeitsberechnung mit Nachbarschaftsbeziehungen

Quelle: Eigene Erstellung

Der Pseudocode für die Methode SimEnhanced ermöglicht eine praktische Umsetzung der oben beschriebenen Abläufe. Hierzu wird zuerst in den gebildeten Pärchen nach den durch Graph 1

bekannten Vorgängern und Nachfolgern gesucht. Anschließend wird deren Distanz zum Originalknoten errechnet und entsprechend der obigen Formeln aufsummiert. In diesem Algorithmus werden die zwei weiter unten beschriebenen Hilfsalgorithmen aufgerufen, welche für die Distanzbestimmung zuständig sind.

Algorithmus: SimEnhanced(mappingToCheck, mappings, edgesG1, edgesG2)

Eingabe:
mappingToCheck Pärchen, deren Editierdistanz berechnet werden soll, mappings enthält alle Pärchen einer möglichen Lösung, edgesG1 alle Kanten des Ausgangsgraphen, edgesG2 alle Kanten des zu vergleichenden Graphen

Ausgabe:
Ermittelte Ähnlichkeit

Variablen:
initalNodeG1 Knoten, dessen Vorgänger/Nachfolger untersucht werden sollen in Graph 1, initalNodeG2 Knoten, deren Vorgänger/Nachfolger untersucht werden sollen in Graph 2, preMappings Pärchen für vorhergehende Elemente, postMappings Pärchen für nachfolgende Elemente,
preCountsSum Anzahl der direkten Vorgänger, postCountSum Anzahl der direkten Nachfolger, ed errechnete Editierdistanz

```
 1  initalNodeG1 = mappingToCheck.NodeG1;
 2  initalNodeG2 = mappingToCheck.NodeG2;
 3
 4  preMappings = {};
 5  postMappings = {};
 6  preCountsSum = 0;
 7  postCountSum = 0;
 8
 9  double ed = 0.0;
10
11  // suche nach vermuteten Vorgängern
12  für i=0 ... edgesG1
13  {
14        e = edgesG1[i];
15
16        falls e.end == initalNodeG1 dann
17        {
18              preCountsSum++;
19
20              für j=0 ... mappings
21              {
22                    mapping = mappings[j];
23
24                    wenn mapping.NodeG1 == e.start dann
25                    {
26                          preMappings.Add(mapping);
27                    }
28              }
29        }
30  }
31
32  // suche nach vermuteten Nachfolgern
33  für i=0 ... edgesG1
34  {
```

```
35          e = edgesG1[i];
36
37          falls e.start == initalNodeG1 dann
38          {
39                  preCountsSum++;
40
41                  für j=0 ... mappings
42                  {
43                          mapping = mappings[j];
44
45                          wenn mapping.NodeG1 == e.end dann
46                          {
47                                  preMappings.Add(mapping);
48                          }
49                  }
50          }
51  }
52
53  für i=0 ... preMappings
54  {
55          mappingPre = preMappings[i];
56
57          distance = NodeDistancePre(initialNodeG2, mappingPre.NodeG2, edgesG2);
58
59          equalPercent = 1 - EdDistanz(mappingPre.NodeG1.Label, map-
60  pingPre.NodeG2.Label)/Max(mappingPre.NodeG1.Label,mappingPre.NodeG2.Label);
61          equalPercentWithDistance = 0;
62          falls distance != 0 dann
63          {
64                  equalPercentWithDistance = equalPercent / distance;
65          }
66          ed = ed + equalPercentWithDistance;
67  }
68
69  für i=0 ... postMappings
70  {
71          mappingPost = postMappings[i];
72
73          distance = NodeDistancePre(initialNodeG2, mappingPost.NodeG2, edgesG2);
74
75          equalPercent = 1 - EdDistanz(mappingPost.NodeG1.Label, mapping-
76  Post.NodeG2.Label)/Max(mappingPost.NodeG1.Label,mappingPost.NodeG2.Label);
77          equalPercentWithDistance = 0;
78          falls distance != 0 dann
79          {
80                  equalPercentWithDistance = equalPercent / distance;
81          }
82          ed = ed + equalPercentWithDistance;
83  }
84
85  retuniert ed/(preCountsSum + postCountSum);
```

Programmausdruck 6: Pseudocode für die Berechnung der Nachbarschaftseditierdistanz

Quelle: Eigene Erstellung

Hier sind nun die zuvor erwähnten Hilfsalgorithmen für die Berechnung der Distanzen angeführt. Da der Algorithmus NodeDistancePost analog zu dem vollständig ausformulierten Pseudocode von NodeDistancePre funktioniert, ist dieser nur verkürzt dargestellt.

Algorithmus: NodeDistancePre(nodeToStart, nodeToSearch, edges)

Eingabe:
nodesToStart speichert den Startknoten, nodeToSearch speichert den Knoten, welcher gesucht werden soll, edges speichert alle Kanten (eine Kante besteht aus einem Start- und einem Endknoten) des zu untersuchenden Graphen

Ausgabe:
Ermittelte Entfernung, 0 wenn der zu suchenden Knoten nicht gefunden wurde

Variablen:
steps ermittelte Entfernungen (Kanten zwischen Ausgangs- und Zielknoten), found Hilfsvariable, um die Schleife zu beenden, nextStep Stack zum Speichern des als nächsten zu untersuchenden Knoten

```
 1  steps = 0;
 2  found = false;
 3
 4  nextStep = {};
 5  nextStep.push(nodeToStart);
 6
 7  solange found == false V Count(nextStep) >= 0
 8  {
 9        steps++;
10
11        nodeForNextStep = nextStep.Pop;
12        für i = 0 ... edges
13        {
14              e = edges[i];
15
16              falls e.start == nodeToSearch V e.end == nodeForNextStep dann
17              {
18                    found = true;
19              }
20
21              falls e.end == nodeForNextStep dann
22              {
23                    nextStep.Push(e.start);
24              }
25        }
26  }
27
28  retuniert steps;
```

Programmausdruck 7: Hilfsalgorithmus für die Bestimmung der Distanz eines Vorgängers

Quelle: Eigene Erstellung

Es folgt der zuvor erwähnte verkürzt dargestellte Pseudocode.

Algorithmus: NodeDistancePost(nodeToStart, nodeToSearch, edges)

Eingabe:
nodesToStart speichert den Startknoten, nodeToSearch speichert den Knoten, welcher gesucht werden soll, edges speichert alle Kanten (eine Kan-

te besteht aus einem Start- und einem Endknoten) des zu untersuchenden Graphen

Ausgabe:
Ermittelte Entfernung, 0, wenn der zu suchenden Knoten nicht gefunden wurde

```
1   //funktioniert analog zu NodeDistancePre, Entfernung wird hier allerdings für
2   nachfolgende Elemente untersucht.
```

Programmausdruck 8: Hilfsalgorithmus zur Bestimmung der Distanz eines Nachfolgers

Quelle: Eigene Erstellung

Ein teilweise vergleichbarer, vereinfachter Ansatz wird in der Arbeit „Process Mining by Measuring Process Block Similarity“ von Bae et al. verwendet, um die Ähnlichkeit ganzer Prozesse zu bestimmen. Problematisch bei dem dort verwendeten Ansatz ist, dass dieser nur bei vollständig übereinstimmenden Knotenelementen funktioniert. (Vgl. Bae, Caverlee, Liu, Rouse, & Yan, 2006, S. 1-17)

5.6 Diskussion und weiterführende Literatur

Zu diesem Kapitel lässt sich abschließend sagen, dass die in der Einleitung gestellten Forschungsfragen und Anforderungen bezüglich der Ähnlichkeitsmessung erfüllt worden sind. So können mit den vorgestellten Algorithmen ähnliche Prozesse identifiziert werden. Außerdem führt die verstärkten Rücksichtnahme auf die Prozessstruktur (vergleiche Abschnitt 5.5), verglichen mit den anderen hier genannten Arbeiten dazu, dass besonders für eine Vereinigung geeignete Prozesse identifiziert werden können. So ermöglicht es die vom Autor hier beschriebene Erweiterung bestehender Ansätze, dass besonders ähnliche Prozesse stärker aus der Masse der nur durchschnittlich ähnlichen Prozesse herausstechen. Dieser Vorteil zeigt sich auch in der Evaluierung der hier beschriebenen Ansätze, welche in Kapitel 8 erfolgt.

Aufgrund des hohen Aufwands und des hohen Bedarfs an Rechenleistung, die eine Ähnlichkeitsmessung benötigt, gilt es, für zukünftige Forschungen noch Verbesserungen in diesem Bereich zu finden. Grundlegen Aspekte wurden hierbei bereits von Yan et al. in „Fast Business Process Similarity Search with Feature-Based Similarity Estimation“ behandelt. Die zuvor genannte Arbeit versucht durch die Identifikation und Überprüfung von sogenannten Features (z.B. das Vorkommen bestimmter Aufgaben) Prozesse zu grob zu klassifizieren und vor allem in die Gruppen "potentiell interessant" und "uninteressant" einzuteilen. Hierdurch sollen Prozesse, welche auf keinen Fall interessant sind, schnell ausgefiltert und nicht weiter untersucht werden. Nur die potentiell interessanten Prozesse werden anschließend mit genaueren Methoden untersucht. (Vgl. Yan, Dijkman, & Grefen, 2010, S. 1-18)

Weitere Ansätze versuchen die Analysemethoden selbst mit einem höheren Durchsatz zu erledigen. So können, wie in „Fast suboptimal algorithms for the computation of graph edit distance“ von Neuhaus et al. beschrieben, Heuristiken eingesetzt werden, um die Messungen zu beschleunigen. Hierbei ist es wie bei dem zuvor beschriebenen Ansatz wichtig, mittels Parametrisierung der Algorithmen die richtige Mischung zwischen Beschleunigung und Genauigkeit zu finden, da es ansonsten zu einer unzureichenden Lösungsqualität kommen kann. (Vgl. Neuhaus, Riesen, & Bunke, 2006, S. 1-10) Eine weitere Möglichkeit wäre die Implementierung eines auf dem Verzweigung und Schranke (engl. Branch and Bound) Konzept (Vgl. Lawler & Wood, 1966, S. 701ff) basierenden Ansatzes. Hierdurch können in dem hier beschriebenen rekursiven Verfahren Teile der entstehenden Baumstruktur eingespart werden. Um eine schnelle Ermittlung einer unteren Schranke zu erreichen, eignet sich die Berechnung der textuellen Editierdistanz der erzeugten

Pärchen. Hierbei kann auf weiterführende Analysen wie die Ermittlung gelöschter Kanten verzichtet werden, um den Durchsatz der Algorithmen zu steigern.

Eine genauere Auflistung, Untersuchung und Bewertung verschiedener Analysemethoden findet sich in der Arbeit „Managing Large Collections of Business Process Models – Current Techniques and Challenges“ von Dijkman et al. Außerdem erfolgt dort eine generelle Betrachtung der Enwicklungen und Richtungen des Managements von Prozessmodellen. (Vgl. Dijkman, Rosa, & Reijers, Managing Large Collections of Business Process Models – Current Techniques and Challenges, 2011, S. 2-9)

6 Vereinigen unterschiedlicher Prozessmodelle

Nachdem in den vorhergehenden Kapiteln die Vorarbeit dafür geleistet wurde, dass, wie in Kapitel 5 beschrieben, ähnliche Prozessmodelle automatisiert gefunden werden können, gilt es nun, diese zu vereinigen (engl. „merge“). Dies ermöglicht es, entsprechend der in der Einleitung formulierten wissenschaftlichen Fragestellung die Anzahl der im Unternehmen befindlichen Prozesse zu reduzieren und führt hierdurch zu den bereits zuvor angesprochenen Vorteilen wie einem verringerten Wartungsaufwand.

Ein mögliches Vorgehen für eine praktische Anwendung wäre es, hierbei die zuvor erstellte Ähnlichkeitsanalyse auf eine Prozessdatenbank anzuwenden. Die hierbei ermittelten Ähnlichkeiten werden anschließend dazu verwenden, automatisch für den Anwender Vorschläge zu generieren, welche Prozesse sich gut für eine Vereinigung eignen würden. Der Anwender muss anschließend nur noch die Entscheidung fällen, ob er einer Vereinigung zustimmt oder nicht.

Es existiert daher eine Menge von Prozessen P_1 bis P_n wobei P_i =(V, E). V ist eine nicht leere Menge der Knoten (Aufgaben, Gateways etc.) und E eine Menge der Verbindungen zwischen den Knoten. Diese Prozesse werden anschließend mittels einer abstrakten Vereinigungsoperation $\cup$ zu einem Prozess P^* vereinigt. Die konkrete Umsetzung von $\cup$ kann dann je nach Umsetzung und Anforderungen unterschiedlich realisiert werden.

$$P^* = \bigcup_{i \in I} P_i$$

Formel 7: Vereinigung von Prozessmengen

Quelle: Eigene Erstellung

Das Ergebnis dieser Vereinigung P^* soll allgemein die verschiedenen Aufgaben der vereinigten Prozesse widerspiegeln. Als Anforderung für die Ergebnisqualität wurde für diese Arbeit festgelegt, dass die Vereinigung einen Prozess erzeugen soll, welcher die zuvor beschriebenen Korrektheitskriterien (siehe 2.3) einhält und möglichst wenig Nachbearbeitung für die Übernahme in den Produktivbetrieb benötigt. Ein Ergebnis zu erzeugen, welches ohne weitere Kontrolle in den Geschäftsalltag übernommen werden kann, ist bei einfachen Prozessen grundsätzlich möglich. Aufgrund der hohen Komplexität der sich im Einsatz befindlichen Prozesse und der großen Bedeutung von Prozessen für den Unternehmenserfolg ist eine abschließende Kontrolle allerdings zumeist anzuraten. Die automatisierte Vereinigung hilft hierbei dann, den entstehenden Arbeitsaufwand für die Vereinigungen deutlich zu reduzieren.

Es gilt nun den Vereinigungsoperator $\cup$ genauer zu definieren. Die einfachste Umsetzung wäre eine aus der Mengenlehre bekannte Vereinigung aller in den Prozessen bekannten Elemente. Das Ergebnis würde hierbei alle Elemente aller vereinigten Prozesse beinhalten. Ein solcher Ansatz wird in „Merging business process models“ von Rosa et al. (Vgl. Rosa, Dumas, Uba, & Dijkman, 2010) beschrieben. Der Vorteil dieses Ansatzes ist, dass er sich sehr einfach realisieren lässt. Dem

gegenüber stehen allerdings zahlreiche Nachteile. So wird hierdurch ein äußerst komplexer Prozess erzeugt. Dadurch, dass alle Elemente vorbehaltslos übernommen wurden, muss der Ergebnisprozess vor dem Produktiveinsatz konfiguriert, also der Kontrollfluss definiert werden. Das Ergebnis ist aufgrund der Komplexität kaum noch zu warten und nur schwer zu erweitern. Alle diese Probleme führen dazu, dass ein solches Vorgehen als nicht praxistauglich anzusehen ist und hiermit keines der Ziele dieser Arbeit, wie ein verringerter Wartungsaufwand, erreicht werden kann.

Zur Verdeutlichung des hierbei verwendeten Vorgehens wird folgende Grafik verwendet. Hierbei werden der Prozess 1 und der Prozess 2 in dem Ergebnisprozess vereint. Basierend auf den oben genannten Ansatz führt dies dazu, dass das Ergebnis alle Pfade der vereinigten Prozesse abbildet und daher deutlich mehr Verbindungen zwischen den Knoten bestehen als in den Originalprozessen. Diese Art der Vereinigung führt dazu, dass nicht nur die zuvor möglichen Ausführungspfade im vereinigten Prozess verfügbar sind, sondern auch noch zusätzliche neue Ausführungsmöglichkeiten entstehen. So besteht im Ergebnisprozess beispielsweise die Möglichkeit, folgende Ausführung durchzuführen: A→ B → C. Dieser Pfad war in keinem der beiden Ausgangsprozesse möglich. Möchte man dies verhindern, müssen redundante Elemente eingeführt werden. So kann für dieses spezielle Szenario das Element B verdoppelt werden, sichtbar im alternativen Ergebnisprozess mit B1, B2, sodass das alternative Ergebnis den zuvor beschriebenen Pfad nicht mehr aufweist. Hiermit sind allerdings noch nicht alle dieser problematischen zusätzlichen Ausführungspfade behoben, sodass beispielsweise mit dem Element Y eine vergleichbare Vorgehensweise angewendet werden müsste.

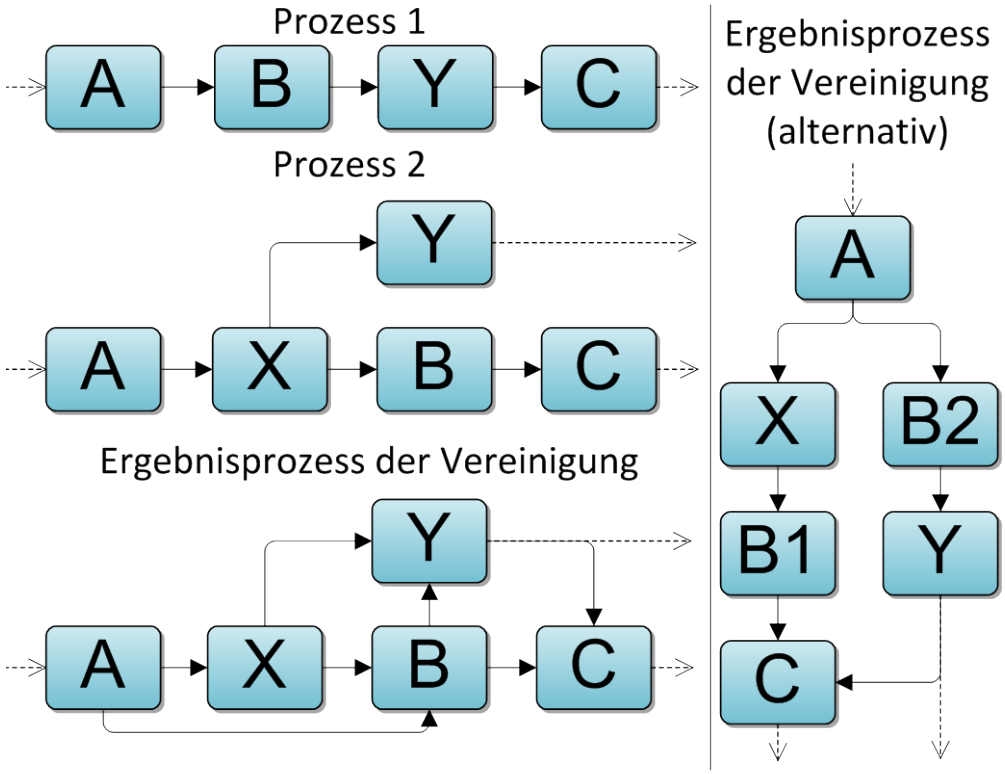

Abbildung 23: Beispiel für eine Vereinigung unter Beibehaltung aller Ausführungspfade

Quelle: Eigene Erstellung

Da sich der zuvor genannte Ansatz für die Anforderungen dieser Arbeit nicht eignet, ist es notwendig, eine alternative Herangehensweise zu finden. Betrachtet man hierzu andere Bereiche der Informatik, zeigt sich, dass das Vereinigen von unterschiedlichen Daten insbesondere in der Soft-

wareentwicklung und in den dort verwendeten Versionsverwaltungssystemen weit verbreitet und gut unterstützt ist. (Vgl. Baerisch, 2005, S. 14ff)

Genützt werden hierbei beispielsweise Algorithmen, welche auf dem "Longest Common Subsequence" (LCS) Algorithmus basieren. Bei diesem handelt es sich um einen Algorithmus, der erstmals von Douglas McIlroy und James W. Hunt in "An Algorithm for Differential File Comparison" 1976 beschrieben wurden. Nach den beiden Forschern wird dieser Ansatz auch als Hunt-McIlroy-Algorithmus bezeichnet. Dieser versucht, die längstmögliche, nicht unbedingt zusammenhängende Teilsequenz mehrerer Zeichenketten zu finden. (Vergleiche hierzu auch die vorhergehenden Abschnitte 3.3.1 und 5.1.)

Dieser Ansatz eignet sich gut für die Analyse von Texten. Die Ergebnisqualität für andere Datenarten ist allerdings auf einem Niveau, welches hinter den Möglichkeiten zurückbleibt. Dies führte beispielsweise dazu, dass für XML-Dateien zahlreiche unterschiedliche Methoden entwickelt wurden, um Unterschiede zwischen diesen zu messen und zu erfassen. (Vgl. Peters, 2005, S. 1-8) Anschließend können diese so erfassten Unterschiede vereinigt werden und hierdurch auch eine Vereinigung der erfassten Datensätze erreichen.

Ähnlich wie die zuvor genannten Ansätze für XML-Dateien wird daher auch in dieser Arbeit angestrebt, einen spezialisierten Algorithmus zu entwickeln. Da einige grundlegende Informationen über die Art und Struktur der Daten während der Entwicklung bekannt sind, soll dies erlauben, ein Ergebnis zu liefern, das die an diese Arbeit gestellten Anforderungen zu einem höheren Grad erfüllt als die zuvor beschriebene generische Lösung.

Allgemein muss bei der Vereinigung von Modellen zwischen der strukturbasierten und der verhaltensbasierten Vereinigung unterschieden werden. Der strukturbasierte Ansatz eignet sich gut für Modelle wie ER-Diagramme, welche keinen klaren Startpunkt besitzen und welche man daher anhand der Struktur ihrer Elemente aufeinander abbilden muss. (Vgl. Sabetzadeh & Nejati, 2006, S. 1-3) Beim verhaltensbasierten Ansatz steht das Verhalten des Modells bei dessen „Ausführung“ im Vordergrund. (Vgl. Uchitel & Chechik, 2004, S. 1-10) Es eignet sich daher für Modelle, die einen Zustand besitzen und sich deshalb beispielsweise als Automaten repräsentieren lassen; entsprechend eignet es sich auch für Prozessmodelle.

Der hier vorgestellte Ansatz setzt eine Kombination strukturbasierter, für die initiale Erkennung von Überlappungen und Zusammenhängen und verhaltensbasierter Methodiken zur anschließenden Verfeinerung der Ergebnisse ein.

Bevor allerdings die eigentlichen Algorithmen beschrieben werden, gilt es noch zu klären, welche Daten für die Verschmelzung benötigt werden (Abschnitt 6.1) und wie vorgegangen werden muss, wenn mehrere Prozesse gleichzeitig miteinander verschmolzen werden sollen (Abschnitt 6.2).

6.1 Drei- bzw. Zwei-Wege-Vereinigung

Eine der wichtigsten Entscheidungen beim Design eines den Anforderungen entsprechenden Algorithmus ist es, ob dieser als Drei- oder als Zwei-Wege-Vereinigung implementiert werden soll. Bei der Zwei-Wege-Vereinigung stehen nur die Informationen der zu vereinigenden Prozesse zur Verfügung. (Vgl. Mens, A State-of-the-Art Survey on Software Merging, 2002, S. 452f) Bei der Drei-Wege-Vereinigung sind auch Informationen über die Eltern der Prozesse (Vergleiche Abschnitt 3.2.5) vorhanden. Als Eltern können hierbei die Prozesse verstanden werden, welche als Basis für die Generierung der nun zu vereinigenden Kindprozesse gedient haben. (Vgl. Mens, A State-of-the-Art Survey on Software Merging, 2002, S. 453-455)

Prinzipiell lässt sich annehmen, dass sich bei den meisten länger im Einsatz befindlichen Prozessen sogenannte Prozessfamilien bilden. Eine Prozessfamilie ist hierbei eine Gruppe von ähnlichen Prozessen, welche sich auf verwandte Themen beziehen. Aufgrund dieses Naheverhältnisses existieren oft Beziehungen zwischen diesen Prozessen, welche dazu führen, dass diese voneinander abgeleitet sind und daher eine Vater-Kind-Beziehung besitzen. (Vgl. Giese, Overdick, & Buhl, 2005, S. 3ff)

Durch die Mehrinformationen (vor allem bezüglich der vollzogenen Modifikationen zwischen Vater- und Kindprozessen), welche bei der Drei-Wege-Vereinigung zur Verfügung stehen, ist es möglich, zusätzliche Aussagen zu treffen. Dieser Umstand wird nun im Anschluss anhand des in Abbildung 24 gezeigten Beispiels verdeutlicht. Es zeigt drei Prozesse, welche sich in einen Vater- und zwei Kindprozesse aufteilen. Die beiden Kindprozesse sollen nun vereinigt werden. Verwendet man einen auf der Zwei-Wege-Vereinigung basierenden Ansatz, dann kann dieser Unterschiede zwischen den Prozessen erkennen, wie z.B., dass der erste Kindprozess das Element "C" beinhaltet, der zweite Kindprozess jedoch nicht. Schwierigkeiten bereitet allerdings herauszufinden, ob das erwähnte Element beispielsweise komplett neu in den ersten Kindprozess hinzukam oder ob es ein bereits länger existierendes Element ist, welches im zweiten gelöscht wurde.

Gelöst kann dieses Problem werden, indem die Informationen des Vaterprozesses herangezogen werden. Dies ermöglicht es zu erkennen, dass es sich bei dem Element "C" um ein komplett neues Element handelt, das zuvor noch nicht aufgetreten ist. Mittels der Informationen des Vaterprozesses kann weiters auch die Automatisierbarkeit der Vereinigung sowie der Auflösung von Konflikten, ein wichtiger Aspekt der hier zu entwickelnden Lösung, der Vereinigungsabläufe, erhöht werden, indem, wenn ein Prozesselement in beiden Kindprozessen eine unterschiedliche Beschreibung besitzt, überprüft werden kann, welcher Kindprozess das Element unverändert vom Vater übernommen hat und welcher es modifizierte. Diese Information kann abschließend dazu verwendet werden, zu entscheiden, ob das modifizierte oder das unveränderte Element übernommen werden soll.

Die folgende Grafik visualisiert die obige Problemstellung. Die grünen Pfeile stellen dabei Arbeitsschritte dar, welche nur bei einer Drei-Wege-Vereinigung durchgeführt werden können.

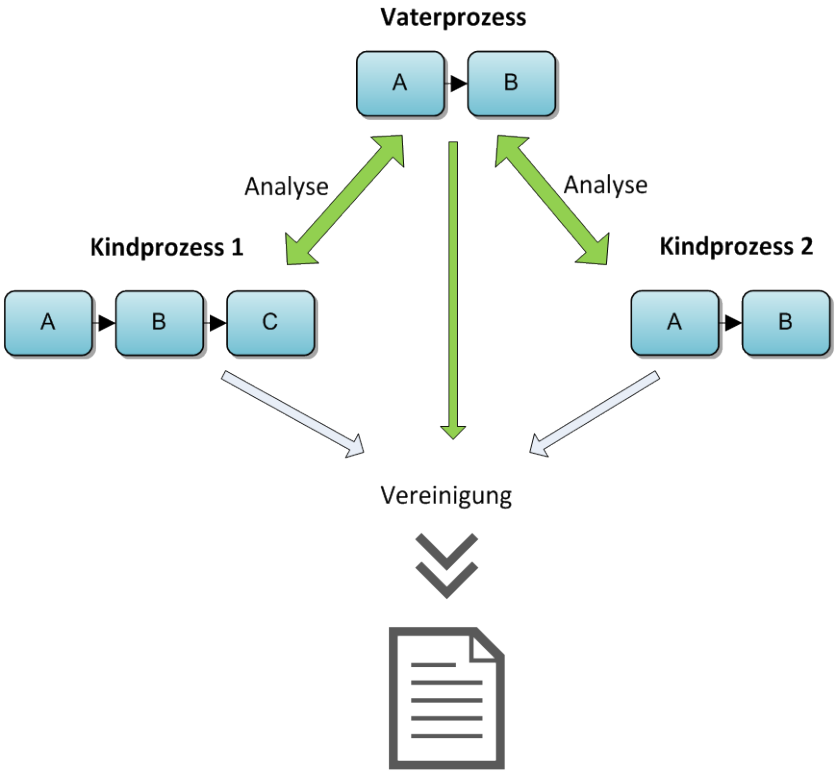

Abbildung 24: Verdeutlichung der Abläufe einer Zwei- bzw. Drei-Wege-Vereinigung

Quelle: Eigene Erstellung

Die für diese Arbeit entwickelten Algorithmen unterstützen aufgrund der weiter oben erwähnten Vorteile den Ansatz der Drei-Wege-Vereinigung. Stehen allerdings ausreichende Informationen zum Vaterprozess nicht zur Verfügung, kann auch eine Zwei-Wege-Vereinigung durchgeführt werden.

6.2 Gleichzeitige Vereinigung mehrerer Prozesse

Nachdem nun geklärt ist, welcher grundlegende Ansatz für die Vereinigung verwendet wird, gilt es noch festzulegen, wie mehrere (Anzahl größer als zwei) Prozesse "gleichzeitig" miteinander verschmolzen werden können. Dass von dieser Möglichkeit Gebrauch gemacht wird, ist wahrscheinlich, befinden sich doch in Unternehmen oft Tausende von Prozessen im Einsatz, welche häufig ähnliche Bereiche abdecken (Vgl. Dongen, Dijkman, & Mendling, 2008, S. 1f). Werden nun mit der im vorherigen Kapitel beschriebenen Ähnlichkeitsmessung mehrere solche Prozesse gefunden, gilt es dann auch, diese gemeinsam in eine neue, einheitliche Form zu überführen.

Für die Lösung dieses Problem gibt es zwei grundsätzliche Möglichkeiten: Die erste Möglichkeit ist eine Implementierung, die alle Elemente in einem einzigen Schritt verarbeitet und vereinigt („n-ary merge“). (Vgl. Zwart, 2009, S. 6) Der Vorteil dieses Ansatzes ist, dass gleichzeitig alle Informationen der zu verarbeitenden Prozesse zur Verfügung stehen. Vorteile ergeben sich hierbei dadurch, dass Konflikte durch Mehrheitsentscheidungen gelöst werden können. Eine solche Mehrheitsentscheidung kann notwendig sein, wenn konkurrierende Änderungen vorliegen. Eine solche konkurrierende Änderung kann beispielsweise entstehen, wenn gleiche Elemente an den gleichen Stellen in unterschiedlicher Weise modifiziert wurden. Der Zustand, der dann bei der größten Anzahl der Prozesse vorliegt, wird aufgrund dessen dann übernommen. Die Nachteile betreffen vor allem einen höheren Aufwand bei der Implementierung und einen erhöhten Bedarf

an Rechenleistung, da dann die Prozesselemente aller zu vereinigenden Prozesse gleichzeitig im Speicher gehalten werden müssen. Verschiedene weitere Möglichkeiten werden im folgenden Abschnitt angeführt.

Die Alternative zu diesem Ansatz ist, von der Menge der zu vereinigenden Prozesse immer eine kleine Anzahl, z.B. zwei Prozesse, herauszugreifen und nur diese zu vereinigen (binary merge). (Vgl. Griffin, Colcheste, Roll, & Studholme, 1994, S. 2f) Dies ermöglicht es, mit einem geringeren Bedarf an Rechenleistung auszukommen. Weiters vereinfacht sich die Implementierung, da bereits während der Konzeption des Algorithmus genau bekannt ist, wie viele Prozesse später maximal gleichzeitig zu behandeln sind. Sollen mehr als die hierbei berücksichtigten Prozesse vereinigt werden, muss dies in mehreren Schritten geschehen, wie dies in Abbildung 25 gezeigt wird. Nachteilig ist, im Vergleich mit der zuvor vorgestellten Alternative, der geringere Informationsgehalt, der dem Algorithmus gleichzeitig zur Verfügung steht.

Die folgende Grafik zeigt im linken Teilbereich das Vorgehen, wenn immer nur ein Teil der Prozesse, hier zwei Prozesse, miteinander vereint werden können. Hierzu werden zu Beginn zwei zu vereinigende Prozesse gewählt und vereint. Anschließend wird das so gewonnene Ergebnis wieder mit einem weiteren Prozess vereint. Dies geschieht so lange, bis alle Prozesse abgearbeitet wurden. Rechts ist dagegen die Vereinigung aller Prozesse in einem Schritt zu sehen, in welchem das links zu sehende gestaffelte Vorgehen nicht notwendig ist. Die Zahnräder repräsentieren hierbei eine Anwendung eines Vereinigungsalgorithmus.

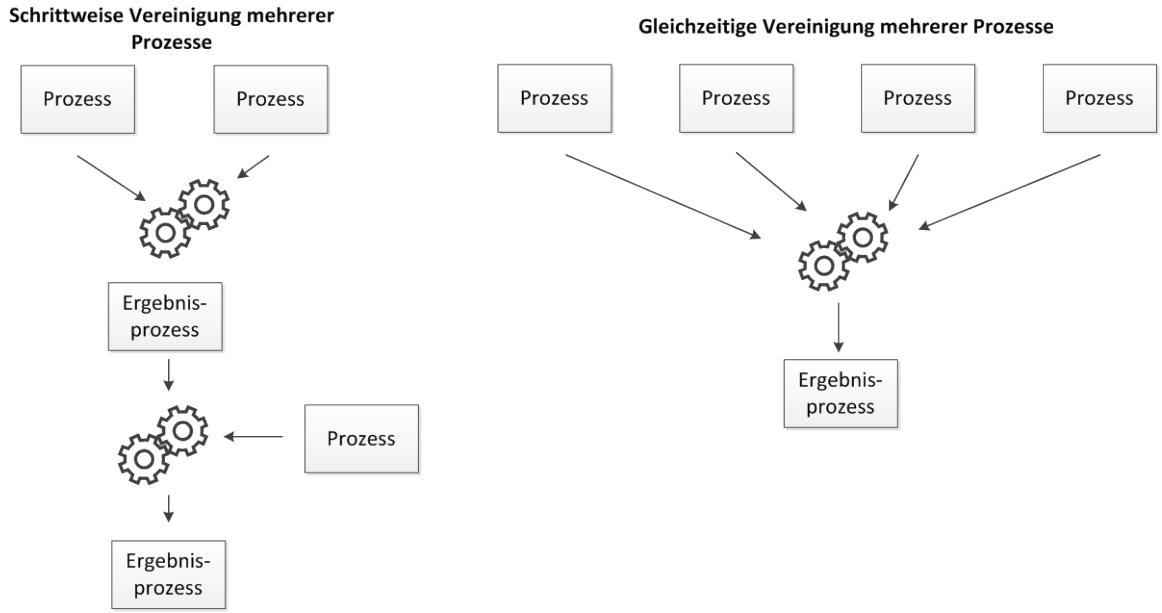

Abbildung 25: Methoden für die Vereinigung mehrerer Prozesse

Quelle: Eigene Erstellung

6.3 Behandlung von Konflikten

Wie zuvor beschrieben, kann es bei der Vereinigung von Informationen – Prozessmodelle bilden hier keine Ausnahme – zu Konflikten kommen. Ein Konflikt tritt auf, wenn nicht eindeutig ersichtlich ist, welche Änderungen von den verschiedenen Prozessen übernommen werden sollen. Beispielsweise kann in zwei zu vereinigenden Prozessen das gleiche Element auf unterschiedliche Art und Weise verändert worden sein. Zu entscheiden, welche der beiden Änderungen nun "gewinnt", gestaltet sich je nach vorhandener Datenbasis schwierig. Zusätzlich gilt es natürlich auch, verschiedene Korrektheitskriterien (Vgl. hierzu Abschnitt 2.3) bei den durch die Vereinigung er-

zeugten Prozessen einzuhalten. Es gilt daher Strategien zu finden, welche es ermöglichen, mit diesen Problemen umzugehen. (Vgl. Bleiholder & Naumann, 2006, S. 10f)

Eine allgemeine Übersicht über verschiedene Methoden zur Konfliktbehandlung ist in der Folge ersichtlich.

- *Konflikte ignorieren*: Dies stellt den einfachsten Ansatz dar. Problematische Elemente werden unverändert übernommen, sodass im Endergebnis bei konfliktbehafteten Änderungen alle möglichen Ausprägungen enthalten bleiben. Dies ergibt komplexe Ergebnisse, welche manuell nachbearbeitet werden müssen. (Vgl. Bleiholder J. , 2010, S. 32ff)
- *Konflikte auflösen*: Hierbei wird versucht, Konflikte zu erkennen und diese mit verschiedenen Strategien aufzulösen. Geschieht dies automatisiert, wird hierdurch, im Vergleich zum vorangegangenen Ansatz, der Endanwender entlastet. Eine exemplarische Liste möglicher Vorgehensweisen ist nachfolgend angeführt. (Vgl. Bleiholder J. , 2010, S. 32ff)
    - *Mehrheitsentscheidung*: Hierbei werden die unterschiedlichen Ausprägungen eines konfliktbehafteten Elements gezählt. Die Ausprägung, welche am öftesten vorkommt, wird anschließend übernommen. (Vgl. Bleiholder J. , 2010, S. 32ff)
    - *Zufallsentscheidung*: Die Wahl der übernommenen Elementausprägung wird zufällig getroffen. (Vgl. Bleiholder J. , 2010, S. 32ff)
    - *Versuchte Vereinigung*: Trotz eines Konfliktes wird hierbei versucht, die verschiedenen Daten, welche zueinander in einem Konflikt stehen, zusammenzuführen. Ein vermutlich nicht optimales Ergebnis wird hierbei in Kauf genommen, um diesen Schritt automatisch durchzuführen. (Vgl. Bleiholder J. , 2010, S. 32ff)
    - *Neueste Information gewinnt*: Die Ausprägungen werden nach ihrer Aktualität gereiht. Der neuesten Ausprägung wird hierbei die höchste Relevanz zugerechnet und diese daher übernommen. (Vgl. Bleiholder J. , 2010, S. 32ff)
    - *Vertrauensbasierte Entscheidung*: Die Wahl des zu übernehmenden Elements zur Auflösung des Konfliktes wird bei dieser Methode vertrauensbasierend getroffen. Dabei wird einem jeden zu vereinigenden Prozess bzw. dessen Ersteller ein unterschiedlicher Wert zugewiesen, welcher es erlaubt, die bestehende Vertrauensbeziehung quantitativ zu messen. Diejenigen Elemente, zu welchen das höchste Vertrauen existiert, werden anschließend übernommen. (Vgl. Gatterbauer & Suciu, 2010, S. 219ff)
    - *Allgemeinen Regeln*: Hierbei werden feste Wenn-Dann-Regeln definiert, welche bestimmen, wie mit bestimmten Konfliktfällen umgegangen werden soll. Eine solche Regel kann sein, dass im Fall einer konfliktbehafteten Lösch/Update Operation immer die Löschung gewinnt und daher das gelöschte bzw. modifizierte Element nicht übernommen wird. (Vgl. Chomicki, Lobo, & Naqvi, 2003, S. 244ff)

Auch mit dem Festlegen von Reizschwellen kann gearbeitet werden. So wäre eine Lösung denkbar, welche mit einer festgelegten Reizschwelle von z.B. 90% arbeitet. Diese würde dann besagen, dass, wenn 90% der Inhalte übereinstimmen, diese als ident zu bewerten wären und dann kein Konflikt vorliegen würde. Die Problematik, dass es hierbei zu Fehlerkennungen kommen kann, ist allerdings bei solchen einfachen Methoden schwer zu vermeiden.

Als weitere Möglichkeit bietet sich auch an zu überprüfen, ob die als unterschiedlich detektierten Werte tatsächliche Unterschiede repräsentieren. Möglicherweise handelt es sich bei den geänderten Bereichen um Synonyme oder die Inhalte besitzen trotz der Änderung noch immer die gleiche semantische Bedeutung. Es lohnt sich daher, auch die im vorherigen Kapitel (Abschnitt 5.1) beschriebenen Ansätzen (Synonyme, Ontologien etc.) hier wieder mit einfließen zu lassen, um die Wahrscheinlichkeit für Fehlentscheidungen zu verringern.

Das Problem, dass es bei der Arbeit mit unterschiedlichen Daten zu Konflikten kommen kann, ist nicht neu. Es verlagert sich hier jedoch in einen neuen Forschungsbereich. Zahlreiche Lösungsansätze wurden bereits für andere Bereiche wie die Vereinigung von Extensible Markup Language (XML) Schemata (Vgl. Pottinger & Bernstein, 2003, S. 1ff) oder von Datenbanksystemen (Vgl. Parent & Spaccapietra, 1998, S. 1ff) entwickelt. Nach deren Adaption können diese anschließend zur Ergänzung der hier exemplarisch vorgestellten Techniken verwendet werden.

Ist eine automatische Entscheidung nicht möglich bzw. nicht mit der geforderten Qualität bzw. Aufgabenstellung zu vereinen, besteht als letzter Ausweg immer auch die Möglichkeit, beim Anwender, also einem Menschen, nachzufragen. Hierzu sind eine entsprechende Aufbereitung der Daten und Wahlmöglichkeiten sowie ein ausreichendes Wissen des Anwenders über die Daten, Ziele und Problemstellungen notwendig.

Auch teilweise gelöschte Elemente (Prozesselemente, welche im Vaterprozess, aber nur in einem Teil der Kinder vorhanden sind) stellen ein Problem dar, da hier unklar ist, ob ein Element bleiben soll oder ebenfalls entfernt werden muss. Bezüglich dieser Herausforderung wurde für die Entwicklung des hier beschriebenen Algorithmus entschieden, dass gelöschte Elemente immer entfernt werden.

Anschließend an die zuvor erfolgte theoretische Bearbeitung des Themas folgt hier nun eine praxisorientierte Betrachtung der im Rahmen dieser Arbeit entwickelten Lösung zur Vereinigung von Prozessen.

6.4 Entwickelte Algorithmen zur Prozessmodellvereinigung

Nachdem mit dem vorangegangenen Teil dieses Kapitels die theoretischen Probleme und Herausforderungen beleuchtet wurden, wird im Folgenden die praktische Seite untersucht. Zur erfolgreichen Erfüllung der gestellten Forschungsfragen ist es notwendig, eine Lösung zu entwickeln, welche mehrere Prozesse vereinigen kann. Der hierbei entstandene Ansatz gliedert sich grob in die folgenden, in Abbildung 26 visualisierten und sequenziell durchlaufenden Schritte. Diese werden, sobald notwendig, auch mit dem entsprechenden Pseudocode unterfüttert.

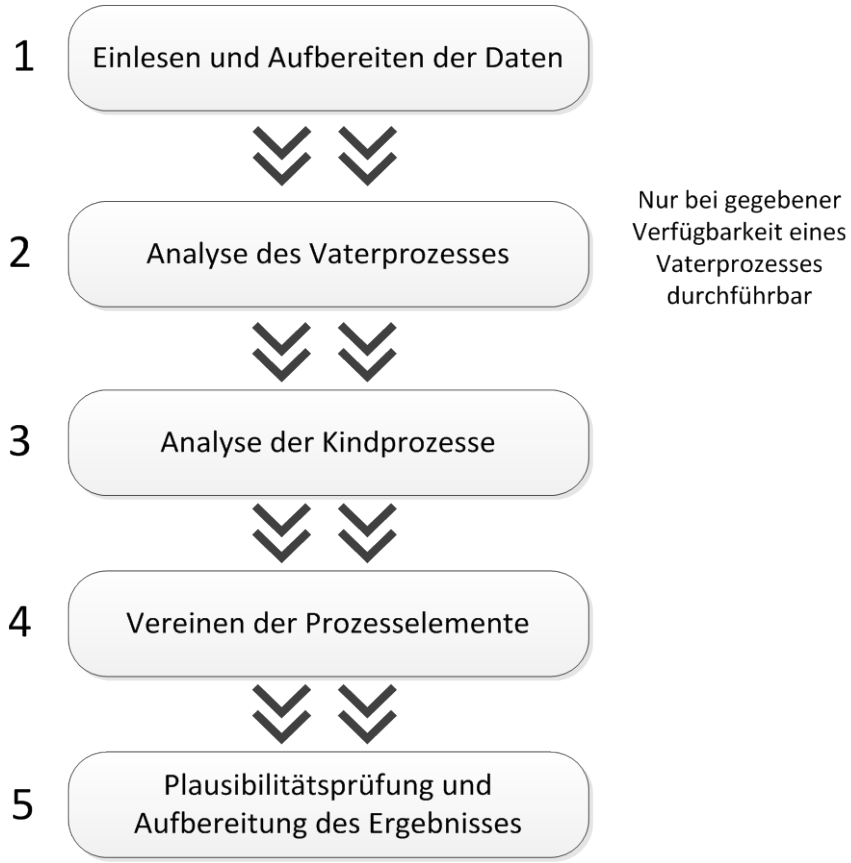

Abbildung 26: Reihenfolge der vollzogenen Schritte zur Prozessvereinigung

Quelle: Eigene Erstellung

Als Endergebnis wird im Gegensatz zum vorhergehenden Kapitel, wo es galt, einen Prozentwert als Repräsentation der Ähnlichkeit zwischen Prozessen zu ermitteln, versucht, basierend auf den zugeführten Prozessdefinitionen ein ausführbares, vereinigtes Prozessmodell zu erzeugen. Natürlich ist es je nach Komplexität der Prozesse nicht möglich, ein "perfektes" Ergebnis zu liefern. Die ermittelte Lösung versteht sich also mehr als Vorschlag mit dem Ziel, dass dieser möglichst wenig Nachbearbeitung benötigt, denn als Lösung, welche sofort in den Unternehmensalltag übernommen werden muss. Hierzu trägt insbesondere auch dazu bei, dass in dieser Arbeit nur der Kontrollfluss beachtet wird.

Es stellt sich jetzt die Frage, wie denn die Ergebnisse bewertet werden können. Insbesonders auch in Hinsicht darauf, einen Vergleich des hier beschriebenen Ansatzes mit anderen Vereinigungsmethodiken, wie mit dem zu Beginn dieses Kapitels beschriebenen klassischen Ansatzes, welcher die Erhaltung aller Ausführungspfade als Ziel hat (Vgl. Abbildung 23). Allgemein können hierbei verschiedene Maßzahlen herangezogen werden, um die Qualität eines Prozesses zu messen. Eine Möglichkeit stellen die von Mendling et al. in „Seven Process Modeling Guidelines“ vorgestellten sieben Best Practice Regeln für Prozessmodellierungstechniken dar. (Vgl. Mendling, Reijers, & Aalst, Seven Process Modeling Guidelines (7PMG), 2010, S. 130ff) Hierbei lassen sich allerdings nur die ersten drei Regeln sinnvoll anwenden. Diese besagen, dass Prozesse aus möglichst wenigen Elementen bestehen und die Anzahl der Pfade pro Element möglichst gering sein sollen. Zusätzlich wird geraten, nur ein Start- bzw. Endelement pro Prozessmodell zu verwenden. Die weiteren Regeln vier bis sieben können nicht verwendet werden, da diese direkt mit der Qualität der verarbeiteten Prozessdaten zusammenhängen – daher sollen OR Elemente vermieden und aussagekräftige Texte verwendet werden. Auf diese Informationen kann der automatische Vereinigungsalgorithmus keinen Einfluss nehmen. Eine Evaluierung anhand dieser und anderer Gesichtspunkte erfolgt in Abschnitt 8.2.

6.4.1 Einlesen und Aufbereiten der Daten

Zu Beginn müssen die zu verarbeitenden Daten in entsprechender Form eingelesen und so den Algorithmen zugänglich gemacht werden. Als Format für die Eingangsdaten wird hierbei der in Kapitel 4 beschriebene Aufbau verwendet. Das gleiche Format bildet auch die Basis für die Ausgangsdaten und dient zur Abbildung des vereinigten Prozesses.

Für eine eindeutige Identifikation der Elemente sind eindeutige IDs angedacht (Vgl. Abschnitt 4.1.1). Diese ermöglichen es unter anderem festzustellen, ob Elemente neu hinzugekommen sind (neue ID tritt auf, welche sich in keinem anderen Prozess findet) oder nur verändert wurden (bekannte ID tritt in mehreren Prozessen auf, aber Inhalte, wie der Beschreibungstext, sind unterschiedlich).

Um eine praxistaugliche Lösung zu finden, welche diese IDs zur Verfügung stellt, wäre eine entsprechende Unterstützung durch die Applikationen notwendig, welche zur Erstellung/Modifikation der Prozesse verwendet werden. Diese Unterstützung müsste insbesondere daraus bestehen, dass, wenn ein neuer Kindprozess aus einem Vaterprozess abgeleitet wird, die IDs von alten Elementen in die Elemente der neuen Kindprozesse mit übernommen werden und nicht verloren gehen.

Eine entsprechende Unterstützung dieses ID basierten Ansatzes ist auch in Praxislösungen wie „AristaFlow“, einer Prozess Management Lösung, gegeben. (Vgl. Atkinson & Stoll, 2008, S. 3f)

Da diese Unterstützung voraussichtlich nicht in allen Fällen gegeben sein wird, lässt sich dieser Umstand auch durch eine Ähnlichkeitsanalyse der verschiedenen Elemente angehen. Hierbei werden die im vorangegangenen Teil der Arbeit beschriebenen Ansätze zum Vergleichen von Texten (LCS, Ontologien etc.) und der Positionen von Elementen im Prozessablauf herangezogen. Gearbeitet werden kann hierbei mit konfigurierbaren Schwellwerten, welche definieren, ab wann ein Element als gleich bzw. verwandt angesehen wird. Basierend auf diesen Informationen kann anschließend unter anderem ermittelt werden, welche Elemente von einem Vater- auf einen Kindprozess übernommen wurden.

Von allen folgenden Analyseoperationen, wie der Suche nach gelöschten Prozesselementen, sind Knoten und Kanten gleichermaßen betroffen, sodass sowohl Änderung an der Knoten- wie auch an der Kantenmengen der zu vereinenden Prozesse erkannt werden. Deshalb ist es auch notwendig, dass sowohl Knoten wie auch Kanten mit eindeutigen IDs versehen sind.

6.4.2 Untersuchen des Vaterprozesses

Stehen Informationen zum Vaterprozess zur Verfügung, wird versucht, diese auszunützen, indem eine Drei-Wege-Vereinigung durchgeführt wird. Sind keine Daten zum Vaterprozess vorhanden, wird dieser Schritt übersprungen, was zum Grundansatz einer Zwei-Wege-Vereinigung führt. (Siehe hierzu Abschnitt 6.1)

Die Prozessinformationen bezüglich des Vaterprozesses werden hierbei, wie bereits im vorhergehenden Abschnitt erwähnt, dafür genützt, um folgende Informationen zu erhalten (wobei sowohl betroffene Knoten wie auch Kanten gesucht werden):

- Welche Prozesselemente wurden im Vergleich zum Vater gelöscht bzw. vollständig neu hinzugefügt?
- Welche Prozesselemente wurden im Vergleich zum Vater verändert (z.B. Modifikation der Attribute eines Elementes) und sind diese Änderung in allen Kindprozessen gleich bzw. unterschiedlich?
- Welche Prozesselemente wurden unverändert übernommen?

Mit diesen Informationen kann wiederum auf mehrere Arten umgegangen werden. Eine Möglichkeit besteht darin, dass, falls ein Element in einem der Kindprozesse im Vergleich zum Vater entfernt wurde, dieses auch im finalen Ergebnisprozess zu entfernen. Falls mehrere Prozesse gleichzeitigt vereinigt werden sollen, ist auch eine Alternative anzudenken, in welcher eine Mehrheitsentscheidung gefällt wird, um zu entscheiden, ob ein Element "gelöscht" wird. Diese kann darauf basieren, einen Schwellwert anzulegen, welcher definiert, dass ein Element dann als gelöscht angesehen werden soll und nicht übernommen wird, falls dieses in mehr als 50% der Kindprozesse ebenfalls nicht mehr vorhanden ist.

Für den hier entwickelten Ansatz wurde der in Abschnitt 6.3 beschriebene Weg gewählt, welcher besagt, dass solche einseitig gelöschten Elemente nicht in das Endergebnis mit einfließen. Man muss sich hierbei des Problems eines möglichen Informationsverlustes bewusst sein.

6.4.3 Analyse der Kindprozesse

Während der allgemeinen Analysephase sind gleich mehrere, teilweise voneinander abhängige Fragestellungen zu klären. Zuerst gilt es zu herauszufinden, wie denn die verschiedenen Prozesselemente der unterschiedlichen Prozesse zueinander in Beziehung stehen. Dazu müssen zuerst folgende Punkte untersucht werden.

- Welche Elemente sind in allen Prozessen ident vorhanden?
- Welche Elemente sind zwar in allen Prozessen vorhanden, weisen aber Unterschiede auf?
- Welche Elemente sind in den Kindprozessen vollständig neu hinzugekommen?

Die so gewonnenen Daten müssen mit den Informationen aus dem Vaterprozess, sofern vorhanden, abgeglichen werden bzw. können zum Teil direkt aus der vorangegangenen Analysephase entnommen werden.

Nachdem diese und die vorhergehenden Problemstellungen gelöst wurden, stehen alle grundlegenden Informationen für die Vereinigung zur Verfügung. Zur Verbesserung der Lösungsqualität (Kriterien zur Messung siehe: Ende der Einleitung des Kapitels 6.4 bzw. die Evaluierung in Abschnitt 8.2) sind allerdings noch weitere Optimierungen anwendbar. Insbesondere ist das automatische Finden und Positionieren von Teilen des neu gebildeten Kontrollflusses notwendig, um zu verhindern, dass unterwünschte bzw. ungeeignete Ausführungsreihenfolgen entstehen.

Dieses Problem zeigt sich am folgenden Beispiel, bei welchem zwei Prozesse vereinigt werden. Der Prozess eins und der Prozess zwei besitzen hierbei die beiden gleichen Elemente A und C. Diese werden aufgrund ihrer identen Eigenschaften, welche entweder anhand der Inhalte wie der Label oder anhand eindeutiger Identifikationsnummern (siehe Abschnitt 6.4.1) festgestellt werden können, anschließend als Ankerpunkte bzw. Referenzen für die Vereinigung verwendet. Hierdurch kommt es zu einer parallelen Ausführung beider Gruppen. Daher es entsteht die in Abbildung 28 gezeigte 4. Ausführungsmöglichkeit.

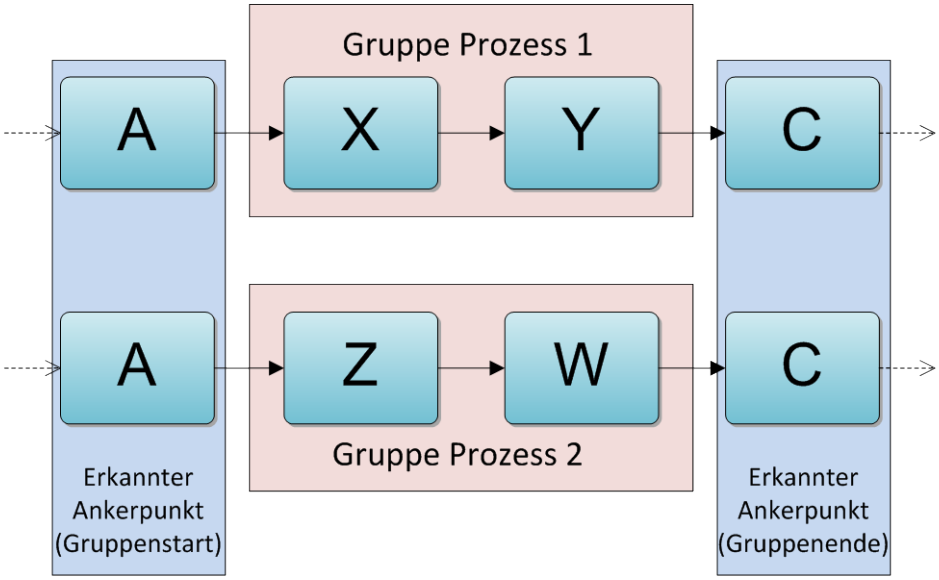

Abbildung 27: Darstellung der Gruppenbildung zur Kontrollflussoptimierung

Quelle: Eigene Erstellung

Dies kann allerdings zu Nebenläufigkeitsproblemen wie inkorrekten Zuständen der zu verarbeitenden Daten (Vgl. Russell, Hofstede, Edmond, & Aalst, 2005, S. 22) führen oder aus organisatorischen Gründen nicht erwünscht oder möglich sein. Zur Verhinderung dieses Problems können jetzt zusätzliche Informationen wie der Datenfluss, welcher zur Erkennung von Abhängigkeiten genützt werden kann, herangezogen werden. Diese stehen hier nicht zur Verfügung, da sich diese Arbeit vor allem auf den Kontrollfluss von Prozessen bezieht.

Als Alternative dazu wird eine Methode verwendet, welche versucht, basierend auf den Verbindungen der Prozesselemente Gruppen zu extrahieren. Eine Gruppe wird hierbei aus neu hinzugekommenen Kindprozesselementen, jeweils für jeden Kindprozess getrennt, anhand deren Beziehungen zueinander gebildet. Die Gruppen werden hierbei maximiert, es wird also versucht, Gruppen mit maximaler Größe, daher maximaler Anzahl an noch verbundenen Elementen, zu finden.

Die so gefundenen Gruppen werden anschließend daraufhin untersucht, ob sich diese überlappen und daher hinsichtlich des zuvor beschriebenen Problems, dass eine Vereinigung der untersuchten Prozesse zu einer parallelen möglicherweise nicht gewünschten Abarbeitung führen würde, anfällig sind. Eine Überlappung wird hierbei durch die Untersuchung der Ankerpunkte bestimmt; hierzu wird insbesondere der startende Ankerpunkt verwendet. Wird eine Überlappung festgestellt, kann hierauf mit verschiedenen Möglichkeiten reagiert werden.

1. *Übernahme keines der Elemente*: Hierbei werden keine Elemente der Kindprozesse übernommen, sodass das endgültige Ergebnis der Vereinigung die problematischen Elemente der überlappenden Gruppen nicht beinhaltet.
2. *Übernahme der Elemente nur eines Kindprozesses*: Bei diesem Ansatz werden nur Teile der Informationen übernommen. Hierbei ist zu entscheiden, welcher der zu vereinigenden Kindprozesse bevorzugt wird.
3. *Serielle Anordnung*: Bei der seriellen Anordnung werden alle Gruppenelemente aller Kindprozesse übernommen. Hier ist zu entscheiden, in welcher seriellen Reihenfolge die Elemente positioniert und damit später ausgeführt werden sollen.
4. *Parallele Anordnung der Gruppen*: Abschließend ist auch eine parallele Anordnung der Elemente möglich, was den zuvor beschriebenen Standardfall darstellt, der auch ohne Gruppensuche erreicht werden kann.

Jede Art für sich kann durch das gezielte Übernehmen bzw. Ignorieren sowie durch das dynamische Generieren/Entfernen der benötigten Kanten zur Erzeugung der Verbindungen erreicht werden. Die Wahl der oben beschriebenen Methode wird hierbei vom Anwender getroffen. Zur Unterstützung ist eine entsprechende Aufbereitung und Visualisierung notwendig. (Vgl. Böhmer, 2011, S. 6)

Die folgende Grafik zeigt eine Übersicht über die zuvor beschriebenen Möglichkeiten zur Anordnung der Gruppen, wobei hier die Nummerierung den Nummern der oben beschriebenen Liste folgt. Die verwendeten Prozesselemente basieren hierbei auf der vorangehenden Abbildung 27.

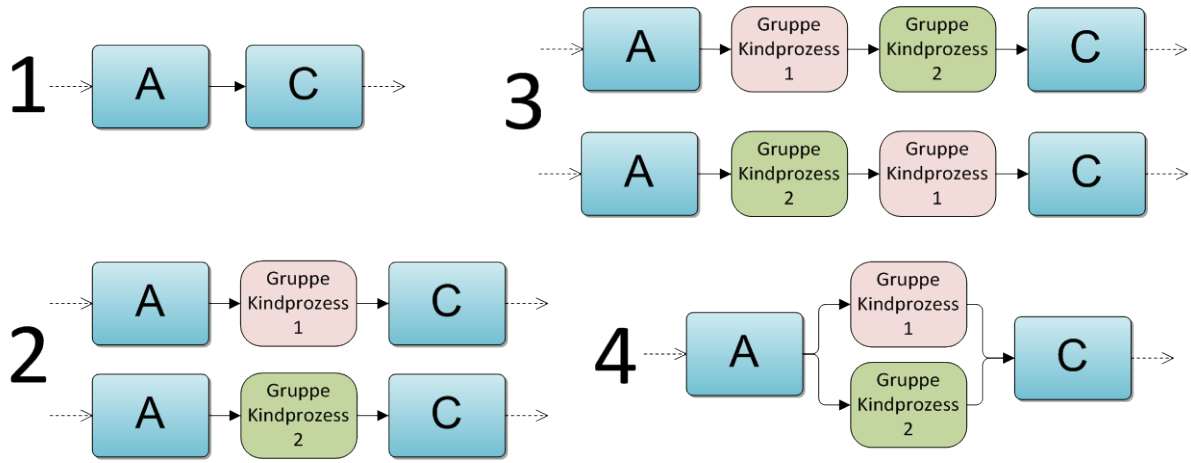

Abbildung 28: Möglichkeiten zur Positionierung von Gruppen im Kontrollfluss

Quelle: Eigene Erstellung

Ein beispielhafter Algorithmus einer Herangehensweise für das zuvor beschriebene Problem ist im folgenden Pseudocode ersichtlich. Dieser besteht aus den Methoden HandleGroups, FindGroups, CompleteGroup sowie GroupsHaveTheSameOrigin, welche gefundene Elemente hinsichtlich der Überlappungen untersucht und, falls notwendig, die Präferenzen des Anwenders erfragt. Einige der Hilfsmethoden werden hier aus Platzgründen nicht vollständig abgedruckt. Deren Aufgabe ist wie folgt:

- *GetStartingArc*: Ermittelt die Kante, welche den Beginn einer Gruppe einleitet.
- *GetElementsConnectedByArc*: Ermittelt die beiden Elemente, welche durch eine Kante verbunden sind.
- *GetArcsConnectedWithElement*: Ermittelt die Kanten, welche mit einem Element verbunden sind.
- *AskUserAboutGroupOrder*: Befragung des Anwenders, wie die gefundenen Gruppen positioniert werden sollen.

Algorithmus: HandleGroups(newElementeChildA, newElementsChildB, finalResult)

Eingabe:
newElementeChildA und newElementsChildB speichern Elemente, welche neu in einem Kindprozess sind, finalResult Menge der Prozesselemente, welche als Endresultat verwendet werden

Variablen:
idetifiedGroupsA und identifiedGroupsB durch die Analyse identifizierte Gruppen, assembledGroupResult Menge der Gruppenelemente unter Bezugnahme der Anwenderentscheidung

```
1   idetifiedGroupsA = FindGroups(newElementeChildA)
2   identifiedGroupsB = FindGroups(newElementsChildB)
3
4   für i=0 ... idetifiedGroupsA
5   {
6         groupa = idetifiedGroupsA[i];
7         für j=0 ... identifiedGroupsB
8         {
9               groupb = idetifiedGroupsA[i];
10              falls GroupsHaveTheSameOrigin(groupa, groupb) dann
11              {
12                    newElementeChildA = newElementeChildA \ groupa;
13                    newElementeChildB = newElementeChildB \ groupb;
14
15                    assembledGroupResult = AskUserAboutGroupOrder(groupa,
16  groupb);
17                    finalResult = finalResult ∪ assembledGroupResult;
18              }
19        }
20  }
```

Programmausdruck 9: Finden von Gruppen und Befragung des Anwenders zur Positionierung

Quelle: Eigene Erstellung

Folgender Algorithmus dient dem Identifizieren der zuvor beschriebenen potentiell problematischen Gruppen.

Algorithmus: FindGroups(elementsToAnalyse)

Eingabe:
elementsToAnalyse Menge der Elemente, welche darauf untersucht werden sollen, ob diese eine Gruppe bilden

Ausgabe: Ermittelte Gruppen

Variablen:
allGroups Menge aller gefundenen Gruppen, groupElement vermutetes Element einer Gruppe

```
1   allGroups = {};
2
3   für i=0 ... elementsToAnalyse
4   {
5         groupElement = elementsToAnalyse[i];
6         group = {};
7
8         CompleteGroup(groupElement, group, elementsToAnalyse)
9         elementsToAnalyse = elementsToAnalyse \ group;
10        i = 0;
11
12        allGroups = allGroups ∪ group;
13  }
```

```
14
15  retuniert allGroups;
```

Programmausdruck 10: Initiiert das Finden von Gruppen

Quelle: Eigene Erstellung

Basierend auf einem Ausgangselement wird versucht, eine möglichst umfassende Gruppe (also eine Gruppe mit möglichst vielen Elementen) zu bilden. Beschränkt werden die Gruppen durch die Referenzpunkte am Start und Ende einer Gruppe, da Elemente, welche wie in Abbildung 27 gezeigt, in den gruppenbildenden Prozessen gleichermaßen vorhanden sind.

Algorithmus: CompleteGroup(startGroupElement, group, elementsToAnalyse)

Eingabe: startGroupElement Element, das als Ausgangspunkt für eine Gruppensuche genommen wird, group Gruppe, die vervollständigt werden soll, elementsToAnalyse Elemente, welche analysiert werden sollen

Ausgabe:
Rückgabe einer vollständig gebildeten Gruppe

Variablen:
connectedArcs Menge von Kanten, welche mit dem initialen Gruppenelement verbunden sind, connectedElements Menge von Elementen, welche mit dem initialen Gruppenelement verbunden sind, newFoundElements Menge von Elementen, welche zuvor bei dieser Gruppensuche noch nicht gefunden wurden

```
 1  group = group ∪ startGroupElement;
 2
 3  connectedArcs = GetArcsConnectedWithElement(startGroupElement, elementsToAna-
 4  lyse, ConnectionType.Both)
 5
 6  connectedElements = {};
 7  für i=0 ... connectedArcs
 8  {
 9        arc = connectedArcs[i];
10        connectedElements = connectedElements ∪ GetElementsConnectedByArc(arc,
11  elementsToAnalyse);
12  }
13
14  newFoundElements = connectedElements \ group;
15
16  für i=0 ... newFoundElements
17  {
18        newStartElement = newFoundElements[i];
19        CompleteGroup(newStartElement, group, elementsToAnalyse);
20  }
21
22  group = group ∪ connectedArcs;
23
24  retuniert group;
```

Programmausdruck 11: Maximierung von potentiellen Gruppen

Quelle: Eigene Erstellung

Hiermit wird bei den zuvor ermittelten Gruppen untersucht, ob es bei diesen zu einer parallelen Ausführung kommen würde und ob daher eine Nachfrage beim Anwender bezüglich der Anordnung notwendig ist.

Algorithmus: GroupsHaveTheSameOrigin(group1, group2)

Eingabe:
group1 und group2 Gruppen, welche hinsichtlich ihrer Überlappung untersucht werden sollen

Ausgabe:
Wert, welcher angibt, ob sich die Gruppen überlappen oder nicht.

Variablen:
haveSameOrigin Rückgabewert für Überlappungsstatus, startingArcGroup1 und startingArcGroup2 Kanten, welche den Beginn einer Gruppe darstellen

```
1   haveSameOrigin = false;
2
3   startingArcGroup1 = GetStartingArc(group1);
4   startingArcGroup2 = GetStartingArc(group2);
5
6   falls startingArcGroup1 != NULL V startingArcGroup2 != NULL V startingArcGroup1
7   == startingArcGroup2 dann
8   {
9         haveSameOrigin = true;
10  }
11
12  retuniert haveSameOrigin;
```

Programmausdruck 12: Überprüfung der Ankerpunkte zweier Gruppen auf Überlappung

Quelle: Eigene Erstellung

6.4.4 Vereinen der Prozesse

Das Vereinen der Elemente lässt sich mit den in Abschnitt 3.3.2 besprochenen Mengenoperationen leicht realisieren. Hierbei werden neue Elemente der Kindprozesse unter Beachtung der zuvor beschriebenen Gruppenbildung übernommen. Bestehende Kantenbeziehungen sowie bei den verschiedenen Elementen gespeicherte Daten bleiben hierbei erhalten. Unproblematisch gestalten sich Elemente, welchen in allen Prozessen in der gleichen Form unterschiedslos vorhanden sind. Diese können ohne weitere Bearbeitung in den Ergebnisprozess eingefügt werden.

Das hierzu notwendige Vorgehen wird in der Folge vereinfacht dargestellt (die Nummerierung orientiert sich hierbei an Abbildung 26). Der erste Schritt (Einlesen und Aufbereiten der Daten) wird hierbei implizit als bereits absolviert angenommen (mit dem Ergebnis Vaterprozess: $P0 \rightarrow [A, X, Y, C\}$, Kindprozess 1: $P1 \rightarrow \{A, Y, C\}$ sowie Kindprozess 2: $P2 \rightarrow \{A, X, B, C\}$), sodass das Beispiel bei Schritt 2 einsteigt. Als Beispielprozess werden die in Abbildung 29 dargestellten Prozesse verwendet. Um das Beispiel besser darstellen zu können, werden hier nur die Knoten der Prozesse behandelt. Ein vergleichbares Vorgehen ist allerdings auch bei den Kanten möglich, wodurch auch die Verschiebung von Elementen innerhalb des Prozesses bearbeitet wird. Abschließend wird eine Vereinigung der als Endergebnis ermittelten Knoten- und Kantenmenge durchgeführt.

2. Analyse des Vaterprozesses: P1: X gelöscht, P2: Y gelöscht
    - Erkannt mit: $P0 \backslash P1 \rightarrow \{X\}$ und $P0 \backslash P2 \rightarrow \{Y\}$. Ergibt Menge gelöschter Elemente mit $R = \{X, Y\}$
3. Analyse der Kindprozesse: P2: B neu eingefügt
    - Erkannt mit: $P1 \backslash P0 \rightarrow \{\}$ und $P2 \backslash P0 \rightarrow \{B\}$. Ergibt Menge neuer Element mit $N = \{B\}$
4. Vereinigung der Prozesse: Ergebnisprozess mit Knotenmenge: $\{A, B, C\}$
    - Erkannt mit: Möglichkeit eins basierend auf Vaterprozess $(P0 \backslash R) \cup N \rightarrow \{A, B, C\}$; Möglichkeit zwei basierend auf Kindprozessen $(P1 \cup P2) \backslash R \rightarrow \{A, B, C\}$
    - Während der Vereinigung ist auch eine Erkennung von Aktualisierungen bestehender Elemente möglich. Daher Elemente die anhand ihrer IDs als in allen Prozessen vorhanden erkannt wurden werden hierbei auf Unterschiede in ihren Inhalten untersucht. Wird hierbei herausgefunden, dass Elemente der Kindprozesse sich von ihren Originalen, Element mit gleicher ID im Vaterprozess, unterscheiden wird die Änderung der Kindprozesse übernommen. Bei sich unterscheidenden Änderungen von in den Kindprozessen enthaltenen Elementen mit gleicher ID wird der Benutzer gebeten eine Entscheidung zu fällen welche Version übernommen werden soll. Alternativ erfolgt eine automatische Übernahme anhand von den Prozessen zugeordneten Prioritäten.
    - Eine Vereinigung ist auch ohne Miteinbeziehung des Vaterprozesses möglich (Möglichkeit zwei), allerdings wäre ohne Vaterprozess hier die Menge R leer und gelöschte Elemente würden nicht erkannt.
5. Zusätzlich sind noch weitere Operationen durchzuführen, welche die Qualität der Endergebnisse erhöhen. Diese lassen sich allerdings aufgrund der hohen Komplexität der Berechnungen nicht mit reinen Mengenoperationen abbilden. (Vergleiche hierzu Abschnitt 6.4.3 – Erkennung und Behebung von möglichen Konflikten der Ausführungsreihenfolge und Abschnitt 6.4.5 – Wiederherstellung unterbrochener Kontrollflussabschnitte).

Bei Elementen, welche in unterschiedlicher Art und Weise in den Kindprozessen modifiziert wurden (da ein Element, welches im Vater- sowie in den Kindprozessen vorhanden ist, jedoch in jedem der Kindprozesse nun unterschiedliche Inhalte besitzt), tritt einer der zuvor beschriebenen Konflikte auf. Eine Lösung ist hier mit den zur Verfügung stehenden Informationen nicht möglich, sodass eine entsprechende Anfrage an einen Anwender gestellt werden muss. Dem gegenüber stehen Elemente, welche im Vaterprozess existieren und nur in einem Kind oder in mehreren Kindern in der gleichen Art und Weise modifiziert wurden. Hier wird angenommen, dass die modifizierten Elemente von der Bedeutung höher sind als die unveränderten, da hier ein zusätzlicher Aufwand in die Modifikation investiert wurde. Das modifizierte Element "gewinnt" deshalb und wird übernommen.

Vergleicht man die Essenz dieses Ansatzes (in Verbindung mit der in dieser Arbeit ausgearbeiteten Vorgehensweise) mit dem zuvor vorgestellten klassischen Ansatz, zeigen sich die unterschiedlichen Herangehensweisen bzw. Ziele. So versucht der klassische Ansatz (Vgl. Abbildung 23), eine Vereinigung aller Ausführungspfade zum Ziel hat, während der hier neu vorgestellte Ansatz eine Vereinigung aller Änderungsoperationen erreichen möchte. Vorteilhaft ist hierbei, dass zur Erreichung dieses Ziels nicht langwierig die verschiedenen Änderungen bzw. Änderungsoperationen gesucht und vereint werden müssen. (Vgl. Lippe & Oosterom, 1992, S. 80ff) (Vgl. Ignat & Norrie, 2004, S. 2ff). Stattdessen kann direkt auf dem Letztstand der Prozessmodelle gearbeitet werden und die Erkennung und Vereinigung der Unterschiede bzw. Änderungen erfolgt implizit.

6.4.5 Plausibilitätsprüfung und Aufbereitung der Ergebnisse

Abschließend sind noch verschiedene Schritte notwendig, welche vor allem darauf abzielen, die Korrektheit der erzeugten Ergebnisprozesse zu verbessern. Hierzu wird zuerst eine Bereinigung

durchgeführt, indem Elemente, welche sichtlich inkorrekt sind, wie Kanten, denen das Start- und/oder End-Element fehlt, aus der Menge der Prozesselemente gelöscht werden. Vergleichbar damit können "unnötige" Elemente entfernt werden. Als unnötig können hierbei Gateways betrachtet werden, welche den Kontrollfluss verzweigen sollen, jedoch nur eine einzelne ausgehende bzw. eingehende Kante besitzen.

Problematisch ist hierbei allerdings, dass hierdurch möglicherweise absichtlich inkorrekt modellierte Elemente ebenfalls entfernt würden, beispielsweise, weil sich ein Prozessmodell gerade erst im Aufbau befindet. Deshalb wird auch eine Möglichkeit geboten, die Bereinigung temporär zu deaktivieren.
Zusätzlich wird ein komplexerer Algorithmus verwendet, um Verbindungen im Kontrollfluss wieder herzustellen, welche durch die vorangegangenen Operationen durchtrennt wurden. Geschehen kann dies, wenn ein Element existiert, welches sowohl im Vater- als auch in den meisten Kindprozessen vorhanden ist, in einem der Kindprozesse jedoch gelöscht wurde. Zusätzlich müssen dann auch noch neue Elemente hinzukommen, welche wiederum nur in einem Teil der Kindprozesse auftreten. Hierdurch kommt es zu der in Abbildung 29 gezeigten Kombination von Faktoren, welche zu einer Lücke im Kontrollfluss führen.

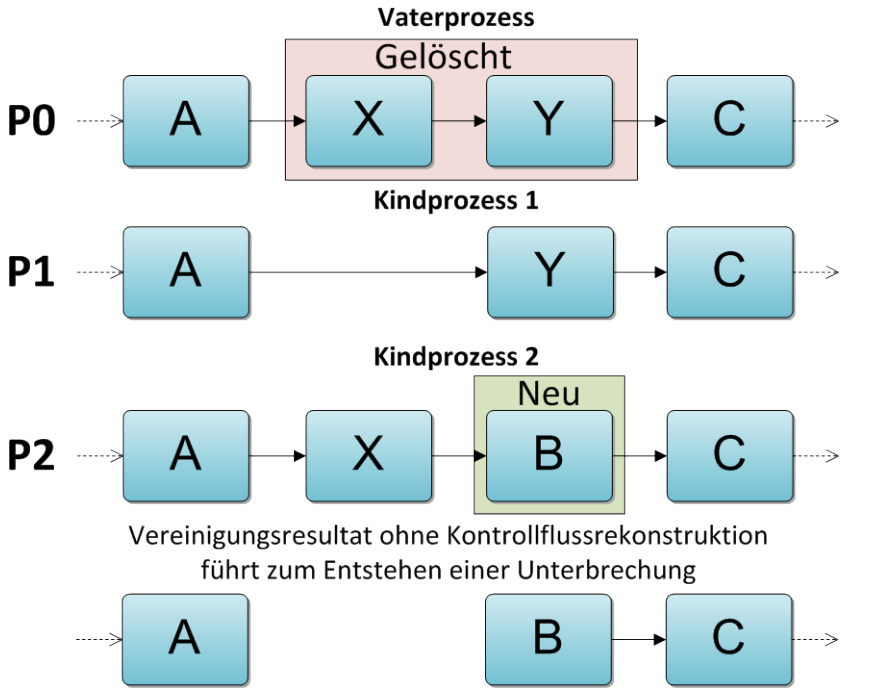

Abbildung 29: Kontrollflussunterbrechung durch Löschoperationen nach Vereinigungen

Quelle: Eigene Erstellung

Das obige Problem lässt sich lösen, indem neue Kanten generiert werden, um die fehlenden Verbindungen zu ersetzen. Hierzu werden Elemente gesucht, denen eingehende bzw. ausgehende Verbindungen fehlen. Start- bzw. Endelementen kommt hierbei eine Sonderrolle zu, da Startelemente keine eingehenden Verbindungen benötigen und Endelemente analog hierzu keine ausgehenden Verbindungen besitzen können. Basierend auf den hierdurch identifizierten Elementen werden anschließend Elemente gesucht, zu denen eine Verbindung am sinnvollsten aufgebaut werden kann. Hierzu wird, ausgehend von den betroffenen Elementen, im zugehörigen Vater- bzw. Kindprozess ermittelt, welche direkten und indirekten Vorgänger/Nachfolger dort existieren. Ist solch ein Element im Ergebnisprozess aufzufinden, so wird zu diesem eine entsprechende Verbindung aufgebaut und hierdurch der Kontrollfluss wiederhergestellt.

Folgende Grafik zeigt zum Vergleich das Ergebnis der klassischen Vereinigung (Beibehaltung aller Ausführungspfade) der in Abbildung 29 dargestellten Prozesse.

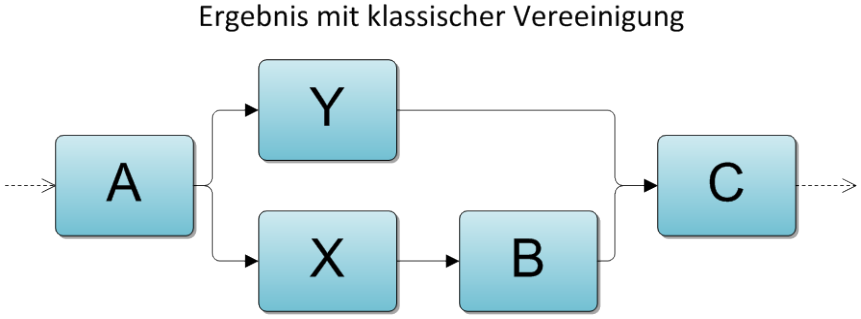

Abbildung 30: Beispiel zum Vergleich klassische Vereinigung mit der vom Autor vorgeschlagenen Vereinigung

Quelle: Eigene Erstellung

Aufgrund der Komplexität dieses Ansatzes wird auch hier der entsprechende Pseudocode für den Algorithmus angegeben. Dieser besteht aus den Methoden RecreateControlFlow, für welchen die Verbindungen wiederhergestellt wird, sowie aus den Methoden SearchElementsToConnectTo für die Suche nach Elementen, welche sinnvoll miteinander verbunden werden können. Zusätzlich werden noch folgende, nicht näher definierte Methoden verwendet.

- *SearchElementsToConnectToWithChilds*: Funktioniert analog zu SearchElementsToConnectTo. Hierbei wird allerdings nicht versucht, die Verbindungen anhand des Vaterprozesses wiederherzustellen, sondern es werden hierbei die Kindprozesse anstelle der Vaterprozesse herangezogen.
- *SearchElementsWithoutSpecificTypeArc*: Ermittelt die Elemente, welche den Beginn/das Ende einer „Lücke“ bilden und daher zur Wiederherstellung des Kontrollflusses neu eingebunden werden müssen.
- *GetArcs* : Ermittelt aus einer Menge von Prozesselementen alle Kanten und liefert diese zurück.

Algorithmus: RecreateControlFlow(partenProcess, mergedProcess, childProzessA, childProzessB)

Eingabe:
parentProcess Vaterprozess der zu vereinigenden Kindprozesse, mergedProzess ist der Prozess, welcher als Endergebnis der Vereinigung entstehen soll, childProzessA und childProzessB stellen die beiden in einem Durchlauf vereinigten Kindprozesse dar

Variablen:
elementsWithoutIncomingArc Menge von Elementen, welche keine ausreichende Einbindung in den Kontrollfluss besitzen, predecessors Elemente, welche zur Wiederherstellung eines korrekten Kontrollflusses als Ausgangspunkt für eine neue Kante verwendet werden, arc neue Kante, um durchtrennte Verbindungen neu zu bilden

```
1   elementsWithoutIncomingArc = SearchElementsWithoutSpecific-
2   TypeArc(ConnectionType.Incoming, mergedProcess);
3
4   für i=0 ... elementsWithoutIncomingArc
5   {
6         elementToCheck = elementsWithoutIncomingArc[i];
```

```
7         predecessors = SearchElementsToConnectTo(ConnectionType.Incoming, parten-
8   Process, mergedProcess, elementToCheck);
9
10        falls predecessors == {} dann
11        {
12              // falls keine passende Verbindung im Vaterprozess gefunden wurde,
13  wird hiermit in den Kindprozessen gesucht
14              predecessors = SearchElementsToConnectToWith-
15  Childs(ConnectionType.Incoming, childProzessA, childProzessB, mergedProcess,
16  elementToCheck);
17        }
18
19        für j=0 ... predecessors
20        {
21              predecessor = predecessors[j];
22              Arc arc = new Arc(predecessor, elementToCheck);
23              mergedProcess = mergedProcess ∪ arc;
24        }
25  }
26
27  /* Ein zum obigen Code vergleichbarer Ablauf findet mit fehlenden ausgehenden
28  Kantenverbindungen statt. Hierbei werden dann mit nachfolgenden Elementen neue
29  Kantenverbindungen geknüpft. Aus Platzgründen wurde auf eine vollständige Ab-
30  bildung verzichtet. */
```

Programmausdruck 13: Wiederherstellung durchbrochener Kontrollflüsse

Quelle: Eigene Erstellung

Der folgende Algorithmus sucht, basierend auf einem rekursiven Ansatz, Elemente, welche dazu verwendet werden können, durchtrennte Kontrollflüsse wiederherzustellen. Die Rekursion wird hierbei dazu verwendet, den Kontrollfluss von Element zu Element stufenweise zurückzuverfolgen, bis das erste passende Element gefunden wurde.

Algorithmus: SearchElementsToConnectTo(connectionType, process-ToSearchIn, mergedProcess, elementToCheck)

Eingabe:
connectionType Art der Kantenverbindung, welche bei der Suche betrachtet werden soll (eingehend oder ausgehend), processToSearchIn Prozess, in dessen Elementen nach passenden Vor- bzw. Nachfolgerelementen für die Lückenschließung gesucht wird, mergedProcess passende Elemente für eine Lückenschließung müssen im vereinigten Ergebnisprozess enthalten sein, deshalb wird dieser übergeben, elementToCheck Element, welches wieder eingebunden werden soll und dessen Nachfolger bzw. Vorgänger bestimmt werden.

Ausgabe:
Returniert eine Menge von Prozesselementen, welche für die Wiederherstellung des Kontrollflusses verwendet werden können.

Variablen:
existingPossibleConnectionPartner ist eine Menge von Elementen, zu denen sich zur Wiederherstellung eines Kontrollflusses eine Verbindung aufbauen lässt, processToSearchInArcs ist eine Menge von Kanten, welche hinsichtlich der von ihnen verbundenen Elementen untersucht werden (Erkennung von Vorgängern/Nachfolgern, welche nicht mehr korrekt in den Kon-

trollfluss eingebunden sind), connectedArcsSearchProcess Ergebnis der zuvor beschriebenen Suche, connectedElements Menge von Elementen, welche mit den identifizierten Kanten verbunden sind. Bei diesen wird anschließend getestet, ob sich diese im vereinigten Ergebnisprozess wiederfinden.

```
existingPossibleConnectionPartner = {}

// Suche Kanten, welche im zu untersuchenden Prozess mit dem wieder einzubin-
denden Element verbunden sind
processToSearchInArcs = GetArcs(processToSearchIn);

connectedArcsSearchProcess = {}
für i=0 ... processToSearchInArcs
{
	arc = processToSearchIn[i];

	falls connectionType == ConnectionType.Incoming dann
	{
		falls IsIncoming(arc, elementToCheck) dann
		{
			connectedArcsSearchProcess = connectedArcsSearchProcess ∪
arc;
		}
	}
	sonst
	{
		falls IsOutgoing(arc, elementToCheck) dann
		{
			connectedArcsSearchProcess = connectedArcsSearchProcess ∪
arc;
		}
	}
}

// Suchen von Elementen, welche über die zuvor gefundenen Kanten mit dem wieder
einzubindenden Element verbunden sind.
connectedElements = {}

für i=0 ... processToSearchIn
{
	connectedElement = processToSearchIn[i];
	für j=0 ... connectedArcsSearchProcess
	{
		arc = connectedArcsSearchProcess[j];
		falls connectionType == ConnectionType.Incoming dann
		{
			falls IsOutgoing(arc, connectedElement) dann
			{
				connectedElements = connectedElements ∪ connectedEle-
ment;
			}
		}
		sonst
		{
			falls IsIncoming(arc, connectedElement) dann
			{
				connectedElements = connectedElements ∪ connectedEle-
ment;
```

```
                    }
                }
        }
}

// Prüfe, ob eines der gefundenen Elemente im vereinigten Prozessergebnis wie-
derzufinden ist.
für i=0 ... conntectedElements
{
        connectedElement = connectedElements[i];
        foundPropertConnection = false;

        für j=0 ... mergedProcess
        {
                mergedElement = mergedProcess[j];
                falls Equals(connectedElement, mergedElement) dann
                {
                        foundPropertConnection = true;
                }
        }

        falls foundPropertConnection == true dann
        {
                existingPossibleConnectionPartner = existingPossibleConnection-
Partner ∪ connectedElement;
        }
}

// Durch die rekursive Suche ist auch eine Wiedereinbindung möglich, falls meh-
rere Elemente aus dem Pfad entfernt wurden.
falls existingPossibleConnectionPartner == {} dann
{
        für i=0 ... connectedElements
        {
                connectedElement = conntectedElements[0];
                falls existingPossibleConnectionPartner / connectedElement == {}
dann
                {
                        searchResults = SearchElementsToConnectTo(connectionType,
processToSearchIn, mergedProcess, connectedElement);

                        für j=0 ... searchResults
                        {
                                searchResult = searchResult[j];
                                falls existingPossibleConnectionPartner / connectedEl-
ement == {} dann
                                {
                                        existingPossibleConnectionPartner = existing-
PossibleConnectionPartner ∪ searchResult;
                                }
                        }
                }
        }
}

retuniert existingPossibleConnectionPartner;
```

Programmausdruck 14: Identifizieren von Kontrollflusslücken und Elementen für eine Schließung dieser

Quelle: Eigene Erstellung

6.4.6 Beispiel für die Anwendung der Vereinigungsabläufe

Folgendes Beispiel zeigt die Vereinigung zweier lebensnaher komplexer Prozesse (mehrere Aufgabenelemente, Schleifen sowie parallele Ausführung von Abläufen), welche in dieser Form im Bankenwesen eingesetzt werden, um den Status verschiedener Bankgeschäfte zu bewerten. Der Prozess selbst besteht grundlegend aus dem Laden von Daten, dem Berechnen von Werten und Prüfen der Ergebnisse. Dies geht wiederum mit möglicherweise notwendigen Parameteranpassungen und Neuberechnungen einher. Anschließend werden die Daten für das Management aufbereitet und den betreffenden Führungspersonen präsentiert.

Verwendet wurden hierbei die zuvor beschriebenen Abläufe. Die Abbildung 31 zeigt einen Vaterprozess. Darauf folgen in Abbildung 32 und Abbildung 33 zwei davon abgeleitete Kindprozesse namens Szenario A und Szenario B. Die Unterschiede zwischen den Vater- und den Kindprozessen wurden nach folgendem Farbschema markiert: Gelöschte Elemente wurden im Vaterprozess mit einem roten R, für engl. **r**emoved, (in Verbindung mit einem A und/oder B markiert, um anzuzeigen, in welchen Kindprozessen es gelöscht wurde, A/B steht hierbei dafür, dass es in beiden Kindprozessen gelöscht wurde). Bei neu hinzugekommenen Elementen folgte in den Kindprozessen ein grünes N, für engl. **n**ew, und bei modifizierten Elementen (Änderung des Inhalts und/oder Position) eine Markierung mit einem orangen M, für engl. **m**odified. Diese beiden Markierungen erfolgten jeweils bei den betreffenden Kindprozessen.

Abbildung 34 zeigt das Ergebnis der Vereinigung der zuvor beschriebenen Prozesse. Dieses eignet sich durchaus für die sofortige Übernahme in die Produktion. Die A bzw. B, welche sich bei den verschiedenen Elementen befinden, bezeichnen den Kindprozess, von welchem ein Element abstammt bzw. den Prozess, von welchem es übernommen wurde. Wurde ein Element unverändert vom Vaterprozess übernommen, ist es mit einem X gekennzeichnet.

Bei der folgenden Abbildung handelt es sich um den zuvor beschriebenen beispielhaften Vaterprozess (Szenario X). Elemente, welche in einem der Kinder gelöscht wurden, sind rot markiert.

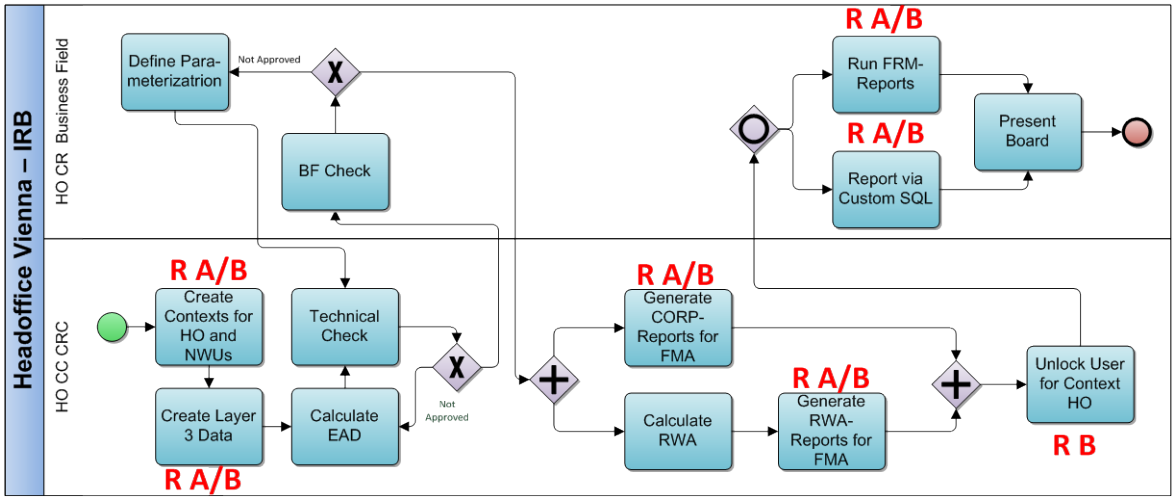

Abbildung 31: Vaterprozess (Szenario X) für die beispielhafte Vereinigung

Quelle: Eigene Erstellung

Die nun folgenden Prozesse zeigen die beiden vom Vaterprozess abgeleiteten Kindprozesse. Als Erstes ist der Kindprozess A (Szenario A) angeführt. Darauf folgt der Kindprozess B (Szenario B). Neue bzw. modifizierte Elemente (aus Sicht des Vaterprozesses) wurden entsprechend der zuvor angegebenen Konventionen mit einem grünen N bzw. einem orangen M markiert.

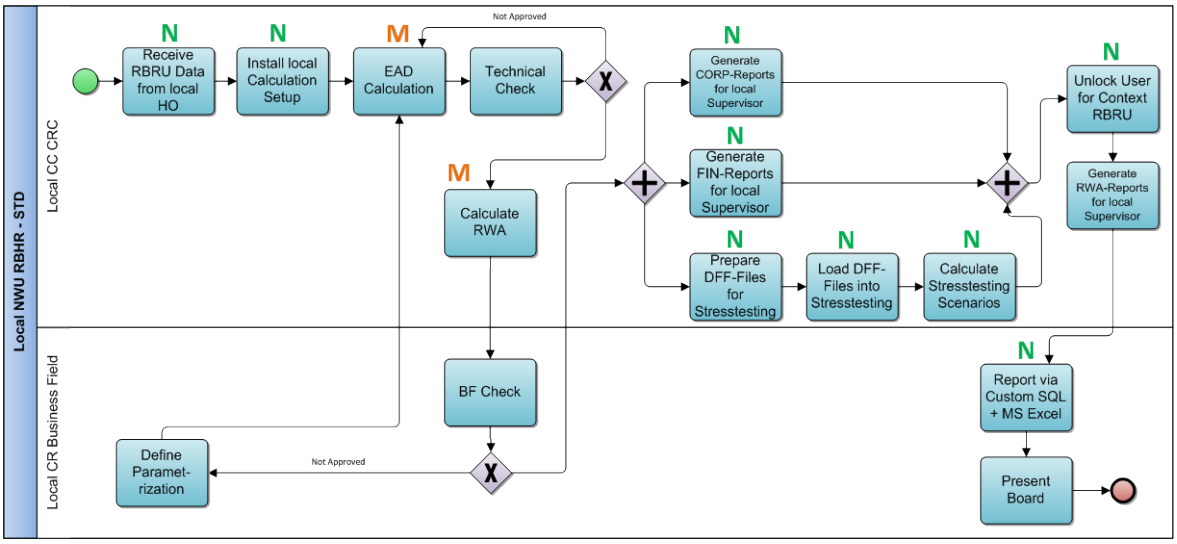

Abbildung 32: Kindprozess A, entwickelt basierend auf Vaterprozess X

Quelle: Eigene Erstellung

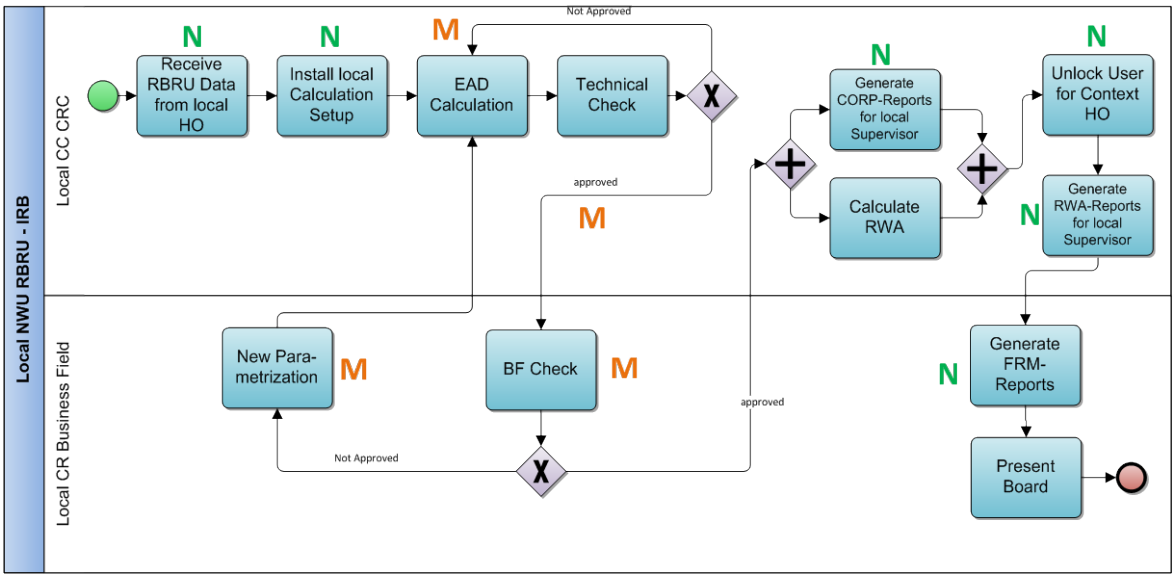

Abbildung 33: Kindprozess B, entwickelt basierend auf Vaterprozess X

Quelle: Eigene Erstellung

Hier ist nun das finale Ergebnis der Vereinigung nach dem Durchlaufen der fünf zuvor beschriebenen Vereinigungsschritte. Jedes übernommene Element erhielt einen Buchstaben zugeteilt, der angibt, von welchem der untersuchten Prozesse es übernommen wurde. Das Ergebnis der Vereinigung lässt sich trotz der komplexen Ausgangsdaten als valide bezeichnen (hinsichtlich der Korrektheitskriterien, welche in Abschnitt 2.3 beschrieben sind) und wäre in dieser Form für die Übernahme in den Produktivbetrieb geeignet. Hierzu trugen insbesondere die zuvor beschriebenen Korrektur- und Analysemethoden für das Finden problematischer Gruppen (wobei zwei Gegebenheiten gefunden wurden, bei denen nach der gewünschten Anordnung gefragt wurde, vergleiche Abschnitt 6.4.3) und die automatische Behebung von Kontrollflussunterbrechungen (es wurde eine vollständig neue Kante, welche in keinem der untersuchten Prozesse vorhanden war, neu generiert, vergleiche Abschnitt 6.4.5) bei.

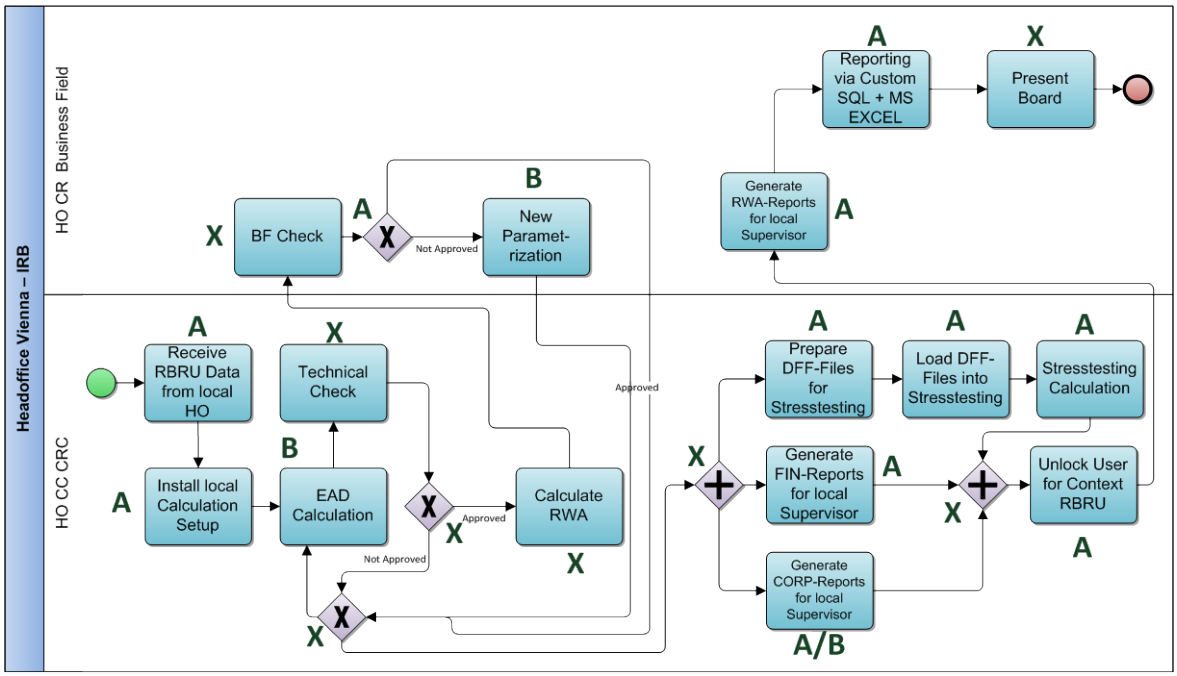

Abbildung 34: Finales Ergebnis der Vereinigung der Prozesse X, A und B

Quelle: Eigene Erstellung

6.5 Diskussion und weiterführende Literatur

Verglichen mit der im vorherigen Kapitel behandelten Ähnlichkeitsmessung wurde das Vereinigen von Prozessmodellen in der Wissenschaft bisher weniger ausführlich bearbeitet. So ist nicht nur eine geringere Anzahl von Arbeiten zu diesem Thema vorhanden, die dort vorgestellten Lösungsansätze sind zumeist auch einfacher aufgebaut und orientieren sich weniger stark an in der Praxis bestehenden Bedürfnissen. In „Merging business process models“ von Rosa et al. (Rosa, Dumas, Uba, & Dijkman, 2010) besteht die Vereinigung beispielsweise darin, alle zu vereinigenden Modelle so in eines zusammenzufassen, dass alle zuvor möglichen Ausführungspfade erhalten bleiben. Hierdurch ergeben sich allerdings sehr komplexe Modelle, welche vor jeder Ausführung konfiguriert und bezüglich des gewünschten Kontrollflusses eingestellt werden müssen. Ein ähnlicher Ansatz findet sich auch in der Arbeit „Merging Event-driven Process Chains“. (Gottschalk, Aalst, & Jansen-Vullers, Merging Event-driven Process Chains, 2009) Eine Evaluierung dieser klassischen Vereinigungsverfahren im Vergleich mit dem in dieser Arbeit vorgestellten Ansatz findet in Abschnitt 8.2 statt. Hierbei zeigt die vom Autor dieser Arbeit vorgeschlagene Herangehensweise Vorteile in praxisrelevanten Szenarien.

Zusätzlich erwähnenswert ist noch die Arbeit „Difference and Union of Models“ von Alanen et al. (Alanen & Porres, 2003), in welcher, vergleichbar mit dieser Arbeit, ebenfalls Metamodelle zur Prozessdefinition eingesetzt werden. Insbesondere wird hierbei mit einem formalisierten Meta Object Facility (MOF) (Vgl. Open Model Group, 2002) -Ansatz gearbeitet. Weiters liegt der Fokus der Arbeit darauf, Grundlagen für ein Versionsverwaltungssystem für Prozessdaten zu erstellen. In der Arbeit „A manifesto for model merging“ von Brunet et al. (Brunet, Chechik, Easterbrook, Nejati, Niu, & Sabetzadeh, 2006) wird ein Framework für das Vereinigen von Modellen vorgestellt. Hierfür werden verschiedene Operationen für die Arbeit mit Modellen betrachtet und mathematisch definiert (merge, match, diff). Zusätzlich werden Teile ihrer Arbeiten in das Framework eingeordnet, so beispielsweise der Umgang mit Entity-Relationship-Modellen (ER-Modelle).

Das Ergebnis dieses Kapitels lässt sich positiv bewerten. So konnte ein Paket an Algorithmen erstellt werden, welche eine Vereinigung verschiedener Prozessmodelle erlauben. Hierbei wurde der zuvor gewählte grundlegende Ansatz der Mengenorientierung erfolgreich eingesetzt.

Für zukünftige sowie weiterführende Arbeiten ist es lohnenswert, vor allem noch nicht verwendete Methoden wie Single-Entry Single-Exit (SESE)-Regionen für die Erkennung von Unterschieden mit einfließen zu lassen. Auch Verbesserungsmöglichkeiten und Erweiterungen zu den hier vorgestellten Lösungen hinsichtlich der Erkennung, Vermeidung und automatischen Korrektur von Konflikten sind zu untersuchen. Dabei kann gleichzeitig auch die Miteinbeziehung zusätzlicher Informationsquellen wie der Datenfluss unterstützend eingebracht werden.

7 Implementierung der beschriebenen Algorithmen

Im Folgenden wird kurz die prototypische Umsetzung der zuvor vorgestellten Algorithmen für die Ähnlichkeitsmessung zwischen Prozessen und deren Vereinigung vorgestellt. Die beiden Kernaspekte der Arbeit wurden nicht nur theoretisch behandelt, sondern auch implementiert.

Für die Ein- und Ausgabe der Informationen wird eine textbasierte Oberfläche genützt. Das Einlesen der benötigten Prozessdaten (welche in dem in Kapitel 4 definierten intermediate Format vorliegen müssen) und das Speichern der bei der Vereinigung erzeugten neuen Prozessmodelle geschieht automatisiert.

Realisiert wurden beide Lösungen in „managed code“ (.NET) bzw. bytecode (JAVA) Programmiersprachen wie .NET (eingesetzt wurde .NET 4.0 bei der Realisierung der Ähnlichkeitsmessung) oder JAVA (eingesetzt bei der Prozessvereinigung in Version 1.6), bei welchen die Ausführung in einer eigenen Laufzeitumgebung geschieht. Die Wahl dieser Methode wurde aufgrund des Umstandes, dass dieser Ansatz eine einfache Programmierung (Speicherverwaltung wird automatisiert gehandhabt) und plattformunabhängige Realisierung ermöglicht, getroffen. Die Plattformunabhängigkeit war hierbei von besonderer Bedeutung, da abwechselnd auf Linux- und Windows-Rechnern entwickelt wurde.

Die folgende Abbildung zeigt das textuelle Konsolen-Interface der Ähnlichkeitsmessung. Gezeigt wird das Ergebnis der Messungen, in welcher mehrere Prozesse verglichen wurden und eine Reihung, basierend auf der ermittelten Ähnlichkeit, erfolgte. Die ermittelte Ähnlichkeit zum Ausgangsprozess (links) wird jeweils in Prozent, in Klammern nach dem Prozessnamen, angegeben.

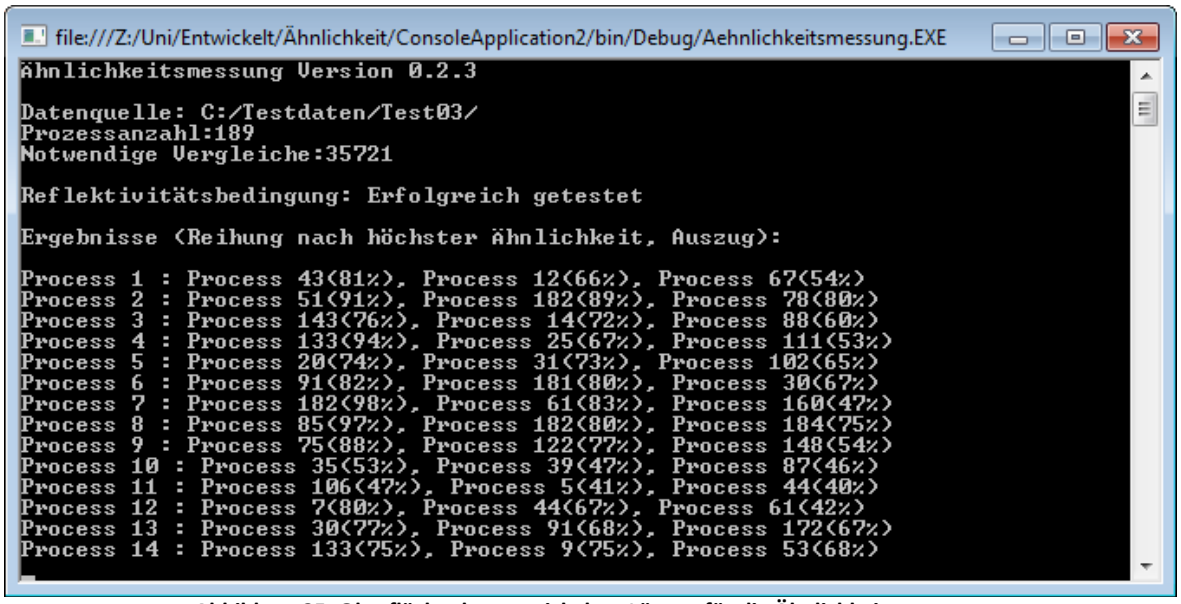

Abbildung 35: Oberfläche der entwickelten Lösung für die Ähnlichkeitsmessung

Quelle: Eigene Erstellung

Einen Einblick in die Benutzeroberfläche der Implementierung der Algorithmen zur Vereinigung von Prozessen ist in folgender Grafik ersichtlich. Gezeigt wird die Darstellung des vereinigten Ergebnisprozesses nach der Durchführung aller hierzu notwendigen Berechnungen.

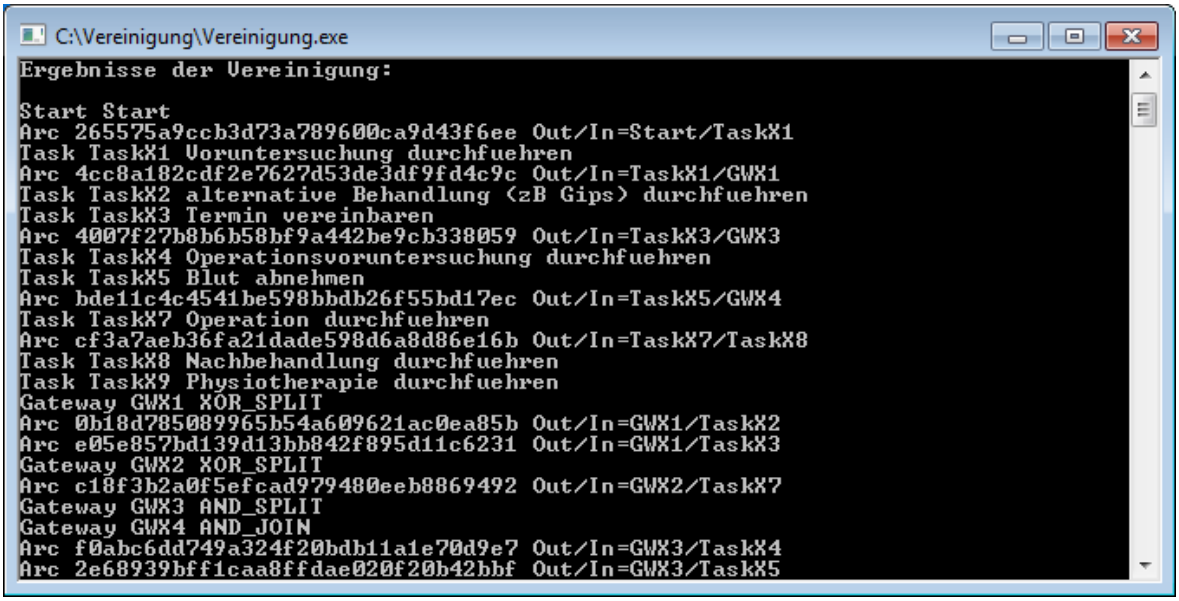

Abbildung 36: Oberfläche der entwickelten Lösung zur Vereinigung von Prozessen

Quelle: Eigene Erstellung

8 Evaluierung der Ergebnisse

Nachdem nun in den vorangegangenen Kapiteln die Lösungswege und Methoden zur Klärung der Fragestellungen (Ähnlichkeitsmessung von Prozessen sowie die Vereinigung von Prozessmodellen) präsentiert wurden, wird nun im Folgenden eine Evaluierung dieser Methoden durchgeführt.

Die Evaluierung der Ähnlichkeitsmessung (siehe Abschnitt 5) erfolgt, indem die in dieser Arbeit vorgeschlagene Verbesserung der Berechnung der strukturellen Unterschiede (Erweiterung der klassischen Substitutionskostenberechnung durch eine umfassende Analyse der Nachbarschaftsbeziehungen) mit einem vergleichbaren Ansatz ohne diese Erweiterung verglichen wird. Hierzu werden die entsprechenden Algorithmen auf eine umfangreiche Prozessdatenbank mit praxisbezogenen Prozessmodellen angewendet, eine Analyse der Ergebnisse und das Herausarbeiten von Unterschieden folgen.

Für die Evaluierung der Vereinigung (siehe Abschnitt 6) wird ein anderes Vorgehen angewandt. Dazu wird die hier entwickelte Lösung anhand verschiedener Kriterien mit den in der Literatur bereits beschriebenen Lösungen aus dem Bereich der Prozessvereinigung verglichen. Das Hauptaugenmerkt fällt hierbei auf für den Praxiseinsatz wichtige Kriterien wie Wartbarkeit, Erweiterbarkeit und den notwendigen Aufwand, um ein vereintes Prozessmodell in den Betrieb übernehmen zu können.

8.1 Evaluierung der Ähnlichkeitsmessung von Prozessmodellen

Wie zuvor beschrieben, wird für die Evaluierung der Ähnlichkeitsmessung ein Vergleich verschiedener Algorithmen durchgeführt. Über die hierbei verwendete Prozessdatenbank dürfen aus Geheimhaltungsgründen nur allgemeine Informationen veröffentlich werden. Insgesamt werden zur

Evaluierung 151 Prozesse analysiert. Die einzelnen Prozesse, allesamt praxisrelevant und im Geschäftsalltag (Bildungssektor) im aktiven Einsatz, haben hierbei zwischen einem und 54 Elementen. In der Prozessdatenbank selbst sind Abläufe von Prozessen mit unterschiedlichster Ähnlichkeit beschrieben, welche nur geringe Unterschiede aufweisen, bis hin zu Prozessen, welche komplett unterschiedliche Themengebiete abdecken. Vergleichbar hiermit sind auch die verschiedensten Komplexitätsgrade der Prozesse. So bestehen einige Prozesse aus nur wenigen sequenziell angeordneten Elementen, andere Prozesse wiederum verwenden komplexe Kombinationen von Schleifen und Verzweigungen.

Vor Beginn der eigentlichen Analyse mussten die Daten entsprechend dem in Kapitel 4 behandelte Ablauf in das in dieser Arbeit beschriebene intermediate Format transformiert werden. Die Informationen selbst waren hierbei anfangs in einem proprietären XML-basierenden Format abgebildet und auf verschiedene Dateien und Ordnerstrukturen verteilt. Der Transformator hatte also die Aufgabe, die Informationen auszulesen, zu kombinieren/validieren und abschließend eine Transformation in das hier entwickelte intermediate Format vorzunehmen. Die so gewonnenen Informationen wurden danach zur Ähnlichkeitsmessung an die Algorithmen weitergereicht. Teilweise waren in den Prozessen Elementtypen enthalten, welche in dem hier beschriebenen Datenformat keine Berücksichtigung erhielten wie Elemente zum Ermitteln von internen Kennzahlen. Um diese Informationen nicht zu verlieren, wurden solche Elemente zur internen Verarbeitung mit Aktivitäten abgebildet.

Für die Ähnlichkeitsmessung selbst wurden die in Kapitel 5 vorgestellten Ansätze bzw. Algorithmen verwendet. Um die Vergleichbarkeit mit den in diesem Kapitel vorgestellten Beispielen zu gewährleisten, wurde auf die gleichen Kosten- und Anteilsparameter zurückgegriffen (welche in Formel 4: Berechnung der Graphenähnlichkeit, verwendet werden). Diese zeigten auch bei den verschiedenen Test sowie experimentellen Untersuchungen verschiedener Parameterkombinationen gute Ergebnisse. Aus Gründen der leichteren Lesbarkeit werden diese hier nun nochmals wiederholt:

Parametername	Parameterwert
Berechnung des Endergebnisses	
Gewichtung entfernte Knoten	0,3
Gewichtung entfernte Kanten	0,2
Gewichtung textuelle Editierdistanz	0,5
Berechnung textuelle Editierdistanz	
Kosten gelöschtes Zeichen	0,5
Kosten neues Zeichen	0,5
Kosten ersetztes Zeichen	1

Tabelle 10: Parameter der Ähnlichkeitsmessungsalgorithmen

Quelle: Eigene Erstellung

Mit den oben definierten Parametern wurden nun alle zur Verfügung stehenden Prozesse analysiert. Da hierbei jeder Prozess mit jedem Prozess verglichen wurde (um Prozesse zu identifizieren, welche einander ähnlich sind), waren insgesamt 22650 Prozessvergleiche notwendig. Alle Tests wurden auf einem Laptop mit Intel Core i5-2520M mit 2,5 Ghz, 8 GB Arbeitsspeicher durchgeführt. Als Betriebssystem wurde Windows 7 SP1 mit .NET 4.5 als Programmierframework verwendet.

Basierend auf den Analysen ergaben sich folgende Werte: Zur Berechnung des Mittelwertes der Ähnlichkeit wurden alle errechneten Ähnlichkeiten summiert und durch die Anzahl der analysierten Prozesse dividiert. Die Rechenzeit stellt einen Mittelwert aus 10 durchgeführten Durchläufen dar.

Methode	Mittelwert der Ähnlichkeit	Rechenzeit
Klassische Methode	59%	20,3 Minuten
Mit Nachbarschaftsbeziehungen	53%	21,7 Minuten

Tabelle 11: Messergebnisse der Evaluierung – Ähnlichkeitsmessung

Quelle: Eigene Erstellung

Dass die durchschnittliche Ähnlichkeit der klassischen Ähnlichkeitsmessungen höher ist als mit der in dieser Arbeit beschriebenen Erweiterung der Einbeziehung der Nachbarschaften verwundert nicht. So kann diese Erweiterung aufgrund des Konzeptes die gemessene Ähnlichkeit nur verringern und nicht erhöhen. Interessanter als diese Durchschnittswerte ist daher der Vergleich der pro Prozess ermittelten Ergebnisse. Hierbei besteht die Hypothese, dass sich ähnliche Prozesse stärker von der „Masse“ daher dem Grundrauschen (dem Mittelwert der Ähnlichkeit) der einander nicht besonders ähnlichen Prozesse abheben.

Zur Überprüfung dieser Annahme wurden 25 Prozesse zufällig aus der für die Tests verwendeten Prozessdatenbank ausgewählt. Die Ähnlichkeit wurde mit beiden Methoden ermittelt; die Ergebnisse wurden anschließend in der folgenden Grafik visualisiert. Eingezeichnet sind pro analysiertem Prozess jeweils die ermittelten Ähnlichkeiten des ähnlichsten Prozesses in Prozent. Hierbei wurde Blau für die Messergebnisse verwendet, welche ohne die in dieser Arbeit beschriebene Erweiterung bezüglich der Nachbarschaftsbeziehungen entstanden sind, und Rot für die Ergebnisse, welche mit der Erweiterung entstanden.

Es zeigt sich, dass die Messergebnisse mit Nachbarschaftserweiterung im Mittel niedriger sind als die ohne Erweiterung. Bei den sehr ähnlichen Prozessen (welche sich dann auch gut für eine Vereinigung eignen) sind die Ergebnisse jedoch gleich hoch bzw. nur geringfügig niedriger als ohne Erweiterung. Hierdurch sind die so gewonnenen Aussagen aussagekräftiger, die Trennschärfe der Ergebnisse wird verbessert und ähnliche Prozesse stechen stärker aus der Masse heraus, wodurch die Anzahl der Prozesse, welche vom Anwender bezüglich einer möglichen Vereinigung betrachtet werden müssen, reduziert wird.

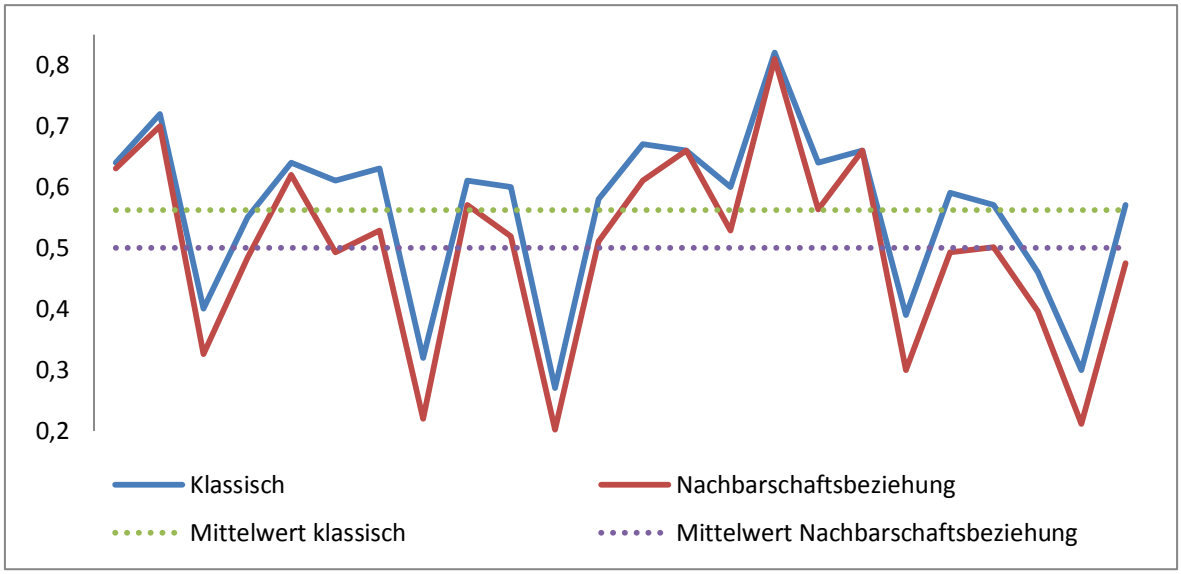

Abbildung 37: Ähnlichkeitsmessung – Vergleich der Methoden

Quelle: Eigene Erstellung

8.2 Evaluierung der Vereinigung von Prozessmodellen

Betrachtet man aktuelle Arbeiten zum Thema Vereinigung von Prozessmodellen wie „Merging Event-driven Process Chains“ von Gottschalk et al. (Gottschalk, Aalst, & Jansen-Vullers, Merging Event-driven Process Chains, 2009) oder „Merging business process models“ von Rosa et al. (Rosa, Dumas, Uba, & Dijkman, 2010) zeigt sich, dass hierbei zumeist auf das Verhalten der Prozesse Wert gelegt wird. Das Ziel der Vereinigung bei diesem Ansatz ist daher, das Verhalten aller vereinigten Prozesse möglichst verlustlos und vollständig zusammenzuführen. Weiterführend wird von den Autoren hierbei auch das Verhalten als Qualitätsmerkmal herangezogen, sodass die Qualität der Vereinigung daran gemessen wird, inwieweit sich das Verhalten der vereinigten Prozesse im vereinigten Prozess abbilden lässt.

Hierdurch ergeben sich innerhalb des vereinigten Prozessmodells zahlreiche Ausführungspfade, welche in unterschiedlicher Art und Weise zu sinnvollen oder auch problematischen bzw. fehlerhaften Lösungen kombiniert werden können. Ein Vorteil dieses Ansatzes ist, dass ein Überblick über die Ausführungspfade und die vereinigten Elemente leicht erlangt werden kann. Dies führt allerdings auch dazu, dass vor einer Ausführung eines Prozesses zuerst eine Konfigurierung, beispielsweise mit konfigurierbaren, ereignisgesteuerten Prozessketten (k-EPK) (Vgl. Rosemann & Aalst, 2007, S. 7ff) erfolgen muss, um den später in der Ausführungsphase zu durchlaufenden Pfad individuell zu definieren.

Dieser Vorgang muss manuell erfolgen, weshalb immer die Gefahr besteht, dass Fehler gemacht werden. Zusätzlich zeigt sich, dass in der Praxis im Umgang mit Prozessen zumeist wenig geschulte Personen mit dem Erstellen und Warten von Prozessen beauftragt werden, was die Wahrscheinlichkeit für Fehler zusätzlich erhöht. (Vgl. Rosemann M. , 2006, S. 249ff) Dass diese Gefahr auch praxisrelevant ist, beweisen Untersuchungen, welche ergaben, dass 3,3 bis 37,5 Prozent aller Prozesse in Geschäftsprozessdaten von Unternehmen fehlerbehaftet sind. (Vgl. Mendling, Empirical studies in process model verification, 2009, S. 218)

Eine weitere praxisrelevante Problematik wird beim längerfristigen Einsatz eines Prozesses sichtbar. Durch Wartung und Erweiterung sind hierbei Modifikationen am Prozess notwendig. Aufgrund der zuvor erwähnten Beibehaltung aller möglichen Ausführungspfade und der hohen Komplexität (große Anzahl an Ausführungspfaden und Elementen) sind diese Aufgaben allerdings sehr aufwändig und fehleranfällig. Dies vor allem deshalb, da auf die entsprechende Konfiguration der Ausführungspfade Rücksicht genommen werden muss bzw. diese ebenfalls überprüft bzw. angepasst werden müssen.

In der Arbeit „What Makes Process Models Understandable?“ von J. Mendling et al. wird eine Studie zur Messung der Verständlichkeit von Prozessmodellen durchgeführt. Diese ergab, dass eine klar erkennbare Strukturierung eine positive Auswirkung auf das Verständnis hat und dass eine hohe Anzahl an Elementen und Verbindungen sowie viele Verbindungen pro Element eine negative Auswirkung haben. (Vgl. Mendling, Reijers, & Cardoso, What Makes Process Models Understandable?, 2007) Dies sind Punkte, bei welchen deutlich wird, dass der in den zuvor genannten Arbeiten verwendete Vereinigungsansatz Schwächen hat.

Um diese Problematik anzugehen, wird in dieser Arbeit ein alternativer Ansatz vorgestellt. Dieser basiert auf den grundlegenden Anforderungen, welche in der Softwareentwicklung an eine Vereinigung gestellt werden. In diesem Bereich der Informatik wird auf Wartung und Erweiterbarkeit sowie auf die einfache Erkennbarkeit von Ausführungspfaden Wert gelegt. (Vgl. Bolton & Johnston, 2009, S. 9ff) Diese Anforderung wurde anschließend für die Entwicklung der in Kapitel 6 angeführten Lösungen zur Vereinigung von Prozessmodellen als richtungsweisend herangezogen.

Dies resultierte in einer Vereinigung, welche nicht mehr darauf abzielt, alle möglichen Ausführungspfade der vereinigten Elemente zu erhalten. Unterschiede in den Ergebnissen bestehen hierbei vor allem abseits der beiden möglichen Extremfälle (alle Elemente der vereinigten Prozesse sind gleich oder keine Prozesselemente stimmen überein). So werden, basierend auf einer Drei-Wege-Vereinigung, auch Informationen aus dem Vaterelement herangezogen, um, soweit dies sinnvoll ist (wenn diese in einem der vereinigten Kindprozesse gelöscht wurden), Elemente aus dem Ergebnisprozess zu entfernen. Auch wurden verschiedene Mechanismen implementiert, welche automatisierte bzw. halb automatisierte Anpassungen des Kontrollflusses und damit der Prozessstruktur ermöglichen. Hierdurch kann bei Bedarf das Ergebnis der Vereinigung noch besser an die speziellen Anforderungen der Prozesse angepasst werden und es ist möglich, für überlappende Prozessteile zwischen verschiedenen Ausführungsreihenfolgen, wie sequenziell oder parallel, zu wählen.

Das Ergebnis der Vereinigung ist durch dieses Vorgehen klar strukturiert und damit leichter zu warten und zu erweitern und bedarf außerdem vor der Ausführung keiner zwingenden Nachbearbeitung.

In der Folge werden die beiden hier vorgestellten Ansätze einander gegenübergestellt. Dabei wird jeweils auf die individuellen Stärken und Schwächen der unterschiedlichen Verfahren eingegangen.

Hier vorgestellter Ansatz: mengenbasierte Vereinigung mit struktur- und verhaltensbasierten Erweiterungen	Vereinigung unter Beibehaltung aller möglichen Ausführungspfade
Vorteile	**Vorteile**
• Direkt ausführbares Ergebnis • Geringere Komplexität der Ergebnisse • Leichtere Wartbarkeit und Erweiterbarkeit der Ergebnisse • Keine zwingende Nachbearbeitung vor der Ausführung • Leichteres Verständnis der Abläufe für den betrachtenden Anwender	• Verhalten der vereinigten Prozesse bleibt vollständig erhalten • Gute Übersicht über Zusammenhänge aller Ursprungsprozesse • Geringer Rechenaufwand für die Vereinigung
Nachteile	**Nachteile**
• Keine vollständige Abbildung aller Pfade • Höherer Rechenaufwand für die Vereinigung	• Sehr komplexer Ergebnisprozess • Schwierige Wartung und Erweiterung • Vor der Ausführung hat zwingend eine Nachbearbeitung zu erfolgen

Tabelle 12: Gegenüberstellung der Vereinigungsansätze

Quelle: Eigene Erstellung

9 Diskussion und Ausblick

In dieser Arbeit wurden neue Vorschläge für die Messung der Ähnlichkeit von Prozessmodellen sowie für die Vereinigung bestehender Prozesse in praxisnahen Anwendungsbereichen erstellt. Es wurden verschiedene Verfahren und Techniken zur Erreichung der zuvor genannten Bereiche angeführt sowie Möglichkeiten, um auftretende Herausforderungen zu meistern. Deshalb werden hier eine Zusammenfassung der Ergebnisse und ein Ausblick für weitere Forschungsmöglichkeiten geboten.

In Kapitel 2 (Prozessmodellierung) wurden grundlegende theoretische Hintergründe zur Prozessmodellierung beschrieben. So wurde dort geklärt, welche Informationen in einem Prozessmodell überhaupt zur Verfügung stehen. Von den drei erwähnten Informationssichten (Kontrollfluss, Datenfluss und Attribute) wird im späteren Verlauf nur noch der Kontrollfluss genauer behandelt. Weiters erfolgte eine Analyse mehrerer Modellierungssprachen, wobei, basierend auf den hierdurch erhaltenen Informationen, ein Metamodell erstellt wurde. Dieses diente anschließend als Grundlage, um das in Kapitel 4 (Intermediate Format) beschriebene Datenformat auszuarbeiten. Dieser Ansatz erlaubt es, von der Modellierungssprache wie z.B. BPMN oder EPK unabhängige Algorithmen zu entwickeln. Zusätzlich wird in Kapitel 2 auch noch auf Korrektheitskriterien für Prozessmodelle eingegangen, wobei diese zur Beurteilung der von den im späteren Teil der Arbeit ausgearbeiteten Vereinigungsalgorithmen erzeugten Prozessmodellen verwendet werden können. Zusätzlich werden von den Prozessmodellen unabhängig erfassbare Informationen behandelt. Diese unabhängigen Informationen fallen hierbei bei der Modifizierung bzw. Ausführung als Änderungs- bzw. Ausführungsprotokolle an.

Kapitel 3 behandelt die für die Ähnlichkeitsmessung wichtigen Bereiche der Prozessmodifikation. Dabei werden unterschiedliche Methoden und Operationen zur Modifikation vorgestellt und analysiert. Hierbei wird auch auf die Problematik der Prozessinstanzmodifikation eingegangen, bei

welcher der jeweilige Ausführungszustand der Instanz sowie die hierbei angewendeten Korrektheitskriterien und Kompensationsmechanismen von großer Bedeutung sind. Auch die verschiedenen Änderungsarten werden ausführlich beschrieben. Zusätzlich erfolgt eine Betrachtung verschiedener Methoden zur Erkennung und Behandlung der Differenzen. Hierbei werden textbasierte Verfahren wie LCS genauso behandelt wie rein mengenbasierte Ansätze. Ein Ansatz, welcher Ereignisprotokolle einsetzt, sowie deren Abbildung und Analyse mittels MXML und XES wird aufgrund seiner Bedeutung ebenfalls beleuchtet. Für die weitere Arbeit wird dann der mengenbasierte Ansatz bevorzugt, da dieser die geringsten Anforderungen an die vorhandene Datenbasis stellt. So kann beispielsweise nicht immer davon ausgegangen werden, dass umfangreiche Ausführungs- und Änderungsprotokolle zur Verfügung stehen.

In Kapitel 5 wird eines der beiden Kernthemen dieser Arbeit, die Ähnlichkeitsmessung zwischen Prozessen, bearbeitet. Im Vergleich mit bestehenden Arbeiten wird hierbei besonders auf die möglichen Optimierungsstrategien für die zu analysierenden Prozessdaten eingegangen. Verschiedene Probleme wie Homonyme, Synonyme oder unterschiedliche Modellierungstechniken werden aufgezeigt und Lösungen vorgestellt. Hierbei werden auch Methoden zur Optimierung der Abläufe und Daten sowie zur Minimierung der Rechenlast durch Filterung der Daten und einer Miteinbeziehung verschiedener Sichten behandelt. Dies hat insbesondere beim Einsatz im Geschäftsalltag und den hierbei vorkommenden umfangreichen Prozessdatenbanken und deren Analyse zur nachfolgenden Vereinigung bestehender Prozesse Vorteile. Die Ähnlichkeitsmessung selbst basiert auf einem hybriden Verfahren, welches die Vorteile einer strukturbasierten Erstanalyse (hohe Geschwindigkeit) mit einem verhaltensbasierten Verfahren (hohe Genauigkeit) verbindet. Durch eigene Erweiterungen (erweiterte Analyse der Nachbarschaftsbeziehungen) werden diese Ansätze anschließend noch ausgebaut.

Im darauffolgenden Kapitel 6, dem zweiten Kernthema dieser Arbeit, wird auf die Vereinigung von Prozessmodellen eingegangen. Hierbei werden praxistaugliche Lösungen vorgestellt, deren Vorteile im Firmenalltag zum Tragen kommen. Als Vorteil ist insbesondere die im Vergleich mit den hier ebenfalls vorgestellten akademischen Lösungen, höhere Qualität hinsichtlich der Erweiter- und Wartbarkeit, zu erwähnen. Auch die von den Anwendern manuell durchzuführende Nachbearbeitungszeit wird bei der hier vorgestellten Lösung minimal gehalten. Dies gelingt, indem Ansätze verwendet werden, welche mit den Modellen mutiger umgehen und versuchen, die Anforderungen, welche in der Softwareentwicklung an Vereinigungsalgorithmen gestellt werden, in die Welt der Prozesse zu transformieren. Auch in der Softwareentwicklung wird erwartet, dass nach einer Vereinigung ein sofort ausführbares Ergebnis entsteht.

Zu beiden Hauptthemen dieser Arbeit erfolgte auch eine prototypische Implementierung der vorgestellten Algorithmen. Diese findet bei der Evaluierung der Ansätze in Kapitel 8 Anwendung und wird im vorangehenden Kapitel 7 kurz vorgestellt.

In dieser Arbeit gelangt es, alle in der Einleitung vorgestellten wissenschaftlichen Fragestellungen zu klären und einen wichtigen wissenschaftlichen Beitrag zu den untersuchten Themenbereichen zu leisten:

- So konnten Lösungen für die drei in der Einleitung vorgestellten Kernaspekte gefunden werden. Die Korrektheit der bei der Vereinigung entstehenden Prozesse wird von den Algorithmen proaktiv unterstützt. Diese erkennen mögliche Probleme und wenden automatisiert Lösungen wie eine Wiederherstellung des Kontrollflusses an oder befragen, falls dies notwendig ist, unter anderem bei möglicherweise konfliktbehafteten Elementanordnungen, den Anwender.

- Um eine Unabhängigkeit der Algorithmen von den Modellierungssprachen zu erreichen, wurde als Lösung ein Metamodell erstellt, das auf in dieser Arbeit durchgeführten Analysen verbreiteter Modellierungssprachen (EPK und BPMN) basiert. Dieses Metamodell diente als Vorlage für ein hier definiertes intermediates Format, welches schlussendlich die modellierungssprachenunabhängige Implementierung ermöglichte.
- Die Automatisierung aller Abläufe konnte ebenfalls erfolgreich umgesetzt werden. Im Bereich der Ähnlichkeitsmessung wurde eine vollständige Automatisierung erreicht, sodass während der Ausführung der Algorithmen keine Nutzerinteraktion nötig ist. Bei den Vereinigungsalgorithmen ist eine adaptive Automatisierung möglich. Falls notwendig, kann hier der Nutzer bei konkreten Entscheidungen unterstützend eingreifen. Alternativ können vor Analysebeginn aber auch vordefinierte Entscheidungen in den Algorithmus eingespielt werden, welche dann bei allen Nutzerentscheidungen automatisiert angewendet werden.

Das zu Beginn dieser Arbeit entwickelte Metamodell basiert vor allem auf Analysen zweier verbreiteter Modellierungssprachen (BPMN und EPK). Für zukünftige Erweiterungen dieser Arbeit wäre es hierbei interessant, weitere Modellierungssprachen wie Unified Modeling Language (UML) Aktivitätsdiagramme (Vgl. Kecher, 2006, S. 213ff), High-Level-Petrinetze (Vgl. International Organization for Standardization, 2000, S. 6ff) oder Business Process Execution Language (BPEL) (Vgl. OASIS, 2007) mit einzubeziehen und diesen dann ebenfalls die entwickelten Algorithmen zugänglich zu machen.

Vergleichbar hiermit ist eine Erweiterung der pro Modell einbezogenen Sichten. So werden im aktuellen Status nur Informationen des Kontrollflusses berücksichtig. Eine Analyse, ob Informationen aus dem Datenfluss sowie die zusätzlichen Attribute (Vgl. Abschnitt 2.1) die Qualität der Ergebnisse verbessern und wie diese eingebunden werden können, ist für zukünftige Arbeiten sicherlich sinnvoll. Bei Miteinbeziehung des Datenflusses wäre es ein Vorteil, wenn mehr Informationen zur automatisierten Erkennung von Abhängigkeiten und zur Rekonstruktionen des Kontrollflusses, wie hier bei den Vereinigungsalgorithmen verwendet, verfügbar wären. Zusätzlich profitiert auch die Ähnlichkeitsmessung davon, da diese mit steigender Informationsmenge ein immer differenzierteres Bild von den Verwandtschaftsbeziehungen der Prozesse zueinander zeichnen kann.

Als logischer Schritt der Weiterführung können auch Integration und Erweiterung der bestehenden Lösungen in ein Versionsverwaltungssystem für Prozesse gesehen werden. Von den Funktionen her kann man sich hierbei an bestehenden Lösungen für die Softwareentwicklung wie Apache Subversion (SVN) (Vgl. The Apache Software Foundation, 2011) oder GIT (Vgl. GIT, 2012) orientieren. In „A manifesto for model merging“ von Brunet et al. (Brunet, Chechik, Easterbrook, Nejati, Niu, & Sabetzadeh, 2006) wird außerdem noch ein spezielles Framework für eine Versionsverwaltung von Prozessen definiert. Hier sind aber naturgemäß große Überschneidungen mit den zuvor erwähnten Funktionen von Versionsverwaltungen in der Softwareentwicklung gegeben. Für eine entsprechende Umsetzung müssten vor allem die Bereiche Datenspeicherung (Entwicklung einer Datenbank zur effizienten Abbildung und Untersuchung von Prozessversionen) und Visualisierung (von Prozessen und Modifikationen/Zusammenhängen) angegangen werden.

Speziell bei der Ähnlichkeitsmessung wäre es sinnvoll, weitere Anstrengungen hinsichtlich einer Performanceoptimierung der Algorithmen vorzunehmen. Ein Forschungspotential ist hier vor allem hinsichtlich heuristischer Methoden zur Beschleunigung der eigentlichen Analyse und bei der Verringerung der zu analysierenden Datenmenge durch vorausgehende Filterung dieser gegeben. Ein Ansatz mittels Verzweigung und Schranke (engl. Branch and Bound) Prinzip wird hierbei

als Anregung in der Diskussion des Kapitels 5 kurz besprochen und zeigte in einer experimentellen Form während der Evaluierung auch ein großes Potential. Weitere Optimierungen sind möglich, wenn man in Betracht zieht, dass nicht alle Ähnlichkeiten gesucht werden, sondern vor allem die ähnlichsten Prozesse, wobei diese dann auch noch nur für eine Vereinigung interessant sind, wenn eine sehr große Ähnlichkeit, beispielsweise größer als 60 Prozent, besteht. Hierdurch können für die Vereinigung uninteressante Prozesse frühzeitig (wenn sich eine niedrige Ähnlichkeit andeutet) erkannt und ausgespart werden.

10 Literaturverzeichnis

Aalst, W. v., & Basten, T. (2002). Inheritance of Workflows – An approach to tackling problems related to change. *Theoretical Computer Science, 1-2*(270), S. 125-203.

Aalst, W. v., & Jablonski, S. (2000). Dealing with workflow change: identification of issues and solutions. *International Journal of Computer Systems Science & Engineering, 15*(5), S. 267-276.

Aalst, W. v., & Weijters, A. (2003). Process mining: a research agenda. *Computers in Industry - Special issue: Process/workflow mining, 3*(53), S. 231 - 244 .

Aalst, W. v., Dongen, B. v., Günther, C., Mans, R., Medeiros, A. A., Rozinat, A., et al. (2007). ProM 4.0: Comprehensive Support for Real Process Analysis. *ICATPN'07 Proceedings of the 28th international conference on Applications and theory of Petri nets and other models of concurrency* , S. 484-494 .

Aalst, W. v., Reijersa, H., Weijtersa, A., & Dongena, B. v. (2006). Business process mining: An industrial application. *Information Systems, 5*(32), S. 713-732 .

Agrawal, R., Gunopulos, D., & Leymann, F. (1998). Mining Process Models from Workflow Logs. *EDBT '98 Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*, S. 469-483.

Alanen, M., & Porres, I. (2003). Difference and Union of Models. *Lecture Notes in Computer Science*(2863), S. 2-17.

Allweyer, T. (2009). *BPMN 2.0 - Business Process Model and Notation.* Norderstedt: Books on Demand.

Atkinson, C., & Stoll, D. (2008). *An Environment for the Orthographic Modeling of Workflow Components.* Mannheim: PRIMIUM.

Bae, J., Caverlee, J., Liu, L., Rouse, B., & Yan, H. (2006). Process Mining by Measuring Process Block Similarity. *BPM'06 Proceedings of the 2006 international conference on Business Process Management Workshops*(4103), S. 141-152.

Baerisch, S. (2005). *Versionskontrollsysteme in der Softwareentwicklung.* Bonn: Informationszentrum Sozialwissenschaften der Arbeitsgemeinschaft Sozialwissenschaftlicher Institute e.V.

Bellman, R., & Dreyfus, S. (2010). *Dynamic Programming.* Princeton: Princeton University Press.

Bereg, S., Kubica, M., Wale, T., & Zhu, B. (2005). RNA multiple structural alignment with longest common subsequences. *COCOON'05 Proceedings of the 11th annual international conference on Computing and Combinatorics*, S. 32-41.

Bibliographisches Institut GmbH. (2012). *Duden*. Abgerufen am 1. 8. 2012 von Duden: http://www.duden.de/rechtschreibung/liefern

Bleiholder, J. (2010). *Data Fusion and Conflict Resolution in Integrated Information Systems.* Potsdam: Hasso-Plattner-Institut.

Bleiholder, J., & Naumann, F. (2006). Conflict handling strategies in an integrated information system. *Proceedings of the Workshop on Information Integration on the Web*.

Böhmer, K. (2011). *Erstellung einer multivarianten Datenanalyse mit Use-Case-basierendem Visualisierungskonzept.* Wien: FH Technikum Wien.

Bolton, G., & Johnston, S. (2009). *IfSQ Level-2 A Foundation-Level Standard for Computer Program Source Code.* Cambridge: IfSQ.

Brunet, G., Chechik, M., Easterbrook, S., Nejati, S., Niu, N., & Sabetzadeh, M. (2006). A manifesto for model merging. *GaMMa '06 Proceedings of the 2006 international workshop on Global integrated model management* , S. 5-12.

Bruza, P., & Weide, T. v. (1993). The Semantics of Data Flow Diagrams. *Proceedings of the International Conference on Management of Data*, S. 66-78.

Cardoso, J., & Aalst, W. v. (2009). *Handbook of Research on Business Process Modeling.* London: IGI Global.

Casati, F., Ceri, S., Pernici, B., & Pozzi, G. (1996). Workflow Evolution. In *Data and Knowledge Engineering* (S. 438-455). Berlin: Springer Verlag.

Caumanns, J. (1999). *A Fast and Simple Stemming Algorithm for German Words.* Berlin: Freie Universität Berlin, Fachbereich Mathematik und Informatik .

Chomicki, J., Lobo, J., & Naqvi, S. (2003). Conflict resolution using logic programming. *IEEE Transactions on Knowledge and Data Engineering, 1*(15), S. 244-249.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms.* Massachusetts: The MIT Press.

Curran, T. A., & Keller, G. (1998). *SAP R/3 Business Blueprint: Business Engineering mit den R/3-Referenzprozessen .* Bonn : Addison-Wesley Verlag .

Davenport, T. H., & Short, J. E. (1995). The new industrial engineering. *Proceedings of 1995 IEEE Annual International*, S. 287-292.

Dehnert, J., & Rittgen, P. (2001). Relaxed Soundness of Business Processes. *CAiSE '01 Proceedings of the 13th International Conference on Advanced Information Systems Engineering* , S. 157-170.

Dehnert, J., & Zimmermann, A. (2005). On the suitability of correctness criteria for business process models. *BPM'05 Proceedings of the 3rd international conference on Business Process Management*(3649), S. 386-391.

Dijkman, R., Dumas, M., & Garcia-Banuelos, L. (2009). Graph Matching Algorithms for Business Process Model Similarity Search. *BPM '09 Proceedings of the 7th International Conference on Business Process Management*, S. 48-63.

Dijkman, R., Rosa, M. L., & Reijers, H. A. (2011). Managing Large Collections of Business Process Models – Current Techniques and Challenges. *Computers in Industry, 2*(63), S. 91-97.

Dongen, B. v., & Aalst, W. v. (2005). A Meta Model for Process Mining Data. *Conference on Advanced Information Systems Engineering – CAiSE*(2), S. 309-320.

Dongen, B. v., Dijkman, R., & Mendling, J. (2008). Measuring Similarity between Business Process Models. *CAiSE '08 Proceedings of the 20th international conference on Advanced Information Systems Engineering*, S. 450-464.

Dongen, B. v., Mendling, J., & Aalst, W. v. (2006). Structural Patterns for Soundness of Business Process Models. *Proceeding EDOC '06 Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference* , S. 116-128.

Dubray, J.-J. (kein Datum). A Novel Approach for Modeling Business Process Definitions. *APCCM '07 Proceedings of the fourth Asia-Pacific conference on Comceptual modelling*(67), S. 71-80.

Ehrig, M., Koschmider, A., & Oberweis, A. (2007). *Measuring Similarity between Semantic Business Process Models.* Karlsruhe: Universität Karlsruhe.

Fellbaum, C. (1998). *Wordnet: An Electronic Lexical Database.* Bradford: Bradford Book.

Galler, J. (1995). Metamodelle des Workflow Managements. *Veröffentlichungen des Instituts für Wirtschaftsinformatik , 121* .

Gatterbauer, W., & Suciu, D. (2010). Data conflict resolution using trust mappings. *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* , S. 219-230 .

Georgakopoulos, D., Hornick, M., & Sheth, A. (1995). *An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure.* Georgia: University of Georgia.

Giese, C., Overdick, H., & Buhl, W. (2005). Realisierungsstrategien für Prozessfamilien: Werkzeuge für Modellierung und Generierung. *Process Family Engineering in Service-Oriented Applications*(15), S. 348-361 .

GIT. (2012). *git --everything-is-local*. Abgerufen am 9. 12 2012 von git: http://git-scm.com/

Gottschalk, F., Aalst, W. M., & Jansen-Vullers, M. H. (2008). Mining Reference Process Models and their Configurations. *OTM Workshops 2008*(1), S. 263-272.

Gottschalk, F., Aalst, W. M., & Jansen-Vullers, M. H. (2009). Merging Event-driven Process Chains. *OTM '08 Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008*, S. 418-426.

Greenfield, P., Fekete, A., Jang, J., & Kuo, D. (2003). Compensation is not enough. *Enterprise Distributed Object Computing Conference, 2003. Proceedings. Seventh IEEE International*, S. 232-239.

Griffin, L., Colcheste, A., Roll, S., & Studholme, C. (1994). Hierarchical Segmentation Satisfying Constraints. *BMVC94: Proceedings of the 5th British Machine Vision Conference*, S. 135-144.

Gubała, T., Bubak, M., Malawski, M., & Rycerz, K. (2006). Semantic-Based Grid Workflow Composition. *Parallel Processing and Applied Mathematics* , S. 651-658.

Günther, C. W. (2009). *XES Standard Definition.*

Hamming, R. W. (1950). Error-detecting and error-correcting codes. *The bell system technical journal, 2*(26), S. 147-63.

Harrington, H. J. (1991). *Business Process Improvement: The Breakthrough Strategy for Total Quality, Productivity, and Competitiveness.* New York: Mcgraw-Hill Professional.

Hunt, J. W., & McIlroy, M. D. (1976). An Algorithm for Differential File Comparison. *Computing Science Technical Report, Bell Laboratories*(41), S. 61-70.

Ignat, C. L., & Norrie, M. C. (2004). Operation-based versus State-based Merging in Asynchronous Graphical Collaborative Editing. *Proceedings of the Sixth International Workshop on Collaborative Editing, Conference on Computer Supported Cooperative Work (CSCW'04)*, S. 105-116.

International Organization for Standardization. (2000). *High-level Petri Nets – Concepts, Definitions and Graphical Notation – ISO/IEC 15909.* Geneva: International Organization for Standardization.

Jeckle, M. C. (2000). Metamodellierung als Ansatz zur Lösung der Modellaustauschproblematik im Umfeld objektorientierter Modellierungssprachen. *Proceedings International Knowledge Technology Forum* , S. 136-156.

Jurafsky, D., & Martin, J. H. (2008). *Speech and language processing.* New Jersey: Pearson Prentice Hall International.

Karnath, M., & Ramamritham, K. (1998). Failure handling and coordinated execution of concurrent workflows. *ICDE '98 Proceedings of the Fourteenth International Conference on Data Engineering* , S. 334-341.

Kecher, C. (2006). *UML 2.0.* Bonn: Galileo Press.

Keller, G., Nüttgens, M., & Scheer, A.-W. (1992). Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). *Veröffentlichungen des Instituts für Wirtschaftsinformatik*(89), S. 117-132.

Kettinger, W. J., Teng, J. T., & Guha, S. (1997). Business process change: a study of methodologies, techniques, and tools. *MIS Quarterly, 1*(21), S. 55-80.

Kim, M., & Notkin, D. (2006). Program element matching for multi-version program analyses. *MSR '06 Proceedings of the 2006 international workshop on Mining software repositories* , S. 58-64.

Küster, J. M., Gerth, C., Fürster, A., & Engels, G. (2008). Detecting and Resolving Process Model Differences in the Absence of a Change Log. *BPM '08 Proceedings of the 6th International Conference on Business Process Management* , S. 224-260.

Lawler, E. L., & Wood, D. E. (8 1966). Branch-and-Bound Methods: A Survey. *Operations Research, 4*(14), S. 699-719.

Levenshtein, V. I. (1965). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady, 8*(10), S. 707-710.

Li, C., Reichert, M., & Wombacher, A. (2008). On Measuring Process Model Similarity based on High-level Change Operations. *Proceedings of the 27th International Conference on Conceptual Modeling*, S. 248-264.

Li, C., Reichert, M., & Wombacher, A. (2011). Mining Business Process Variants: Challenges, Scenarios, Algorithms. *Data & Knowledge Engineering, 5*(70), S. 409-434.

Lippe, E., & Oosterom, N. v. (1992). Operation-based merging. *SDE 5 Proceedings of the fifth ACM SIGSOFT symposium on Software development environments*, S. 78-87.

Loskiewicz-Buczak, A., & Uhrig, R. E. (1995). Information Fusion by Set Operation. *SIMULATION, 1*(65), S. 52-66.

Ly, L. T., Rinderle, S., & Dadam, P. (2006). Semantic correctness in adaptive process management systems. *Proc. 4th Int'l conf. on Business Process Management (BPM'06)*(1), S. 193-208.

Medeiros, A. A., Aalst, W. v., & Weijters, A. (2008). Quantifying process equivalence based on observed behavior. *Data & Knowledge Engineering, 1*(64), S. 55-74.

Mendling, J. (2009). Empirical studies in process model verification. *Transactions on Petri Nets and Other Models of Concurrency II*, S. 208-224.

Mendling, J., Reijers, H. A., & Cardoso, J. (2007). What Makes Process Models Understandable? *Proceeding BPM'07 Proceedings of the 5th international conference on Business process management* , S. 48-63.

Mendling, J., Reijers, H., & Aalst, W. M. (2010). Seven Process Modeling Guidelines (7PMG). *Information and Software Technology, 2*(52), S. 127-136.

Mens, T. (2002). *A State-of-the-Art Survey on Software Merging.* Brussels: Vrije University Brussels.

Mens, T. (2002). A State-of-the-Art Survey on Software Merging. *IEEE Transactions on Software Engineering, 5*(28), S. 449-462.

Microsoft Corporation. (2012). *Enumerable Class*. Abgerufen am 1. 9. 2012 von MSDN: http://msdn.microsoft.com/en-us/library/bb345746

Mino, M., Tartakovski, A., & Bergmann, R. (2007). Representation and structure-based similarity assessment for agile workflows. *ICCBR '07 Proceedings of the 7th international conference on Case-Based Reasoning: Case-Based Reasoning Research and Development* , S. 224-238.

MIT, ERCIM, & Keio. (27. April 2007). *SOAP Version 1.2*. Abgerufen am 1. 8. 2012 von W3C: http://www.w3.org/TR/soap

Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S., & Zave, P. (2007). Matching and Merging of Statecharts Specifications. *29th International Conference on Software Engineering* , S. 54--64.

Neuhaus, M., Riesen, K., & Bunke, H. (2006). Fast suboptimal algorithms for the computation of graph edit distance. *SSPR'06/SPR'06 Proceedings of the 2006 joint IAPR international conference on Structural, Syntactic, and Statistical Pattern Recognition* , S. 163-172.

Nüttgens, M., & Rump, F. J. (2002). Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). *Entwicklungsmethoden für Informationssysteme und deren Anwendung - EMISA*, S. 64-77.

OASIS. (11. 4 2007). *Web Services Business Process Execution Language Version 2.0*. Abgerufen am 9. 12 2012 von http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

Object Management Group. (2010). *OMG Unified Modeling Language (OMG UML).* Needham: OMG Available Specification.

Object Management Group. (2011). *Business Process Modeling Notation.* Needham: OMG Available Specification.

Open Model Group. (April 2002). *Meta Object Facility*. Abgerufen am 10. 10. 2012 von http://www.omg.org

Oracle Corporation. (2011). *Map (Java Platform SE 6)*. Abgerufen am 1. 9. 2012 von JavaDoc: http://docs.oracle.com/javase/6/docs/api/java/util/Map.html

Paice, C. D. (1994). An evaluation method for stemming algorithms. *SIGIR '94 Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval* , S. 42-50.

Parent, C., & Spaccapietra, S. (1998). Issues and approaches of database integration. *Communications of the ACM, 5*(41), S. 166-179.

Peters, L. (2005). *Change Detection in XML Trees: a Survey.* Twente: University of Twente Faculty of Electrical Engineering, Mathematics and Computer Science.

Pottinger, R. A., & Bernstein, P. A. (2003). Merging models based on given correspondences. *VLDB '03 Proceedings of the 29th international conference on Very large databases*, S. 862-873 .

Preschler, J., Weidlich, M., & Mendling, J. (September 2012). The Projected TAR and its Application to Conformance Checking. *EMISA 2012*, S. 151-164.

Priyam, A., Karan, B. M., Sahoo, G., & Anwar, S. (2009). An Optimal Approach towards Sequence Analysis. *Contemporary Engineering Sciences, 8*(2), S. 383-390.

Richter, M. M. (1999). Classification and Learning of Similarity Measures. *SEKI Report*(18), S. 92-100.

Rinderle, S., Bassil, S., & Reichert, M. (2006). A Framework for Semantic Recovery Strategies in Case of Process Activity Failures. *Eighth International Conference on Enterprise Information Systems (ICEIS'06): Databases and Information Systems Integration*, S. 136-154.

Rinderle, S., Jurisch, M., & Reichert, M. (2007). On Deriving Net Change Information From Change Logs – The DELTALAYER-Algorithm –. *Automatica*, S. 364-381.

Rinderle, S., Reichert, M., & Dadam, P. (2004). Correctness criteria for dynamic changes in workflow systems – a survey. *Data & Knowledge Engineering, 1*(50), S. 9-34.

Rinderle, S., Reichert, M., & Dadam, P. (29. Oktober 2004). Disjoint and Overlapping Process Changes: Challenges, Solutions, Applications. *Proceedings 12th International Conference Cooperative Information Systems*, S. 101-120.

Rinderle, S., Reichert, M., & Dadam, P. (2004). On Dealing with Structural Conflicts between Process Type and Instance Changes. *Proc. 2nd. Int'l Conf. Business Process Management (BPM'04)*(3080), S. 274-289.

Rinderle, S., Reichert, M., Jurisch, M., & Kreher, U. (2006). On Representing, Purging, and Utilizing Change Logs in Process Management Systems. *BPM'06 Proceedings of the 4th international conference on Business Process Management* , S. 241-256.

Rinderle-Ma, S., Reichert, M., & Jurisch, M. (2011). On Utilizing Web Service Equivalence for Supporting the Composition Life Cycle. *International Journal of Web Services Research*, S. 41-67.

Rosa, M. L., Dumas, M., Uba, R., & Dijkman, R. (2010). Merging business process models. *OTM'10 Proceedings of the 2010 international conference on the move to meaningful internet systems*, S. 93-113.

Rosemann, M. (2006). Refactoring Large Process Model Repositories: part A. *Business Process Management Journal*, S. 249-254.

Rosemann, M., & Aalst, W. v. (2007). A configurable reference modelling language. *Information Systems, 1*(32), S. 1-23.

Russell, N., Hofstede, A. H., Edmond, D., & Aalst, W. M. (2005). Workflow Data Patterns: Identification, Representation and Tool Support. *ER'05 Proceedings of the 24th international conference on Conceptual Modeling*, S. 353-368.

Sabetzadeh, M., & Nejati, S. (2006). TReMer: A Tool for Relationship-Driven Model Merging. *ACM SIGSOFT Software Engineering Notes, 6*(31), S. 1-2.

Sanfeliu, A., & Fu, K.-S. (1983). A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics, 3*(13), S. 353-362.

Scheer, A.-W. (1992). *Architektur integrierter Informationssysteme. Grundlagen der Unternehmensmodellierung.* Berlin: Springer.

Scheer, A.-W. (2001). *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen.* Berlin: Springer.

Schmidt, W. (2008). *Deutsche Sprachkunde.* Berlin: IFB Verlag.

Shafranovich, Y. (Oktober 2005). *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. Abgerufen am 15. 6. 2012 von IETF: http://tools.ietf.org/html/rfc4180

Stentz, A. (1994). Optimal and Efficient Path Planning for Partially-Known Environments. *IEEE International Conference on Robotics and Automation*, S. 3310-3317.

The Apache Software Foundation. (2011). *Apache™ Subversion®*. Abgerufen am 9. 12. 2012 von Subversion: http://subversion.apache.org/

The PHP Group. (31. 8 2012). *similar_text*. Abgerufen am 1. 9. 2012 von PHP Manual: http://php.net/manual/en/function.similar-text.php

Tichy, W. F. (1983). The String-to-String Correction Problem with Block Moves. *ACM Transactions on Computer Systems (TOCS), 4*(2), S. 309-321.

Uchitel, S., & Chechik, M. (2004). Merging partial behavioural models. *SIGSOFT '04/FSE-12 Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering* , S. 43-52.

Weber, B., Reichert, M., & Rinderle-Ma, S. (2006). Change Patterns and Change Support Features – Enhancing Flexibility in Process-Aware Information Systems. *Data & Knowledge Engineering, 3*(66), S. 438-466.

Weber, B., Wild, W., Reichert, M., & Dadam, P. (20. 4. 2007). ProCycle – Integrierte Unterstützung des Prozesslebenszyklus. *KI Journal*(12), S. 9-15.

Weidlich, M., Mendling, J., & Weske, M. (2010). *Computation of Behavioural Profiles of Processes Models.* Potsdam: Hasso Plattner Institute.

Weske, M. (2007). *Business Process Management.* Berlin: Springer.

Wilbur, W. J., & Sirotkin, K. (1992). The automatic identification of stop words. *Journal of Information Science, 1*(18), S. 45-55.

Wombacher, A. (2006). Evaluation of technical measures for workflow similarity based on a pilot study. *Proceedings of the 2006 Confederated international conference on On the Move to Meaningful Internet Systems*, S. 255-272.

Yan, Z., Dijkman, R., & Grefen, P. (2010). Fast Business Process Similarity Search with Feature-Based Similarity Estimation. *OTM'10 Proceedings of the 2010 international conference on On the move to meaningful internet systems*(1), S. 60-77.

Zahran, S. (1998). *Software Process Improvement: Practical Guidelines for Business Success.* Essex: Addison-Wesley Professional .

Zhang, K., & Shasha, D. (1989). Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM Journal on Computing, 6*(18), S. 1245-1262.

Zwart, J.-W. (2009). Structure and order: asymmetric merge. *The Oxford Handbook of Linguistic Minimalism*, S. 96-118.

11 Abbildungsverzeichnis

12 Tabellenverzeichnis

13 Formelverzeichnis

14 Programmausdruckverzeichnis

15 Anhang

15.1 Lebenslauf

Lebenslauf mit Konzentration auf die akademische Laufbahn

Persönliche Daten

Name	Böhmer
Vorname	Kristof
Geburtsdatum	07.09.1987
Geburtsort	St. Pölten
Staatsangehörigkeit	Österreich
Anschrift	Stolzenthalergasse 13/19, 1080 Wien
Telefon	0676/9653062

Ausbildung

1998-2002	Unterstufe des BRG Ringstraße Krems
2002-2007	HTBLA Krems, Abteilung Informationstechnologie
Juni 2007	Reifeprüfung
2007-2011	FH Technikum Wien/Studienrichtung Informatik
Juni 2009	Abschluss Bachelor Informatik
März 2011	Abschluss Master Informationsmanagement und Computersicherheit
2011-2013	Universität Wien Studienrichtung Wirtschaftsinformatik
Jänner 2013	Verleihung des Ingenieurtitels
2013	Abschluss Master Wirtschaftsinformatik

15.2 Kurzfassung

Aufgrund formaler Richtlinien der Universität Wien wird die Kurzfassung dieser Arbeit hier im Anhang hinterlegt.

In dieser Diplomarbeit werden grundlegende Verfahren zum Aufbau eines Versionsverwaltungssystems für Prozessmodelle beschrieben, entworfen und implementiert. Hierzu erfolgt in den ersten Kapiteln eine Aufbereitung der theoretischen Hintergründe. Insbesondere wird hierbei auch ein Metamodell entworfen, welches in Kapitel 4 dazu verwendet wird, ein intermediate Format zu entwerfen, welches es erlaubt, alle Algorithmen unabhängig von den zu verarbeitenden Modellierungssprachen zu gestalten.

Bezüglich der Ähnlichkeitsmessung werden in Kapitel 5 Erweiterungen zu bestehenden Verfahren beschrieben, welche auf einer stärkeren Miteinbeziehung der Nachbarschaftsbeziehungen zwischen den Elementen basieren.

Für die Vereinigung bestehender Prozessmodelle, Kapitel 6, wird ein gänzlich neuer Ansatz vorgestellt, welcher Anforderungen, die in der Softwareentwicklung gestellt werden (das erzeugte Ergebnis muss ohne Nachbearbeitung weiterzuverwenden sein), mit denen einer Prozessmodellverarbeitung vereint. Dabei wird mit einer Kombination von mengenorientierten, strukturbasierten und verhaltensanalysierenden Ansätzen gearbeitet.

Bei der Evaluierung der entwickelten Verfahren zeigt sich, dass mit der in dieser Arbeit vorgestellten Erweiterung, welche die Nachbarschaftsbeziehungen der Elemente mit einbezieht, die Relevanz der Ergebnisse erhöht werden kann.

Bei der Vereinigung wird deutlich, dass die hier beschriebene Lösung Vorteile im praktischen Einsatz besitzt. So reduziert sich der manuell durchzuführende Nachbearbeitungsaufwand, und die Wartung und Erweiterbarkeit der erzeugten Prozessmodelle sind auf einem höheren Niveau als die im wissenschaftlichen Umfeld verbreiteten Ansätze, welche zum Vergleich behandelt werden.

Schlagwörter: Prozessmanagement, Ähnlichkeitsmessung, Vereinigung, Metamodell, intermediate Format

15.3 Abstract

Aufgrund formaler Richtlinien der Universität Wien wird der Abstract dieser Arbeit hier im Anhang hinterlegt.

This thesis designs and implements some of the basic algorithms necessary to create a version control system for process models. The first chapters provide the necessary theoretical background information. The 2nd chapter also describes the design of a meta model, based on evaluation of various modeling languages which will later be used to design an intermediate file format (see chapter 4) which allows to implement all the algorithms independent of different modeling languages.

Chapter 5 deals with the measurement of the similarity of processes. It describes some already known algorithms along with a new extension to existing concepts based on a greater involvement of the neighborhood relationships between the elements.

In chapter 6 a completely new approach is presented for the merging of existing processes. It is based on the idea that the result of the merge must be able to be reused easily and that only a minimal amount of post processing must be done manually by the user. To succeed in this task a combination of quantity-oriented, structure- and behavior-based approaches are used.

This thesis also contains an evaluation of the created algorithms. The evaluation of the similarity measurement algorithms shows, that the expressiveness of the results could be enhanced if the neighborhood information is included.

The evaluation also shows that the process merging solution described here has advantages in practical use. It reduces the manual finishing work to be performed and the maintenance and expansion of the generated process models is to be treated as a more important factor than the wildly used scientific algorithms which are used for comparison purpose.

Tags: Process management, similarity measurement, merging, meta model, intermediate format