



DISSERTATION

Titel der Dissertation

A Multi-Domain Framework for
Community Building Based on
Data Tagging

Verfasser

Bojan Božić, MSc.

angestrebter akademischer Grad

Doktor der technischen Wissenschaften
(Dr.techn.)

Wien, im Februar 2014

Studienkennzahl lt. Studienblatt:	A 786 880
Dissertationsgebiet lt. Studienblatt:	Informatik
Betreuer:	Ao. Univ.-Prof. MMag. Dr. Werner Winiwarter

For Silvia, Lea, Ema, and Pia.

Acknowledgements

I would like to express my gratitude to my advisor Prof. Werner Winiwarter. I couldn't imagine a better person to work with. During the last years he was very supportive, helpful, and had the right advice to each problem I might have come up with (hence, really earned the designation advisor). Also I would like to thank all of my colleagues for their support and many interesting and helpful discussions. Especially, Gerald Schimak the coordinator of the TaToo project which made this thesis even possible, Gerhard Dünnebeil who did a lot of coding and development effort for the time series toolbox, Denis Havlik who had a first idea for the thesis topic, Sigi Kluckner and Sergiu Gordea who came up with several interesting discussions and inspirations. I am grateful to the head of my business unit Andrea Nowak for her support and the opportunity to take half a year off in order to complete my thesis.

Special thanks to my friends Christian Wagner for mental support and Franz Knipp who opened his office for me, gave me asylum and hence provided a constructive, productive, but most of all, peaceful environment for a lot of hours of writing. I owe my parents in law, Căcilia and Alois Vukovich, a debt of gratitude for their mental and social support when they shared meat and mead and supported my wife during my absences. Thanks to my sister and brother in law Daria and Andreas Vukovich for helping out when help was most needed. Great thanks to my parents Verica and Branko Božić for giving me life and bringing our family to Austria where I've got the opportunity to build a family on my own. My brother, Andrej Božić, has also helped out when needed, thanks bro.

Lea, Ema, and Pia, my lovely and beautiful daughters. I am so glad to be your dad. Thank you for all the positive distractions you caused, for showing me what life is all about, and keeping me grounded. I love you so much. But most of all I am thankful to my wife Silvia Božić who was always there for me and shared with me all lows and highs of a PhD student's life. Silvia, I would never had made it if it wasn't for you. You gave me your selfless love, passion, support, encouragement, company, and so much more all the way long. Thank you, I love you!

Abstract

In time series processing there is a gap between resources and information, and users (domain experts) who need access to this information. The resources are available on the Web, and attempts have been undertaken to close this gap by using different approaches. However, none of them was to integrate Semantic Web technologies as a possible solution to the problem.

There are a lot of existing user communities which have interest in certain information. The communication between the owner of data, who is not able to provide her data to the appropriate community, and the communities themselves is actually a big difficulty and needs improvement. As far as the data owner is not capable of assigning data to interested people, the community needs to play a proactive role.

Therefore, a technology is needed which enables a community to get to the information it needs for their research. This is where a multi-domain framework for community building through tagged data can be a solution to the problem. The idea is to use meta information for data and have it evaluated and tagged by groups of users. Relevant information for a user group can be dispensable for other groups. Semantic annotations have to be processed here and used to associate groups and data. A multi-domain framework can be a benefit and provide a cycle of information enrichment of different domain ontologies in a multi-domain context. The semantic annotations can be used for assignment of time series with meta information about the annotator, the time of the annotation, the region where the annotation has been performed, etc. to the appropriate user. Relying on this information, the time series can be processed, used for filtering, and analyzed. The results can be used to build groups of users and communities as well as virtual habitats around specific topics.

Finally, there would be a world of virtual communities with experts for certain research topics having easy and fast access to relevant information and possibilities to exchange knowledge and research results.

Zusammenfassung

In der Zeitreihenverarbeitung gibt es eine Lücke zwischen den Ressourcen und Informationen, sowie Benutzern, die Zugriff auf diese Informationen benötigen. Die Ressourcen sind im Internet verfügbar und es wurden Versuche unternommen diese Lücke durch die Verwendung unterschiedlicher Technologien zu schließen. Jedoch keine davon war die Integration von Semantic Web Technologien um zu einer Lösung dieses Problems zu gelangen.

Es gibt eine Vielzahl von bestehenden User-Communities, die Interesse an bestimmten Informationen haben. Die Kommunikation zwischen den Eigentümern von Daten, die nicht in der Lage sind ihre Daten der entsprechenden Community zur Verfügung zu stellen, und den Benutzern dieser Daten, ist tatsächlich eine große Schwierigkeit und muss verbessert werden. Da der Eigentümer der Daten nicht in der Lage ist eine Zuordnung seiner Daten zu interessierten Gruppen zu treffen, muss die Community selbst eine aktive Rolle spielen.

Daher wird eine Technologie benötigt, die es ermöglicht, einer wissenschaftlichen Community alle benötigten Informationen für ihre Forschung verfügbar zu machen. Hierbei kann ein Multi-Domain Framework für Community Building durch Datentagging eine Lösung für das Problem sein. Die Idee ist, Meta-Informationen für Daten zu verwenden und durch Benutzer auswerten und taggen zu lassen. Relevante Informationen für eine Benutzergruppe können für andere Gruppen uninteressant sein. Semantische Annotationen müssen hier verarbeitet und genutzt werden, damit Gruppen mit den für sie relevanten Daten assoziiert werden können. Ein Multi-Domain Framework kann hier von Vorteil sein und einen Zyklus von Informationsbereicherung für Domänenbenutzer in verschiedenen Domänen mit unterschiedlichen Ontologien und vielsprachigen Kontexten bieten. Die semantischen Annotationen können verwendet werden um Zeitreihen mit Meta-Informationen über den Autor, die Zeit der Annotation, die Region in der die Annotation durchgeführt worden ist etc. dem richtigen Benutzer zuzuordnen. Unter Berücksichtigung dieser Informationen können Zeitreihen verarbeitet, gefiltert und analysiert werden. Die Ergebnisse können verwendet werden, um Gruppen von Be-

nutzern und Communities, sowie virtuelle Lebensräume rund um bestimmte Themen zu erschaffen.

Dies würde eine Welt virtueller Gemeinschaften von Experten für bestimmte Forschungsthemen mit einfachem und schnellem Zugriff auf relevante Informationen und Möglichkeiten zum Austausch von Wissen und Forschungsergebnissen kreieren.

Copyright Notice

I have tried my best to find all copyright owners of images and to collect their agreement to use the content in this thesis. Should there be a copyright infringement however, please inform me about it.

Contents

I	Introduction and Backgrounds	1
1	Introduction	3
1.1	Motivation	3
1.1.1	Time Series Processing Problem	3
1.1.2	Semantic Web Problem	4
1.1.3	Community Building Problem	4
1.2	Contributions	5
1.2.1	Contributions to Time Series Processing	5
1.2.2	Contributions to Semantic Web	6
1.2.3	Contributions to Community Building	6
1.3	Organization of Thesis	7
2	State of the Art	9
2.1	Time Series Processing	9
2.1.1	Background	9
2.1.2	Approaches	11
2.1.3	Methods	14
2.1.4	Literature	16
2.2	Semantic Web	17
2.2.1	Background	17
2.2.2	Basic Standards	19
2.2.3	Vocabularies	23
2.2.4	Languages	26
2.2.5	Web Service Standards	28
2.2.6	Other Standards	31
2.2.7	Semantic Web Frameworks	32
2.2.8	Ontology Editors	34
2.2.9	Ontology Mapping Tools	37
2.2.10	Reasoners	38
2.2.11	Browsers	40
2.2.12	Annotation Tools	41

2.2.13	Other Tools	42
2.2.14	Literature	48
2.3	Community Building	49
2.3.1	Background	50
2.3.2	Existing Platforms	51
2.3.3	Literature	57

II Implementation 59

3 Time Series Processing Language 61

3.1	Time Series Processing Language	61
3.2	Language Specification	66
3.2.1	letter	66
3.2.2	digit	66
3.2.3	integer	66
3.2.4	float	67
3.2.5	number	67
3.2.6	string	67
3.2.7	stmt	67
3.2.8	pipe	68
3.2.9	generator	68
3.2.10	ts_param_id_list	68
3.2.11	formal_parameter	69
3.2.12	expression_list	69
3.2.13	assign_expression	69
3.2.14	if_expression	70
3.2.15	logic_expression	70
3.2.16	add_expression	71
3.2.17	mult_expression	72
3.2.18	expression	72
3.2.19	ts_slice	73
3.2.20	property_access	73
3.2.21	interval	73
3.2.22	numeric_interval	74
3.2.23	logical_interval	74
3.2.24	time_interval	74
3.2.25	bra	75
3.2.26	ket	75
3.2.27	log_index_expression	75
3.2.28	int_with_time	76

3.2.29	time_index_expression	76
3.2.30	every_phrase	76
3.3	Architecture	79
3.3.1	Lexer	81
3.3.2	Parser	84
3.3.3	Interpreter	86
3.4	Expressions	88
3.5	Benchmarks	93
3.6	Profiling	100
4	Semantic Framework	103
4.1	Semantic Repository	106
4.1.1	Implementation	106
4.1.2	Semantic Repository Benchmarks	108
4.2	Connectors	110
4.2.1	Connector Interface	112
4.2.2	Web Site Connector Class	112
4.2.3	RDFa Parser Class	114
4.2.4	RDFa Parser Stylesheet	117
4.2.5	Harvesting Benchmarks	120
4.3	Ontology Mapping	123
4.4	Semantic Processing	131
4.5	User Interaction and Annotation	138
4.5.1	User Interaction	138
4.5.2	Annotation	140
4.6	Future Improvements	142
5	Community Building	143
5.1	Authentication	145
5.2	Group Generation	149
5.3	User and Time Series Rating	151
5.4	Sharing of Time Series among Users and User Groups	154
5.5	SPARQL Endpoint	159
5.6	Reasoning in Community Building	161
5.6.1	Assignment of Users to Groups	162
5.6.2	Suggestions of Time Series to Users	163
5.6.3	Assignment of Time Series to Groups	163
5.6.4	Suggestions of Related Users to Users	164

III	Use Cases and Validation	167
6	Validation of Semantic Time Series Processing	169
6.1	Climate Change Twin Regions – Discovery Platform	171
6.1.1	Introduction	171
6.1.2	Ontology	173
6.1.3	Use Case	181
6.2	Anthropogenic Impact and Global Climate Change	192
6.2.1	Introduction	192
6.2.2	Ontology	193
6.2.3	Use Case	202
IV	Conclusions	211
7	Conclusions	213
7.1	Results and Contributions	213
7.2	Future Work and Research	216
V	Bibliography	219
VI	Appendix	233
A	Time Series Processing Language Specification	235
B	RDFa Parsing Stylesheet	239
C	Curriculum Vitae	259

List of Figures

2.1	An environmental time series with threshold plotted in R. . . .	10
2.2	The extended, 3D Semantic Web layer cake (by Benjamin Nowack).	18
2.3	Difference of the view on a web document between browsers (left) and humans (right) [Group et al., 2012].	21
3.1	Letter.	66
3.2	Digit.	66
3.3	Integer.	67
3.4	Float.	67
3.5	Number.	67
3.6	String.	68
3.7	Statement.	68
3.8	Pipe.	68
3.9	Generator.	69
3.10	Time series parameter id list.	69
3.11	Formal parameter.	69
3.12	Expression list.	70
3.13	Assignment expression.	70
3.14	IF expression.	71
3.15	Logic expression.	72
3.16	Addition expression.	73
3.17	Multiplication expression.	73
3.18	Expression.	74
3.19	Time series slice.	74
3.20	Property access.	75
3.21	Interval.	75
3.22	Numeric interval.	76
3.23	Logical interval.	76
3.24	Time interval.	77
3.25	First part of bracket.	77
3.26	Second part of bracket.	77

3.27	Logical index expression.	77
3.28	Integer with time unit.	78
3.29	Time index expression.	78
3.30	Every phrase.	78
3.31	The language processing architecture.	80
3.32	Workflow of a time series processor.	81
3.33	The web service based architecture.	82
3.34	The Semantic Web architecture.	82
3.35	Execution time of representative expressions for the 3 main classes.	97
3.36	Execution time of the representative class 3 expressions for the 7 subclasses.	99
4.1	UML component diagram of the Semantic Framework.	105
4.2	Typical workflow example for using the semantic repository.	106
4.3	Visualized results of the benchmark tests.	109
4.4	UML class diagram representing the architecture of the Web Site Connector component.	111
4.5	Vizualized results of the harvesting benchmark test.	122
4.6	Relation between bridge ontology and domain ontologies.	129
4.7	Bridge ontology for Semantic Time Series Processing (created in Protégé and visualized in RDF Gravity).	130
4.8	Output of a consistency check of our bridge ontology in Protégé using Pellet.	137
4.9	Screenshot of the Semantic Time Series Processing web portal.	139
4.10	Window for annotation of semantic time series.	141
5.1	Semantic tagging and building of groups with common interests.	144
5.2	The login window of the Semantic Time Series Processing portlet.	146
5.3	User settings screen which is presented to the user after her first login.	148
5.4	An example of group suggestions in the Groups menu.	149
5.5	An example of a detailed view for a group (in this case the group 'Water').	150
5.6	User annotation and rating window after selecting a user from the table in the group overview.	152
5.7	The annotation and rating window for time series.	153
5.8	Overview of the time series data presented as a line chart.	153
5.9	SPARQL endpoint of the Semantic Time Series Processing portal.	160

6.1	Use case of the prototype.	170
6.2	Screenshot of the Climate Twins application [Ungar et al., 2011].	172
6.3	Enlarged screenshot of the Climate Twins control panel [Ungar et al., 2011].	173
6.4	Building class of the AIT ontology.	174
6.5	Climate class of the AIT ontology.	174
6.6	ClimateAdaptation class of the AIT ontology.	175
6.7	ClimateMitigation class of the AIT ontology.	175
6.8	ClimateModel class of the AIT ontology.	176
6.9	Energy class of the AIT ontology.	176
6.10	MeasureType class of the AIT ontology.	177
6.11	MeteorologicalPhenomena class of the AIT ontology.	177
6.12	PrecipitationValueExpression class of the AIT ontology. .	178
6.13	Reliability class of the AIT ontology.	178
6.14	SpatialExpression class of the AIT ontology.	179
6.15	TemperatureExpression class of the AIT ontology.	179
6.16	TemperatureType class of the AIT ontology.	179
6.17	TemporalExpression class of the AIT ontology.	180
6.18	ThermalProcess class of the AIT ontology.	181
6.19	Weather class of the AIT ontology.	181
6.20	A simplified overview of the most important parts of the ontology.	182
6.21	Sample politician user with Climate Twins ontology.	184
6.22	Sample scientist user with Climate Twins ontology.	185
6.23	Sample business manager user with Climate Twins ontology. .	186
6.24	A list of time series with the topics “Climate Change” and “Air Pollution”.	187
6.25	Screenshot of the Climate Change and Air Pollution groups. .	191
6.26	Screenshot of the SVOD Web portal.	194
6.27	Screenshot of the GENASIS Web portal.	195
6.28	Cancer class of the MU ontology.	196
6.29	BreastCancer class of the MU ontology.	196
6.30	Compound class of the MU ontology.	197
6.31	Data class of the MU ontology.	198
6.32	Disease class of the MU ontology.	199
6.33	EpidemiologicalMeasures class of the MU ontology.	199
6.34	Matrix class of the MU ontology.	201
6.35	ProjectType class of the MU ontology.	201
6.36	Risk class of the MU ontology.	202
6.37	Simplified overview of the MU domain ontology.	203
6.38	Screenshot of filter settings provided by User 1.	207

6.39 Screenshot of the filtering results as they are presented to the user.	209
--	-----

List of Tables

2.1	Overview of time series processing approaches.	14
2.2	Overview of time series processing methods.	16
2.3	Overview of basic standards.	24
2.4	Overview of vocabularies.	25
2.5	Overview of languages.	28
2.6	Overview of web service standards.	31
2.7	Overview of other standards.	32
2.8	Overview of Semantic Web frameworks.	34
2.9	Overview of ontology editors.	37
2.10	Overview of ontology mapping tools.	39
2.11	Overview of reasoners.	40
2.12	Overview of browsers.	41
2.13	Overview of annotation tools.	42
2.14	Overview of other tools.	47
2.15	Overview of community building platforms.	58
3.1	General expressions and their meanings.	62
3.2	Demonstration of time series processing results.	65
3.3	Execution time of the time series processing expressions (Part 1).	94
3.4	Execution time of the time series processing expressions (Part 2).	95
3.5	Overview and categorization of class 3 expressions.	98
4.1	Results of the benchmark tests for the semantic repository. . .	110
4.2	Benchmark results for harvesting websites.	121
4.3	Ontology mapping tools overview according to our requirements.	125
4.4	Overview of reasoners according to our requirements.	133

Part I

Introduction and Backgrounds

Chapter 1

Introduction

Nowadays we are facing an unbelievable amount of information flooding the World Wide Web. Recent research in the field of Semantic Web technologies brought us quite far providing an attempt to tackle a lot of problems the Web introduced during its evolution. However, the Semantic Web approach is still far away from being the solution to all current problems concerning the World Wide Web. When we go deeper in potential fields of application of Semantic Web technologies it gets harder to find the right solution. An ideal example is time series processing. While facing the problem of bringing together time series data and their interpretation with the expert user, we can use methods from the fields of time series processing, Semantic Web, and community building to close this gap. This thesis combines those three fields and presents a possible solution to the problem.

1.1 Motivation

The problem this thesis aims to solve is threefold. As already described, the thesis combines three disciplines of Computer Science, which are Time Series Processing, Semantic Web, and Community Building. Hence, there are also three problems, which need to be explained in the following subsections.

1.1.1 Time Series Processing Problem

Time Series Processing has a long tradition in many different scientific areas, such as computer science, mathematics, economics, etc.

The latest developments in this field are data mining, time series analysis, forecasting of future data, and handling flexible data formats and large amounts of data.

Although the combination of state-of-the-art non-linear time series analysis and prediction techniques, which are based on mathematical models, and dynamic data mining techniques provide acceptable results, there is currently no way to perform such tasks without very large computing capacities (which are of course very expensive as well).

This is where a combination with Semantic Web technologies can make a significant improvement. By processing time series together with an ontology for a specific domain, a lot of computational power can be saved, as time series data can be interpreted dynamically. Furthermore, users are able to annotate and tag resources. This way they can improve the assignment to a domain, which results in more efficient time series forecasting. The Time Series problem and the approach to solve it are described in detail in Chapter 3.

1.1.2 Semantic Web Problem

In the field of Semantic Web development and research work have made a lot of progress recently. There are a lot of standards, technologies, tools, and projects, which are very promising or already established. A state of the art overview of Semantic Web technologies is given in Chapter 2. Furthermore, new ideas are needed to refresh current developments and to combine Semantic Web with other disciplines, which may take advantage of its technologies.

However, there is still no complete software solution to be used for the time series processing field of application. In this thesis a Semantic Time Series Processing Framework is proposed. The framework is based on the idea of semantic time series processing, and is discussed in detail together with the concept of semantically-enriched time series in Chapter 4.

1.1.3 Community Building Problem

The problem with Community Building is not bound to the research field of Web 2.0, which is also called Social Web. The developments in this field are growing steadily, software solutions are booming, and there is not much to worry about at the moment. The only point, which could be criticized is that Community Building technologies are not being used in other fields of application, where they can make significant contributions.

In this thesis possibilities of using Community Building techniques for defining groups of interest for different views on semantically-enriched time series data, as well as categorization of the groups in certain domains, will be discussed. Chapter 5 deals with the exploitation of Community Building to solve problems in Time Series Processing and Semantic Web. Use cases

for this approach and validation of the findings can be found in Chapter 6, they also have been published in Božić and Winiwarter [2013b].

1.2 Contributions

As three different fields of Computer Science are dealt with, this thesis delivers contributions to three different fields as well. They can be subdivided into contributions to Time Series Processing, contributions to the Semantic Web, and contributions to Community Building.

In fact, contributions means that the results of the thesis bring new scientific findings to these fields and generate new ideas, as well as new ways to tackle problems in the handled areas of research.

To measure and prove the contributions, improvements, and new ideas accomplished, the previously mentioned prototypes are the method of choice. Therefore, in this place, the contributions are being explained by means of the prototypes and the motivation to develop them. Furthermore, validation of the developed methodology is accomplished by using ontologies from an FP7 research project.

The contributions to various fields of research are discussed in detail in Chapter 7 “Conclusions”.

1.2.1 Contributions to Time Series Processing

Regarding Time Series Processing, before the implementation of an own language to fulfill this task started, there were thoughts to use already implemented languages and therefore to save time by reusing existing technologies, instead of developing something new from scratch. The usage of languages for numerical computations, such as Octave ¹, MATLAB ², or Mathematica³ was considered. However, a decision has been made, not to use anything already existing, but to implement a new language taking into account all the specialities and idiosyncrasies of Time Series Processing.

The prototype implemented to prove the contributions to Time Series Processing is a dynamic, extensible, and easy to use language specifically developed for the problem of Time Series Processing. It can be extended to use other languages, as it is modular and therefore more powerful than general purpose languages for numerical computation. The main contribution to the field of Time Series Processing is to have an all-in-one solution for all kinds

¹<http://www.gnu.org/software/octave/>

²<http://www.mathworks.com/products/matlab/>

³<http://www.wolfram.com/mathematica/>

of problems in this area. The input for the language are time series and an expression, which defines how they should be processed. This is a boundary to other, general-purpose languages, as they are not able to work with time series directly. The output is an arbitrary number of time series containing the results of the calculations, as defined by the expression. An exact description of the language and its benefits and contributions can be found in Chapter 3 “Time Series Processing Language”. The requirements analysis and language specification have also been published in Schimak et al. [2011] and Božić et al. [2012].

1.2.2 Contributions to Semantic Web

In the field of Semantic Web technologies there are a lot of tools, technologies, standards, and projects, as Chapter 2 “State of the Art” demonstrates. However, the problem is that there is no general framework, which can be used by some client who is willing to enrich her application easily, flexibly, and quickly by using only one single framework or toolbox from one single vendor in terms of time series processing.

The Semantic Framework prototype, as a contribution to the Semantic Web, is therefore implemented to demonstrate how such an architecture could be accomplished. It consists of a Data Access component for access to a semantic repository, databases and connectors to other resources; a Semantic Processor for reasoning and processing of semantically-enriched time series; an Annotation and Tagging component for handling annotations from users; and a Semantic Discovery component for discovery of semantic resources, generally time series, which can be tagged and processed. The application using the framework to be enriched by Semantic Web technologies, can be a Web Service, a library, or any kind of stand-alone application. All details about the prototype and its advantages can be found in Chapter 4 “Semantic Framework”. The framework description has also been published in Božić [2011], the ontology mapping and reasoning in Božić and Winiwarter [2013a].

1.2.3 Contributions to Community Building

Community Building can be seen as an additional outcome of the thesis. The prototype, which provides contributions to Community Building, is a direct result of the other two prototypes, which means that it contributes to Community Building based on the functionality which is offered by Time Series Processing and Semantic Web technologies.

The principle is quite obvious: If one imagines time series as data input and semantics as enrichment of the data input, as well as users and groups

of scientists, which are specifically interested in one certain aspect of the time series, it is relatively clear that there must be communities of people, in a so-called domain, which a certain view on a time series is dedicated to. Therefore, the contribution to Community Building is a semi-automatic approach to generate communities using a certain, common interest around semantically-enriched time series. The prototype for Community Building is simply integrating this approach, which it thereby makes available to the field of Community Building. The Community Building prototype is introduced in Chapter 5 “Community Building”. It has also been published in Božić and Winiwarter [2012] and Božić [2012].

1.3 Organization of Thesis

This introduction, being the first Chapter of the thesis, aims at addressing a number of problems in three different fields, which this thesis combines, suggesting a new scientific approach to bring us a step further and refresh the Semantic Web movement by offering a couple of new ideas to this exciting research area. It starts with the motivation for writing this thesis, continues by giving an overview of scientific contributions, and ends with a description of the organization of the thesis.

The second Chapter “State of the Art” contains an extensive state-of-the-art analysis of current standards, technologies, tools, and projects. It aims to provide an as much as possible complete overview of previous work and to present how far the research in the Semantic Web has come until today.

Chapter 3 “Time Series Processing” sets the stage for the main part of the thesis by introducing time series; methods of traditional time series processing; a modern Time Series Processing language, implemented within the scope of this thesis; and expressions in Time Series Processing defining current possibilities and weaknesses of Time Series Processing.

The fourth Chapter “Semantic Framework” introduces a new concept of Time Series Processing. It shows the difference to traditional time series, how ontologies are used to enrich time series, how tagging and annotation is performed, and what the technical details and benefits are. Further it introduces a language to fulfill the complex task of processing Semantic Time Series. The chapter provides a specification of the language, details about the implementation, and sample expressions showing the capabilities of the language.

In Chapter 5 “Community Building” the context of Community Building is presented which is relevant for the thesis, but also the usage of the Semantic Framework to enable Community Building, different Community Building

methods, use cases, and examples.

Chapter 6 “Validation of Semantic Time Series Processing” shows use cases based on ontologies developed in TaToo’s validation scenarios “Climate Change Twin Regions – Discovery Platform” and “Anthropogenic Impact and Global Climate Change”.

Finally, the last Chapter 7 “Conclusions” provides a summary of the thesis’ contributions to research in the fields of Time Series Processing, Semantic Web, and Community Building, as well as a preview of future research topics and fields based on the findings of this thesis.

Chapter 2

State of the Art

The first part of the thesis is a thorough state-of-the-art analysis. Therefore, this chapter provides a background on time series processing, Semantic Web, and community building; it presents relevant literature in these fields and lists existing technologies. We remind the reader that this can only be a snapshot, since technology in computer science is developing rapidly, and there are a lot of different solutions; therefore, we only claim to list the most popular standards, methods, tools, and technologies to our best knowledge. Parts of this chapter have already been published in Božić [2011].

2.1 Time Series Processing

Time series processing is the application of time series analysis in computer science. This means that time series analysis methods are implemented in several programming languages and used to perform various calculations with time series values.

As the problem area is slightly different from the view point of computer science, compared to mathematics, we concentrate in this section on this special point of view.

2.1.1 Background

Our understanding of time series is that time series consist of slots. Each slot has a certain time stamp and a value. Hence, the simplest time series is a table with two rows, where the first row contains the time stamps and the second row the values. The intervals between time stamps do not have to be equally spaced. Additionally, slots, as well as whole time series, can have certain properties. For time series this means that they are able to

carry additional information, which is valid for slots and their values, and is specific for the whole time series. In the case of slots, this means that they are able to transport more than one value for a certain time stamp.

A simple time series, plotted in R^1 , is shown in Figure 2.1. It consists of environmental measurements of a certain hazardous material for the last 20 years. The red line defines the threshold, which, when exceeded, leads to the fact that the air quality is considered dangerous.

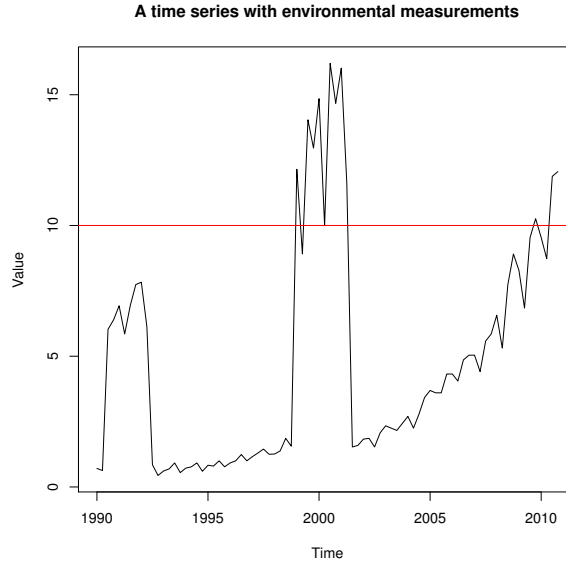


Figure 2.1: An environmental time series with threshold plotted in R.

To process time series data, we use certain expressions. For example, Equation 2.1 shows the processing of a time series with awareness of threshold exceeding. The input is a time series with measurements of ozone concentration in Vienna. The expression defines that the property warning should be set to “ACTIVE” if the value of 10 ppm (parts per million) is exceeded, and to “INACTIVE” if the value is within limits.

$$\begin{aligned}
 @Ozone_Vienna < warning &= \text{“ACTIVE”} \\
 &\text{if } Ozone_Vienna[n] > 0.1 \text{ ppm} \\
 &\text{otherwise } warning = \text{“INACTIVE”} > \quad (2.1)
 \end{aligned}$$

Another example is shown in Equation 2.2, where the acidity of water in the Danube is measured. The input time series contains values of the

¹<http://www.r-project.org>

measurement. The expression calculates mean values of pollution for one week. Therefore, the selected time series data lasts from past one week to the current week, where every week one slot is taken. The result is a time series of mean values with a constant interval (one value per week).

$$@Acidity_Danube < Acidity_Danube[t - 1\ week \ ..\ t].mean > \text{ every } 1\ week \quad (2.2)$$

Finally, Equation 2.3 shows the interconnection of two expressions with the pipe operator. This means that the two expressions are calculated one after another, and the output of the first is used as input of the second.

$$@Time_Series < A[n] * 2 > || @Time_Series < A[n] / 2 > \quad (2.3)$$

As we can see from the previous expressions, time series processing can be very powerful and can help us with processing measurements and deriving decisions from the results. The processing can take place in real-time by streaming a time series or with a time series whose data is already complete. In our approach it does not matter in which domain the processing itself takes place or from which domain the data originates.

2.1.2 Approaches

As time series contain data, time series analysis means the same as data analysis. The only difference is that time series data is structured by time. In the following some of the most popular approaches for analysis of data in time series are introduced and described.

Forecasting

Machine learning models have started to play a more and more dominant role in time series processing, especially in the area of forecasting [Ahmed et al., 2010]. The first approaches emerged when neural network models had been developed to replace classical statistical models. After that, other models have been extended as well, such as support vector machines, decision trees, etc. All of them together are called machine learning models ([Hastie et al., 2009] and [Alpaydin, 2010]). Theoretical understanding of models has been developed widely during the last years, and also the validation of models through examination and performance comparison has been an important goal. For the user, the important point is to provide less choices, but to

increase quality and usefulness of the proposed models, as well as to provide knowledge about strengths and weaknesses.

In machine learning, especially its large scale studies, most of the research has been done regarding the classification domain [Caruana and Niculescu-Mizil, 2006]. Sharda and Patil [1992] have compared neural networks to ARIMA² on the M-competition³ time series data. Hill et al. [1996] have also considered the M-competition data and have compared neural networks with traditional methods. Swanson and White [1995] have applied their comparison to nine US macroeconomic series. Alon et al. [2001] have analyzed neural networks versus other traditional methods, such as Winters exponential smoothing⁴, Box-Jenkins ARIMA, and multivariate regression, on retail sales data. Callen et al. [1996] have compared neural networks with linear models for 296 quarterly earnings time series. Zhang et al. [2004] tackle a common weakness, but with additional variables. Tersvirta et al. [2004] consider neural networks, smooth transition auto-regressions, and linear models for 47 macroeconomic series. Although not all outcomes of these studies are clear, generally it can be said that neural networks are able to outperform classical linear techniques.

Spatial Data Analysis

There are two ways of performing spatial data analysis: the first one is vector-based and the second one raster-based [Raju, 2005]. For spatial analysis in time series processing, we need spatial relationships between map features and attribute data. Spatial analysis has some important tasks for operations, the most important of them are described in the following list:

- *Single layer operations*: operate on single data layers and correspond to queries as well as alterations of data.
- *Multi layer operations*: operate on multiple layers and manipulate spatial data.
- *Topological overlays*: are operations for multiple layers. They collect features from different layers and combine them to form new maps, provide new features and information, and therefore improve already existing maps.

²<http://people.duke.edu/~rnau/411arim.htm>

³<http://www.neural-forecasting.com/m-competitions.htm>

⁴<http://www.slideshare.net/ForecastIT/forecast-it-5-winters-exponential-smoothing-4370646>

- *Point pattern analysis*: examines and evaluates spatial patterns as well as point features.
- *Network analysis*: designed for connected networks and their line features, deals with transportation problems or location analysis, e.g. routing, optimization, path finding, etc.
- *Surface analysis*: uses spatial distribution of three-dimensional surface information.
- *Grid analysis*: processes spatial data in regularly spaced form.
- *Fuzzy spatial analysis*: fuzzy set theory based. The fuzzy set theory is a generalized Boolean algebra for situations with gradual transition zones, which are used for class separation without using conventional crisp boundaries.

Time Series Analysis

In time series analysis, the main goal is to extract statistics out of one or more time series on the one hand, and to work with that data on the other hand. Another important functionality is filtering of time series data. This means that we want to present only selected parts of time series to the user, or to provide a certain view (e.g. select some properties which are specifically relevant to the user) of the data, etc. However, time series processing is systematic, which means that we apply a specified procedure in order to extract statistical and mathematical solutions to a problem. This problem can be bound to time correlation, distribution of observation, transformation of data, and data selection.

A lot of (maybe even all) data in computer science, but also in other fields (e.g. medicine, finance, social sciences, etc.) can be modelled as time series. Here, time series processing can provide approaches to solve the problems. Lately, a field which is very important in the context of this thesis has become more and more popular as a field of application in time series analysis, namely the environmental field.

Overview

Table 2.1 presents an overview of approaches in Time Series Processing. The approaches are listed in alphabetical order, and their category and purpose are described.

Name	Category	Purpose
Forecasting	Machine learning model	Used to calculate future time series values.
Spatial Data Analysis	Data analysis approach	Used to analyze spatial data.
Time Series Analysis	Data analysis approach	Used to analyze time bound data.

Table 2.1: Overview of time series processing approaches.

2.1.3 Methods

The methods in time series processing are subdivided into frequency-domain and time-domain methods. Therefore, at first these two main fields are explained. Additionally, a description of some of the most popular correlation and analysis methods is given.

Frequency-Domain Methods

Frequency-domain methods assume that points of interest for users of time series relate to natural variations, which can be found in most data. This is often caused by phenomena of interest (biological, physical, or environmental). Actions like winds or changes of water temperature can cause reactions of the environment they have an impact on. Therefore, the frequency-domain methods analyze and extend studies in periodicities by measuring frequent changes and their impacts.

Time-Domain Methods

For time-domain methods the assumption that correlation between points in time can be described by the dependence of current and past values is made. They focus on modeling of future values of time series, which can be predicted by using current and past values. The starting point is linear regression of present values on their own past values and on past values of other time series.

Auto-Correlation

Auto-Correlation is a kind of cross correlation (see below), where the signal cross-correlates with itself. It describes similarity between functions and the time separation between them. Auto-correlation is often used in time series processing to find patterns (e.g periodic signals and noise, etc.).

Cross-Correlation

Auto-correlation has the problem that there is a peak at a lag of zero. Therefore, cross-correlation includes a standardizing factor, which leads to correlation values between -1 and +1. The term cross-correlation refers to the correlation between two random variables, while the correlation of a vector is a correlation matrix between the components of that vector.

Spectral Analysis

Spectral analysis partitions periodic variation and therefore influences variance associated with periodicity of interest. The power spectrum is the variance profile over frequency. Time- and frequency-domain methodologies produce often similar answers for long time series, but the performance is better in the time domain. Sometimes the frequency domain is just more convenient, but could also be covered in a more efficient way by time domain methods.

Wavelet Analysis

Wavelet analysis is a rather new trend in time series processing. It helps to analyze localized variations of power in time series [Torrence and Compo, 1998]. When a time series is divided into time-frequency space, variability and time dependency can be determined. This analysis method is popular in geophysics and used there for many different fields of application (e.g. tropical convection, oscillation, temperature prediction, etc.). Wavelet analysis lacks quantitative results so far, however, it has been regarded as an interesting diversion by many scientists and is able to produce pure qualitative results. The problem is that wavelet analysis transforms one-dimensional time series to diffuse two-dimensional images of time-frequency. Therefore a misconception can be assumed. The solution could be an arbitrary normalization and statistical significance tests.

Overview

Table 2.2 presents an overview of methods in Time Series Processing. The methods are listed in alphabetical order, and their category and purpose are described.

Name	Category	Purpose
Auto-Correlation	Correlation method	Correlates a variable with itself.
Cross-Correlation	Correlation method	Correlates two random variables.
Frequency-Domain Methods	Analysis method	Measure frequent changes.
Spectral Analysis	Analysis method	Partitions periodic variations.
Time-Domain Methods	Analysis method	Predictions based on time changes.
Wavelet Analysis	Analysis method	Analyzes localized variations.

Table 2.2: Overview of time series processing methods.

2.1.4 Literature

Shumway and Stoffer [2010] offer an exhaustive overview of the current state of the art in Time Series Processing. Modern techniques in time series analysis are described, such as categorical time series analysis, the spectral envelope, multivariate spectral methods, long memory series, non-linear models, longitudinal data analysis, ARCH models, resampling techniques, wavelets, stochastic volatility, etc.

It is also worth to mention developments like non-linear system identification and characterization of time series, which depends on appropriate pre-processing of data. A neural network architecture is used to extract NLPCs (non-linear principal components) [Rico-Martinez et al., 1996].

The time series modeling approach developed by Lehtokangas et al. [1996] has a neural network structure with three layers and uses a general autoregressive model; Figwer [1997] presents an approach for the simulation of wide-sense stationary random time series defined by their power spectral density; and the study of Zhang et al. [2001] presents an experimental evaluation of neural networks for non-linear time series forecasting.

As Fazlollahi et al. [1997] state, the effectiveness of decision support systems (DSS) is enhanced through dynamic adaptation of support to the needs of the decision maker, to problem, and decision context. Chuang and Yadav [1998] developed an integrated conceptual model of an adaptive decision support system (ADSS).

The state-of-the-art in Semantic Web and Web Mining is developing very fast. These two research areas are more and more combined together; as a result, Web Mining is being improved by exploiting semantic structures in the Web, and its techniques are used for building the Semantic Web [Stumme et al., 2006]. Therefore, it would improve the situation to use time series data, which is enriched by semantics, in order to combine these two fields.

This means that semantically enriched time series are used to transport data from Web Mining applications and services to Semantic Web applications and back.

As time series pattern extraction and processing is a very complex process [Yuan and Huang, 2001], it could be simplified by a Service Oriented Architecture approach for sensor networks (from which time series currently originate in general). Therefore, the data model would be enriched with semantic concepts, and complex decision support systems could be implemented [Amato et al., 2010]. Patel-Schneider and Horrocks [2007] show that the Semantic Web is best realized by using standard logics. This would also be the case when using semantic time series, as time series processing is always based on standard logics.

2.2 Semantic Web

Semantic Web is a collective name for standards, technologies, and tools, which can be used to add meaning to data. The idea behind the Semantic Web is that information should not be interpreted by intelligent algorithms or artificial intelligence approaches, but rather be structured beforehand in a way that it can be processed and used by simple applications or, simply said, information should be machine-understandable.

2.2.1 Background

The World Wide Web, as we know it and as it nowadays exists, is undergoing massive changes (at least this is the prediction of experts) [Berners-Lee et al., 2001]. The future promises an evolution of the current human-readable Web towards a machine-understandable and meaningful Web which can be processed by simple software implementations based on the structure it has. Semantic Web approaches can be used to fulfill this by structuring information and therefore preparing it for further processing. The biggest (and probably also only) challenge is to convince the user to provide meta-information and annotate data she creates or to tag automatically generated data. The Semantic Web is an extension of the Web we use today with the goal to bring it towards the direction of automated processing of data.

Figure 2.2 shows the structure of Semantic Web technologies⁵. The technology stack is an extension of the originally presented (simpler), so-called layer cake. The linked data pillar represents a selection of Semantic Web technologies that are used for linked data. The Semantic Web technology

⁵<http://bnode.org/blog/2009/07/08/the-semantic-web-not-a-piece-of-cake>

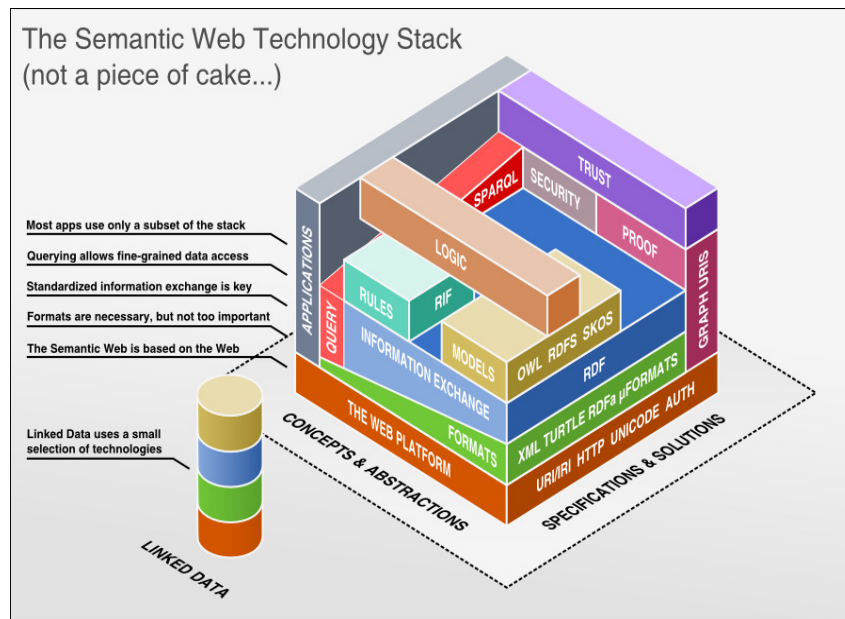


Figure 2.2: The extended, 3D Semantic Web layer cake (by Benjamin Nowack).

graphic classifies the technologies into the following categories: web platform, formats, information exchange, query, models, rules, applications, proof, security, trust, and logic.

The Semantic Web works with structured information and inference rules, which it uses to generate new information out of the already existing knowledge. This approach is called reasoning, which accepts that information is not always complete and that there is not a central instance which controls all knowledge. The goal remains to model the real world as complete as possible, but also to take into account that there might be always missing information. However, it is important to have a language (and logic) to describe a great complexity of properties and objects and to add additional vocabularies where needed. The most important standards to fulfill this task are the eXtensible Markup Language (XML)⁶ and the Resource Description Framework (RDF)⁷. XML allows the user to define her own structure for her documents and RDF allows to express meaning which is described by triples (subject – predicate – object) of elementary sentences. All resources are identified by a Universal Resource Identifier (URI).

⁶<http://www.w3.org/XML/>

⁷<http://www.w3.org/RDF/>

A very important part of the Semantic Web world are ontologies. This is because information about resources and their relations are modeled and stored in ontologies. Since more than one ontology can be used by an application simultaneously, ontology mapping needs to be implemented. Ontology mapping is a method to combine multiple ontologies by finding out which common definitions they have (e.g. same classes with different names). Moreover, ontologies consist of a taxonomy and inference rules. Ontologies can therefore improve web searching by helping to retrieve pages which refer to a precise concept and not pages with ambiguous keywords.

While this sounds very powerful so far, it must be noticed that only implementations of programs which collect content from different kinds of sources, process information, and exchange results with other software can reveal the Semantic Web's real power. Such software programs are called agents. As the Semantic Web grows, more and more agents are being implemented and the challenge is to guarantee that agents which are not designed to work together can communicate from scratch. This is only possible if a unified language is used. This language is currently on the one hand the Resource Description Framework Schema (RDFS)⁸, and on the other hand the Web Ontology Language (OWL)⁹ (see next section).

The extensibility of the Semantic Web by simply adding new URIs for concepts makes it easy for users to create new concepts. The concepts are enabled by a unifying logical language and progressively linked into the Web. The structure opens up the knowledge, whose meaning can be analyzed by software agents.

2.2.2 Basic Standards

In the following, basic standards of the Semantic Web are explained. These standards are well known in the Semantic Web community and recommendations of the World Wide Web Consortium. They are the basis for advanced Semantic Web methodologies such as ontology mapping and reasoning.

Extensible Markup Language (XML)

The Extensible Markup Language¹⁰ describes XML documents, which can be defined by users and contain structured data, and partially the behavior of software for processing XML documents. XML is a subset of its prede-

⁸<http://www.w3.org/TR/rdf-schema/>

⁹<http://www.w3.org/TR/owl-features/>

¹⁰<http://www.w3.org/TR/2008/REC-xml-20081126/>

cessor Standard Generalized Markup Language (SGML)¹¹. Therefore, XML documents are also valid SGML documents.

XML documents contain units, which consist of parsed and unparsed data. Parsed data is markup and character data (CDATA). The markup defines the document's layout and logical structure, which can be constrained by XML elements.

XML documents can be read and accessed by XML processors. XML processors, in turn, are used by applications which use XML documents as input and/or output.

XML Schema

XML Schema¹² describes valid formats of XML data-sets¹³. The schema defines the meaning and purpose of XML elements, i.e. it defines which elements are allowed at which place in the document, which attributes may be defined for which elements, how many occurrences of attributes and elements are allowed, etc.

The first requirement on XML documents is to be “well formed”. This means that there must not be any syntactic errors in the document definitions and content. After that, validity checks could be done to find out whether a document is also “valid”. Validation is only possible if there is some schema definition against which documents can be checked. In “well formed” documents there has to be exactly one root element, every sub-element needs to have start and end tags, and they have to be properly nested. A “valid” document is “well formed” and needs to conform to certain production rules.

Resource Description Framework (RDF)

The Resource Description Framework (RDF)¹⁴ is the basic technology to create triples for the Semantic Web [Brickley and Guha, 2000]. Furthermore, it is the standard to exchange information for machines on, but also off, the Web. RDF can be expressed in XML and thus uses this standard to describe resources very often, but the resources themselves are not restricted to any format. Fields of application are discovery of resources, cataloging and archiving, in cooperation with intelligent agents, or simply describing resources and adding machine-readable meta-information. RDF is considered one of the key technologies to build the “Web of Trust” as well. The

¹¹<http://www.w3.org/MarkUp/SGML/>

¹²<http://www.w3.org/XML/Schema>

¹³<http://lucas.ucs.ed.ac.uk/xml-schema/>

¹⁴<http://www.w3.org/RDF/>

admission of RDF as a W3C recommendation has led to a wide acceptance and usage in the Semantic Web community.

Resource Description Framework in attributes (RDFa)

Web sites are interconnected and designed to be human-readable and understandable [Group et al., 2012]. Still most of web page developers use HTML and its elements, which can be interpreted and presented by web browsers. The links are defined in HTML and can be easily recognized by humans or web crawlers, but there is no way to find out where a link leads until it is followed. Therefore, without artificial intelligence techniques there is no way for machines to interpret web sites and reduce labor for human users, e.g. automate the search and filtering process or make the presentation for humans more efficient and compact (increase information density).

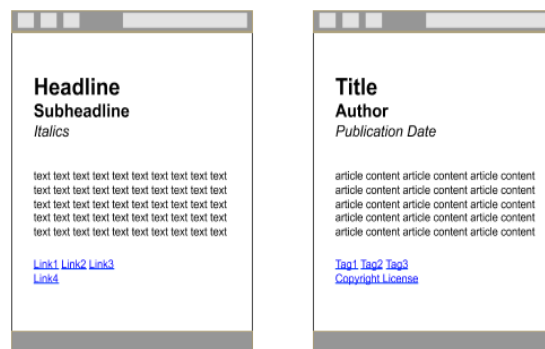


Figure 2.3: Difference of the view on a web document between browsers (left) and humans (right) [Group et al., 2012].

Figure 2.3 shows the difference between the perception of a web site for a browser and a human user. In HTML the browser has information about how the content can be visualized, but is not able to recognize the content itself (which is an easy task for a human user).

If browsers would be in the position to take tags and annotations of web sites into account, they could present information to users more naturally, and could show content from different pages on one page as a summary for users. For example, if a person wants information on a restaurant, she could retrieve the name of the restaurant, the opening hours, the address (including a map), and the menu on only one web site. This would require that the browser is able to interpret all sites it collects information from to be able to generate such a summary for the user.

This is exactly what RDFa¹⁵ targets. Through the possibility to add RDF triples (meta-information) directly in HTML, it allows a semantic web browser to interpret text on web sites. This is possible for simple pages (e.g. people’s business cards), but also for complex sites (e.g. content management systems or social networks).

Resource Description Framework Schema (RDFS)

RDFS is an extension of the Resource Description Framework. It provides classes and properties for the description of ontologies. Its goal is to structure web resources in form of triples, which can be stored in a triple store and retrieved with the query language SPARQL (SPARQL Protocol and RDF Query Language).

Web Ontology Language (OWL)

The acronym for the Web Ontology Language¹⁶ is OWL. This might sound a little bit strange, since the logical acronym would be WOL, but this “joke” could be recognized by Winnie-the-Pooh fans. In this cartoon, the owl writes its name wrong by interchanging O and W. Since the OWL writes WOL, the acronym for the Web Ontology language is OWL instead of WOL. OWL is a W3C recommendation and a popular and widely-used standard in the Semantic Web community [McGuinness and van Harmelen, 2004]. It provides the means (semantics) for defining triples and therefore enables automatic processing and integration of information for machines on the Web. The most commonly used syntax is RDF/XML, which can be easily processed by Web applications due to its XML background. OWL is an extension of RDF and a complete ontology engineering language, which is also used to describe documents on the Web.

In contrast to other standards, OWL fulfills all criteria in order to be a complete Web Ontology Language:

- XML does not cover information about the meaning of a document.
- XML Schema extends XML by the data type concept and forces a certain structure on XML documents.
- RDF enables the definition of triples (subject – predicate – object) and can be used with XML syntax.

¹⁵<http://www.w3.org/TR/xhtml1-rdfa-primer/>

¹⁶<http://www.w3.org/TR/owl-features/>

- RDF Schema describes hierarchies of attributes and classes and therefore extends RDF.
- OWL extends RDF(S) and provides means to describe relations between classes, equality, cardinality, characteristics and stylization of attributes and classes, etc.

In order to reduce complexity for unexperienced users, OWL has three subsets of expressivity: OWL Lite, OWL DL, and OWL Full (OWL Lite \subset OWL DL \subset OWL Full).

SPARQL Protocol And RDF Query Language (SPARQL)

SPARQL¹⁷ is a recursive acronym and stands for SPARQL Protocol and RDF Query Language. It is important to RDF, because of its ability to retrieve RDF data from a knowledge base (RDF triple store). The RDF Data Access Working Group has developed SPARQL and its use cases and requirements [Prud'hommeaux and Seaborne, 2008]. The main task, as defined by the working group, is the generation of queries and retrieval of results (very closely related to SQL). The SPARQL Query Results XML Format Specification defines a SPARQL SELECT and ASK query format to represent results in XML.

Overview

Table 2.3 shows an overview of the most popular basic standards in the Semantic Web area.

2.2.3 Vocabularies

Vocabularies are predefined collections of classes and attributes that provide a basis for building a domain ontology for a certain task. In the following, we describe the most popular vocabularies in the Semantic Web.

Dublin Core (DC)

Dublin Core¹⁸ was first introduced during a workshop of librarians and computer scientists to improve resource discovery [Weibel et al., 1998]. This was continued in the so-called Dublin Core Metadata Workshop Series. Dublin Core's goals are to make creation and maintenance of resources easier, to

¹⁷<http://www.w3.org/TR/rdf-sparql-query/>

¹⁸<http://dublincore.org/>

Name	Category	Purpose
Extensible Markup Language (XML)	Markup language	Used to describe structured data.
XML Schema	Markup definition language	Used to validate XML documents.
Resource Description Framework (RDF)	Resource description language	Used to define triples.
Resource Description Framework in attributes (RDFa)	Resource description language	Used to define triples in HTML.
Resource Description Framework Schema (RDFS)	Resource description language	Used to define triples.
Web Ontology Language (OWL)	Resource description language	Used to define triples.
SPARQL Protocol and RDF Query Language	RDF query language	Used to query a knowledge base.

Table 2.3: Overview of basic standards.

understand the semantics, to conform to standards, to enable extensibility, and to improve interoperability. The Dublin Core vocabulary is maintained by the Dublin Core Metadata Initiative.

Friend of a Friend (FOAF) Vocabulary

The FOAF¹⁹ vocabulary aims to describe the world by using simple web based ideas [Brickley and Miller, 2010]. Descriptions in FOAF are categorized into properties, which are classes and links. Therefore, FOAF is a dictionary of terms, where every term is a class or property. The idea is that other projects provide other classes and other properties, but to link them to FOAF, so that the modeled world grows a bit every time a new vocabulary is linked. FOAF is published on the web as RDF/XML or RDFa. It can be imported into ontologies or web sites to use its classes and properties. In fact, FOAF tries to build a network of people with published FOAF profiles on various web sites and in various ontologies. There is no guarantee that such profiles are not falsified, but the probability that the information is useful is quite high and the ability to merge documents and combine descriptions is an interesting approach. The FOAF vocabulary describes people, groups,

¹⁹<http://www.foaf-project.org/>

documents, etc.

Simple Knowledge Organization System (SKOS)

SKOS's²⁰ goal is to provide a framework which can improve machine readability, i.e. format knowledge to a structure that can be understood by machines [Miles et al., 2005]. SKOS is a standard and vocabulary with RDF(S) attributes and classes, which can be used to define knowledge structure and generate RDF graphs. It supports different kinds of Knowledge Organization Systems (KOSs), these are vocabularies, taxonomies, glossaries, schemas, etc.

Semantically Interlinked Online Community (SIOC)

SIOC²¹ tries to improve adoption by community sites and usage with existing ontologies [Breslin et al., 2006]. The challenge is to reach a critical mass of users for the SIOC ontology. This could be done by automatically created properties and easily understandable concepts. Another challenge is to get users for the ontology by mappings and interfaces to other widely used ontologies (e.g. DC, FOAF, etc.). Finally, there are open questions like scalability, update of topics, etc.

Overview

Table 2.4 presents an overview of the most popular vocabularies in the Semantic Web.

Name	Category	Purpose
Dublin Core (DC)	Vocabulary	Description of resources
Friend of a Friend Vocabulary (FOAF)	Vocabulary	Description of friends and communities.
Simple Knowledge Organization System (SKOS)	Vocabulary	Define knowledge structure.
Semantically Interlinked Online Community (SIOC)	Vocabulary	Used for community sites.

Table 2.4: Overview of vocabularies.

²⁰<http://www.w3.org/2004/02/skos/>

²¹<http://sioc-project.org/>

2.2.4 Languages

The languages subsection describes some popular languages from different fields of application such as markup, resource description, knowledge representation, and query.

Darpa Agent Markup Language (DAML) + Ontology Interface Layer (OIL)

DAML+OIL²² is the predecessor of RDF and also a semantic markup language [van Harmelen et al., 2001]. It is an early W3C standard, before RDF(S) extended the standard with a richer syntax and more possibilities. DAML+OIL is based on description logics and includes modeling primitives like frame-based languages.

Gleaning Resource Descriptions from Dialects of Languages (GRDDL)

XML documents are written in many different languages (“dialects”), e.g. XHTML, XML, or RDF [Connolly et al., 2007]. GRDDL²³ is used to describe resources from dialects of languages. It is based on existing standards (XML and RDF) and can be used to extract data from documents. It allows that documents, defined with their namespaces, include gleanable data. Roughly speaking, the goal is to understand dialects across different domains.

Knowledge Interchange Format (KIF)

Knowledge Interchange Format (KIF)²⁴ is a language for interchange of knowledge between different programs [Genesereth et al., 1992]. It is not intended for interaction with humans, but for different programs, which can interact with users and other applications. The idea behind it is, that when a program reads a knowledge base, the information should be represented according to the problem (i.e. it should be integrated in the GUI of the program). KIF is then the interchange format between different programs, which means that a program deals with knowledge and as soon as it wants to interoperate with another program, it has to convert the data into KIF. Everything else is done program-internally.

KIF can be compared to Postscript, which is used by text and graphics programs for interprocess communication, but also for communication with printers. It is a programmer readable representation and can be understood

²²<http://www.w3.org/TR/daml+oil-reference>

²³<http://www.w3.org/2004/01/rdxh/spec>

²⁴<http://logic.stanford.edu/kif/>

by programs and printers. KIF is not as efficient as specialized knowledge representation languages, but it is readable by programmers.

Open Knowledge Base Connectivity (OKBC)

Open Knowledge Base Connectivity (OKBC)²⁵ is a protocol which provides an interface for knowledge representation systems [Chaudhri et al., 1998]. It is layer oriented and enables development of generic tools that operate on top of knowledge representation systems. Implementations are based on many common programming languages, such as Java, C, and Common Lisp. Knowledge bases can be accessed locally or over the network. OKBC extends languages which support knowledge sharing and focuses on knowledge representation system operations.

Sesame RDF Query Language (SeRQL)

SeRQL²⁶ is the implementation of requirements from RDF implementers. It was developed by the SESAME project with the goal to unite the advantages of already existing RDF query languages [Broekstra and Kampman, 2003]. It is designed to go beyond experimental query languages, lay the focus on the design of the language, and take into account syntactic and semantic specifications. However, SeRQL is not the most popular RDF query language. SPARQL is by far better known and more widely used.

Semantic Web Rule Language (SWRL)

The Semantic Web Rule Language (SWRL²⁷) is a combination of OWL DL and OWL Lite, but it contains elements of the Rule Markup language as well [Horrocks et al., 2004]. Since it contains Horn-like rules, it makes the combination of Horn-like rules and the OWL knowledge base possible. Its syntax extends the OWL syntax and the OWL model-theoretic semantics. The rules which it proposes are body and head (while both consist of at least one atom) concepts, where rules that hold for the body also must hold for the head. Empty statements are treated as true. Atoms can be OWL descriptions, properties, and variables, individuals, or data values. SWRL supports an XML syntax (like most other languages in the Semantic Web), or more precisely OWL RDF/XML syntax.

²⁵<http://www.ai.sri.com/~okbc/>

²⁶<http://www.w3.org/Submission/SWSF-SWSL/>

²⁷<http://www.w3.org/Submission/SWRL/>

Name	Category	Purpose
Darpa Agent Markup Language Ontology Interface Layer (DAML+OIL)	Markup language	Description of ontologies.
Gleaning Resource Descriptions from Dialects of Languages (GRDDL)	Resource description language	Extraction of data from documents.
Knowledge Interchange Format (KIF)	Knowledge language	Language for interchange of knowledge.
Open Knowledge Base Connectivity (OKBC)	Knowledge representation language	Provides interface for knowledge representation systems.
Sesame RDF Query Language (SeRQL)	RDF query language	Used to query a knowledge base.
Semantic Web Rule Language (SWRL)	Web rule language	Definition of ontologies.

Table 2.5: Overview of languages.

Overview

Table 2.5 presents an overview of the Semantic Web languages introduced in this subsection.

2.2.5 Web Service Standards

Web Services are pieces of functionality which are accessible over the Web [De Bruijn et al., 2006b]. Current technologies such as the Web Service Definition Language (WSDL)²⁸ allow to describe the functionality offered by a Web Service on a syntactic level. Semantic descriptions of Web Services are required to automate of tasks, such as discovery, composition, and execution. Since Semantic Web technology enables this formal description of Web content, the combination of Semantic Web with Web Services is the natural next step to be taken. The web service standards subsection describes standards in the Semantic Web that are directly related to the modeling of web services.

²⁸<http://www.w3.org/TR/wsdl>

Web Service Modeling Ontology (WSMO)

Web Service Modeling Ontology (WSMO)²⁹ is an ontology for describing aspects of Semantic Web services [De Bruijn et al., 2006a]. WSMO is based on the Web Service Modeling Framework (WSMF), which consists of ontologies, web service goals, web service descriptions, and mediators. The Web Service Modeling Ontology provides specifications for semantic web services, which try to combine Semantic Web and web service technology in order to reach the possibility of machine-understandability on the web. Such services need to follow special design principles:

- *Web compliance*: Usage of URI and namespaces, support of XML and other W3C recommendations.
- *Ontology-basis*: Usage of ontologies for resource description and data interchange.
- *Decouplement*: Isolated definition of resources.
- *Mediation-centralit*y: Handling of heterogeneities.
- *Role separation*: Different contexts for different users.
- *Description and implementation*: Separation of service description and implementation.

The combination of Semantic Web technologies and web services is often referred to as Semantic Web Services. In this context, the Web Service Modeling Ontology WSMO provides a conceptual model for the description of various aspects of services towards such Semantic Web Services (SWS).

Web Service Modeling Language (WSML)

The Web Service Modeling Language (WSML)³⁰ provides formal syntax and semantics for the Web Service Modeling Ontology WSMO. It is based on logical formalisms, such as description logics, first-order logic, and logic programming, in order to model so-called Semantic Web Services. The language has different variants depending on logic formalisms which are: Core, DL, Flight, Rule, and Full. WSML-Core is an intersection of description logics and Horn logic. All other variants increase expressiveness towards description logics and logic programming. Finally, WSML-Full unifies both and is the most expressive WSML variant.

²⁹<http://www.w3.org/Submission/WSMO/>

³⁰<http://www.w3.org/Submission/WSML/>

WSML supports human-readable syntax, XML and RDF syntax; and thus supports interoperability with Web and Semantic Web applications. Furthermore, WSML provides mappings between WSML ontologies and OWL ontologies. Therefore, it is also interoperable with OWL ontologies through a common subset of OWL and WSML.

Semantic Web Services Language (SWSL)

SWSL³¹ is based on logic and used to specify semantic web services [Battle et al., 2005]. It is a W3C recommendation and includes SWSL-FOL and SWSL-Rules, two sublanguages which are first order logic (FOL) and rule-based (Rules). FOL specifies the web service ontology and Rules can be used for specification or implementation. The Semantic Web Service Ontology defines domain dependent constructs, which makes SWSL itself independent.

SWSL-FOL as a first-order logic language appears to be more suitable for specifying ontologies, and SWSL-Rules can be better used for programming. However, both languages can be used together, SWSL allows to use SWSL-FOL in SWSL-Rules or vice-versa, since there is a bridge for both of them written in the other language.

Semantic Annotations for WSDL (SAWSDL)

Semantic Annotations for WSDL (SAWSDL)³² is a standard, which defines the addition of annotations to WSDL documents (e.g. operations) [Farrell and Lausen, 2007]. Its attributes are compatible with the WSDL 2.0 extensibility framework. SAWSDL defines how WSDL interfaces and operations can be annotated and then improves the discovery of a web service in a registry. Therefore, annotations can be used during discovery and composition. SAWSDL attributes can be applied to WSDL elements, but also to XML Schema elements. The annotations refer from a WSDL or XML Schema to an ontology. These annotations do not depend on the language of the ontology or other mapping languages.

Overview

Table 2.6 provides an overview of standards from the Semantic Web domain which are relevant for web service development.

³¹<http://www.w3.org/Submission/SWSF-SWSL/>

³²<http://www.w3.org/2002/ws/sawSDL/>

Name	Category	Purpose
Semantic Web Services Language	Web service language	Specification of semantic web services.
Web Service Modeling Language	Modeling language	Language for web service modeling.
Web Service Modeling Ontology	Ontology	Ontology for web service modeling.
Semantic Annotations for WSDL	Annotation language	Adds annotations to web services.

Table 2.6: Overview of web service standards.

2.2.6 Other Standards

In the following, we describe other standards which are important for the Semantic Web community but do not fit in one of our previously introduced categories.

METHONTOLOGY

METHONTOLOGY³³ is a methodology for ontology engineering from scratch or by the reuse of other ontologies [Fernández-López et al., 1997]. The ontologies are constructed at the knowledge level. The contents of the framework are identification of ontology development process, identification of main activities, evolving prototypes, the methodology for performing activities, the outcomes, and the evaluation. These are the steps which need to be performed in order to develop an ontology.

OpenSearch

OpenSearch³⁴ can be used to learn about the public search engine interface [Clinton et al.]. It provides description documents with URL templates that indicate how requests should be made. Common search engines can use this information to add meta-data to their search results in many different content formats. In general, formats can be used to improve the discovery and usage of search engines and to syndicate search results.

³³<http://semanticweb.org/wiki/METHONTOLOGY>

³⁴<http://www.opensearch.org/Home>

Rule Interchange Format (RIF)

The Rule Interchange Format (RIF)³⁵ is a standard for rule exchange between systems in the Semantic Web defined by a working group of the W3C [Kifer, 2008]. This is a difficult task, since already developed rule systems are very diverse, especially regarding their syntax and semantics. Some of them extend each other, other systems are incompatible, etc. This fact makes it hard to achieve interoperability. The idea is to develop dialects with similar syntax and semantics, to prevent isolated applications. Dialects could be dynamically added if there is enough demand. In fact, other systems should be able to map their languages to RIF (as dialects) most easily, and interoperate by finding a common dialect. The format of RIF is XML.

Overview

Other relevant standards in the Semantic Web area are summarized in Table 2.7.

Name	Category	Purpose
METHONTOLOGY	Ontology engineering	Used to develop ontologies.
Open Search	Search engine interface	Improve search engine discovery.
Rule Interchange Format	Rule exchange language	Interchange of rules and dialects.

Table 2.7: Overview of other standards.

2.2.7 Semantic Web Frameworks

This subsection introduces the most important general frameworks in the Semantic Web area.

Jena API

Jena³⁶, as an RDF platform, has support for ontologies which is restricted to formalisms based on RDF [Apache Software Foundation, 2010]. In particular, this means RDF(S), OWL, and DAML+OIL. In general, Jena provides the

³⁵http://www.w3.org/standards/techs/rif#w3c_all

³⁶<http://jena.apache.org/>

tools for building Semantic Web applications. The main components of the framework are:

- API for reading and writing of RDF data in different formats (XML, N-triples, and Turtle),
- ontology API for OWL and RDFS ontologies,
- rule-based inference engine for reasoning in RDF and OWL,
- stores for large numbers of RDF triples,
- query engine for processing SPARQL queries.

SESAME

SESAME³⁷ is an architecture for efficient storage and expressive querying of large quantities of RDF meta-data [Fensel et al., 2005]. It can be used for parsing, storing, inferencing, and querying of RDF data. SESAME can be deployed on top of relational databases, in-memory, file systems, and keyword indexers, and provides full support of the SPARQL query language and the most popular RDF file formats such as RDF/XML, Turtle, N-Triples, etc.

OntoStudio

OntoStudio³⁸ is an environment for the development of semantic solutions and a platform to create knowledge engineering tools [Weiten, 2009]. It includes a number of functionalities, such as reasoning, text mining, ontology learning, and management of meta-data. OntoStudio provides design time support, which means that it concentrates on keeping the time to develop an ontology at a minimum. It includes additional functionality such as semantic wrapping and integrating existing data sources.

RDF2Go

RDF2Go³⁹ is a Java library which provides abstract data access methods to RDF triples stored in a triple store [Schied et al., 2010]. It uses common adapter classes to access different triple stores. At the moment, RDF2Go delivers adapter classes for Jena and Sesame and can easily be extended to other triple stores. Communication between SMW (Semantic Media Wiki)

³⁷<http://www.openrdf.org/>

³⁸<http://www.semafora-systems.com/en/products/ontostudio/>

³⁹<http://semanticweb.org/wiki/RDF2Go>

and the triple store connector is done via SPARQL and the SPARUL extension. Initial Loading of RDF data from SMW into the triple store is triggered with a SPARUL LOAD command on part of SMW. The connector handles this event by reading the semantic data directly from SMW's database tables due to performance reasons and a missing function for retrieving the wiki's semantic data as a whole via HTTP.

Overview

Table 2.8 shows an overview of the Semantic Web Frameworks discussed in this subsection.

Name	Category	Purpose
Jena API	Semantic Web library	Reasoning and knowledge base.
SESAME	RDF framework	RDF storage and querying.
OntoStudio	Semantic development platform	Reasoning, text mining, and ontology learning.
RDF2Go	RDF library	Access to RDF triple store.

Table 2.8: Overview of Semantic Web frameworks.

2.2.8 Ontology Editors

Ontologies are definitely one of the most important parts in Semantic Web technologies, and enable semantic interoperability and data integration [Haase et al., 2008]. Nowadays, they are produced in large numbers and have great complexity (large RDF and OWL files). Ontologies capture domain knowledge and provide an understanding of the domain for reuse and interoperability [Sure et al., 2002]. However, it is difficult to build ontologies, because they are still very complex and formal, and require that the developer possesses a lot of expert knowledge in ontology engineering. The key point is to enable reuse of ontologies in such a way that it is not necessary to reinvent the wheel. In the following, we describe the most widely used ontology editors.

Protégé

Protégé⁴⁰ is a free, open-source platform with a growing user community and a collection of tools for the creation of domain models and knowledge-based

⁴⁰<http://protege.stanford.edu/>

applications with ontologies [Stanford Center for Biomedical Informatics Research, 2010]. As core functionality, Protégé implements a rich collection of knowledge-modeling structures and actions, which support the visualization and manipulation of ontologies in different formats. Protégé can be customized to provide domain-friendly support for generation of knowledge models and data input. Furthermore, it can be enhanced by plug-ins and a Java-based API and used to create knowledge-based tools and applications.

The Protégé platform supports two main kinds of ontology modeling:

- The Protégé Frames Editor makes it possible for a user to create and to fill ontologies, which are frame-based and support the Open Knowledge Base Connectivity (OKBC) protocol. In this model, an ontology consists of a set of classes, which are organized in a hierarchy to represent concepts of a domain; a set of slots, which are associated to classes and describe their attributes and relations; and a set of instances of those classes.
- The Protégé OWL Editor enables users to create ontologies using the Web Ontology Language (OWL). An OWL ontology can contain descriptions of classes, attributes, and instances. With such an ontology, the formal OWL semantics specifies how its logical consequences can be derived. This can be based on one single document or multiple, distributed documents, which are combined through OWL mechanisms.

NeOn Toolkit

The NeOn toolkit⁴¹ is a state-of-the-art integrated modeling environment for ontology development and modeling. It supports network ontology management and collaboration during the ontology engineering process as well as engineering of contextualized semantic applications. Modern semantic applications are open and ontologies are not large and monolithic, but small and dynamic, importing a lot of already popular and well-known general purpose ontologies. The NeOn Toolkit can be used to produce distributed content and provides methods for managing heterogeneous content.

OntoEdit

OntoEdit⁴² as an ontology engineering environment, attempts to solve the problem of requiring expert knowledge for ontology engineering by combining ontology development with collaboration and inferencing. The focus is

⁴¹http://neon-toolkit.org/wiki/Main_Page

⁴²<http://www.semafora-systems.com/en/products/>

on three main steps in ontology development: specification, refinement, and evaluation. First, all requirements are collected, then the description is refined in iteration cycles and formalized to a representation language, and finally the resulting ontology needs to be evaluated according to the specification. OntoEdit supports the user in all of these phases.

OntoGen

OntoGen⁴³ is an ontology editor, which is semi-automatic and data-driven [Fortuna et al., 2007]. It focuses on editing topic ontologies and combines text mining techniques with an efficient user interface, which enables the user to spend less time for her workflow and work more efficiently.

The system is able to suggest concepts, relations, and names as well as to automatically assign instances and visualize them. However, the user can still control the system and make modifications to the ontology at any time. The data represents the domain for which the ontology is made. OntoGen supports automatic extraction of instances and co-occurrences from the data.

SMORE

SMORE (Semantic Markup, Ontology and RDF Editor)⁴⁴ is a tool which tries to make knowledge engineering look like web page authoring [Kalyanpur et al., 2006a]. It blurs the line between content creation and semantic annotation, but also supports ontology use, modification, combination, and extension. SMORE provides built-in support for semantic markup (e.g. a WYSIWYG editor) and enables the user to create and deploy web pages. Furthermore, it is able to generate semantic markup, to let users compose triples (in this case triples are composed of plain natural language), and to link ontological elements and user-defined terms.

Controlled Language for Ontology Editing (CLOnE)

Using CLOnE, users can design, create, and manage information spaces and do not need to have much knowledge about XML, RDF, OWL, or other standards [Funk et al., 2007]. CLOnE's goal, as a natural language processor, is to use normal language to specify logical data or semantic knowledge. It is based on GATE's (see Subsection 2.2.24) information extraction and natural language processing, and accepts user input as valid or warns her of errors.

⁴³<http://ontogen.ijs.si/>

⁴⁴<http://www.mindswap.org/2005/SMORE>

The parsing is deterministic and, therefore, usual performance measures are not relevant.

Overview

Table 2.9 presents an overview of the presented ontology editors.

Name	Category	Purpose
Protégé	Ontology editor	Ontology development.
NeOn Toolkit	Ontology editor	Ontology development.
OntoEdit	Ontology editor	Ontology development.
OntoGen	Ontology editor	Ontology development.
SMORE	Ontology editor	Creation of ontologies.
CLOnE	Ontology language	Use natural language for logical data.

Table 2.9: Overview of ontology editors.

2.2.9 Ontology Mapping Tools

Semantic mapping between two or more ontologies is needed in order to make agents and services interoperable [Ehrig and Staab, 2004]. A number of different approaches and methods for ontology mapping have already been developed. The main difference between the methods is their effectiveness and efficiency. The clue is that ontologies become more and more compact and dynamic (they include a lot of already existing definitions), but there is no effective ontology mapping method, which is specialized for such a situation. Important ontology mapping, merging, and management tools are described in the following.

OntoMap

OntoMap⁴⁵ is a web site, which provides access to ontologies and mappings between them [Kiryakov et al., 2001]. It facilitates easy access, understanding, and reuse of resources, and captures most of the semantics from upper-level models. Currently, OntoMap supports online browsing and export into DAML+OIL. Important contents of the OntoMap framework are upper-level ontologies, knowledge representation, concept mapping, and services.

⁴⁵<http://www.ontotext.com/research/ontomap>

OntoMerge

OntoMerge⁴⁶ answers subclass queries on merged results effectively [Wang et al.]. It can answer queries on multiple ontologies and uses ontology merging for ontology translation. The merge takes place by taking the union of axioms of the ontologies. To avoid ambiguities, XML namespaces are used. Also, bridging axioms are added for relations of terms between ontologies. Demand-driven or data-driven inference can be performed, which means that forward- and backward-chaining is supported.

PROMPT

The PROMPT⁴⁷ framework consists of a number of tools like iPrompt (for ontology merging), AnchorPrompt (for ontology alignment), PromptDiff (for ontology versioning), PromptFactor (for factoring out complete sub-ontologies), etc. [Noy and Musen, 2003]. These are all tools for multiple-ontology management and provide a uniform user interface. The advantages are the following: uniform view on tasks in multiple-ontology management, one framework for all tasks, interactive ontology merging, and correlation discovery between concepts from different ontologies.

Quick Ontology Mapping (QOM)

QOM is a method, which takes efficiency over effectiveness, and therefore, addresses exactly the point where ontology mapping tools and approaches differ. Where other tools try to obtain the best possible results, QOM has the goal to get results as quickly as possible. The loss of quality is marginal most of the time but the gain in performance can be quite notable.

Overview

In Table 2.10 the introduced ontology mapping, merging, and management tools are summarized.

2.2.10 Reasoners

This subsection presents some popular reasoning tools and frameworks.

⁴⁶<http://cs-www.cs.yale.edu/homes/dvm/daml/ontology-translation.html>

⁴⁷<http://protege.stanford.edu/plugins/prompt/prompt.html>

Name	Category	Purpose
OntoMap	Ontology mapping tool	Mapping of domain ontologies.
OntoMerge	Query tool	Answering of queries on multiple ontologies.
PROMPT	Ontology management	Merging, alignment, versioning, and factoring.
Quick Ontology Mapping	Ontology mapping	Efficient mapping of ontologies.

Table 2.10: Overview of ontology mapping tools.

OWLIM

OWLIM⁴⁸ is a storage and reasoning infrastructure for ontologies and structured meta-data [Kiryakov et al., 2005]. It supports a part of OWL, which is nearly as large as OWL DLP⁴⁹. OWLIM has native RDF engines and is implemented in Java. It uses SESAME and Jena, supports RDF(S), OWL 2 RL⁵⁰, and query evaluation. A large number of research projects use OWLIM, since it is available under a variety of licenses (OWLIM-Lite, OWLIM-SE, and OWLIM-Enterprise).

OntoBroker

OntoBroker⁵¹ has the following core elements [Decker et al., 1999]:

- Ontologies are the most central part.
- Ontocrawler extracts knowledge from web sites.
- The inference engine exploits formal semantics and enables automatic reasoning.
- RDF-Maker exploits the inference engine and generates an RDF representation.
- The query interface enables interactive formulation of queries.

OntoBroker is a system for extraction, reasoning and generation of meta-data. The meta-data and ontologies are domain specific and depend on

⁴⁸<http://www.ontotext.com/owlim>

⁴⁹<http://www.ontotext.com/rdfs-rules-owl>

⁵⁰http://www.w3.org/TR/owl2-profiles/#OWL_2_RL

⁵¹<http://www.semafora-systems.com/en/products/ontobroker/>

specific content. The reasoning service derives information which is specified in web resources.

Overview

Table 2.11 shows an overview of the presented reasoners.

Name	Category	Purpose
OWLIM	Reasoner & KB	Reasoning and knowledge archiving.
OntoBroker	RDF tool	RDF extraction, reasoning, and generation

Table 2.11: Overview of reasoners.

2.2.11 Browsers

The following subsection presents different browsers for various Semantic Web purposes.

IsaViz

IsaViz⁵² is a visual browsing tool for RDF models [Telea et al., 2003]. It uses Jena's RDF API for reading RDF models and the GraphViz package for graph layout. Some features are text-based search, copy and paste, model editing, nodes and arcs editing, textual property browsing, and graph views. Therefore, IsaViz is a state-of-the-art RDF model browsing tool. The drawback is, that it is not easily extensible due to its rigid architecture.

Magpie

Web browsing consists of two tasks: to find the right website and to derive meaning from the content [Dzbor et al., 2003]. Until now, research focused on information retrieval or semantic-enabled mechanisms for supporting the search for web resources. Much less attention has been paid to the second problem. Therefore, Magpie⁵³, which supports the interpretation of websites, was developed.

Magpie provides aligned sources of knowledge, which deliver fast access to important background information about a web resource. It associates an

⁵²<http://www.w3.org/2001/11/IsaViz/>

⁵³<http://projects.kmi.open.ac.uk/magpie/main.html>

ontology-based, semantic layer automatically to a web resource and allows relevant services to be called in a standard browser. Hence, Magpie can be seen as a first step towards a semantic web browser.

Swoop

Swoop⁵⁴ is a web ontology browser and editor and can be used for OWL ontologies [Kalyanpur et al., 2006b]. The UI for Swoop is a browser, which most users are familiar with and are therefore able to use for ontology creation. Swoop supports multiple ontologies, various OWL representations, reasoning, and open-world semantics. Furthermore, it can perform collaborative annotation, data markup, ontology refactoring, and debugging. Finally, the accessibility of Swoop to newbies and expert users makes it interesting for the design of robust ontology modeling and analysis.

Overview

Table 2.12 summarizes the three presented browsers.

Name	Category	Purpose
IsaViz	RDF browsing tool	Graphical visualization of RDF.
Magpie	Semantic web browser	Background information about web resources.
Swoop	Web ontology browser	Creation of ontologies.

Table 2.12: Overview of browsers.

2.2.12 Annotation Tools

This subsection discusses two popular annotation tools.

S-CreaM

S-CreaM stands for Semi-automatic Creation of Metadata and is an annotation framework which aligns conceptual markup with semantic and indicative tagging [Handsuh et al., 2002]. The framework has the following purposes:

1. Solve the underspecification of automatically produced tagging, which can be done by discourse representation.

⁵⁴<http://code.google.com/p/swoop/>

2. Turn semantic tags into conceptual markup.
3. Produce indicative tags for exploitation within a learning cycle.

SemTag

SemTag performs automatic tagging of large corpora [Dill et al., 2003]. It can be applied to millions of web pages and then automatically generates disambiguated tags, which are published to the web. As a very large semantic tagging platform, it demonstrates the viability of bootstrapping a web scale semantic network and resolves ambiguities in a natural language corpus.

Overview

Table 2.13 shows an overview of the two popular annotation tools.

Name	Category	Purpose
S-CreaM	Annotation framework	Automatic tag generation.
SemTag	Annotation framework	Automatic tag generation.

Table 2.13: Overview of annotation tools.

2.2.13 Other Tools

All other remaining, popular tools which are relevant for the Semantic Web area are listed in the following.

AeroDAML

AeroDAML is a prototype tool for experimentation with producer side NLP (Natural Language Processing) applications [Kogut and Holmes, 2001]. It is a knowledge markup tool, which applies NLP extraction techniques and generates DAML annotations from web sites automatically. Furthermore, AeroDAML links nouns and relationships with classes and properties in DAML ontologies. In AeroDAML the weakness of DAML, that annotation of web pages and other resources takes a lot of time, is fixed. The idea is to move the annotation task from the consumer to the producer site and therefore increase the motivation to provide annotations, since resource producers are highly interested in an improved discovery of their resources.

CS AKTive Space

CS AKTive Space⁵⁵ is a Semantic Web application which has geographically distributed content, diverse ownership, syntactic and semantic heterogeneity, and real-world data [Shadbolt et al., 2004]. It provides an information overview of who is doing what and where in UK university-based computer science research, and helps users to get a quick overview of the information space. When a user has found an information space, she has the possibility to find people's locations and to contact them. This is especially helpful when, e.g. organizing workshops. Furthermore, the application fulfills the following tasks: acquiring content, developing scalable RDF storage, developing ontologies to mediate heterogeneous data sources, querying facilities, facilitating knowledge-processing services over the harvested content, and semantically directing interaction design.

Delicious

Delicious⁵⁶ is a social bookmarking system, which recently gained a lot of interest in academia and the rest of the world [Wetzker et al., 2008]. It has already a large number of users and derives its success from the ability to centrally store user bookmarks, which the users find somewhere on the web. This also makes the improvement of services like trend indication, advanced web search, and automated recommendations possible. Also, a great amount of user annotations can be gained by such systems to improve the semantic part of the web. The process as such is called collaborative tagging. Delicious, due to its early acceptance by users, can help to analyze the users' behavior regarding bookmarking.

Esperanto

The Esperanto⁵⁷ project has the goal to create a bridge between the Semantic Web and the World Wide Web [Benjamins et al., 2003]. While the current web is created for human users and therefore can only be interpreted by humans (HTML is only for the design and presentation of web content), the Semantic Web claims to be machine-readable and understandable. A lot of resources, pages, and information is out there in the Web. The problem arises when this information needs to be processed by computers automatically. Therefore, Esperanto provides tools and techniques for publishing on the Semantic Web. This way static and dynamic web pages, but also multimedia

⁵⁵<http://www.aktors.org/technologies/csaktivespace/>

⁵⁶<https://delicious.com/>

⁵⁷<http://ontoweb-lt.dfki.de/projects/esperanto.htm>

and multilingual content can be considered. To show how this could work, Esperanto implements showcases, which can be used as a proof-of-concept.

flickr

Flickr⁵⁸ is one of many tagging applications, which have been rapidly growing during the recent few years [Schmitz, 2006]. More and more users are participating in tagging applications and communities. Two issues are currently open for tagging systems: The first one is that their user interfaces are bad when the vocabulary is hierarchic and closed; the second one is that strict tree concepts do not reflect usage and intent. The first issue can be fixed by dynamic ontologies and better user interfaces. The second issue is caused by models which force users too much in one direction or the other, but can be solved by using faceted ontologies.

Flickr addresses both issues and enables the user to share and most importantly tag pictures with dynamic ontologies. It also supports tag-based search, refined search, clustering performance, open vocabularies, etc.

GATE

The General Architecture for Text Engineering (GATE)⁵⁹ has the following goals [Cunningham et al., 1996]:

- Support for information transfer between Language Engineering (LE) modules on the highest possible level.
- Support for integration of modules, which have been developed using an arbitrary language and on widely-used platforms.
- Support for evaluation and refinement of LE modules, and of systems, which build upon them having a unified, easy to use graphical interface, which provides data visualization and corpora management.

According to these goals, GATE provides three main elements to fulfill the tasks: GDM, the GATE Document Manager, which is based on the TRIPSTER Document Manager; CREOLE (Collection of Reusable Object for Language Engineering), a collection of LE modules, which are integrated into the system; and GGI (GATE Graphical Interface), a development tool for LE R & D, which provides an integrated access to services of other components; but also visualization and debugging tools.

⁵⁸<http://www.flickr.com/>

⁵⁹<http://gate.ac.uk/>

Haystack

Haystack is a user interface for semantic web applications which lets users interact with RDF [Quan et al., 2003]. Users are able to manage their documents, emails, appointments, tasks, etc. The user interface hides RDF internally from the user and shows her a variety of different types of information. The idea is to acquire users which are not familiar with ontology development and description logics. Haystack is built as an extensible platform that allows new functionality to be easily integrated.

MediaWiki

The technology and idea of wikis is very popular nowadays. They are used to collaborate and exchange knowledge [Krötzsch et al., 2006]. One important goal is to organize collected knowledge and share information. MediaWiki⁶⁰ leverages semantic technologies and therefore improves knowledge and information access. MediaWiki is also the extension of traditional wiki syntax by means of the Semantic Web.

Ontolingua

Ontolingua⁶¹ makes it possible to write ontologies in a canonical format with the goal to facilitate the translation into other representations and reasoning systems [Gruber, 1992]. This makes maintenance of the ontologies easier, because the system takes care that it is translated into different required formats. In general, syntax and semantics are based on KIF (Knowledge Interchange Format), which is therefore the standard Ontolingua format. Ontolingua extends KIF with standard primitives for defining classes and relations, organizing knowledge, and creating object-oriented hierarchies. Currently, besides KIF, Ontolingua translates into LOOM, Epikit, and Algernon.

QuizRDF

QuizRDF is an information-seeking system which combines browsing and querying RDF annotations with traditional keyword querying [Davies and Weeks, 2004]. It uses RDF(S) to specify and populate ontologies and RDF annotations, which are indexed with the text of annotated resources. The result allows full-text search and RDF queries. QuizRDF enables full exploitation of Semantic Web technologies in knowledge management as well

⁶⁰<http://www.mediawiki.org/wiki/MediaWiki>

⁶¹<http://www.ksl.stanford.edu/software/ontolingua/>

as searching and browsing functionality. The idea is to exploit RDF annotations if they are there, but if they are not, still to have a possibility to provide full-text search.

SOA4All

Service-oriented architectures (SOA) have become very popular during the last years [Domingue et al., 2008]. The SOA4All project⁶² has the goal to provide means and methods for implementing large-scale SOA infrastructures. In service-oriented architectures, software building blocks are placed on a base of services. The services are as loosely coupled as possible and publish their interfaces on the Web in order to be discovered by other services. SOAs' advantages are the increase in flexibility, dynamic reaction to changes, usage of Web standards, and better interoperability. The vision is to have a so-called "Service Web" where transparent services can be found for every possible task. Therefore, it is necessary for software engineers to follow SOA principles as closely as possible.

Text2Onto

Text2Onto⁶³ is a redesigned and reengineered version of the TextToOnto system. It is a toolbox for learning from text [Cimiano and Völker, 2005]. The software targets problems which occur in ontology learning: probabilistic ontology models and data driven change discovery. Ontologies provide shared understandings of different domains and are key to semantic-driven modeling. Text2Onto aims to solve the following shortcomings:

- Dependency on specific or proprietary ontology models.
- Interaction with end-users has been neglected.
- Lack of robustness.
- Probabilities for learning structures are needed.

WSMO Studio

WSMO Studio⁶⁴ is an open source integrated modeling environment for semantic web services [Dimitrov et al., 2007]. It is based on the Web Service

⁶²<http://www.soa4all.eu/>

⁶³<http://code.google.com/p/text2onto/>

⁶⁴<http://sourceforge.net/projects/wsmostudio/>

Modeling Ontology (WSMO) and can be used as Eclipse⁶⁵ plugin. The goal is to make semantic web service technology easily usable. The focus should be set on multiple aspects of semantic web service technology (e.g. service composition, ontology management, semantic discovery, etc.).

Overview

Table 2.14 shows an overview of the Semantic Web tools described in this subsection.

Name	Category	Purpose
AeroDAML	Knowledge markup tool	Applying NLP extraction techniques.
CS AKTive Space	Semantic Web application	Showing who does what and where in academia in the UK.
Delicious	Bookmarking system	Central storage for user bookmarks.
Esperanto	Publishing tool	Publishing semantic content on the Web.
flickr	Tagging web application	Publishing and tagging of photos.
GATE	Text engineering	Text engineering technology.
Haystack	User interface	UI for semantic applications.
MediaWiki	Semantic Web wiki	Wiki extension with Semantic Web technology.
Ontolingua	Ontology converter	Translation of ontologies into other formats.
QuizRDF	Information seeking system	Browsing and querying of RDF.
SOA4All	Service-oriented architecture	Enabling service-orientation.
Text2Onto	Text learning toolbox	Ontology learning problems.
WSMO Studio	Integrated modeling environment	Creation of semantic web services.

Table 2.14: Overview of other tools.

⁶⁵<http://www.eclipse.org/>

2.2.14 Literature

Daconta et al. [2003] provide an overview of the future of Web Services and Knowledge Management, but also explain how Semantic Web technologies, like RDF(S), OWL, and SPARQL, build on them, what they exactly are, and what they aim for now and in the future. XML technologies build the basis for the Semantic Web. Almendros-Jiménez [2008] investigates an extension of XQuery for querying RDF documents.

In Hitzler et al. [2008] the focus is set on background knowledge of Semantic Web technologies. The book deals with introductions to RDF(S), OWL, and SPARQL, but also provides in-depth knowledge about formal semantics, predicate logic, extensional semantics, and automated inferencing.

Semantic Web technologies are discussed in more detail in Davies et al. [2000]. The book concentrates on using RDF and OWL combined with XML to describe the content of Web documents. The idea is to build more intelligent software systems for exploitation of information on the Web. Additionally, the application of Semantic Web technologies to knowledge management is explained.

A more specialized description of the link between Semantic Web and Knowledge Management is provided by Davies et al. [2009]. Fundamental research, tools, and applications in Semantic Web and Knowledge Management are introduced, and an insight in current research activities is given. Additionally, a short summary of the most relevant Semantic Web standards and tools can be found in Geist et al. [2011] as well.

As a real-world use case, Golbreich et al. [2006] present a method developed for migrating the Foundational Model of Anatomy (FMA) from its representation with frames to its logical representation. Noy and Rubin [2007]’s Foundational Model of Anatomy (FMA) represents the result of manual and disciplined modeling.

Implementation details for Semantic Web technologies are presented in Segaran et al. [2009]. They describe how to implement a reasoner, a semantic repository, and how to use Semantic Web technologies like RDF, RDFa, OWL, SPARQL, etc. In general, the use of semantic programming techniques is introduced to enrich current Web applications.

In Allemang and Hendler [2008] a resource for ontology developers is provided, which goes beyond the scope of technical standards documents. It provides information to build up practical knowledge in ontology engineering. The book explains ontology development techniques for real-world problems, and provides creative solutions and highly illustrative examples together with detailed instructions for applications in RDF, RDFS, and OWL, among others. Automatic metadata generation may provide a solution to the problem

of inconsistent and unreliable metadata describing resources on the Web, as explained by Jenkins et al. [1999]. The work of Lukasiewicz and Straccia [2008] shows that ontologies play a crucial role in the development of the Semantic Web.

Semantic Web technologies and latest developments are described in Ankolekar et al. [2007]. New tools, technologies and projects are being introduced as well as attempts undertaken to combine the concepts of Semantic Web and Web 2.0. Although this development is very positive and should be acknowledged, as it leads us directly to the next generation of the Web, there is still some place for improvement.

Other developments, such as Tsoft, a graphical interactive analysis software package for analysis and processing of time series [Van Camp and Vauterin, 2004], and a novel web prototype that enables three activities central to the Semantic Web visions, namely organizing, sharing and discovering⁶⁶ [Garcia-Castro et al., 2010], are very interesting extensions of the concepts discussed here.

A very important technology related to the Semantic Web are web services. Gibbins et al. [2004] describe that the web services world consists of loosely-coupled distributed systems, which use service descriptions to adapt to changes. Agarwal et al. [2004] explain that the way web services currently are developed, places them beside rather than within the existing World Wide Web. High quality domain ontologies are essential for successful employment of Semantic Web services, as Sabou et al. [2005] describe.

An important topic for building web services and Semantic Web technologies is security. The results of Shekarpour and Katebi [2010]’s paper are a review and analysis of well known methods of trust modeling and evaluation. Viinikka et al. [2009] found out that the main use of intrusion detection systems (IDS) is to detect attacks against information systems and networks.

In our work, the presented Semantic Web technologies can be used not only to improve time series processing, but also to add community building functionalities and therefore provide means for structuring data in a way that the right information can easily be distributed to the right addressees. The next section presents platforms and approaches which show how the community building task can be realized.

2.3 Community Building

To overcome distance and lack of time for meeting friends and family in person, nowadays people are spending more and more time online to shop, play,

⁶⁶<http://www.scientifik.info/livingdocument>

discuss, or spend time on other kinds of online entertainment [Kim, 2000]. The Web is becoming a village with people gathering around communities with certain common interests. At the same time, this is a great opportunity for social networks and Web community platforms.

2.3.1 Background

[Kim, 2000] found nine strategies, which characterize successful and sustainable communities:

- *Define a purpose:* Understand why and for whom you build a community, then express your vision and design.
- *Build gathering places:* After defining the purpose, gathering places are needed for members to start working together.
- *Create member profiles:* Good member profiles help members to get to know each other. They improve trust, relationships, and enable personalized services.
- *Design roles:* Newcomers need guidance, and more experienced members leadership and possibilities to contribute to the community.
- *Develop a leadership program:* Required to acquire community leaders, who greet, encourage, teach, answer questions, etc. They are one of the most important parts of a community.
- *Encourage etiquette:* Conflicts need to be avoided or at least there has to be a plan how to deal with them. Therefore, basic rules are needed to control the behavior of users.
- *Promote events:* Events are very important for community members to come together on a regular basis. They also help community members to develop stable relationships.
- *Integrate rituals:* Most communities need the possibilities to carry out rituals to celebrate certain events. This functionality makes it possible to set up an online culture.
- *Facilitate subgroups:* For large communities technologies for creating and managing subgroups are required as well. This can improve member loyalty and distinguish the community from its competition.

2.3.2 Existing Platforms

Based on the strategies for building successful communities and social networks presented in the previous subsection, the following list describes some of the most popular platforms currently available.

Facebook

Facebook⁶⁷ and other social network platforms allow their users to present themselves through profiles, create their own social networks, and connect with others [Ellison et al., 2007]. In fact, Facebook was originally intended to connect students of a university to each other (to be more precise, it was about comparing two students to each other in terms of how good they are looking). Registered persons can interact with people they already know, or meet people they don't know yet. Facebook has the idea that everyone has his/her own profile and presents himself/herself to the rest of the world. Then they gather friends, who they may know or not know in real-life, and build groups of users, so-called social communities, the members of which have the possibility to post comments on other members' profiles, join groups with common interests, and share whatever information about themselves with others.

The Facebook developer documentation⁶⁸ provides a lot of functionality for application developers:

- *Core Concepts:*
 - *Social Design:* How to use Facebook to create social experience.
 - *Social Plugins:* Provide social experience to users with HTML.
 - *Open Graph:* Map content and user actions, and publish user activity to timelines.
 - *Social Channels:* integrate social channels and drive growth and engagement with app, site, or content.
 - *Login:* Connect with users on app or website (JavaScript or mobile SDKs can be used to speedup registration).
 - *Graph API:* Read and write data to Facebook and have a simple and consistent view of the social graph.
- *Advanced Topics:*

⁶⁷<http://www.facebook.com/>

⁶⁸<http://developers.facebook.com/docs/>

- *Dialogs*: Simple, consistent interface to display dialogs to users.
 - *FQL*: Facebook Query Language enables usage of an SQL-style interface to query the data exposed by the Graph API.
 - *Internationalization API*: Take advantage of the community translations framework to translate app or website.
 - *Payments*: Payment system to pay for digital and virtual goods in games and apps across Facebook.
 - *Ads API*: makes creation and management of ads on Facebook possible, without using the Facebook Advertising Manager tool.
 - *Chat API*: Integrate Facebook Chat into a Web-based, desktop, or mobile instant messaging product (connection via Jabber/XMPP service).
- *SDK & Tools*:
 - *JavaScript SDK*: Enables access to features of the Graph API and Dialogs via JavaScript.
 - *iOS SDK*: Provides Facebook Platform support for apps on Apple mobile devices.
 - *Android SDK*: Provides Facebook Platform support for Android apps.
 - *PHP SDK*: Provides Facebook Platform support to PHP-based web apps.
 - *Tools*: Provides tools such as App Dashboard, Graph API Explorer, App Insights, JavaScript Test Console, etc.

Google+

Google+⁶⁹ enables users to share content with only selected users [Kairam et al., 2012]. To achieve this, connected users are organized in so-called “circles” and when someone publishes content, she can decide which circles are able to see it. It is possible to create profiles, to connect to friends, family, and simply known people or to follow topics of interest. Since Google+ is merged with Hangouts⁷⁰ the platform can also be used to chat (in text, voice, or video) with others.

The Google+ Web API⁷¹ has the following components:

⁶⁹<https://plus.google.com/>

⁷⁰<https://www.google.at/hangouts/>

⁷¹<https://developers.google.com/+/>

- *+1 button*: Visitors are able to recommend content or a web site on Google Search and share it on Google+.
- *Badge*: Link from a web site to a Google+ profile page and improvement of Google search results.
- *Follow button*: Button that can be placed on a web site to automatically follow a Google+ profile.
- *Interactive posts*: Users can share a site or app and provide a link to it. For example RSVP (répondez s'il vous plaît) for an event.
- *Share*: Meant to use for content that someone wants to share but not +1.
- *Sign-in*: OAuth⁷² token for making API requests on behalf of signed in users.
- *Snippet*: Code generation for a page that indicates images and text.
- *JavaScript Client API*: Access to the Google+ libraries through JavaScript.
- *Supported languages*: This is the internationalization API, which enables developers to use functionality in different languages.

Diaspora*

Diaspora^{*73} is an alternative social networking platform. It is completely open source and developed in Ruby-on-Rails⁷⁴. The driving idea behind the project is, that every user is the sole owner of her own data. Therefore, Diaspora* offers the following functionality:

- *Hashtags*: Hashtags allow users to label and follow their interests.
- *Reshare*: Can be applied to posts users want to share with others.
- *Mentions*: With the @ character users can mention others and can get their attention.
- *Love*: By loving something users can show their appreciation to a topic.

⁷²<http://oauth.net/>

⁷³<http://diasporaproject.org/>

⁷⁴<http://rubyonrails.org/>

- *Personal profile*: Users can share personal information with others via a profile.
- *Aspects*: Sharing information with only a certain group of people through aspects (every connected person belongs to an aspect).
- *Host own data*: Personal data does not have to be on a central server. Every user is allowed to host an own server.

MySpace

MySpace⁷⁵ makes it possible for everyone to quickly build a web page that represents himself/herself through a profile and to connect with friends who have profiles on their own [Hinduja and Patchin, 2008]. Additionally, multimedia enhancements, like pictures, video, and audio files can be uploaded to extend the profiles. The profile pages are a kind of blogs, where users can post recent news and activities. Users can also leave comments on other profiles or communicate by personal messages.

Developers can use JavaScript or OSML⁷⁶ APIs for creating applications and games for MySpace⁷⁷. Furthermore, MySpace supports development of OpenSocial⁷⁸-based (OpenSocial is a component model for cloud based social applications) applications.

Twitter

Twitter⁷⁹ is a social network, which can be used to stay connected to friends, family, and co-workers through computers and mobile devices [Huberman et al., 2008]. Twitter allows users to post short messages (maximum 140 characters) which can then be read by any other user. Users can follow each other and get automatic updates about messages from people they are following. The links between Twitter users are directed, because if user1 follows user2, this does not automatically mean that user2 follows user1 back.

The Twitter platform APIs⁸⁰ provide access to Twitter data. Twitter APIs are constantly developed further and each of them represents a facet of Twitter. The following list introduces the Twitter APIs:

⁷⁵<https://myspace.com/>

⁷⁶<https://sites.google.com/site/opensocialdraft/Home/osml-tags>

⁷⁷<http://developer.myspace.com/>

⁷⁸<http://opensocial.org/>

⁷⁹<http://www.twitter.com/>

⁸⁰<https://dev.twitter.com/start>

- *Website API*: Enables integration of Twitter into web sites. This means that web sites can use the “tweet button” to post a tweet through a web site and the “follow button” to automatically follow the account represented through the web site.
- *Search API*: Allows querying for Twitter content. This means finding tweets by keywords, by mentioned user, or by author user.
- *REST API*: Access to timelines, status updates, and user information. For example, through the REST API an application can build a graph of people whom a certain user is following. Also, tweets can be posted, retweeted, favorited, replied to, etc.
- *Streaming API*: Provides data intensive, real-time communication to data mining or analytics applications. Allows large quantities of keywords, tracking, geo-tagged tweets, or public user statuses. This can be done through an HTTP connection.

XING

XING⁸¹ is a social networking platform for professional use that offers the following functionality to its users:

- *Contacts*: Connect to colleagues and coworkers.
- *Jobs & Careers*: Find a job or a qualified person for a vacancy.
- *Events*: Manage seminars, workshops, courses, and conferences.
- *Groups*: Join or create groups of common interest.
- *Companies*: Profiles for companies.
- *Projects*: Find open projects and apply to them as freelancer.

The XING API⁸² provides a possibility for developers to develop applications with XING integration and make API calls for profiles of users, contacts, bookmarks, recommendations, and geolocations. The supported technologies are REST, OAuth, JSON, and XML.

⁸¹<http://www.xing.com/>

⁸²<http://devblog.xing.com/api-2/api-release/>

LinkedIn

LinkedIn⁸³ can be compared to XING, but it rather targets people from academia than industry. However, it is a social network and a platform for job discovery and business opportunities. LinkedIn enables profile management, contacting other users, and discovery of information. The platform also has a RESTful API for developers⁸⁴.

Mendeley

Mendeley⁸⁵ is an academic social network and document (publication) organizer. It manages references and helps users to organize research, collaboration in research, and discovery of other research results and publications.

Mendeley provides the following functionality:

- Automatic bibliography generation.
- Online collaboration with peers.
- Import of papers from external resources.
- Discovery of relevant papers.
- Apps for mobile devices to access papers on the go.

Research Gate

ResearchGate is a social networking platform for scientists to improve collaboration in science in order to get better scientific results⁸⁶. In the beginning of ResearchGate two researchers discovered that working with a peer on the other side of the world was a quite complicated task. ResearchGate enables its users to create profiles and present themselves and their work to others, connect with peers from their global scientific community, and manage their publications. Furthermore, it makes discovering interesting research papers and getting fast answers for problems in their work or methodology easier. Finally, it can be also used as a career center, i.e. to find jobs in academia.

⁸³<http://www.linkedin.com/>

⁸⁴<http://developer.linkedin.com/rest>

⁸⁵<http://www.mendeley.com>

⁸⁶<http://www.researchgate.net/aboutus>AboutUs.html>

Academia.edu

Academia.edu⁸⁷ is a site where scientists can get in touch with academic peers⁸⁸. It uses social networking techniques to allow academics mostly from universities around the world to make connections by creating profiles and adding peers. It can be used by students to exchange information about lectures, by professors to communicate with researchers from similar fields, or by universities to find qualified personnel. Further, it is very useful to exchange knowledge with people who are far away or to reconnect with peers you met at conferences.

Slashdot

Originally, Slashdot⁸⁹ is a technology news website, but it has developed a community and goes into the direction of social networks [Kunegis et al., 2009]. The platform has editors and users who write short articles about news in the technology sector, and allows other users to comment the articles. Additionally, it has a “Zoo” feature, which lets users add friends and tag other users. In contrast to many other social network platforms, Slashdot allows to rate other users positively and negatively.

Slashdot has several channels like Slashdot BI (Business Intelligence), Slashdot Cloud, and Slashdot Data Center. Furthermore, it has a popular Job Center with a lot of vacancies in the technology sector.

Overview

Table 2.15 provides an overview of the most popular community building platforms. The platforms are listed alphabetically together with their categories.

2.3.3 Literature

Kim [2000] explains why communities form and grow, and shows how a social network, a catalyst for community growth, can be made. In this book all the popular strategies for bringing people in and retaining them are covered.

In the field of aggregation, extraction, and visualization of online social networks a tool called Flink has been developed, which is able to employ semantic technology for reasoning about personal information extracted from

⁸⁷<http://academia.edu/>

⁸⁸<http://www.killerstartups.com/social-networking/academia-edu-where-scholars-meet/>

⁸⁹<http://slashdot.org/>

Name	Category
Academia.edu	Scientific social network
Diaspora	Private social network
Facebook	Private social network
Google+	Private social network
LinkedIn	Business social network (science)
Mendeley	Papers management and scientific social network
MySpace	Private social network
Research Gate	Scientific social network
Slashdot	Technology news network
Twitter	Social communication network
XING	Business social network (industry)

Table 2.15: Overview of community building platforms.

web pages, emails, publication archives, and FOAF⁹⁰ profiles [Mika, 2005a]. In ontology-based knowledge management, the SEKT⁹¹ project produced very interesting results. Also current issues in social ontologies [Mika and Gangemi, 2004], and a discussion on the relation between sociability and semantics [Mika, 2005b] are important and could be of high interest.

To apply such changes or integrate a new technology or concept like this to the Semantic Web, it is necessary to understand the evolution of the Semantic Web [Zhou et al., 2010]. One of the steps in the evolution is the attempt to combine ideas from Social Web and Semantic Web [Gruber, 2007]. This means that the “collective intelligence” of the Social Web would be a perfect extension of the “knowledge representation and reasoning” of the Semantic Web. Here again, the “collective intelligence” such as comments, groups of friends, or interests could be represented by time series. Comments have time stamps, a content, and other properties which all also occur in classic time series. Friends and groups can be related and assigned to time series. Interests can be used as filters to present only certain parts of a time series. More details on this topic are presented in Chapter 5 “Community Building”.

⁹⁰<http://www.foaf-project.org>

⁹¹<http://www.sekt-project.com>

Part II

Implementation

Chapter 3

Time Series Processing Language

The Time Series Processing Language is a state-of-the-art language for processing different kinds of time series. It is implemented in Python and able to handle a large amount of different operations and calculations on various time series. Various time series means that the data which is processed can have different formats, such as numbers, strings, images, video clips, etc.

The language itself is defined by its syntax, which further defines which expressions for time series processing can be formulated by the language. It consists of three different components: a lexer, a parser, and an interpreter.

The *lexer* specifies the syntax of the language and defines which terms and expressions can be used. The *parser* is responsible for parsing an expression according to the rules set in the *lexer* and for constructing a data structure of the parsing results. Finally, the interpreter takes the time series and executes the expressions in the data structure from the parser.

Parts of this chapter have already been published in Božić and Winiwarter [2012].

3.1 Time Series Processing Language

The language on which this research is based is originally a classical time series processing language. This means that it is a generic language for processing time series data.

The language supports homogeneous (with fixed time grids) and heterogeneous time series processing. Time series can have very complex data structures. It is also possible to work with time patterns, time intervals, and single slots. The complex types of aggregation can be performed with

predefined, but also with user-defined functions.

Expression	Meaning
<code>< [n].sin * 2 + 3 ></code>	Calculation is applied to all slots.
<code>A, B < A + 2 * B ></code>	Combination of two time series (aggregation).
<code>< [n] > every 2 hours</code>	Projection to a fixed time grid.
<code>< (t .. t-2).mean > every 1 hour</code>	Sliding mean value.
<code>< [n]->hot if [n].temperature > 100 otherwise [n]->cold ></code>	Filtering, classification.

Table 3.1: General expressions and their meanings.

Some common example expressions are shown in Table 3.1. The slot selection is based either on the index ([n] stands for every slot one-by-one based on the index) or on the time ((t) stands for every slot based on time). If not every slot is selected, the selection can be also based on a range (e.g. [n-1 .. n] means from slot n-1 up to the current slot n). The first expression calculates the sine of the value from each slot, multiplies it by 2 and adds the value 3. Expression 2 specifies two time series, where each slot of time series A is added to the doubled value of each slot from time series B. The usage of a time grid is shown in expression 3, where only the slot default value is taken every 2 hours and copied to the output time series. Expression 4 calculates a mean value for each slot and the previous two slots, but only every 1 hour. Finally, the last expression sets the slot default value to “hot” if the temperature is higher than 100, and to “cold” otherwise. The general functionality can be consolidated in the following list:

- **Arithmetic calculation:** Operations such as addition, subtraction, multiplication and division on time series and slots, e.g.:

$$< A[n] * 2 + B[n] ** 3 + 2 * 5 > \quad (3.1)$$

This expression takes two time series (A and B), multiplies all values of time series A by 2, exponentiates all values of time series B by 3, multiplies 2 and 5, and adds each resulting value of A to B and to 10. The result is a new time series with each slot holding a result.

- **Time patterns, slot and range selection:** The times at which new values have to be generated can be controlled via a time pattern

language, e.g.:

$$< (t) > \text{ every 3 secs} \quad (3.2)$$

The expression takes a time series and returns a new time series with one value every 3 seconds.

- **Conditionals:** Conditionals, i.e. expressions where the evaluation depends on a condition, are not written with if cascades or ternary operators, but instead individual postfix if clauses are used, e.g.:

$$\begin{aligned} &< [n] \text{ if } n.depth < -100 \text{ m} \\ &\text{or } [n] * 0.01 \text{ if } n.depth > 100 \text{ m} \\ &\text{or } 0 \text{ otherwise } > \end{aligned} \quad (3.3)$$

The expression takes a time series and returns a new one with the same value for each slot if the depth property of the slot is lower than -100 m, the same value multiplied by 0.01 if the depth property is larger than 100 m, and 0 if the value of the slot is between -100 m and 100 m.

- **Slot selection:** For slot selection an index is used which is positive if counted from the first value and negative if counted from the last value, e.g.:

$$< [n + 1] > \quad (3.4a)$$

or

$$< [n - 1] > \quad (3.4b)$$

The expressions take a time series and create two new ones with all slots shifted to the right (3.4a) or to the left (3.4b).

- **Aggregation:** Slots can also be addressed in groups using a logical range. This is only used together with aggregations, e.g.:

$$< [n - 2 .. n].sum > \quad (3.5)$$

The expression takes a time series and creates a new one with the sum of all slots beginning with the slot with index **n-2** until the slot at the current position **n**. Supported aggregations are, for example: **sum** (building the sum of values from a given range), **prod** (building the product), **count** (counting all valid values; value \neq None), **min** (finding the minimum value), **max** (finding the maximum value), **mean** (calculating a mean value).

- **Mean calculation:** Some aspects of functionality can be generated by combining other functionality aspects, e.g. mean calculation can be achieved by combining time patterns and aggregation:

$$< [n - 1 \text{ hour} .. n].\text{mean} > \text{ every } 30 \text{ minutes} \quad (3.6)$$

The expression takes a time series and returns a new one with hourly mean values, which has a slot for each half hour mean calculated.

Table 3.2 shows how expressions change time series. It shows input time series, expressions, and modified time series for the previously explained example expressions. The first column contains the time series used as input, the second column the used expression, and the third column the results of the Time Series Processor (TSP – see Section 3.3 “Architecture”). Time series are represented as tables with an identifier at the left of the table. Each column is a slot of the time series. The first row shows the time stamps and the second row the values of slots.

Input Time Series					Expression	Output Time Series				
A	00:00:00	00:00:01	00:00:02		< A[n] * 2 + B[n] ** 3 + 2 * 5 > (3.1)	R	00:00:00	00:00:01	00:00:02	
	0	1	2				37	76	139	
B	00:00:00	00:00:01	00:00:02		< (t) > every 3 secs (3.2)	R	00	03	06	
	3	4	5				3	2	4	
A	00	01	02	03	04	05	06			
	3	1	6	2	0	5	4			
A	00:00:00	00:00:01	00:00:02		< [n] if n.depth < -100 m or [n] * 0.01 if n.depth > 100 m or 0 otherwise > (3.3)	R	00:00:00	00:00:01	00:00:02	
	-150 m	150 m	50 m				-150 m	1.5 m	0 m	
A	00:00:01	00:00:02	00:00:03		< [n+1] > (3.4a)	R1	00:00:02	00:00:03	00:00:04	
	0	1	2				1	2	None	
B	00:00:01	00:00:02	00:00:03		< [n-1] > (3.4b)	R2	00:00:00	00:00:01	00:00:02	
	3	4	5				None	3	4	
A	00:00:00	00:00:01	00:00:02		< [n-2 .. n].sum > (3.5)	R	00:00:00	00:00:01	00:00:02	
	1	2	3				1	3	6	
A	00:00:00	00:15:00	00:30:00		< [n - 1 hour .. n].mean > every 30 minutes (3.6)	R	00:00:00	00:30:00	01:00:00	
	1	2	3				0	1.2	3	
→	00:45:00	01:00:00	01:15:00			→	01:30:00	02:00:00		
	4	5	6				4	2.8		
→	01:30:00	01:45:00	02:00:00							
	2	1	0							

Table 3.2: Demonstration of time series processing results.

The time series processing language has been implemented in the Python programming language, to guarantee the ease of extensibility and interoperability with other programming languages. Therefore it is usable as a standalone library on major platforms¹.

3.2 Language Specification

In the following, we present a formal description of the time series processing language. The grammar of the language is specified in Extended Backus-Naur Form (EBNF), and we have added the complete source code to the appendix (see Appendix A). The presented specification is shown in form of railroad diagrams from the EBNF grammar according to the specified rules.

3.2.1 letter

The first and simplest rule is a letter, which can be lower or upper case (Figure 3.1).

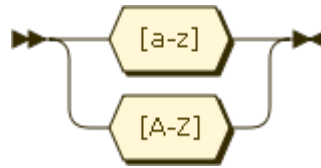


Figure 3.1: Letter.

3.2.2 digit

The digit rule matches a digit from 0 to 9 (Figure 3.2).



Figure 3.2: Digit.

3.2.3 integer

The integer rule matches an arbitrary number of digits preceded by an optional minus sign (Figure 3.3).

¹Currently Java, .Net, and Python



Figure 3.3: Integer.

3.2.4 float

The float rule is matched by two arbitrary numbers of digits separated by a dot (Figure 3.4).

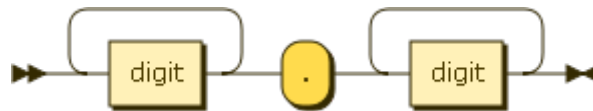


Figure 3.4: Float.

3.2.5 number

The number rule is matched by an integer or a float (Figure 3.5).

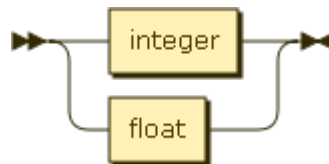


Figure 3.5: Number.

3.2.6 string

A string can be an arbitrarily long combination of letters, digits, and underscores (Figure 3.6).

3.2.7 stmt

A statement consists of a time series parameter id list and a pipe (Figure 3.7).

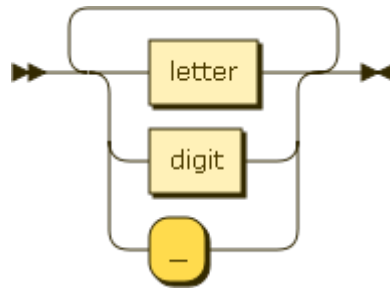


Figure 3.6: String.



Figure 3.7: Statement.

3.2.8 pipe

As shown in Figure 3.8, a pipe can be a generator or a pipe and a generator separated by the pipe operator (`|`).

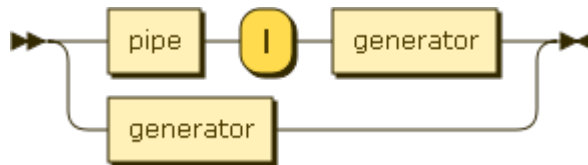


Figure 3.8: Pipe.

3.2.9 generator

A generator is matched by the start (`<<`) and end (`>>`) operators with an expression list in between, or by a generator and an every phrase (Figure 3.9).

3.2.10 ts_param_id_list

The time series parameter id list is matched by a formal parameter or another time series parameter id list followed by a formal parameter (Figure 3.10).

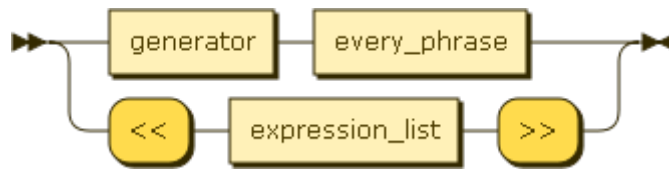


Figure 3.9: Generator.

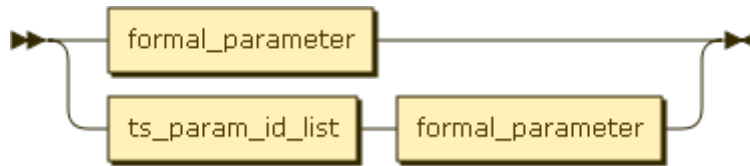


Figure 3.10: Time series parameter id list.

3.2.11 formal_parameter

The formal parameter starts with an at sign (@), which is followed by a letter, and optionally, by an arbitrarily long sequence of letters, digits, and underscores (Figure 3.11).

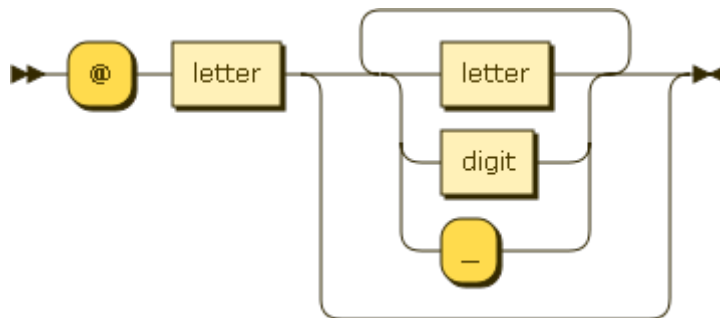


Figure 3.11: Formal parameter.

3.2.12 expression_list

The expression list can consist of an assignment expression or an expression list and assignment expression separated by a semicolon (Figure 3.12).

3.2.13 assign_expression

There are three ways to match an assignment expression (Figure 3.13): The easiest one is an if expression. The second possibility is a string (starting

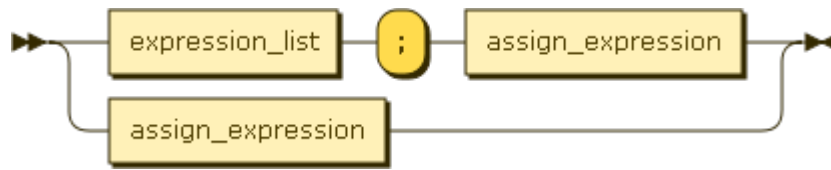


Figure 3.12: Expression list.

with a letter, and containing several letters or digits) for the time series id followed by the copy operator for all properties and an interval. The third option is an if expression with an assignment operator followed by a string for the property.

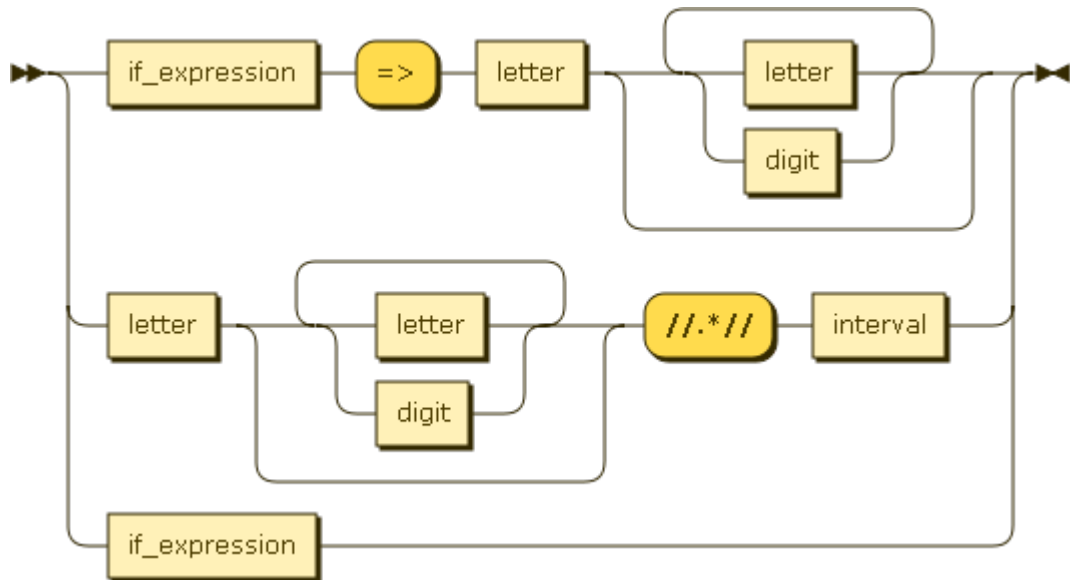


Figure 3.13: Assignment expression.

3.2.14 if_expression

The if expression can be a single logic expression or a logic expression followed by the IF keyword, an if expression, the OTHERWISE keyword, and a second logic expression (Figure 3.14).

3.2.15 logic_expression

A logic expression can be one of the following (Figure 3.15):

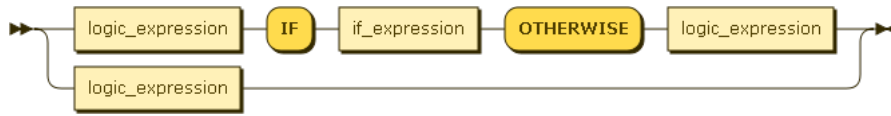


Figure 3.14: IF expression.

- logic expression AND add expression: a logic expression (containing logic operators) and an addition expression (containing addition, subtraction, or multiplication operators) are combined by the logic and operator (the true slots are set to 1 and the false slots are set to 0).
- logic expression OR add expression: a logic expression and an addition expression are combined by a logic or operator.
- logic expression XOR add expression: a logic expression and an addition expression are combined by a logic exclusive or operator.
- add expression $>$ add expression: two addition expressions are compared by the greater than operator.
- add expression $<$ add expression: two addition expressions are compared by the lower than operator.
- add expression \geq add expression: two addition expressions are compared by the greater than or equals operator.
- add expression \leq add expression: two addition expressions are compared by the lower than or equals operator.
- add expression $==$ add expression: two addition expressions are compared by the equals operator.
- add expression $!=$ add expression: two addition expressions are compared by the not equals operator.
- add expression: one single addition expression

3.2.16 add_expression

The addition expression consists of an addition expression and a multiplication expression combined by a plus sign or a minus sign (for addition and subtraction), or just a single multiplication expression (Figure 3.16).

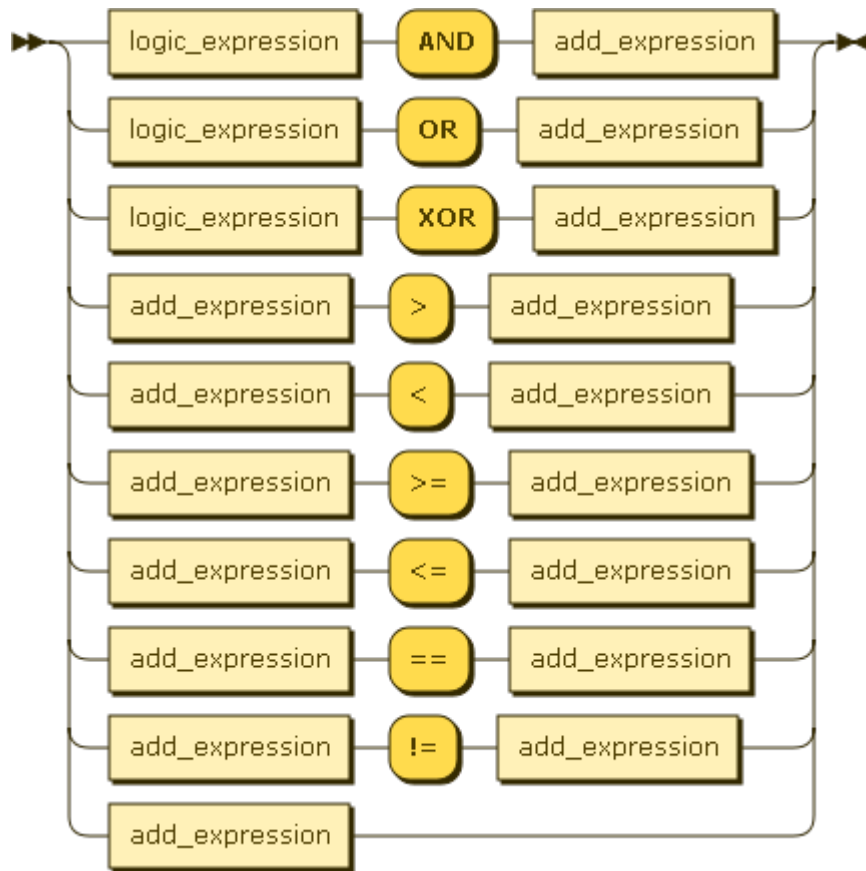


Figure 3.15: Logic expression.

3.2.17 `mult_expression`

The multiplication expression matches another multiplication expression combined with a general expression by a division or multiplication sign, as well as a single general expression (Figure 3.17).

3.2.18 `expression`

A general expression can be one of the following options (Figure 3.18):

- Two expressions combined by a power operator (**).
- An optional minus sign followed by an expression.
- A number.
- A string.

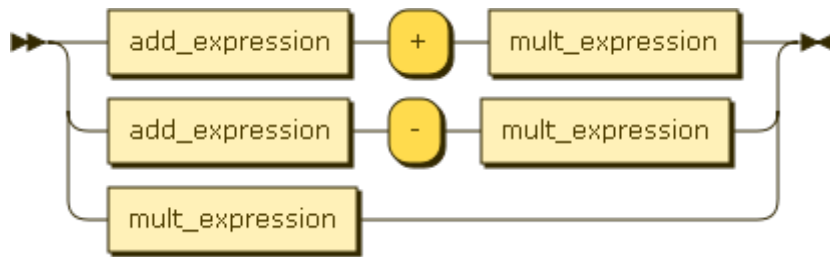


Figure 3.16: Addition expression.

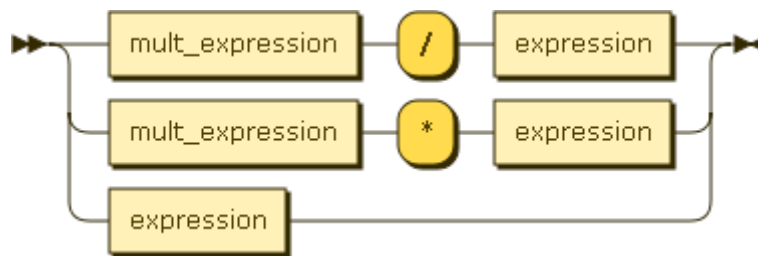


Figure 3.17: Multiplication expression.

- A time series slice.
- An if expression in parenthesis.
- A formal parameter followed by an expression in parentheses.
- A simple None (null pointer).

3.2.19 ts_slice

A time series slice consists of a property access and interval (Figure 3.19).

3.2.20 property_access

As shown in Figure 3.20, the property access can be simply a string (for the default property) or a string (defining the time series id) followed by a dot and another string (defining the name of the property).

3.2.21 interval

The interval can be a numeric, logical, or time interval (Figure 3.21).

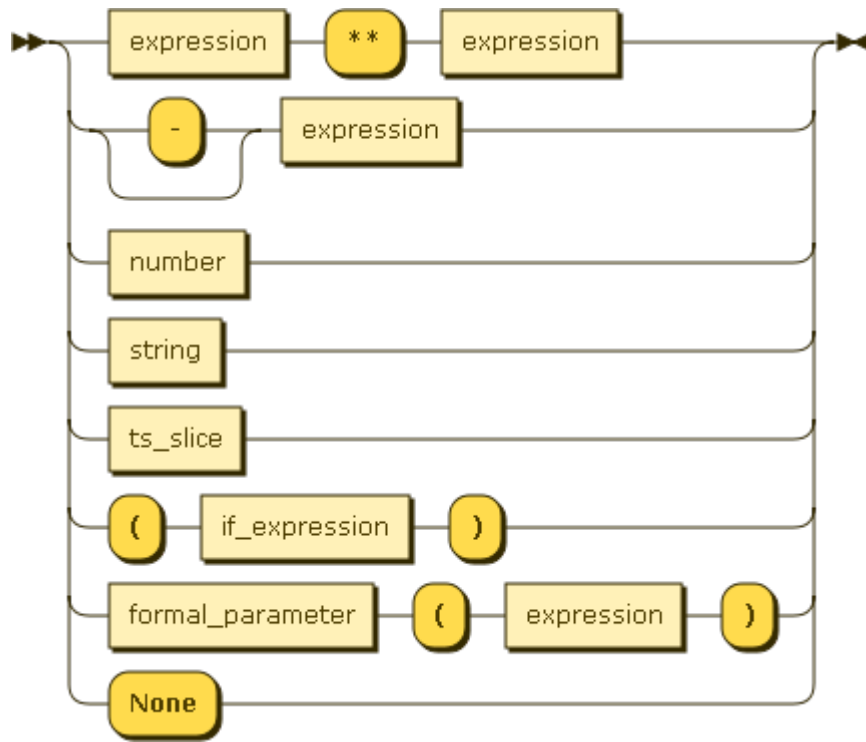


Figure 3.18: Expression.



Figure 3.19: Time series slice.

3.2.22 numeric_interval

A numeric interval is an integer in square brackets (Figure 3.22).

3.2.23 logical_interval

Figure 3.23 shows that a logical interval is a logical index expression in square brackets (for a single slot) or two logical index expressions in square brackets separated by two dots (for a slot range).

3.2.24 time_interval

The time interval is defined by a time index expression in square brackets, where the brackets can be open or closed, or two time index expressions in

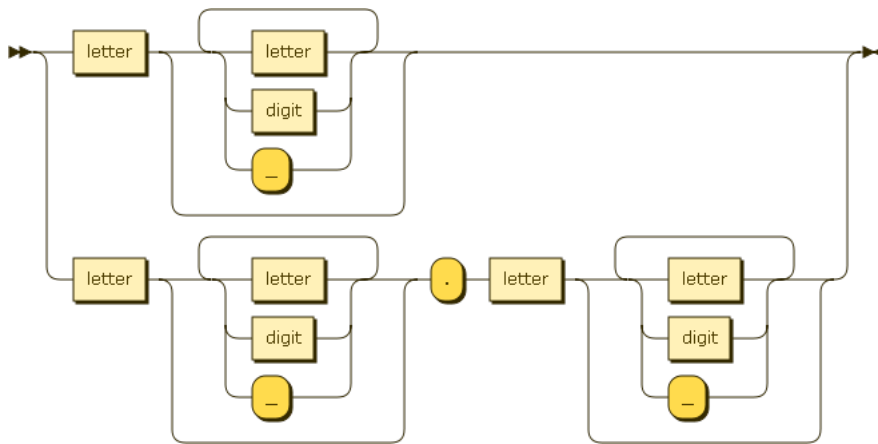


Figure 3.20: Property access.

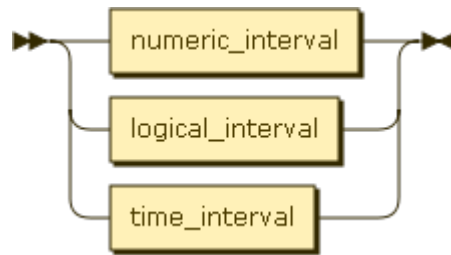


Figure 3.21: Interval.

brackets separated by two dots for a time range selection (Figure 3.24).

3.2.25 bra

Defines the starting square bracket, which can be open or closed (Figure 3.25).

3.2.26 ket

Defines the ending square bracket, which can be open or closed (Figure 3.26).

3.2.27 log_index_expression

The logical index expression can be an i for the current index, an $i + \text{integer}$ for future indices, or $i - \text{integer}$ for past indices (Figure 3.27).



Figure 3.22: Numeric interval.

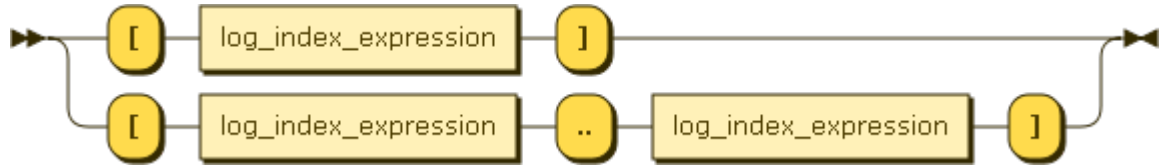


Figure 3.23: Logical interval.

3.2.28 `int_with_time`

The integer with time is used for **every** statements to define the time period in which a certain expression is executed (Figure 3.28). It consists of an integer followed by ms (milliseconds), secs (seconds), sec (second), mins (minutes), min (minute), hours, hour, days, day, weeks, or week.

3.2.29 `time_index_expression`

The time index expression is a `t` for the current point in time, `t + integer` with time for future points in time, or `t - integer` with time for past points in time (Figure 3.29).

3.2.30 `every_phrase`

As shown in Figure 3.30, the every phrase consists of the **every** keyword followed by an integer with time or the **every** keyword followed by an integer with time, an @ sign, and another integer with time (for the definition of a special start point for the measurement, e.g. **every 1 minute @ 15 seconds**).

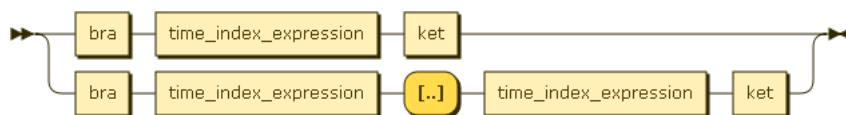


Figure 3.24: Time interval.

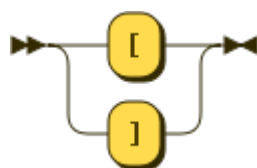


Figure 3.25: First part of bracket.

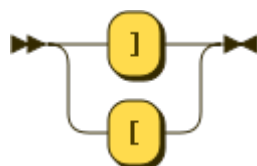


Figure 3.26: Second part of bracket.

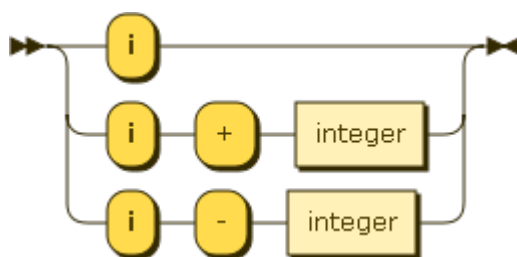


Figure 3.27: Logical index expression.

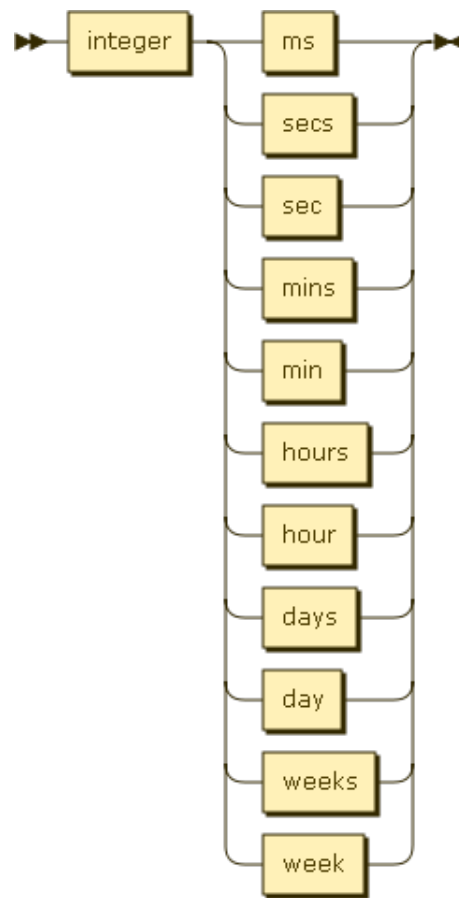


Figure 3.28: Integer with time unit.

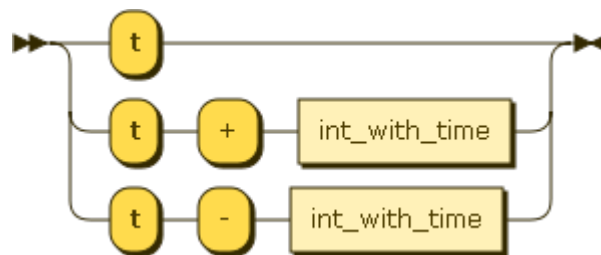


Figure 3.29: Time index expression.

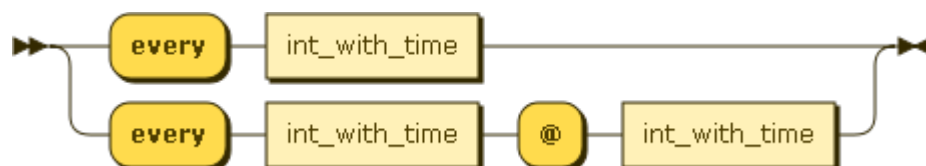


Figure 3.30: Every phrase.

3.3 Architecture

Before talking about the architecture as a whole, it is first necessary to explain which kind of time series our language understands. While the language makes no assumptions about the provenance of a time series, it has the abstract expectation that it is a linear array of chronologically ordered slots.

The time information within the slot may not be just a time stamp (with a system specific precision). A time duration marks the temporal extension of the slot. That duration can be positive or negative, depending on whether the validity of the slot reaches into the future or the past. Slots also have a logical “time”, which is the index in the series (starting with 0).

The payload in the slot has the form of key/value pairs. Keys are either simple identifiers or take the form of qualified names (Qnames – used to reference particular elements within XML documents) or Internationalized Resource Identifiers (IRIs – the internationalized form of Unified Resource Identifiers). Values are either anything of the former or literals such as strings, integers, floats or application-specific objects such as images and matrices. They can also be time durations or time patterns.

When slots are combined into a time series, obviously their time stamp and their signed durations have to be honored. Any temporal overlaps have to be resolved to arrive at a functional time series, i.e. one which can deliver one slot for one particular time stamp.

Figure 3.31 shows the architecture of our time series processing language. The main component is the Time Series Processor (TSP), which coordinates the whole workflow of time series processing. It triggers the parser and the interpreter, gathers time series data from a client platform or from a data source directly, and dynamically loads user-defined extensions. The reason why we have chosen an interpreter is that the time series need to be processed not only as a whole, but also in real-time. Therefore, we needed a possibility to implement a runtime-based interpreter to handle time series “on-the-fly”.

The principle of operation for a time series processor is shown in Figure 3.32. The TSP has an arbitrary number of time series and an expression as input, it calculates the expression on the input time series and provides an arbitrary number of output time series as a result.

The workflow of time series processing is that the client platform (Java, .NET, etc.) pulls data from a data source and provides it to the TSP together with an expression. The TSP first checks if there are user-defined modules to load and triggers the parser to parse the expression. After the parser returns the result in form of a predefined data structure, the TSP delivers the structure and the time series to the interpreter. The interpreter returns the result of the processing back to the TSP. The TSP forwards the result

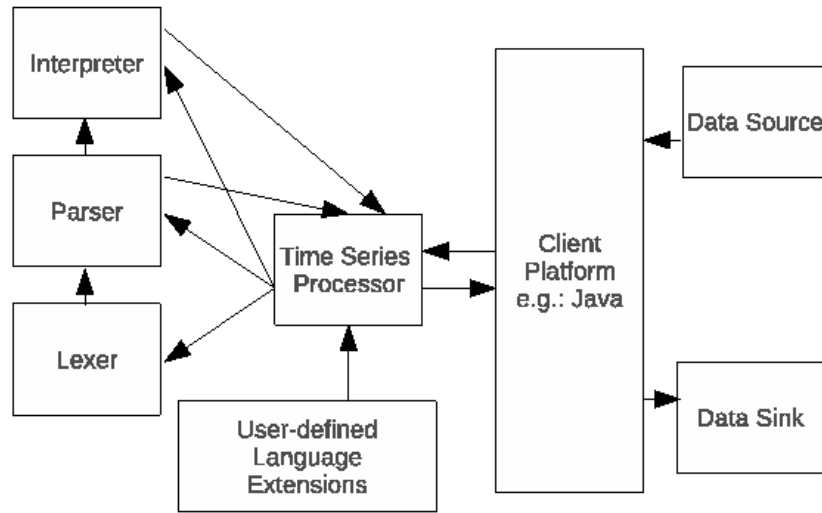


Figure 3.31: The language processing architecture.

to the client platform, which delivers it to the data sink.

Apart from the general architecture of the language, there are also a couple of possible Semantic Web oriented architectures. As the Time Series Processor is available as a Web Service as well, the language could be used web-based as shown in Figure 3.33. Here, a client application communicates with a web service representing the TSP. The client application provides time series and expressions; the web service communicates with a server, which has the parser and interpreter deployed, and provides the expression for parsing and the parsing output together with the time series data to the interpreter. Optionally, time series data can be saved temporarily in a database². The result of the interpreter processing is returned to the web service first, and then by the web service back to the client application.

Finally, Figure 3.34 presents a possible Semantic Web architecture, with language extensions for Semantic Web functionality. Again, the central component is the TSP. It organizes the communication with the parser and interpreter, semantic language extensions, and the data stores. After getting time series data and expressions, it parses the expression. The parser now needs to understand the semantics in the expression and uses additionally the semantic processor extension to process the parser output. When new resources

²For performance reasons.

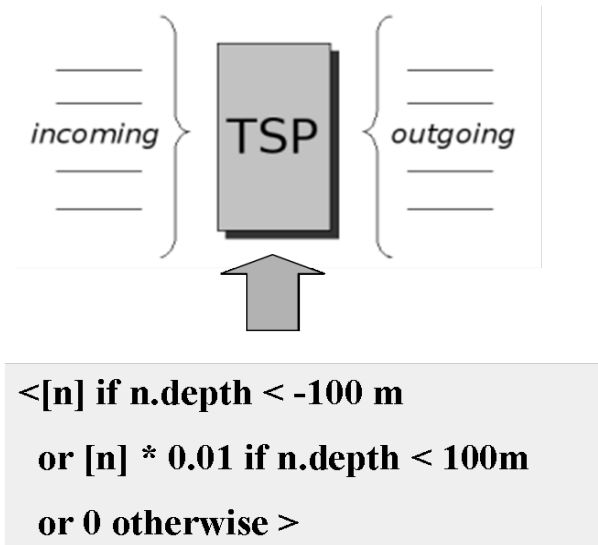


Figure 3.32: Workflow of a time series processor.

are needed, it uses the discovery processor (which implements semantic discovery and is used to discover new resources), and, if there is something to tag, the tagging processor (which is responsible for adding annotations to resources). For storage of data, an ontology store as well as a Resource Description Framework (RDF) store can be used. The results are interpreted by the interpreter and returned to the TSP.

The main goal of the system architecture as a whole is to support the high level of expressiveness and the user-friendly syntax of the language to facilitate the ease of extensibility and to allow meaningful data models. The following subsections describe the three main components of the language: lexer, parser, and interpreter.

3.3.1 Lexer

The task of the lexer is to convert a sequence of characters into a sequence of tokens, and thereby prepare the lexical expression for further processing in the parser. The lexer is implemented using the PLY³ Python module. PLY implements the lex and yacc parsing tools in Python.

Our lexer implementation can be split up into the following parts:

- **Token definition:** Definition of all reserved keywords and operators.

³<http://www.dabeaz.com/ply/>

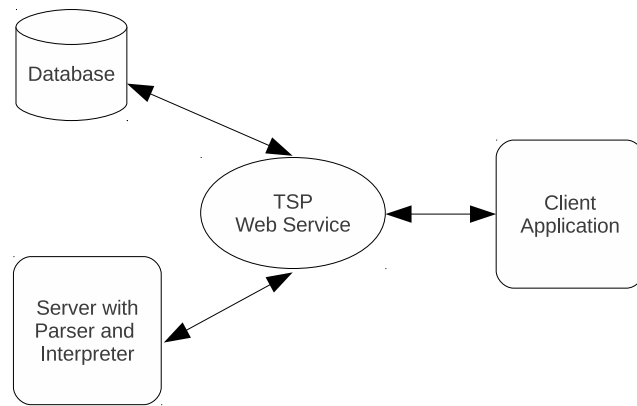


Figure 3.33: The web service based architecture.

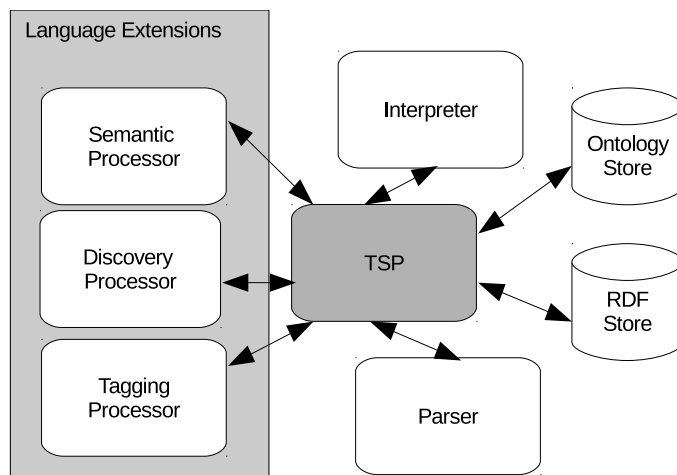


Figure 3.34: The Semantic Web architecture.

- **States definition:** Definition of possible lexer states (e.g. quote or comment).
- **Ruleset definition:** Definition of rule sets (what happens if a keyword or operator are matched).

The following code snippet shows the definition of tokens:

```

1 tokens=( 'NONE', 'STARTGEN', 'ENDGEN', 'LT', 'GT',
2          'SQUARE_BRA', 'SQUARE_KET',
3          'I', 'T',
4          'IF', 'OTHERWISE', 'ASSIGN',
5          'FLOAT', 'INTEGER', 'STRING', 'TS_PARAM_ID',
6          'TS_ID',
7          'AND', 'OR', 'XOR', 'PLUS', 'MINUS', 'TIMES',
8          'DIVIDE', 'POW',
9          'EQUALS', 'LTE', 'GTE', 'EQ', 'NEQ',
10         'SEMICOLON',
11         'ROUND_BRA', 'ROUND_KET', 'EVERY',
12         'FUNC_OR_ACCESS',
13         'DOIDOT', 'AT', 'SERPIPE', 'PROPCOPY',
14         'INT_WITH_TIME' )

```

Listing 3.1: Definition of tokens.

The tokens specify datatypes such as **NONE** and **STRING**; operators such as **LT** (<), **ASSIGN** (=), and **SQUARE_KET** (|); and also keywords such as **IF** and **EVERY**. This is necessary for the lexer to know which expressions can be used (i.e. how many rulesets will be needed).

The definition of states is for setting the lexer in a special state where operators and keywords may be interpreted differently (e.g. ignored when they appear in the comments state). There are two common cases in which this is necessary:

1. If a section is defined as comment, no interpretation should be made about the content of the section.
2. If a section is defined as quote (e.g. in string assignment), then keywords should be ignored and escape sequences should be respected.

The ruleset definition part of the lexer is used to actually generate the tokens. Every single token has an own defined method in the ruleset part. The following code snippet shows such a method for a floating point number:

```

1 def t_FLOAT(t):
2     r '\d+\.\d+([eE][+-]?\d+)? '
3     try:
4         t.value = float(t.value)

```

```

5         except ValueError:
6             print("Not a valid float %s" % t.value)
7             t.value = 0
8         return t

```

Listing 3.2: Ruleset for floating point numbers.

Listing 3.2 shows the implementation of the `t.FLOAT` method in Python code. This method is called every time the lexer finds a floating point number which matches the regular expression `'\d+\.\d+([eE][+-]?\d+)?'`. This is a decimal number of arbitrary length followed by a dot and another decimal number of arbitrary length. Optionally, the exponent can be added at the end of the floating point number. The `try` block attempts to convert the string value to a Python float datatype. If this is successful, the value is set. If the attempt fails, the value is set to 0. Finally, the token is returned.

3.3.2 Parser

For every predefined token combination a parser method is called. The parser is also implemented by using the Python PLY module and consists of the following parts:

- **Parser definitions:** Definition of pipes and generators (basic concepts for parsing).
- **Precedence logic:** Definition of precedence rules in expressions.
- **Function calls:** Calls to functions according to defined token combinations (core parsing functionality).

The parser definitions are responsible to define certain basic parsing concepts. The following listing shows the pipe concept, as defined in the parser definitions:

```

1 def p_pipe_serial(p):
2     '''pipe : pipe SERPIPE generator'''
3     generatorList=p[1]
4     newGenerator=p[3]
5     generatorList.append(newGenerator)
6     p[0]=generatorList

```

Listing 3.3: Pipe concept in parser definitions.

In Listing 3.3 the pipe operator is defined. The `p_pipe_serial` method is called every time the parser finds a `'pipe SERPIPE generator'` token pattern. Then the `pipe` token is set as the list of generators and the generator

token is appended to the list. The return value of the processing (`p[0]`) is set to be the generator list.

The parser looks at defined precedence rules every time it finds shift/reduce conflicts. This means that if the parser finds, e.g. ‘expression MINUS expression’ and the next token would be ‘DIVIDE’, it makes a shift. But if the operators are defined manually, the parser has a shift conflict. Therefore, some manually defined operators need to be added to the precedence tuple.

```

1 precedence = (
2     ('left', 'IF', 'OTHERWISE'),
3     ('left', 'GT'),
4     ('right', 'POW'),
5     ('right', 'UMINUS'),
6 )

```

Listing 3.4: Nested precedence tuple.

Listing 3.4 shows some operators added to the precedence tuple. The first element of each nested tuple denotes to which expression the operator is applied. It can be the left expression, as in ‘IF’, ‘OTHERWISE’, and ‘GT’ (greater than) in our example, or the right expression as in ‘POW’ (power) or ‘UMINUS’ (unary minus). The precedence is ascending from top to bottom, i.e. ‘IF’ and ‘OTHERWISE’ have the lowest precedence and ‘UMINUS’ the highest.

The main part of the parser are the function calls. They define the structure of the parsing tree and generate the output of the parser. In general, the parser output is a tree of dictionaries (hashes) and lists (arrays) as Python structure, containing the expression. An example of such a function call method is the following listing.

```

1 def p_if(p):
2     'if_expression : logic_expression IF if_expression OTHERWISE
   logic_expression'
3     trueExpr = p[1]
4     ifExpr = p[3]
5     otherExpr = p[5]
6     p[0] = {AST.TYPE: 'IFOT', "operation": 'IF', "condition":
   ifExpr, "true": trueExpr, "false": otherExpr}

```

Listing 3.5: Example function call.

Listing 3.5 shows the ‘IF ... OTHERWISE’ expression function call. It matches the tokens `logic_expression IF if_expression OTHERWISE logic_expression`, which means that there has to be an expression (first `logic_expression`) which will be executed if the `if_expression` is true; if the `if_expression` is false, the second `logic_expression` will be executed. In the Python source code we set `trueExpr` to the first logic expression (`p[1]`), `ifExpr` to the

if_expression (p[3]) and otherExpr to the second logic_expression (p[5]). After that, we create the corresponding tree structure {AST.TYPE: 'IFOT', 'operation': 'if', 'condition': ifExpr, 'true': trueExpr, 'false': otherExpr}.

As a next step, we will have a deeper look into the tree structure:

```

1 {
2   'generators':
3   [
4     {
5       'type': 'generator',
6       'expressions':
7       [
8         (
9           {
10            'type': 'func_call',
11            'id': 'log',
12            'arg_list': (3,)
13          },
14          None
15        )
16      ],
17      'every': None
18    }
19  ],
20  'type': 'stmt',
21  'parameters':
22  {
23    'A': None
24  }
25 }
```

Listing 3.6: The output of a parsing process – a dictionary-list-tree structure.

The tree structure in Listing 3.6 shows an expression with a **generator**, a **type**, and a **parameter**. The generator is of type **generator**, has an expression and no **every** (i.e. no time pattern). The expression is a function call and has the id **log**, which means that it is a logarithm function. In the argument list, it has as value the integer '3'. The type of the expression is **stmt** (statement), and the only parameter is the time series 'A' which is a null pointer.

3.3.3 Interpreter

After parsing we have a Python dictionary-list-tree structure, as seen above. This is an efficient representation of the expression that can be used to process time series. Therefore, the interpreter has the possibility to evaluate this

expression on one or more time series (the exact number of time series to be processed depends on the number of parameters in the expression tree). An example of the interpreter usage is shown in the following listing.

```

1  p = parser.Parser()
2
3  ast = p.parse(expression)
4
5  i = interpreter.Interpreter()
6
7  result = i.evaluate(ast, timeSeries)

```

Listing 3.7: Interpreter usage.

Listing 3.7 clearly shows the workflow of time series processing. There are exactly 4 steps which need to be run through one by one:

1. Instantiation of a parser object (line 1).
2. Parsing of the expression (the input is a string holding the expression) by calling the `parse()` method of the parser object. The return value is the previously introduced tree structure (line 3).
3. Instantiation of an interpreter object (line 5).
4. Interpreting the expression tree on one or more time series instances (line 7). The result is one or more new time series, and the parameters are `ast` (= expression tree) and `timeSeries` (a number of time series).

This is the view from the user perspective. This means, all a user has to do is to parse the expression and to evaluate the result on her time series. In fact, those two steps can also be done in only one method. Then a user would only need to pass the expression string together with the time series to the method and get the resulting time series as return value.

If we go a step deeper and have a look into the evaluate method of the interpreter, we can see the Python code, which explains the workflow of the interpreter very well, in the next listing.

```

1  def evaluate(self, ast, tss_in):
2
3      tssDic = tss_in
4
5      type=ast[AST.TYPE]
6      parameters=ast["parameters"]
7      generators=ast["generators"]
8
9      for generator in generators:

```

```

10         tsResult = self.constructResultTS(generator , tssDic)
11         tsResult = self.interprete(generator , tssDic ,
12         tsResult)
13         tssDic = {"_": tsResult}
14
15     return tsResult

```

Listing 3.8: Implementation of the evaluate method of the interpreter.

The main method of the interpreter is shown in Listing 3.8. In the evaluate method the input time series are saved to the time series dictionary. The type of expression is set to AST.TYPE (which is stmt – currently the only supported type). In the next step the parameters and generators are set. There can be an arbitrary number of parameters (input time series) and generators (expressions – since it is also possible to pipe, i.e. to provide a number of expressions which are processed one after the other). For every generator the resulting time series is constructed, and the result is saved to the time series after interpreting the generator on the appropriate time series dictionary. Then the result is saved back to the dictionary and, finally, all resulting time series are returned to the method caller.

3.4 Expressions

In the following list we present 58 expressions of our time series processing language which cover the whole functionality and demonstrate how the time series processor works. Since the expressions, when processed, use all implemented modules and methods, they are used for unit tests, which have a code coverage of 100 %, as well.

1. @A << XXX >>: Invalid expression – used to test exception handling.
2. @A << A[i] >>: Copy operation, the values of the time series A are copied to the result time series.
3. @A << A//.*//[i] >>: Copy operation for the whole time series. This means that not only the primary values (referenced by index [i]) are copied, but also all properties of the time series.
4. @A << mean(A[t-10min..t]); A//.*//[i] >> every 10 mins: Mean calculation for 10 minute mean values from current point in time (t) to 10 minutes in the past. Additionally, all properties of the input time series are copied to the resulting time series.

5. `@I @A << A[i] >>`: Input of two time series and selection of the default value of the second one.
6. `@A @I << A[i] >>`: Input of two time series and selection of the default value of the first one.
7. `@A << A[i]*2; A[i]*3 >>`: The input time series is processed through two expressions (separated by a semicolon). In the first one all values are multiplied by 2 and in the second one all values are multiplied by 3.
8. `@A << A[t]*2 >> every 1 sec`: The values of the input time series are multiplied by 2 in a time grid of 1 second (every second the value is selected and multiplied by 2).
9. `@A @B << A[i] + B[i] >>`: The default values of time series B are added to the default values of time series A.
10. `@A @B << A[i] - B[i] >>`: The default values of time series B are subtracted from the default values of time series A.
11. `@A @B << A[i] * B[i] >>`: The default values of time series B are multiplied by the default values of time series A.
12. `@A @B << A[i] / B[i] >>`: The default values of time series A are divided by the default values of time series B.
13. `@A @B << A[i] ** B[i] >>`: The default values of time series A are raised to the power of the default values of time series B.
14. `@A << A[i+1] >>`: Time series A is shifted by one slot to the left.
15. `@A << A[i-1] >>`: Time series A is shifted by one slot to the right.
16. `@A << A[i] * 2 >>`: The default value of time series A is multiplied by 2.
17. `@A << A[i] * 0.5 >>`: The default value of time series A is multiplied by 0.5.
18. `@A << A[i] + 2 >>`: The default value of time series A is increased by 2.
19. `@A << A[i] + 2.4 >>`: The default value of time series A is increased by 2.4.

20. `@A << A[i] ** 2 + 5 >>`: The default value of time series A is raised to the power of 2 and increased by 5.
21. `@A << A[i] + "X" >>`: The string "X" is concatenated to the default value of time series A.
22. `@A << -A[i] >>`: Change the sign of the default value of time series A.
23. `@A << A[2] >>`: Select the third slot of time series A (logical index – numbering starts with 0).
24. `@A << mean(A[i-1 .. i]) >>`: For every slot of time series A, calculate the mean value of the current slot and the previous slot.
25. `@A << max(A[i-1 .. i]) >>`: For every slot of time series A, calculate the maximum value of the current slot and the previous slot.
26. `@A << min(A[i-1 .. i]) >>`: For every slot of time series A, calculate the minimum value of the current slot and the previous slot.
27. `@A << mean(A[t .. t + 1 min]) >> every 1 min`: For every slot of time series A in 1 minute interval, calculate the mean value of the current slot (time index) and all slots one minute in the future.
28. `@A << A[t .. t + 1 min].mean >> every 1 min`: For every slot of time series A in 1 minute interval, calculate the mean value of the current slot (time index) and all slots one minute in the future.
29. `@A << mean(A[t .. t + 10 sec]); A//.*//[t..t + 10 sec] >> every 1 sec`: For every slot of time series A in 1 second intervals, calculate the mean value of the current slot and all slots 10 seconds in the future. Further copy all properties of time series A to the resulting time series.
30. `@A << mean(A[t .. t + 1 sec]) >> every 1 sec`: For every slot of time series A in 1 second intervals, calculate the mean value from the current slot to the slot 1 second in the future including left and right borders.
31. `@A << mean(A[t .. t + 1 sec]) >> every 1 sec`: For every slot of time series A in 1 second intervals, calculate the mean value from the current slot to the slot 1 second in the future excluding left and including right border.

32. `@A << mean(A[t .. t + 1 sec]) >> every 1 sec`: For every slot of time series A in 1 second intervals, calculate the mean value from the current slot to the slot 1 second in the future including left and excluding right border.
33. `@A << mean(A)t .. t + 1 sec]) >> every 1 sec`: For every slot of time series A in 1 second intervals, calculate the mean value from the current slot to the slot 1 second in the future excluding left and right borders.
34. `@A @B << A[i] and B[i] >>`: For every slot of time series A and B, compare the default values of the slots with the logical AND operator and set 1 to the resulting time series if both values are non-zero, and 0 otherwise.
35. `@A @B << A[i] or B[i] >>`: For every slot of time series A and B, compare the default values of the slots with the logical OR operator and set 1 to the resulting time series if at least one of the values is non-zero, and 0 otherwise.
36. `@A @B << A[i] xor B[i] >>`: For every slot of time series A and B, compare the default values of the slots with the logical XOR operator and set 1 to the resulting time series if one value is non-zero and the other value zero, and 0 otherwise.
37. `@A @B << (A[i] > B[i]) >>`: For every slot of time series A and B, compare the default values of the slots with the greater than operator and set 1 to the resulting time series if the corresponding value of A is greater than B, and 0 otherwise.
38. `@A @B << A[i] < B[i] >>`: For every slot of time series A and B, compare the default values of the slots with the lower than operator and set 1 to the resulting time series if the corresponding value of A is lower than B, and 0 otherwise.
39. `@A @B << A[i] >= B[i] >>`: For every slot of time series A and B, compare the default values of the slots with the greater than or equal operator and set 1 to the resulting time series if the corresponding value of A is greater than or equals B, and 0 otherwise.
40. `@A @B << A[i] <= B[i] >>`: For every slot of time series A and B, compare the default values of the slots with the lower than or equal operator and set 1 to the resulting time series if the corresponding value of A is lower than or equals B, and 0 otherwise.

41. `@A @B << A[i] == B[i] >>`: For every slot of time series A and B, compare the default values of the slots with the equals operator and set 1 to the resulting time series if the corresponding value of A is equal to B, and 0 otherwise.
42. `@A @B << A[i] != B[i] >>`: For every slot of time series A and B, compare the default values of the slots with the not equal operator and set 1 to the resulting time series if the corresponding value of A is not equal to B, and 0 otherwise.
43. `@A << A[i] if A[i] >= 3 otherwise 0 >>`: For every slot of time series A, take the slot value if it is greater than or equals 3, otherwise take 0.
44. `@A @B << A[i] if A[i] < B[i] otherwise B[i] >>`: For every slot of time series A and B, take the slot value of A if it is lower than the value of B, otherwise take B.
45. `@A << 1 if (A[i] > 1) otherwise 0 => value:value >>`: For every slot of time series A, set the property “value:value” to 1 if the default value is greater than 1, otherwise set the property to 0.
46. `@A << A[i]*2 if (A[i] > 2) otherwise A[i]**2 >>`: For every slot of time series A, multiply the default slot value by 2 if it is greater than 2, otherwise raise it to the power of 2.
47. `@A @B @C << A[i] + C[i] if B[i] == A[i] otherwise B[i]*C[i] >>`: For every slot of time series A, B, and C, add the value of C to the value of A if A equals B, otherwise multiply B by C.
48. `@A << A[i] if (A[i] > 1) otherwise None >>`: For every slot of time series A, take the default value of A, if it is greater than 1, otherwise take the null pointer (`None` in Python).
49. `@A << A[i] => mean >>`: For every slot of time series A, set the “mean” property of the slot to the default value.
50. `@A << mean(A[i-1 .. i]) => mean >>`: For every slot of time series A, calculate the mean value for the current and previous value and set it to the “mean” property of the current value.
51. `@A << "set" if (A[i] > 10) otherwise "reset" => command >>`: For every slot of time series A, set the property “command” to the string “set” if the default value of the slot is greater than 10, otherwise set the value to “reset”.

52. `@A << A[i]+log(3) => value >>`: For every slot in time series A, add the logarithm of 3 to the default slot value and store it in the “value” property.
53. `@A << log(A[i]) >>`: For every slot in time series A, calculate the logarithm of the default value.
54. `@A << A[i] * 2; 10 => value_test>>`: For every slot of time series A, multiply the default value by 2 and set the “value_test” property to 10.
55. `@A << A[i] * 2 >> | << _[i] / 2 >>`: For every slot of time series A, multiply the default value by 2, take the resulting time series through a pipe to the next expression and divide the default slot value by 2.
56. `@A << count(A[i]) >>`: For every slot of time series A, prepare the values for counting (i.e. if the value is non-zero and not a null pointer, set the result to 1, otherwise set to 0).
57. `@A << count(A[t]) >> every 1 sec`: For every slot of time series A, prepare the values for counting in 1 second intervals.
58. `@A << A[i] if (A[i] > 2) otherwise None >> | << count(_[i]) >>`: For every slot of time series A, take the default slot value if it is greater than 2, otherwise take None. Put the resulting time series through the pipe operator to the next expression and count all default slot values.

3.5 Benchmarks

This section gives an overview of the capabilities of the processor in terms of performance. We present several tests to see how fast the language can process time series of different size and expressions of different complexities. The tests are run on a Quad Core i7 machine with 8 GB main memory.

#	Cl.	Expression	100k	200k	300k	400k	500k	600k	700k	800k	900k	1000k
1	3	@A << A[i] >>	2.42	4.80	7.19	9.68	12.12	14.45	16.88	19.33	21.83	24.88
2	3	@A << A[/.*/[i] >>	3.82	7.93	12.09	15.92	19.78	24.47	28.32	31.64	35.45	39.97
3	3	@A << mean(A[t-10min..t]); A[/.*/[i] >> every 10 mins	0.06	0.12	0.19	0.25	0.31	0.38	4.44	4.68	4.18	5.22
4	3	@B @A << A[i] >>	2.25	4.72	7.33	9.40	12.49	14.20	16.44	19.74	21.28	23.85
5	3	@A @B << A[i] >>	2.35	4.65	7.08	9.95	11.84	14.07	17.61	18.94	22.16	24.80
6	3	@A << A[i]*2; A[i]*3 >>	8.53	17.23	26.06	34.60	43.09	52.40	61.19	69.03	78.24	86.65
7	2	@A << A[t]*2 >> every 1 sec	11.11	44.40	99.61	176.94	275.03	398.26	542.43	709.99	896.66	1105.21
8	3	@A @B << A[i] + B[i] >>	6.69	13.30	19.91	26.61	33.20	40.03	48.05	53.90	61.48	67.01
9	3	@A @B << A[i] - B[i] >>	6.61	13.62	20.00	27.60	34.18	39.90	47.60	54.26	60.30	67.76
10	3	@A @B << A[i] * B[i] >>	6.73	13.60	20.08	26.60	33.23	40.02	46.42	53.24	61.63	68.84
11	3	@A @B << A[i] / B[i] >>	6.58	13.62	20.07	26.73	33.74	40.04	46.77	53.79	61.44	68.72
12	3	@A @B << A[i] ** B[i] >>	6.60	13.35	20.06	26.81	33.89	40.08	46.82	53.73	61.30	66.86
13	3	@A << A[i+1] >>	2.27	4.71	7.02	9.46	11.86	14.09	17.16	18.90	22.15	24.76
14	3	@A << A[i-1] >>	2.24	4.79	7.41	9.46	11.71	14.21	17.51	19.09	21.25	23.86
15	3	@A << A[i] * 2 >>	4.48	9.29	13.79	18.92	23.65	28.43	32.17	36.79	41.63	46.18
16	3	@A << A[i] * 0.5 >>	4.79	9.74	14.66	19.66	24.43	29.31	34.23	40.04	44.04	48.94
17	3	@A << A[i] + 2 >>	4.49	9.26	13.78	18.32	23.19	27.87	32.98	36.77	41.15	45.77
18	3	@A << A[i] + 2.4 >>	4.72	9.74	14.54	19.40	24.26	29.77	33.87	38.66	43.68	49.39
19	3	@A << A[i] ** 2 + 5 >>	7.23	14.76	22.61	29.82	37.41	44.43	52.61	60.08	66.45	74.22
20	3	@A << -A[i] >>	3.76	7.30	10.97	14.86	18.82	22.07	25.63	29.28	33.05	37.43
21	3	@A << mean(A[i-1 .. i]) >>	4.67	9.20	13.75	18.04	22.51	27.22	31.57	36.39	41.89	46.33
22	3	@A << max(A[i-1 .. i]) >>	4.34	8.96	13.53	17.92	22.71	26.83	32.07	35.61	40.01	44.89
23	3	@A << min(A[i-1 .. i]) >>	4.36	8.85	13.36	18.36	22.91	26.62	32.00	35.47	39.88	44.69
24	2	@A << mean(A[t .. t + 1 sec]) >> every 1 sec	11.37	44.84	100.31	178.40	279.97	402.97	545.86	713.44	903.94	1082.11
25	3	@A << mean(A[t .. t+1min]) >> every 1 min	0.43	1.71	2.76	4.70	7.05	8.55	12.16	15.63	17.75	22.77
26	3	@A << mean(A[t-1min .. t]) >> every 1 min	0.51	1.53	2.52	4.68	7.02	8.96	12.11	15.61	17.71	21.91
27	1	@A << mean(A[t .. t+10sec]); A[/.*/[t..t+10sec] >> every 1 sec	27.18	98.62	213.74	369.52	575.74	844.07	1136.51	1463.27	1875.21	2293.99
28	2	@A << mean(A[t .. t + 1 sec]) >> every 1 sec	11.57	45.73	101.35	180.21	280.76	402.77	551.00	712.64	928.57	1143.03

Table 3.3: Execution time of the time series processing expressions (Part 1).

#	Cl.	Expression	Length									
			100k	200k	300k	400k	500k	600k	700k	800k	900k	1000k
29	2	@A << mean(A[t .. t + 1 sec]) >> every 1 sec	11.81	46.46	104.35	182.62	285.80	411.29	558.98	723.22	914.75	1094.57
30	2	@A << mean(A[t .. t + 1 sec]) >> every 1 sec	11.39	44.79	99.73	176.07	275.39	389.42	526.60	687.85	868.36	1071.80
31	2	@A << mean(A[t .. t + 1 sec]) >> every 1 sec	11.10	43.58	97.25	172.56	269.021	386.44	527.87	687.75	868.50	1073.49
32	3	@A @B << A[i] and B[i] >>	6.59	13.17	19.73	26.69	32.90	39.41	46.72	52.54	59.04	66.82
33	3	@A @B << A[i] or B[i] >>	6.49	13.33	19.70	26.29	33.34	39.48	46.71	52.48	59.24	66.06
34	3	@A @B << A[i] xor B[i] >>	6.48	13.21	19.77	26.77	32.89	40.02	46.00	52.68	59.25	65.91
35	3	@A @B << (A[i] > B[i]) >>	6.47	13.14	20.01	26.32	32.90	39.41	46.00	52.69	59.97	65.81
36	3	@A @B << A[i] < B[i] >>	6.48	13.17	20.03	26.29	32.75	39.45	45.93	52.59	60.13	65.70
37	3	@A @B << A[i] >= B[i] >>	6.46	13.13	19.95	26.24	33.26	39.98	45.89	52.56	59.18	65.75
38	3	@A @B << A[i] <= B[i] >>	6.48	13.15	20.00	26.36	32.86	40.01	46.04	53.36	59.19	66.63
39	3	@A @B << A[i] == B[i] >>	6.47	13.09	19.62	26.24	33.29	39.36	45.86	53.43	59.06	65.77
40	3	@A @B << A[i] != B[i] >>	6.60	13.13	19.97	26.30	32.89	39.38	45.95	52.61	59.99	65.80
41	3	@A << A[i] if A[i] >= 3 otherwise 0 >>	7.15	14.51	21.99	29.06	36.17	43.47	50.81	58.21	66.05	72.43
42	3	@A @B << A[i] if A[i] < B[i] otherwise B[i] >>	11.52	23.27	35.13	46.53	58.58	70.31	81.50	92.95	104.47	117.33
43	3	@A << 1 if (A[i] > 1) otherwise 0 => value:value >>	6.81	13.75	20.94	27.46	34.36	41.17	48.85	55.03	61.91	69.68
44	3	@A << A[i]*2 if (A[i] > 2) otherwise A[i]**2 >>	12.36	24.93	37.70	49.93	62.20	75.33	87.32	99.89	112.45	124.85
45	3	@A @B @C << A[i] + C[i] if B[i] == A[i] otherwise B[i]*C[i] >>	18.72	36.69	54.71	73.20	89.52	108.33	124.22	144.13	158.92	177.12
46	3	@A << A[i] if (A[i] > 1) otherwise None >>	6.59	13.18	19.82	26.37	33.75	40.10	46.60	53.29	59.38	66.94
47	3	@A << A[i] => mean >>	2.18	4.62	6.97	9.29	11.60	13.95	16.19	18.42	21.14	23.51
48	3	@A << mean(A[i-1 .. i]) => mean >>	4.42	8.48	12.90	17.22	21.75	26.40	30.28	34.88	39.64	43.99
49	3	@A << "set" if (A[i] > 10) otherwise "reset" => command >>	7.34	14.32	21.59	28.27	35.92	42.00	48.77	56.54	64.05	70.43
50	3	@A << A[i]+log(3) ==> value >>	6.64	13.25	20.23	27.09	33.86	40.36	47.89	54.92	61.63	68.23
51	3	@A << log(A[i]) >>	4.21	8.53	12.61	16.79	20.99	25.21	29.61	34.66	39.50	41.94
52	3	@A << A[i] * 2; 10 ==> value.test >>	5.00	9.97	15.02	19.97	25.40	30.60	34.39	39.20	44.25	48.74
53	3	@A << A[i] * 2 >> << -[i] / 2 >>	9.36	18.70	28.23	36.70	46.72	56.12	65.83	73.78	84.35	93.47
54	3	@A << count(A[i]) >>	3.92	8.16	12.26	16.38	20.41	24.32	28.68	32.61	36.60	40.61
55	2	@A << count(A[i]) >> every 1 sec	11.17	45.11	101.08	177.57	276.48	397.81	552.29	710.05	893.17	1106.03
56	3	@A << A[i] if (A[i] > 2) otherwise None >> << count(-[i]) >>	6.68	13.45	19.87	26.85	33.50	39.84	46.68	53.25	60.09	67.10

Table 3.4: Execution time of the time series processing expressions (Part 2).

Tables 3.3 and 3.4 show the results of the benchmark tests. The expressions have been processed for time series with a raising number of slots (from 100 000 up to 1 000 000 in steps of 100 000 slots). The result is the execution time in seconds. We have subdivided the tests into three classes: class 1 is the most execution time consuming expression (expression number 27 in Table 3.3), class 2 are expressions with execution times between 1071.80 and 1143.03 seconds for 1 000 000 slots, and class 3 are all other expressions (which have a clearly smaller execution time). We have picked one representative per class, which is marked in grey and italic in the table.

Figure 3.35 shows the representatives of the three categories defined in Tables 3.3 and 3.4. On the y-axis we have the time of execution in seconds and on the x-axis the number of slots of the time series. The legend shows the color of the line for the appropriate expression. Expression number 27 has clearly the highest execution time, because it combines several expensive operations on the input time series: it calculates mean values based on time range selection, copies all properties of the time series based on time range selection, and triggers the processing every second. This is an extreme case and a very rare expression structure. Therefore, the high time consumption is acceptable. Expression number 55 has an average time consumption, since it uses time selection and triggers the count function every second. This is a more common usage of the language but still occurs only in approximately 12% of the cases (based on use case definitions for the language, which are not subject of this thesis). Most commonly, simple expressions such as expression number 19 are used. These expressions show linear growth in execution time and can be processed very quickly (no notable waiting times for the user). Therefore, the representative expressions show that the scalability of the language is good enough for time series processing of time series with up to 1 000 000 slots.

Since there is a large number of class 3 expressions, we provide an overview in Table 3.5 and subdivide them into 7 subclasses to show how they differ. The representatives are again highlighted in gray and printed in italic. Class 1 is represented by an if expression (number 45) with three input time series which takes significantly more time than the other expressions. Expression 42 represents class 2 and is an if expression with two input time series. Expression 6 performs two parallel calculations in one expression (class 3), expression 36 a comparison between two time series (class 4), and expression 15 performs one calculation on a single time series (class 5). The expressions 1 (simple iteration over slots) and 25 (mean calculation) have a similar time cost for 1 000 000 slots, but expression 25 scales better, since the significant growth in time starts at 700 000 slots compared to 500 000 slots for expression 1.

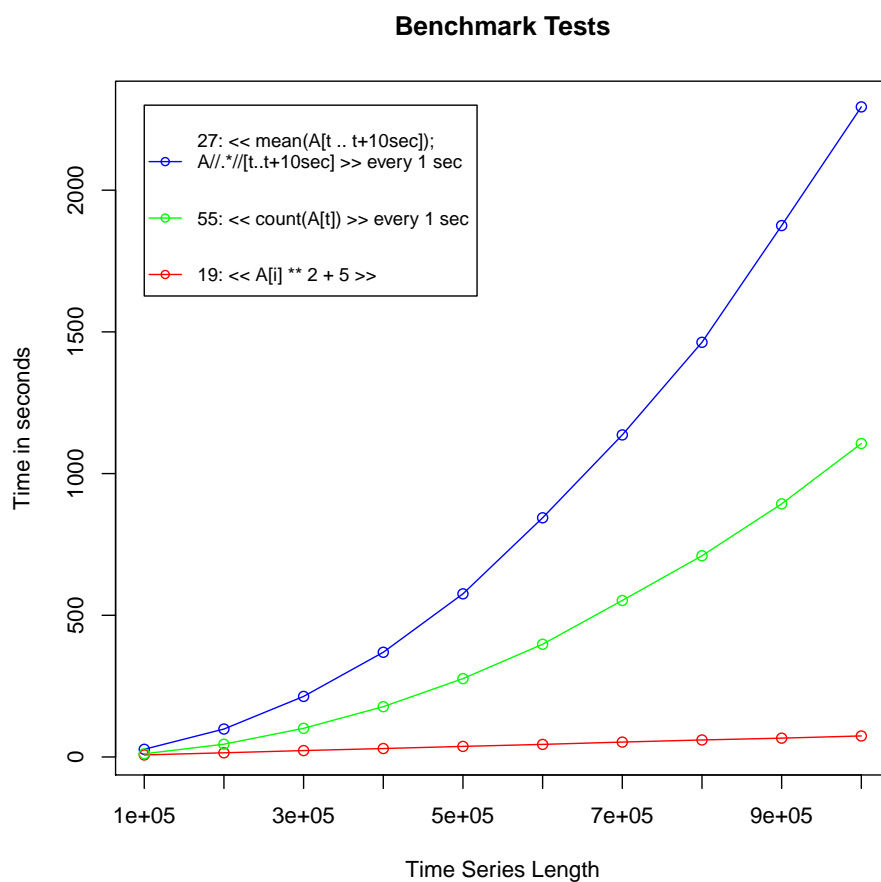


Figure 3.35: Execution time of representative expressions for the 3 main classes.

Figure 3.36 shows the different representatives of subclasses of class 3 expressions. The x-axis shows the number of slots and the y-axis the execution time in seconds. The legend shows the number and expression for each color in the graph.

#	Cl.	Expression	100k	500k	1000k
1	6	@A << A[i] >>	2.42	12.12	24.88
2	5	@A << A[/.*/[i] >>	3.82	19.78	39.97
3	7	@A << mean(A[t-10min..t]); A[/.*/[i] >> every 10 mins	0.06	0.31	5.22
4	6	@B @A << A[i] >>	2.25	12.49	23.85
5	6	@A @B << A[i] >>	2.35	11.84	24.80
6	3	@A << A[i]*2; A[i]*3 >>	8.53	43.09	86.65
8	4	@A @B << A[i] + B[i] >>	6.69	33.20	67.01
9	4	@A @B << A[i] - B[i] >>	6.61	34.18	67.76
10	4	@A @B << A[i] * B[i] >>	6.73	33.23	68.84
11	4	@A @B << A[i] / B[i] >>	6.58	33.74	68.72
12	4	@A @B << A[i] ** B[i] >>	6.60	33.89	66.86
13	6	@A << A[i+1] >>	2.27	11.86	24.76
14	6	@A << A[i-1] >>	2.24	11.71	23.86
15	5	@A << A[i] * 2 >>	4.48	23.65	46.18
16	5	@A << A[i] * 0.5 >>	4.79	24.43	48.94
17	5	@A << A[i] + 2 >>	4.49	23.19	45.77
18	5	@A << A[i] + 2.4 >>	4.72	24.26	49.39
19	4	@A << A[i] ** 2 + 5 >>	7.23	37.41	74.22
20	5	@A << -A[i] >>	3.76	18.82	37.43
21	5	@A << mean(A[i-1 .. i]) >>	4.67	22.51	46.33
22	5	@A << max(A[i-1 .. i]) >>	4.34	22.71	44.89
23	5	A << min(A[i-1 .. i]) >>	4.36	22.91	44.69
25	7	@A << mean(A[t .. t+1min]) >>	0.43	7.05	22.77
26	7	@A << mean(A[t-1min .. t]) >>	0.51	7.02	21.91
32	4	@A @B << A[i] and B[i] >>	6.59	32.90	66.82
33	4	@A @B << A[i] or B[i] >>	6.49	33.34	66.06
34	4	@A @B << A[i] xor B[i] >>	6.48	32.89	65.91
35	4	@A @B << (A[i] > B[i]) >>	6.47	32.90	65.81
36	4	@A @B << A[i] < B[i] >>	6.48	32.75	65.70
37	4	@A @B << A[i] >= B[i] >>	6.46	33.26	66.75
38	4	@A @B << A[i] <= B[i] >>	6.48	32.86	66.63
39	4	@A @B << A[i] == B[i] >>	6.47	33.29	65.77
40	4	@A @B << A[i] != B[i] >>	6.60	32.89	65.80
41	4	@A << A[i] if A[i] >= 3 otherwise 0 >>	7.15	36.17	72.43
42	2	@A @B << A[i] if A[i] < B[i] otherwise B[i] >>	11.52	58.58	117.33
43	4	@A << 1 if (A[i] > 1) otherwise 0 => value:value >>	6.81	34.36	69.68
44	2	@A << A[i]*2 if (A[i] > 2) otherwise A[i]**2 >>	12.36	62.20	124.85
45	1	@A @B @C << A[i] + C[i] if B[i] == A[i] otherwise B[i]*C[i] >>	18.72	89.52	177.12
46	4	@A << A[i] if (A[i] > 1) otherwise None >>	6.59	33.75	66.94
47	6	@A << A[i] => mean >>	2.18	11.60	23.51
48	5	@A << mean(A[i-1 .. i]) => mean >>	4.42	21.75	43.99
49	4	@A << "set" if (A[i] > 10) otherwise "reset" => command >>	7.34	35.92	70.43
50	4	@A << A[i]+log(3) => value >>	6.64	33.86	68.23
51	5	@A << log(A[i]) >>	4.21	20.99	41.94
52	5	@A << A[i] * 2; 10 => value.test >>	5.00	25.40	48.74
53	3	@A << A[i] * 2 >> << -[i] / 2 >>	9.36	46.72	93.47
54	5	@A << count(A[i]) >>	3.92	20.41	40.61
56	4	@A << A[i] if (A[i] > 2) otherwise None >> << count(.[i]) >>	6.68	33.50	67.10

Table 3.5: Overview and categorization of class 3 expressions.

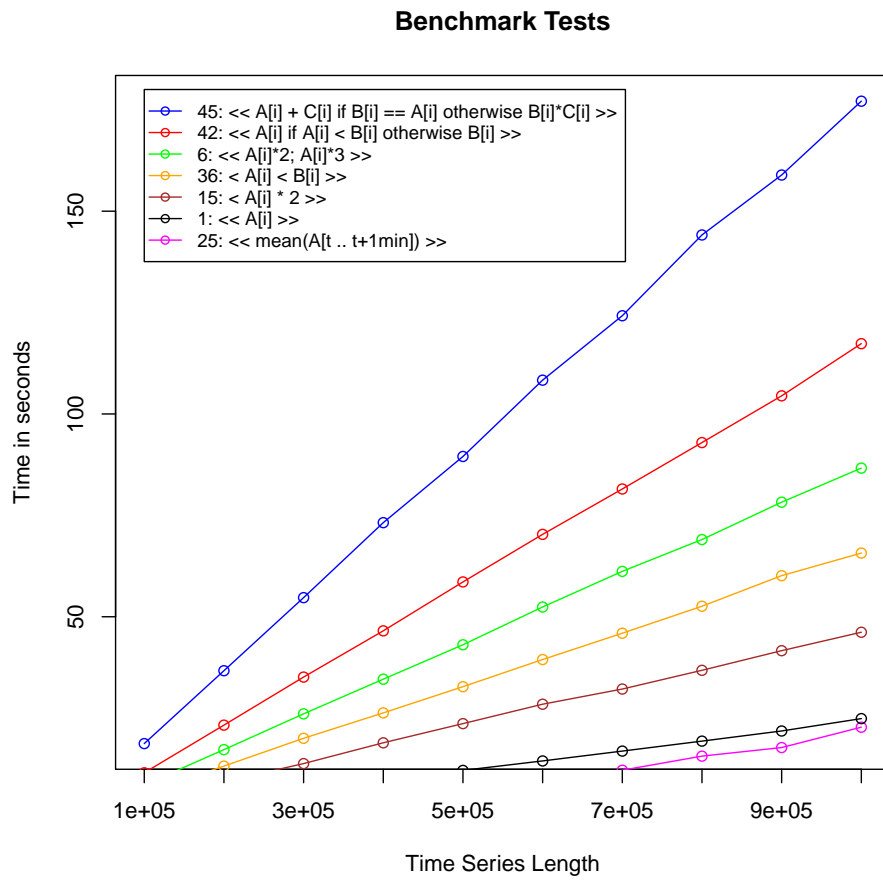


Figure 3.36: Execution time of the representative class 3 expressions for the 7 subclasses.

3.6 Profiling

The profiling of our implementation helps us to analyze the performance of time series processing in order to find bottlenecks. We have a look at method calls and how much execution time they take. Listing 3.9 shows the most time-consuming methods as generated by the Python cProfiler module which is part of the Python standard library. The first column shows the number of calls for the analyzed method, the second column the total time of execution of the method during the whole process runtime, the third column the time per call for the method, and the fourth column the name and location of the method.

```

1 12229933596 function calls (12229919887 primitive calls) in
   7207.406 seconds

2
3   Ordered by: standard name
4
5      ncalls   tottime  percall  filename:lineno(function)
6           1      1.895   1.895    Helper.py:125(constructTS)
7    19801980     2.451   0.000    Helper.py:48(get)
8    19801980     6.836   0.000    Helper.py:63(setSlot)
9    1000990     1.706   0.000    Helper.py:66(setValue)
10   19804955     3.545   0.000    Helper.py:79(getTSProperty)
11   39603960    24.220   0.000    Helper.py:93(getSlot)
12     1980   1152.844   0.582    Helper.py:99(slice)
13  2000818315   199.572   0.000    PythonTimeStamp.py:13(==)
14  2000818313   910.029   0.000    PythonTimeStamp.py:27(<)
15   101011880     8.511   0.000    PythonTimeStamp.py:34(hash)
16  5961783962   401.883   0.000    PythonTimeStamp.py:43(asMillis)
17     990      1.641   0.002    aggregates.py:61(calc)
18     1980    30.009   0.015    ast_interpreter.py:300
19                                     (physical_slice)
20   19801980     8.604   0.000    utils.py:138(getSlotFromTSAsDic)
21   19801980    26.897   0.000    utils.py:149(getSlotAsDic)
22  2000903859   115.956   0.000    {isinstance}
23     3962    27.178   0.007    {method 'keys' of 'dict' objects}
24     3961   717.660   0.181    {sorted}

```

Listing 3.9: Results of profiling the source code of the time series processor implementation.

According to the profiling results the `slice` method clearly takes the most processing time. This is a very clear signal for code optimization since other time consuming methods like `sorted`, `__lt__`, and `asMillis` are imported standard library methods and therefore cannot be improved by the integrator. Slicing is the procedure in our time series processor which is used to select a certain range of a time series. It is the implementation of the indexing and

time intensive because of the involvement of multiple future or past slots for every slot in a time series.

Apart from profiling, we also recorded the tracing to see which lines of code are executed most frequently. For this task we used the Python trace module. Listing 3.10 shows the results of the most often executed part of the interpreter. On the right side of the listing, the analyzed line of code is shown, and on the left side the number of calls to the corresponding line.

```

1 >>>>>> def physical_slice(self, tsNode, ts, t, leftInfo,
2         rightInfo):
3
4 1980:     leftOffset=leftInfo["offset"]
5 1980:     leftOffset=utils.normalizeTimeValue(leftOffset)
6
7 1980:     rightOffset=rightInfo["offset"]
8 1980:     rightOffset=utils.normalizeTimeValue(rightOffset)
9
10 1980:     tLeft=t.asMillis()+leftOffset[0]
11 1980:     tRight=t.asMillis()+rightOffset[0]
12
13 1980:     tsLeft=TimeStamp(tLeft)
14 1980:     tsRight=TimeStamp(tRight)
15
16
17 1980:     ti=TimeInterval(TimeInterval.Openness.CLOSED,
18         tsLeft, tsRight, TimeInterval.Openness.CLOSED)
19
20     # slicing of time series implementation
21 1980:     slice=ts.slice(ti)
22
23 1980:     resultSlice=[]
24
25 1980:     nodeType=tsNode[AST.TYPE]
26 1980:     if nodeType=="PROPCOPY":
27     990:         valueKey="temperature"
28     else:
29     990:         valueKey = utils.getValueKey(tsNode, ts)
30
31 1980:     allTS = slice.getTimeStampsArray()
32
33 19803960: for tStamp in allTS:
34 19801980:     slotDic=utils.getSlotFromTSAsDic(tStamp, ts)
35 19801980:     resultSlice.append((tStamp, slotDic))
36
37
38 1980:     leftOpenness=leftInfo[AST.TYPE]
39 1980:     rightOpenness=rightInfo[AST.TYPE]
40

```

```

41 1980:         if leftOpenness=="OPEN" and len(resultSlice)>0
42             and resultSlice[0][0]==tsLeft:
43 >>>>>>         del resultSlice[0]
44
45 1980:         if rightOpenness=="OPEN" and len(resultSlice)>0
46             and resultSlice[-1][0]==tsRight:
47 >>>>>>         del resultSlice[-1]
48
49             # Remove time stamps
50 19803960:         for i in xrange(len(resultSlice)):
51 19801980:             resultSlice[i]=resultSlice[i][1]
52
53 1980:         return (resultSlice , valueKey)

```

Listing 3.10: Results of tracing for the most frequently used part of the interpreter.

As we can see, the tracing shows as well that slicing is the most time consuming part of the source code. With nearly 20 million executions, the loop for gathering slots from the dictionary and appending result slices, as well as the loop for removing time stamps from slices shows potential for optimization. However, this is out of scope of this thesis and therefore will be covered in future publications.

Chapter 4

Semantic Framework

The Semantic Framework Prototype represents a complete solution for the implementation of Semantic Web applications. It consists of a set of components providing all needed tools to build web services, web portals, libraries, and applications enriched with Semantic Web technology.

In particular, the functionality of the framework is: *Semantic Repository* and *Connectors* (Data Access), *Ontology Mapping*, *Semantic Processing* (Reasoning), *Annotation*, and *User Interaction* (Web Portal).

The *Semantic Repository* is responsible for the storage and retrieval of semantically relevant resources metadata. These are time series metadata which could originate from any field of science and describe any aspect of a time series. The role of the *Semantic Repository* is to load a certain ontology into the knowledge base, as well as additional triples which represent information about time series. It interacts with the *Semantic Processor*, *Ontology Mapping*, and *Connectors* for loading new data to the store.

The *Connectors* are components which connect different data sources to our semantic framework. The most fundamental connector is the *Semantic Repository* which provides a connection to a knowledge base by the usage of specific libraries. Another example is an RDFa connector for collecting RDFa triples from web sites.

We use *Ontology Mapping* techniques to connect ontologies from different domains to each other. This is done by the implementation of a bridge ontology and enables the dynamic extension of our system with multiple domain ontologies which makes it possible to improve reasoning and extend the horizon for time series metadata.

The *Semantic Processing* component processes annotations and tags. It gets resources from the *Semantic Repository* component, associates annotations from the user to these resources, or generates annotations by itself. The component interacts with the *Ontology Mapping* component and with

the *Semantic Repository* to gain access to underlying data bases.

The *Annotation* component's responsibility is to add additional meta information to discovered Web resources. It can be triggered by the user, who provides meta information about the resource, or by the *Semantic Processor*, in which case the annotation is done automatically. The component interacts with a web portal, to allow annotations by users, and with the *Semantic Processor*, to allow automatic annotations.

User Interaction is implemented via a web portal. This approach allows us to present the user with customized portlets according to her specific group affiliation and interests. Furthermore, the portal approach makes resource discovery and annotation functionality available to the user in a natural and intuitive way.

Figure 4.1 shows a UML component diagram presenting the architecture of the Semantic Framework. The *Semantic Repository* retrieves RDF data from the *Connectors* and both components belong to the *Data Management* package. The *Semantic Repository* provides the retrieved triples to the *Semantic Processor* and *Ontology Mapping*, which are in the *Semantic Processing* package. The *Semantic Processor* consumes annotations from the *Annotation* component. *Ontology Mapping* and *Semantic Processor* provide semantic data to the *User Interaction* component, which also consumes annotations from the *Annotation* component and belongs together with the *Annotation* component to the *Visualization* package.

Parts of this chapter have already been published in Božić and Winiwarter [2013a].

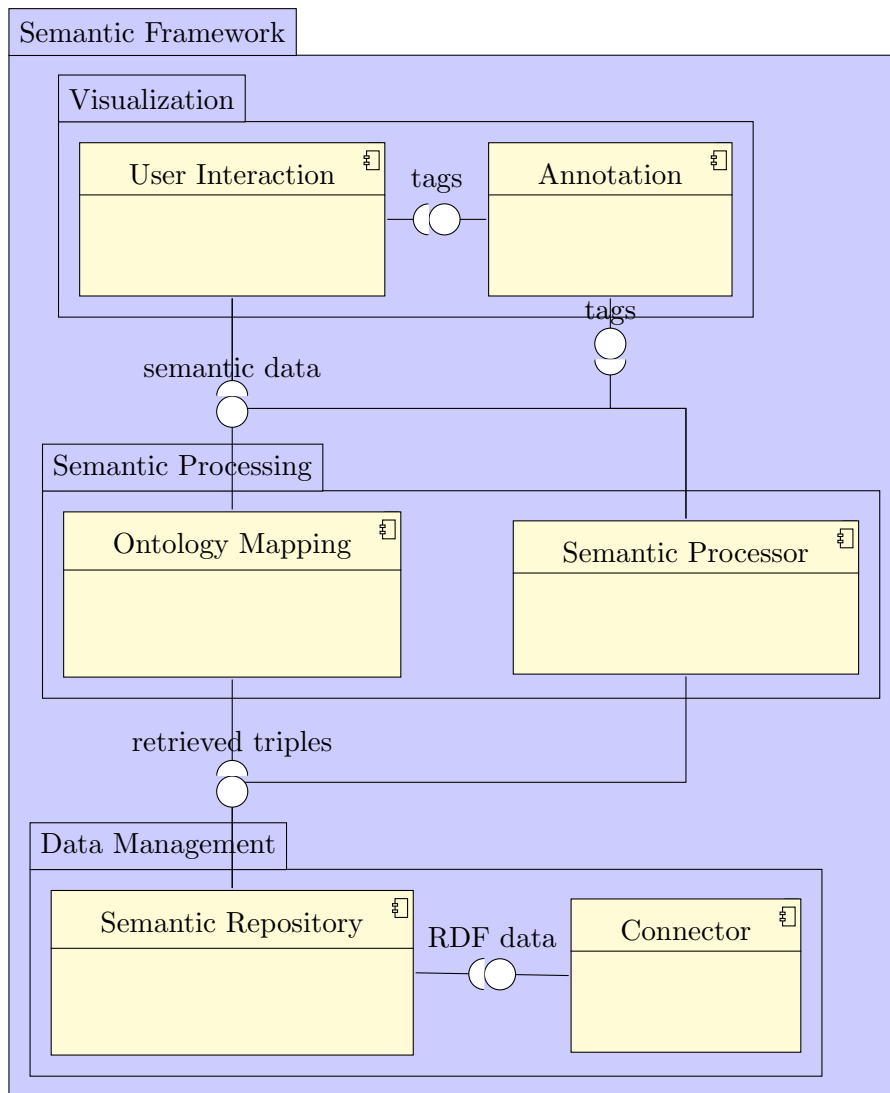


Figure 4.1: UML component diagram of the Semantic Framework.

4.1 Semantic Repository

The Semantic Repository is implemented by using RDFlib¹. RDFlib is a Python framework for building and managing a knowledge base (comparable to SESAME or the Jena API). We use a MySQL² database to make the knowledge base persistent.

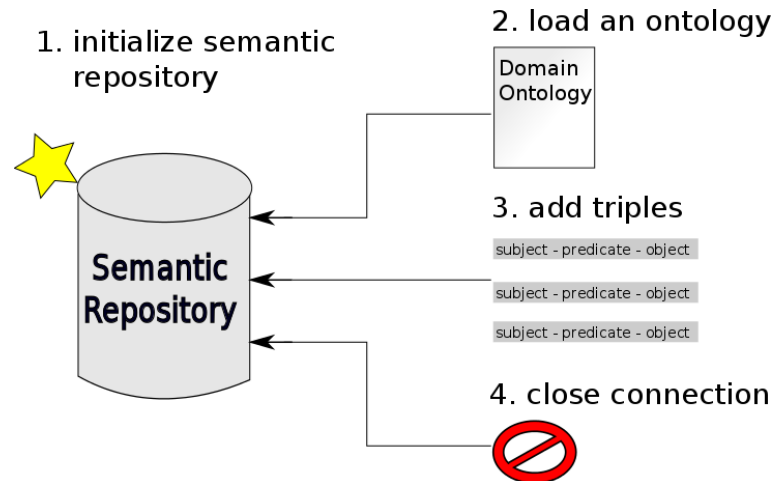


Figure 4.2: Typical workflow example for using the semantic repository.

A typical workflow example for the usage of the semantic repository is shown in Figure 4.2. The workflow consists of 4 steps which represent the main functionality of the component. In the first step a new semantic repository is initialized by calling the constructor of the class, the second step shows the loading of an ontology with already predefined triples, the third step represents adding new triples to the repository, and the fourth step closes the connection. Details of the implementation as well as some performance tests are presented in the following subsections.

4.1.1 Implementation

Listing 4.1 shows the initialization of the Semantic Repository.

```

1 def __init__(self):
2     self.store = plugin.get('SQLAlchemy', Store)
3     (identifier = self.ident)
4     self.g = Graph(self.store, identifier = self.ident)
  
```

¹<http://code.google.com/p/rdflib/>

²<http://www.mysql.com/>

```
5 self.g.open(self.uri, create = True)
```

Listing 4.1: Initialization of the Semantic Repository.

To initialize the Semantic Repository (which is implemented as a Python class), the constructor of the according class is called. At first, the SQLAlchemy plugin which belongs to RDFlib, is called and generates a store object. In the next line, this store is used to create a graph which is responsible for the triple management. Finally, the graph is opened by using the previously set MySQL URI.

In Listing 4.2 the process of loading an already existing ontology is shown. The `load_ontology` method simply calls the `load` method of the graph and passes the location of the ontology file, as well as a string describing the format, to it.

```
1 def load_ontology(self, input_graph, input_format):
2     self.g.load(input_graph, format=input_format)
```

Listing 4.2: Loading an ontology.

A code snippet from the method for adding new triples with data properties is shown in Listing 4.3. Here, we check which of the supported namespaces the predicate belongs to and add the triple (which has a literal object) to the graph. After that, we commit the change and serialize the graph in N-Triple format.

```
1 def add_data_triple(self, s, p, o):
2     if p.startswith('dc:'):
3         self.g.add((self.STSP[s[5:]], self.DC[p[3:]],
4                     Literal(o)))
5     self.g.commit()
6     self.g.serialize(format = 'nt')
```

Listing 4.3: Add a triple with data property.

Listing 4.4 shows how triples with object properties are added. Here, the predicate and object are checked to which of the supported namespaces they belong (if clause has been shortened). The triple is added to the graph which is then committed and serialized.

```
1 def add_object_triple(self, s, p, o):
2     s = self.STSP[s[5:]]
3
4     if p.startswith('dc:'):
5         p = self.DC[p[3:]]
6
7     if o.startswith('dc:'):
8         o = self.DC[o[3:]]
9
```

```

10     self.g.add((s, p, o))
11     self.g.serialize(format = 'nt')
12     self.g.commit()

```

Listing 4.4: Add a triple with object property.

The `get_triples` method is shown in Listing 4.5. It passes the SPARQL query string to the `query` method of the graph and returns the resulting graph to the caller.

```

1 def get_triples(self, query):
2     return self.g.query(query)

```

Listing 4.5: Get triples from repository.

The `close` method in Listing 4.6 is used to destroy the graph and, after that, to close the connection.

```

1 def close(self):
2     self.g.destroy(self.uri)
3     try:
4         self.g.close()
5     except:
6         pass

```

Listing 4.6: Close connection to repository.

The Semantic Repository is responsible for the storage of RDF triples for semantic time series. We use this component to feed in the ontology, to add new meta information about time series, and to retrieve time series triples. For example, a connector gathers an ontology from a web site via RDFa and passes it to the Semantic Repository which then stores the triples to the database.

4.1.2 Semantic Repository Benchmarks

Table 4.1 and Figure 4.3 summarize the results of the benchmark tests for the semantic repository tested on a Quad Core notebook with 8 GB of main memory. The benchmarks are tested for each relevant method of the repository where N represents the number of triples and each other column one of the called methods. The time has been measured in seconds.

The visualization of the benchmarks tests show that the methods `load_ontology`, `add_data_triple`, and `add_object_triple` perform nearly linear as expected, the reasonable limit for adding new triples at once is around 100,000, while `get_triples` performs very well up to 100,000 triples.

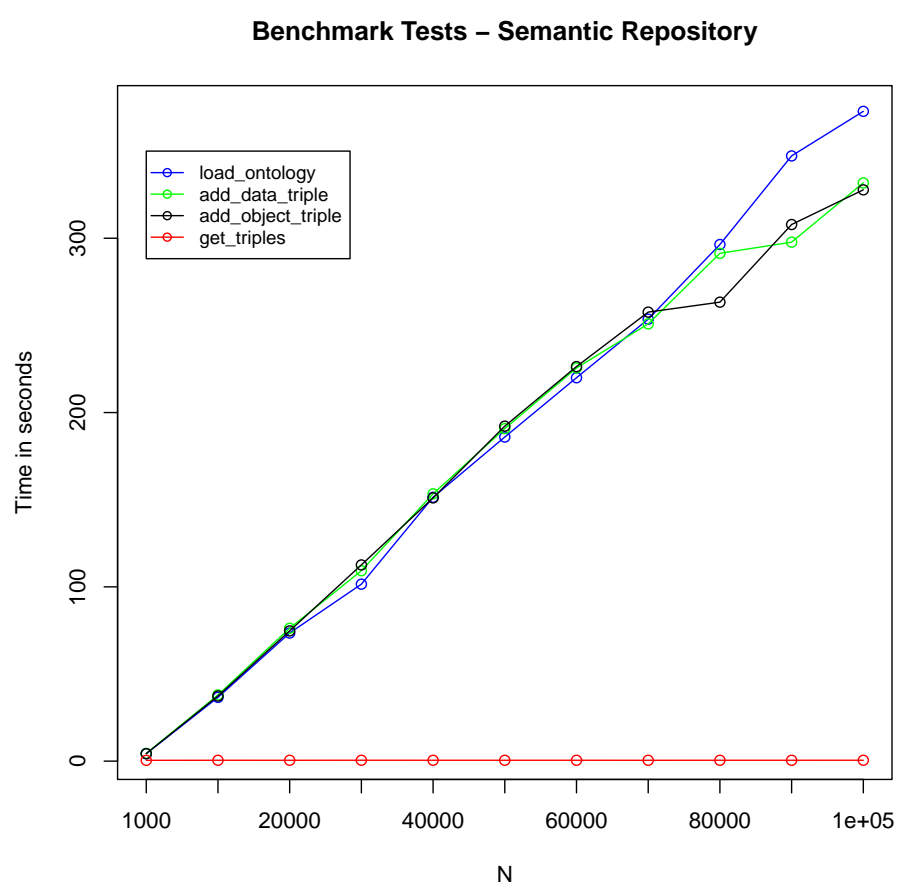


Figure 4.3: Visualized results of the benchmark tests.

N	load_ontology	add_data_triple	add_object_triple	get_triples
1,000	4.241s	4.407s	4.329s	0.471s
10,000	36.588s	37.980s	37.381s	0.497s
20,000	73.520s	76.238s	74.735s	0.493s
30,000	101.543s	109.259s	112.583s	0.511s
40,000	151.390s	153.371s	151.039s	0.499s
50,000	185.954s	190.788s	192.198s	0.498s
60,000	219.977s	225.573s	226.372s	0.502s
70,000	253.575s	250.861s	257.627s	0.496s
80,000	296.411s	291.349s	263.343s	0.503s
90,000	347.253s	297.775s	307.880s	0.500s
100,000	372.820s	331.760s	327.784s	0.515s

Table 4.1: Results of the benchmark tests for the semantic repository.

4.2 Connectors

Connectors are used to add new triples to our Semantic Repository. This process is called harvesting. We will show the Web Site Connector, which extracts RDFa triples from web sites and adds them to the repository, to demonstrate the principle of connectors. Since web sites do not yet provide regular RDF data, the Web Site Connector represents a kind of a compromise between HTML scraping and RDF. This means that the RDFa data is extracted out of HTML code and parsed into RDF triples. The following declarations explain the implementation of the Web Site Connector.

The Web Site Connector is implemented in Java and consists of the following subcomponents:

- Connector Interface: specifies the harvest method which every connector needs to implement.
- General Web Site Connector class: reads the list of web sites which need to be harvested and triggers the harvesting process.
- RDFa Parser class: Opens the stylesheet and implements the methods which are called every time the web site matches a certain stylesheet element.
- RDFa Parser stylesheet: Defines the RDFa elements for web sites to be parsed and integrated into the knowledge base.

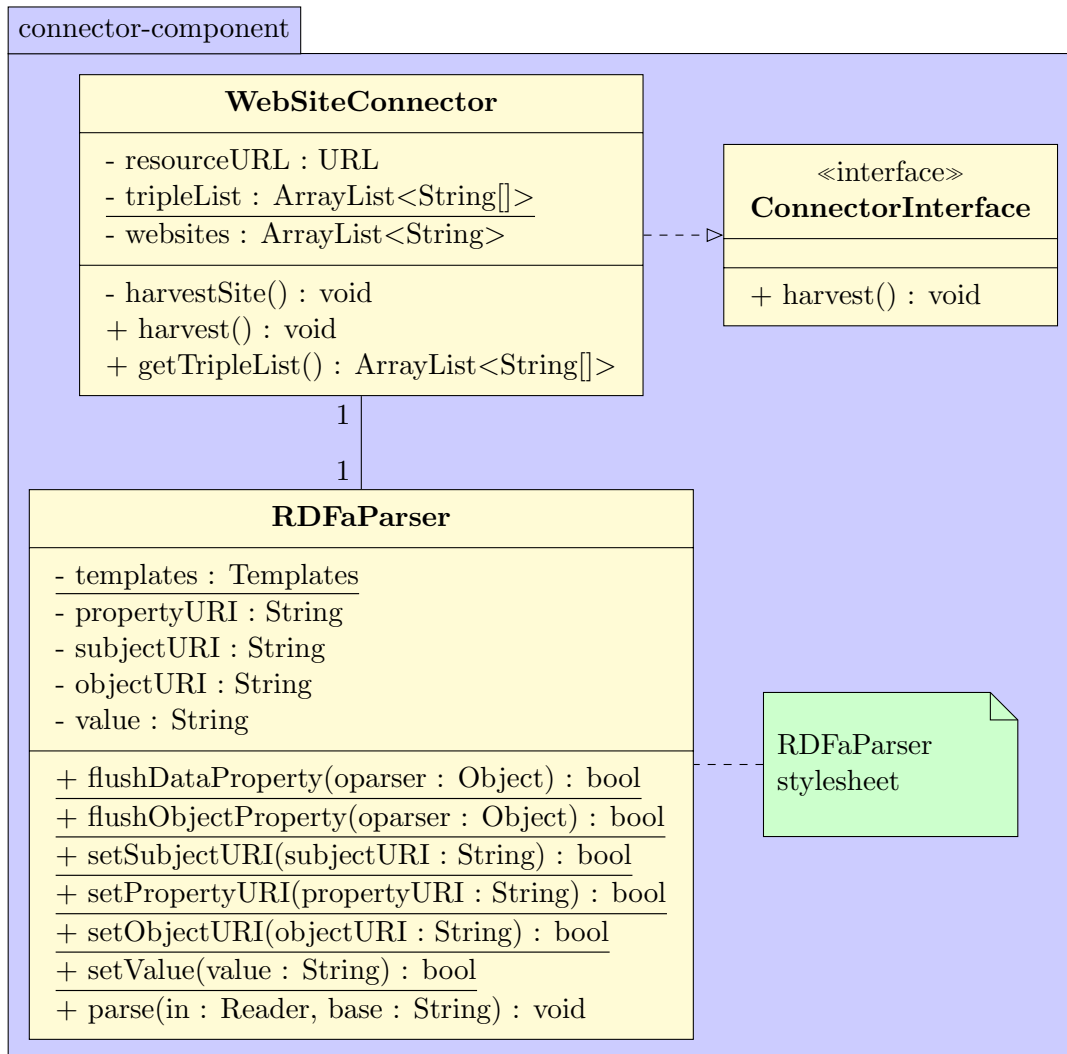


Figure 4.4: UML class diagram representing the architecture of the Web Site Connector component.

Figure 4.4 shows the architecture of the Web Site Connector. For clarity reasons, we illustrated only the most important methods and attributes in the RDFaParser class representation.

4.2.1 Connector Interface

The Connector interface simply defines the harvesting method which every single harvester needs to implement in order to be added to the Semantic Framework and to communicate with the Semantic Repository. Listing 4.7 shows the interface in Java code.

```

1 public interface ConnectorInterface extends Serializable {
2     void harvest() throws HarvestingException;
3 }

```

Listing 4.7: The Connector interface.

4.2.2 Web Site Connector Class

The Web Site Connector class is the main entry point to the connector. It implements the Connector interface and triggers the harvesting process. Listing 4.8 shows the constructor of the Web Site connector class which parses the XML document with a list of web sites to harvest and saves this list to the `websites` property of the class.

```

1 public WebSiteConnector() throws ParserConfigurationException ,
   SAXException , IOException {
2     DocumentBuilderFactory dbf = DocumentBuilderFactory.
3         newInstance();
4     DocumentBuilder db = dbf.newDocumentBuilder();
5     Document doc = db.parse(new File("src/main/resources/
input_web_sites.xml"));
6     doc.getDocumentElement().normalize();
7     NodeList sites = doc.getElementsByTagName("website");
8     for(int i = 0; i < sites.getLength(); i++) {
9         Node website = sites.item(i);
10        websites.add(website.getChildNodes().item(0).
11            getNodeValue());
12    }
13 }

```

Listing 4.8: Constructor of the Web Site Connector.

In Listing 4.9 the implementation of the Connector interface method `harvest` is shown. It goes through the list of web sites, checks whether the URL is valid, and calls the `harvest_site` method which does the harvesting work.

```

1 @Override
2 public void harvest() throws HarvestingException {
3     for(String url : this.websites) {
4         try {
5             this.resourceURL = new URL(url);
6         }
7         catch(MalformedURLException e){}
8         harvest_site();
9     }
10 }

```

Listing 4.9: Harvest method implementation.

Most of the logic is implemented in the `harvest_site` method (Listing 4.10 shows the most important code snippets of the method). This method creates a new parser and overrides the callback methods from the parser (these are called every time an RDFa triple is found) and adds the triple (object or data) to the list of triples. In the next step, the parser is started by passing the reader and URL base, and a new resource object is created with the resource URL. Finally, the resource is added to the annotation library (see “User Interaction and Annotation”).

```

1 private void harvest_site() throws HarvestingException {
2     RDFaParser parser = new RDFaParser() {
3         @Override
4         public void reportDataProperty(String subjectURI, String
5             subjectNodeID, String propertyURI, String value, String
6             datatype, String lang)
7         {
8             String[] triple = new String[3];
9             triple[0] = subjectURI;
10            triple[1] = propertyURI;
11            triple[2] = value;
12            WebSiteConnector.triple_list.add(triple);
13        }
14
15        @Override
16        public void reportObjectProperty(String subjectURI,
17            String subjectNodeID, String propertyURI, String
18            objectURI, String objectNodeID) {
19            String[] triple = new String[3];
20            triple[0] = subjectURI;
21            triple[1] = propertyURI;
22            triple[2] = objectURI;
23            WebSiteConnector.triple_list.add(triple);
24        }
25    };
26
27    parser.parse(reader, base);

```

```

28     BasicResource resource = new BasicResource();
29     resource.setURI(this.resourceURL.toString());
30
31     TaggingLibrary tl = new TaggingLibrary("http://localhost
:8080/openrdf-sesame", "database", "user", "password");
32     tl.connect();
33     tl.addResource(resource, "en");
34 }

```

Listing 4.10: Implementation of the harvest_site method.

Additionally, there is a method for returning the collected triples as a String array (see Listing 4.11).

```

1 public ArrayList<String[]> getTripleList() {
2     return WebSiteConnector.triple_list;
3 }

```

Listing 4.11: Method for returning the list of harvested triples.

4.2.3 RDFa Parser Class

The RDFa parser class is responsible for the initialization of the stylesheet and for listening to the callback methods. Listing 4.12 shows the parse method which is the main method of the class. It loads the stylesheet by loading the XSL file, creates a transformer factory, and sets the parsing templates. Furthermore, the method creates a transformer instance, sets the parameters, and transforms the input stream. After this procedure, the RDFa parser class listens to method calls caused by transformation results.

```

1 public void parse(Reader in, String base) throws IOException,
TransformerException, TransformerConfigurationException {
2     if(RDFaParser.templates == null) {
3         StreamSource source = new StreamSource(this.getClass().
4             getClassLoader().getResource("RDFaParser.xsl").
5             openStream());
6         TransformerFactory factory = TransformerFactory.
7             newInstance();
8         RDFaParser.templates = factory.newTemplates(source);
9     }
10
11     Transformer transformer = RDFaParser.templates.
12         newTransformer();
13     transformer.setParameter("parser", this);
14     transformer.setParameter("url", base);
15     transformer.transform(new StreamSource(in), new StreamResult(
        new OutputStream() { public void write(int b) throws
        IOException {} }));

```

16 }

Listing 4.12: The parse method.

In Listing 4.13 the `flushDataProperty` method is presented. The method is called every time the parser finds an RDFa data property triple. We do not need certain attributes set at this point, therefore the attributes are all set to `null`. The only necessary action here is to call the `reportDataProperty` method and pass the RDFa information.

```

1 public static boolean flushDataProperty(Object oparser) {
2     RDFaParser parser = (RDFaParser) oparser;
3     parser.reportDataProperty(parser.subjectURI,
4         parser.subjectNodeID, parser.propertyURI, parser.value,
5         parser.dataType, parser.language);
6     parser.propertyURI = null;
7     parser.subjectURI = null;
8     parser.subjectNodeID = null;
9     parser.objectURI = null;
10    parser.objectNodeID = null;
11    parser.value = null;
12    parser.dataType = null;
13    parser.language = null;
14
15    return true;
16 }
```

Listing 4.13: Method for flushing data property triples.

The method for flushing object property triples is shown in Listing 4.14. It is called every time the parser finds an RDFa object property triple. We do not need certain attributes set at this point, therefore the attributes are all set to `null`. The only necessary action here is to call the `reportObjectProperty` method and pass the RDFa information.

```

1 public static boolean flushObjectProperty(Object oparser) {
2     RDFaParser parser = (RDFaParser) oparser;
3     parser.reportObjectProperty(parser.subjectURI,
4         parser.subjectNodeID, parser.propertyURI,
5         parser.objectURI, parser.objectNodeID);
6     parser.propertyURI = null;
7     parser.subjectURI = null;
8     parser.subjectNodeID = null;
9     parser.objectURI = null;
10    parser.objectNodeID = null;
11    parser.value = null;
12    parser.dataType = null;
13    parser.language = null;
14
15    return true;
```

```
16 }
```

Listing 4.14: Method for flushing object property triples.

The code in Listing 4.15 shows the `reportDataProperty` and `reportObjectProperty` methods which are empty in order to be overridden by the connector.

```
1 public void reportDataProperty(String subjectURI, String
    subjectNodeID, String propertyURI, String value, String
    dataType, String lang) {}
2 public void reportObjectProperty(String subjectURI, String
    subjectNodeID, String propertyURI, String objectURI, String
    objectNodeID) {}
```

Listing 4.15: Methods for reporting data and object property triples.

The methods in Listing 4.16 are used to set certain attributes which are parsed. These are the data type, the language, the object node ID, the object URI, the property URI, the subject node ID, the subject URI, and the value.

```
1 public static boolean setDataType(Object parser, String dataType
    ) {
2     ((RDFaParser)parser).dataType = dataType;
3     return true;
4 }
5
6 public static boolean setLanguage(Object parser, String language
    ) {
7     ((RDFaParser)parser).language = language;
8     return true;
9 }
10
11 public static boolean setObjectNodeID(Object parser, String
    objectNodeID) {
12     ((RDFaParser)parser).objectNodeID = objectNodeID;
13     return true;
14 }
15
16 public static boolean setObjectURI(Object parser, String
    objectURI) {
17     ((RDFaParser)parser).objectURI = objectURI;
18     return true;
19 }
20
21 public static boolean setPropertyURI(Object parser, String
    propertyURI) {
22     ((RDFaParser)parser).propertyURI = propertyURI;
23     return true;
24 }
25
```

```

26 public static boolean setSubjectNodeID(Object parser, String
    subjectNodeID) {
27     ((RDFaParser)parser).subjectNodeID = subjectNodeID;
28     return true;
29 }
30
31 public static boolean setSubjectURI(Object parser, String
    subjectURI) {
32     ((RDFaParser)parser).subjectURI = subjectURI;
33     return true;
34 }
35
36 public static boolean setValue(Object parser, String value) {
37     ((RDFaParser)parser).value = value;
38     return true;
39 }

```

Listing 4.16: Methods for setting the attributes.

4.2.4 RDFa Parser Stylesheet

The most important parts of the used XSL stylesheet are presented in the following listings. The whole stylesheet can be found in the Appendix.

Listing 4.17 shows how RDFa elements are matched. It searches for attribute properties and returns the namespace for an object property. The template needs to find the right namespace prefix and selects the default vocabulary.

```

1 <!-- match RDFa element -->
2 <template match="*[attribute::property or attribute::rel or
    attribute::rev or attribute::typeof]" mode="rdf2rdfxml" >
3 <!-- returns the namespace for an object property -->
4 <template name="get-relrev-ns">
5     <param name="qname" />
6     <variable name="ns_prefix" select="substring-before(translate
    ($qname, '[]', ' '), ':')"/>
7     <choose>
8         <when test="string-length($ns_prefix)>0">
9             <call-template name="return-ns">
10                 <with-param name="qname" select="$qname"/>
11             </call-template>
12         </when>
13         <!-- returns default_voc if the predicate is a reserved value
            -->
14         <otherwise>
15             <variable name="is-reserved">
16                 <call-template name="check-reserved">
17                     <with-param name="nonprefixed">

```

```

18         <call-template name="no-leading-colon">
19             <with-param name="name" select="$qname" />
20         </call-template>
21     </with-param>
22 </call-template>
23 </variable>
24 <if test="$is-reserved='true'">
25     <value-of select="$default_voc" />
26 </if>
27 </otherwise>
28 </choose>
29 </template>

```

Listing 4.17: Matching an RDFa element.

Listing 4.18 demonstrates the generation of RDF statements. The parameter names are set to subject, predicate, object, datatype, attribute, and language; and the namespace and names of the predicate are set. The main part of the generation is choosing the elements of the statement. The predicate, subject, language, datatype, and value are selected and the corresponding setter methods of the parser class are called. The datatype could be not an XML literal, the content could be in an attribute or in the text nodes of the element, and the datatype could be omitted. The last action is the call to the `flushDataProperty` method with a parser object. This has to be done in order to send the RDF data to the RDFa parser.

```

1 <!-- generate an RDF statement -->
2 <template name="property" >
3     <param name="subject" />
4     <param name="predicate" />
5     <param name="object" />
6     <param name="datatype" />
7     <param name="attrib" />
8     <param name="language" />
9
10    <!-- get namespace and name of the predicate -->
11    <variable name="predicate-ns">
12        <call-template name="get-property-ns">
13            <with-param name="qname" select="$single-predicate" />
14        </call-template>
15    </variable>
16
17    <variable name="predicate-name">
18        <call-template name="get-predicate-name">
19            <with-param name="qname" select="$single-predicate" />
20        </call-template>
21    </variable>
22
23    <choose>

```



```

24 <when test="string-length($predicate-ns)>0">
25   <!-- there is a known namespace for the predicate -->
26   <value-of select="java:websiteconnector.RDfaParser.
    setPropertyURI($parser,$predicate-name)"/>
27 </choose>
28   <when test="starts-with($subject,'blank:nod: ')">
29     <value-of select="java:websiteconnector.RDfaParser.
    setSubjectNodeID($parser,substring-after($subject,'
    blank:node: '))"/>
30   </when>
31   <otherwise> <value-of select="java:websiteconnector.
    RDfaParser.setSubjectURI($parser,$subject)"/> </otherwise>
32 </choose>
33 <if test="string-length($language)>0">
34   <value-of select="java:websiteconnector.RDfaParser.
    setLanguage($parser,$language)"/>
35 </if>
36 <choose>
37   <when test="$datatype='http://www.w3.org/1999/02/22-
    rdf-syntax-ns#XMLLiteral'">
38   <choose>
39     <when test="$attrib='true'"> <!-- content is in an
    attribute -->
40     <value-of select="java:websiteconnector.RDfaParser.
    setDataType($parser,$datatype)"/>
41     <value-of select="java:websiteconnector.RDfaParser.
    setValue($parser,normalize-space(string($object)))/>
42     </when>
43     <otherwise> <!-- content is in the element and may include
    some tags -->
44     <value-of select="java:websiteconnector.RDfaParser.
    setDataType($parser,$datatype)"/>
45     <value-of select="java:websiteconnector.RDfaParser.
    setValue($parser,$object)"/>
46     </otherwise>
47   </choose>
48   </when>
49   <when test="string-length($datatype)>0">
50     <!-- there is a datatype other than XMLLiteral -->
51     <value-of select="java:websiteconnector.RDfaParser.
    setDataType($parser,$datatype)"/>
52   <choose>
53     <when test="$attrib='true'"> <!-- content is in an
    attribute -->
54     <value-of select="java:websiteconnector.RDfaParser.
    setValue($parser,normalize-space(string($object)))/>
55     </when>
56     <otherwise> <!-- content is in the text nodes of the
    element -->

```

```

57     <value-of select="java:websiteconnector.RDFAParser.
setValue($parser,$object)"/>
58     </otherwise>
59 </choose>
60 </when>
61 <otherwise> <!-- there is no datatype -->
62     <choose>
63     <when test="$attrib='true'"> <!-- content is in an
attribute -->
64     <value-of select="java:websiteconnector.RDFAParser.
setValue($parser,normalize-space(string($object))"/>
65     </when>
66     <otherwise> <!-- content is in the text nodes of the
element -->
67     <value-of select="java:websiteconnector.RDFAParser.
setValue($parser,$object)"/>
68     </otherwise>
69 </choose>
70 </otherwise>
71 </choose>
72 </when>
73 </choose>
74 <value-of select="java:websiteconnector.RDFAParser.
flushDataProperty($parser)"/>
75 </template>

```

Listing 4.18: Generation of an RDF statement.

4.2.5 Harvesting Benchmarks

After explaining the implementation and source code of the component, this section provides an example of usage of the connector, as well as benchmarks for harvesting web sites. The Web Site Connector uses an XML file providing a list of web sites to be harvested. For the benchmark tests we used the web site of the TaToo project³. A typical use case for the connector consists of the following steps:

1. Provide a list of web sites to be harvested in the XML file (input_web_sites.xml).
2. Create an instance of the `WebSiteConnector` class.
3. Call the `harvest()` method.

³<http://www.tattoo-fp7.eu/tatooweb/>

Number of calls to website	Number of har- vested triples	Time in milliseconds
100	2,900	62,236
200	5,800	124,739
300	8,700	174,902
400	11,600	228,480
500	14,500	279,981
600	17,400	346,914
700	20,300	393,402
800	23,200	456,925
900	26,100	514,100
1000	29,000	560,990

Table 4.2: Benchmark results for harvesting websites.

4. Collect the results by calling the `getTripleList()` method (the output is a two-dimensional matrix representing a list of triples).

Table 4.2 and Figure 4.5 show the results of the harvesting benchmark tests. The performance for 100 to 1000 websites with up to 29,000 triples is nearly linear and has a reasonable time consumption of 560,990 milliseconds for 29,000 triples.

The semantic repository and harvesting components are essential for meta data management and collection. Therefore, we started by explaining them and their implementation together with some benchmark tests. The next step in the framework development is the mapping of different ontologies as well as reasoning with existing triples to generate new meta data. Since this is an important issue, the next sections present a thorough analysis of already existing solutions and explanations of our approach to tackle these problems.

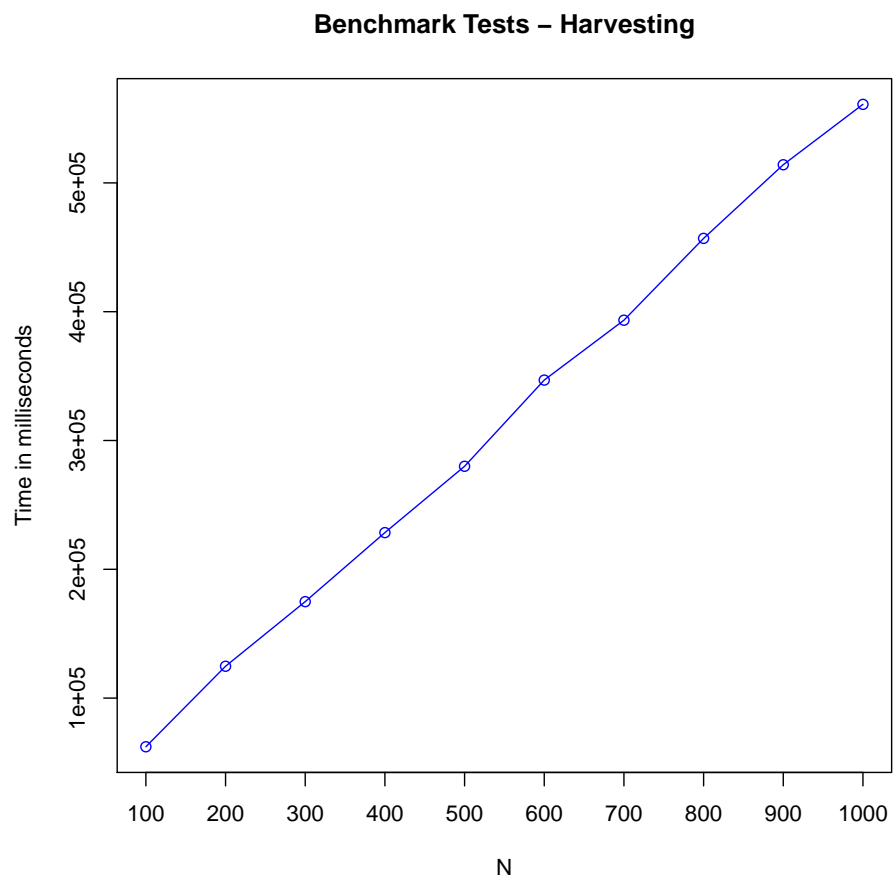


Figure 4.5: Vizualized results of the harvesting benchmark test.

4.3 Ontology Mapping

The main task of ontology mapping is to unite two vocabularies of two ontologies, which are related to a certain degree. This means that the expected output is one common ontology, which contains both vocabularies and has the whole power of expression. Therefore, the two ontologies need a common domain of discourse. In our case this common domain is time series processing. Furthermore, the related ontologies need to respect mathematical structure and ontological axioms. They are populated with contents specific to their community. The mapping is done by assigning symbols of one ontology to symbols of the other. Alternatively, new relations between the ontologies can be built. This process is then called ontology alignment.

Our approach for ontology mapping is the usage of a so-called *semantic bridge ontology (SBO)*. To construct such a bridge ontology, first of all it is necessary to accumulate information and populate an ontology of mapping constructs. This means that the first step is to find common items for the ontologies which need to be mapped. Hereby, existing constructs should be maintained and exploited, new additional constructs should be minimized, and the acceptance of general semantic web tools and vocabularies maximized.

Our workflow for building such a bridge ontology is the following:

1. Analyze the ontologies that should be mapped and find common entities.
2. Generate a common ontology with time series processing vocabulary.
3. Define mappings and relations between entities of the ontologies.
4. Analyze the resulting ontology and improve efficiency of relations and links.

Possible relations of two attributes from two ontologies (a_1 and a_2) are: a_1 is equivalent to a_2 , a_1 includes a_2 , and a_1 is disjoint with a_2 . In the case of mapping of local ontologies, which is our point of interest since domain ontologies are local ontologies, the advantages are the interoperability enablement, highly dynamic and distributed environments and mediation between distributed data.

We have collected some requirements which are needed to map semantic time series ontologies and had a look into some popular ontology mapping tools and frameworks. Table 4.3 presents an overview of the results. There are three types of ontology mapping frameworks: ontology mapping of a local and global ontology (global), ontology mapping of two local ontologies (local),

and ontology merging and alignment (merging and alignment). Since domain ontologies are local ontologies the first requirement is that two local ontologies should be mapped. In our table we called the kind of ontologies which should be mapped the *approach*. The next requirement is the *technique*, which means how the mapping is being done for source and destination ontologies. We found out that the best technique for mapping of semantic time series ontologies would be to use a bridge. Since our ontology mapping should be visualized to the user and user interaction is required, we need a *user interface* as well. Finally, we have the requirement of a *service-oriented* architecture, because we want to implement web services which use ontology mapping technology. In Table 4.3 we used the color red to signalize that a requirement is not met, the color yellow to signalize that it is only partially met, and green to signalize that it is fully met.

Name	Approach	Technique	User Interface	Service-oriented
CROSI Mapping System (CMS)	merging and alignment	alignment	command line	no
CTXMATCH	local	logical deduction	GUI	no
CHIMAERA	merging and alignment	linguistic matches	web GUI	yes
FCA-Merge	merging and alignment	linguistic analysis	GUI	no
GLUE	local	machine learning	no	no
Learning Source Description (LSD)	global	machine learning	no	no
Lexicon-based Ontology Mapping (LOM)	local	lexical similarity	Prolog API	no
Mediator Environment for Multiple Information Sources (MOMIS)	global	name equality	GUI	no
Ontology Mapping Enhancer (OMEN)	local	Bayesian Net	framework	no
Ontology Mapping Framework (MAFRA)	local	bridge	GUI	yes
OntoMorph	merging and alignment	pattern matching	Protégé plugin	no
PROMPT	merging and alignment	heuristic-based analyzer	Protégé plugin	no
Quick Ontology Mapping (QOM)	local	dynamic programming approach	toolbox	no
SMART	merging and alignment	linguistic similarity	tool	no

Table 4.3: Ontology mapping tools overview according to our requirements.

We investigated 14 different ontology mapping tools and frameworks. All of them are popular and widely used in the Semantic Web community, but also in some fields of application. Our idea was to pick one which covers all aspects important for our work, and implement a similar approach. Table 4.3 lists all ontology mapping tools we shed a light on in alphabetical order.

- The *CROSI*⁴ project produced a survey on the state-of-the-art of semantic integration systems, a framework for characterizing semantic integration systems, an architecture for developing semantic integration systems tuned to ontology mapping systems, and a dedicated ontology mapping system (CMS - CROSI Mapping System). CMS does not use a bridge ontology, but merging and alignment techniques. It can be used via command line or Win32 interface and is not service-oriented.
- *CTXMATCH* is in principle only an algorithm, but there is also an application which implements the algorithm⁵. However, the contextmatch application uses a local approach, this means that the mapped ontologies are both local (domain) ontologies. The used technique is logical deduction, the application provides a GUI, but is not service-oriented.
- *CHIMAERA* is a software system for creating and maintaining distributed ontologies on the web and supports merging multiple ontologies together and diagnosing individual or multiple ontologies⁶. The approach is ontology merging and alignment, to fulfill this task it uses linguistic matches. Furthermore, CHIMAERA has a web UI and is service-oriented.
- *FCA-Merge*⁷ is a new method for merging ontologies following a bottom-up approach and offering a global structural description of the merging process. It performs merging and alignment of ontologies by using linguistic analysis. FCA-Merge has an implementation with a GUI, which is not service-oriented.
- *GLUE*⁸ is a system that employs learning techniques to semi-automatically create semantic mappings between ontologies. It uses machine learning methods for mapping local ontologies. GLUE has no UI and is not service-oriented.

⁴<http://www.aktors.org/crosi/>

⁵<http://sun.aei.polsl.pl/~niedbyk/oaei2006/>

⁶<http://www.ksl.stanford.edu/software/chimaera/>

⁷<http://disi.unitn.it/~accord/RelatedWork/Matching/FCA01.pdf>

⁸<http://homes.cs.washington.edu/~pedrod/papers/hois.pdf>

- *Learning Source Description (LSD)*⁹ applies a machine learning approach as well; this time only for global and local ontologies. It has no user interface and is not service-oriented.
- *Lexicon-based Ontology Mapping (LOM)* is a semi-automatic lexicon-based ontology mapping tool that supports a human mapping engineer with a first-cut comparison of ontological terms between the ontologies to be mapped, based on their lexical similarity¹⁰. It is used for local ontologies and applies mappings based on lexical similarity. LOM can be used through a Prolog API and its architecture is not service-oriented.
- *Mediator Environment for Multiple Information Sources (MOMIS)*¹¹ is a framework for information extraction and integration from structured and semi-structured data sources, which is used for mapping of a global and a local ontology and based on name equality. It has a GUI, but is not service-oriented.
- *Ontology Mapping Enhancer (OMEN)*¹² is based on a set of meta-rules that captures the influence of the ontology structure and the existing matches to match nodes that are neighbours to matched nodes in the two ontologies. OMEN is used for mapping of local ontologies and based on a Bayesian Net. From user perspective, it is a non-service-oriented framework.
- *Ontology Mapping Framework (MAFRA)*¹³ allows to create semantic relations between two (source and target) ontologies, and translate source ontology instances into target ontology instances. MAFRA fulfills all our requirements, since it works on two local (domain) ontologies, uses the bridge technique, has a graphical user interface and is service-oriented.
- *OntoMorph*¹⁴ provides a powerful rule language to represent complex syntactic transformations and a rule interpreter to apply them to arbitrary knowledge representation language expressions. It is used for merging and alignment of ontologies, based on pattern matching, presents itself to the user as a Protégé plugin and is not service-oriented.

⁹<http://www.cs.wisc.edu/~anhai/talks/sigmod01-talk.ppt>

¹⁰http://www.daml.org/tools/#ont_mapping

¹¹<http://www.dbgroup.unimo.it/Momis/>

¹²http://link.springer.com/chapter/10.1007\%2F11574620_39

¹³<http://mafra-toolkit.sourceforge.net/>

¹⁴<http://www.isi.edu/~hans/ontomorph/presentation/ontomorph.html>

- *PROMPT*¹⁵ allows users to compare versions of an ontology, move frames between projects, merge two ontologies into one and extract parts of an ontology. It is developed for merging and alignment of ontologies, implemented as a heuristic-based analyzer, can be used as a Protégé plugin and is not service-oriented.
- *Quick Ontology Mapping*¹⁶ is a trade off between efficiency and effectiveness. The approach is to have slightly less quality but therefore by far more performance. QOM is used to map local ontologies, uses a dynamic programming approach, provides tools for users and is not service-oriented.
- *SMART*¹⁷ is a specialized tool for ontology mapping and alignment, which uses an algorithm based on a set of ontology merging and alignment operations. SMART is based on linguistic similarity and not service-oriented.

After analyzing all listed technologies and frameworks, we decided that the MAFRA approach would come very close to meeting our needs. However, the goal was not to use an already existing framework, but to apply such an approach for ontology mapping in our own environment. Therefore we developed our own bridge ontology, which should have a GUI and a service-oriented architecture for local ontologies.

Our first step for building a bridge ontology was to define all common classes and properties, which are valid for all possible time series ontologies. This is then the bridge ontology and can be used as a common interface for all domain ontologies. Every individual domain ontology needs to inherit from the bridge ontology, this means that it has to define all classes which are also defined in the bridge ontology. Therefore, an ontology graph can be constructed, which has the bridge ontology in the center (see Figure 4.6).

Figure 4.6 shows the relation between a bridge ontology and domain ontologies. The idea is that the bridge ontology defines all concepts (classes and properties) which are common to all domain ontologies. The main part is about describing the concept of time series. The bridge ontology is the center point in the ontology architecture and unites all ontologies. This concept is similar to the picture of *ambassadors* of different countries and a *translator*, who is able to speak all languages and therefore makes interaction

¹⁵<http://protege.stanford.edu/plugins/prompt/prompt.html>

¹⁶<http://www.scs.carleton.ca/~armyunis/knowledge-managment/papers/QOM-Quick%20Ontology%20Mapping.pdf>

¹⁷<http://protege.stanford.edu/publications/OntologiesAndTools/tsld022.htm>

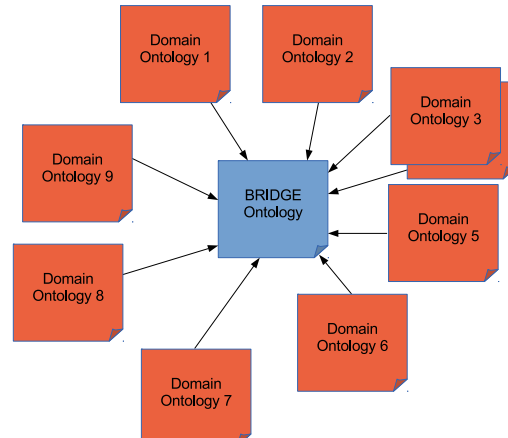


Figure 4.6: Relation between bridge ontology and domain ontologies.

and communication between every single ambassador possible. We call this concept *domain-bridge-domain* or *ambassador-translator-ambassador* communication.

During the mapping approach a domain ontology which defines all classes and properties from the bridge ontology (same URIs) is created. The ontology is extended according to the needs of the domain, new class definitions with URIs and properties are added, and classes from the bridge ontology are connected. When the ontology is loaded to the semantic time series processor, it delivers resulting time series from related domains as well.

The structure of our bridge ontology is illustrated in Figure 4.7. We developed the bridge ontology in Protégé¹⁸ and produced the illustration in Figure 4.7 with RDF Gravity¹⁹. The idea was, as already mentioned, to collect all common classes and properties of semantic time series and model them into a bridge ontology, which can then be used to map all domain ontologies to each other by enforcing the definition of bridge classes in every domain ontology. Figure 4.7 shows the classes and properties of our bridge ontology. *Classes* are shown in blue rectangles with a yellow circle and the letter “C”, whereas *properties* are shown in red rectangles with a dark red triangle and the letter “P”.

The following classes have been defined in the bridge ontology:

- **TimeSeries:** the `TimeSeries` class represents one time series in the system. The time series contains data which is used by a certain domain user for her specific area of interest.

¹⁸<http://protege.stanford.edu/>

¹⁹<http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html>

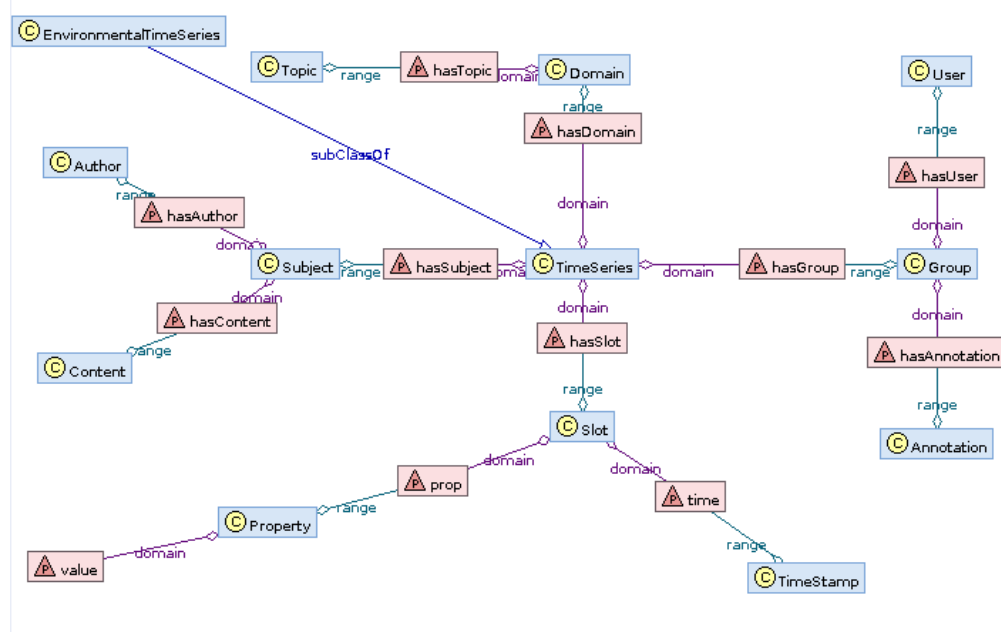


Figure 4.7: Bridge ontology for Semantic Time Series Processing (created in Protégé and visualized in RDF Gravity).

- **Environmental Time Series:** the `EnvironmentalTimeSeries` class is a special time series. It represents obviously time series data for the environmental area (e.g. water quality, air quality, etc.).
- **Domain:** the `Domain` class contains the domain of a time series. A time series can also belong to more than one domain, but it has to have at least one.
- **Topic:** the topic of a domain is represented by the `Topic` class. It specifies which topic a certain domain belongs to, e.g. “Water Quality”.
- **Subject:** the `Subject` class represents a subtopic for a time series (e.g. if a time series has the topic “Air Quality”, a possible subject could be “Ozone Concentration”).
- **Author:** this class represents the `Author` of a subject.
- **Content:** the `Content` class represents a description of the subtopic (subject).
- **Group:** this class represents the user group of the domain where the time series belongs to. In the case of multiple domains, there are also

multiple user groups.

- **User:** the `User` class represents all users, who are interested in the time series and belong to the right group.
- **Annotation:** represents all annotations generated by a group of interest for the time series.
- **Slot:** the slot is a part of a time series, in fact it is exactly one measurement (which can consist of several properties or values, of course).
- **TimeStamp:** every slot has exactly one time stamp, which represents the point in time when the measurement took place.
- **Property:** a property contains a measurement value and belongs to a certain slot. While every property belongs to exactly one slot, a slot can have an arbitrary number of properties.

4.4 Semantic Processing

In our understanding, reasoning is the generation of new RDF triples from already existing ones. The interesting part for our project is reasoning with multiple ontologies interrelated with semantic mappings. This means that we want to reason (or infer new RDF triples) between two domain ontologies by using the bridge ontology.

Generally, reasoning is based on description logic. Ontologies correspond to description logic theories (T-boxes) and semantic mappings correspond to bridge rules. T-boxes contain all necessary information to define the terminology of a domain, so the task of the reasoner then is to find domain relations and to make interpretations.

To integrate reasoning for semantic time series into our functionality, we followed the approach not to implement our own reasoner, but to evaluate existing solutions and pick the best reasoner for our purpose. Our requirements regarding a reasoner were first of all that it should be free and open source. The *license* should be such that we would have no problems when using it for our system. Furthermore, the reasoner should support *OWL-DL*, since our ontologies are implemented using the DL version of OWL. It should have fair *expressivity* and use a state-of-the-art *algorithm* for reasoning. Finally, the reasoner should have *consistency checking* and *rule support*. Table 4.4 shows an overview of reasoners according to our criteria. The meaning of the cell background colors is the following: green – requirement is met, yellow – requirement is met only partially or unknown, and red – requirement is

not met. For background knowledge and term definitions in reasoning and description logic, we refer to Baader [2003].

Name	Licensing	OWL-DL	Expressivity	Algorithm	Consistency checking	Rule support
BaseVISor	free and closed	no	R-entailment	rule-based	yes	yes
Bossam	free and closed	no	-	rule-based	-	yes
Cyc	free and closed	yes	High-order	first-order	yes	yes
Hoolet	free and open	yes	-	first-order	yes	yes
Pellet	free and open	yes	SROIQ(D)	tableau	yes	yes
KAON2	free and closed	yes	SHIQ(D)	resolution & datalog	yes	yes
RacerPro	non-free and closed	yes	SHIQ(D-)	tableau	yes	yes
Jena	free and open	no	-	rule-based	no	yes
FaCT(++)	free and open	yes	SROIQ(D)	tableau	yes	no
SweetRules	free and open	no	-	rule-based	no	yes
OWLIM	non-free and closed	no	R-entailment	rule-based	yes	yes
OntoBroker	non-free and closed	yes	SHIQ(D)	resolution & datalog	yes	yes
HermiT	free and open	yes	SHOIQ+	hypertableau	yes	yes

Table 4.4: Overview of reasoners according to our requirements.

We looked into 13 different reasoners and selected one that fitted all our requirements:

- *BaseVISor*²⁰ is a forward-chaining inference engine specialized to handle facts in the form of RDF triples with support for OWL 2 RL and XML Schema Datatypes, and provides a Java API. It is free, but closed source and has no OWL-DL support. BaseVISor uses a rule-based algorithm and has consistency checking and rule support.
- *Bossam*²¹ is a RETE-based rule inference engine with native support for reasoning over OWL ontologies, SWRL ontologies, and RuleML rules. It is free and closed source and has no OWL-DL support. It has a rule-based algorithm and rule support.
- *Cyc*²² is a platform with a Java-based inference engine, which is free and closed source, has OWL-DL support and high-order expressivity. The reasoner is based on a first-order algorithm with high-order extensions and has consistency checking and rule support.
- *Hoolet*²³ is an implementation of an OWL-DL reasoner that uses a first order prover. It is free and open source, has a first-order algorithm, consistency checking and rule support.
- *Pellet*²⁴ is an OWL 2 reasoner and provides standard and cutting-edge reasoning services for OWL ontologies. It supports OWL-DL, is free and open source, has SROIQ(D) expressivity, a tableau algorithm, consistency checking and rule support.
- Reasoning in *KAON2*²⁵ is implemented by novel algorithms which reduce a SHIQ(D) knowledge base to a disjunctive datalog program. KAON2 is free and closed source and has OWL-DL, consistency checking, and rule support.
- *RACER*²⁶ stands for Renamed A-box and Concept Expression Reasoner, and Racer Pro is the name of the commercial software. It provides a query language, resource bounded computation, extended support for OWL-DL, a model-checking facility, and native access from Java and LISP.

²⁰<http://www.vistology.com/basevisor/basevisor.html>

²¹<http://bossam.wordpress.com/about-bossam/>

²²<http://www.cyc.com/platform/opencyc>

²³<http://owl.man.ac.uk/hoolet/>

²⁴<http://clarkparsia.com/pellet/>

²⁵<http://kaon2.semanticweb.org/>

²⁶<http://www.racer-systems.com/products/racerpro/>

- *Jena*²⁷ is a Java framework for building Semantic Web applications and has a rule-based inference engine for reasoning with RDF and OWL data sources. It is free and open source, has no OWL-DL and rule support, but supports consistency checking. The Jena reasoner uses a rule-based algorithm.
- *FaCT++* is the new generation of the well-known FaCT OWL-DL reasoner. FaCT++²⁸ uses the established FaCT algorithms, but with a different internal architecture. Additionally, FaCT++ is implemented using C++ in order to create a more efficient software tool, and to maximize portability.
- *SweetRules*²⁹ provides translation and interoperability between a variety of rule and ontology languages, highly scaleable backward and forward inferencing, and merging of rulebases/ontologies. Its rule-based reasoner is free and open source, has no OWL-DL support and consistency checking, but rule support.
- *OWLIM*³⁰ is a family of semantic repositories, but has also its own reasoner implementation. The reasoner is non-free and closed source, has no OWL-DL support, is rule-based and has consistency checking and rule support.
- *OntoBroker*³¹ is a Semantic Web middleware and inference machine for processing ontologies that supports all of the W3C Semantic Web recommendations. It has also a reasoner, which is non-free and closed, has OWL-DL, consistency checking, and rule support, SHIQ(D) expressivity, and a resolution and datalog algorithm.
- *HermiT*³² is a reasoner for ontologies written using the Web Ontology Language (OWL). It is free and open source, has OWL-DL, consistency checking and rule support, SHOIQ+ expressivity and is based on a hypertableau algorithm.

As we can see in Table 4.4, only Pellet and HermiT fully meet our requirements. However, it turned out that Pellet was our reasoner of choice, since in addition to meeting all our requirements, in contrast to HermiT, it

²⁷<http://jena.apache.org/>

²⁸<http://owl.man.ac.uk/factplusplus/>

²⁹<http://sweetrules.projects.semwebcentral.org/>

³⁰<http://www.ontotext.com/owlim>

³¹<http://www.semafora-systems.com/en/products/ontostudio/>

³²<http://www.hermit-reasoner.com/>

is seamlessly integrated into Protégé, which makes its usage very convenient. To summarize, Pellet provides OWL-DL entailment, supports SROIQ(D) expressivity for reasoning, uses tableau as reasoning algorithm, has consistency checking and rule support for SWRL-DL safe rules, and last but not least, it is free and open source.

In semantic time series reasoning, our approach is not to implement a new reasoner, but to use Pellet as already existing solution. We also use a bridge ontology, which we defined manually according to the MAFRA approach. The reasoner is used to find new inter-domain triples, i.e. new connections between two domain ontologies. As a next step, we show how Pellet works on the bridge ontology itself.

Listing 4.19 shows a simplified version of our bridge ontology in Turtle syntax.

```

1 @prefix stsp: <http://www.semantic-time-series-
2   processing.com/Bridge.owl#> .
3
4 stsp:TimeSeries stsp:hasSubject stsp:Subject ;
5   stsp:hasGroup stsp:Group ;
6   stsp:hasDomain stsp:Domain ;
7   stsp:hasSlot stsp:Slot .
8
9 stsp:Subject stsp:hasContent stsp:Content ;
10   stsp:hasAuthor stsp:Author .
11
12 stsp:Group stsp:hasUser stsp:User .
13   stsp:hasAnnotation stsp:Annotation .
14
15 stsp:Domain hstsp:hasTopic stsp:Topic .
16
17 stsp:Slot stsp:time stsp:TimeStamp ;
18   stsp:prop stsp:Property .
19
20 stsp:Property stsp:value xsd:string .
21
22 stsp:EnvironmentalTimeSeries rdfs:subClassOf
23   stsp:TimeSeries .

```

Listing 4.19: Simplified bridge ontology.

The Turtle RDF code describes a **TimeSeries** which has a **Subject**, **Group**, **Domain**, and several **Slots**. The **Subject** is a comment like description by a person and therefore consists of an **Author** and a **Content**. A group of users, which are interested in the time series, is described by the **Group** class, which consists of several **Users**, who create **Annotations**. The area of application or field, where the time series thematically belongs to, is described by the **Domain** class. The **Domain** class has also a **Topic**. Every time series

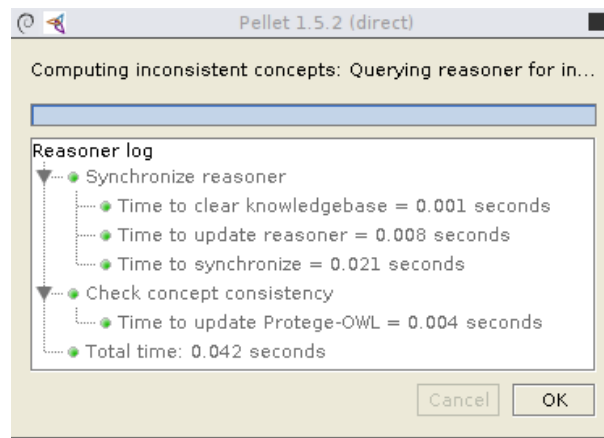


Figure 4.8: Output of a consistency check of our bridge ontology in Protégé using Pellet.

consists of a number of slots, which hold measurement data and the time of measurement. All this is represented by the `Slot` class and its elements (`TimeStamp` for the measurement time and `Property` for the values). Finally, there is a class for special time series, the `EnvironmentalTimeSeries`. This class inherits from the `TimeSeries` class.

First of all, we tested Pellet’s consistency checking in Protégé 3.5, which comes with Pellet 1.5.2 preconfigured, on our bridge ontology. Figure 4.8 shows the result of the consistency check. It shows that our bridge ontology is consistent.

After we were sure that our ontology is consistent, we used the Protégé Reasoner API to perform reasoning in Listing 4.20.

```

1 InputStream inputStream = new FileInputStream("
2   SemanticTimeSeriesBridge.rdf+xml.owl");
3
4 OWLModel owlModel = ProtegeOWL.
5   createJenaOWLModelFromInputStream(
6     inputStream);
7
8 ReasonerManager reasonerManager =
9   ReasonerManager.getInstance();
10
11 ProtegeReasoner reasoner = reasonerManager.
12   createProtegeReasoner(owlModel,
13     ProtegePelletOWLAPIReasoner.class);
14
15 reasoner.classifyTaxonomy();
16
17 reasoner.computeEquivalentConcepts();

```

```

18
19 reasoner.computeInconsistentConcepts();
20
21 reasoner.computeInferredHierarchy();
22
23 reasoner.computeInferredIndividualTypes();

```

Listing 4.20: Usage of the Reasoner API.

In the first step, we create a new input stream by passing the file name of our RDF/XML bridge ontology. Then we use the input stream to load the ontology into an `OWLModel` by calling the static method on the `ProtegeOWL` class. In the next step we create a `ReasonerManager` object by calling the singleton method of the class. We create the reasoner by passing the model and choosing the right reasoner (in our case this is Pellet). After this step the setup of our reasoner is done. We can start working with it by calling the `classifyTaxonomy()` method first and assigning this task to the reasoner. Finally, we can perform the real reasoning and compute equivalent concepts, inconsistent concepts, inferred hierarchy, and individual types.

4.5 User Interaction and Annotation

For user interaction we use a web portal which is based on Vaadin³³ and runs on a Tomcat³⁴ 7 server. In this subchapter we present the user interface in general (concentrating on the design and some implementation details) and the annotation processes through the portal as a showcase of the portal functionality.

4.5.1 User Interaction

The Semantic Time Series Processing Web Portal is the main entry point for users to interact with the framework. Figure 4.9 shows a screenshot of the main window.

³³<https://vaadin.com/>

³⁴<http://tomcat.apache.org/>

Semantic Time Series Processing

Upload new ontology

Choose Topic

Quality

Choose...

Upload

Time Series found:

TITLE	CATEGORY	START	END	VALUES	EDIT
CO2 Concentration	Air Quality	Jan 1, 1970	Aug 30, 2013	1,328	Edit
Ozone Concentration	Air Quality	Jan 1, 1970	Aug 30, 2013	3,423	Edit
Organic Pollutants	Water Quality	Jan 1, 1970	Aug 30, 2013	2,189	Edit
Cryptosporidium	Water Quality	Jan 1, 1970	Aug 30, 2013	1,895	Edit
Dissolved Oxygen	Water Quality	Jan 1, 1970	Aug 30, 2013	1,598	Edit

Ontology graph

```

graph TD
    Sunlight -- subClassOf --> SolarEnergy
    SolarEnergy -- subClassOf --> WindEnergy
    WindEnergy -- subClassOf --> WindSto
    SolarEnergy -- type --> WindEnergy
    WindEnergy -- type --> WindSto
  
```

Figure 4.9: Screenshot of the Semantic Time Series Processing web portal.

The functionality of the portal enables the user to load an ontology of her user group, view a generated graph of the ontology, and select a topic according to the ontology. The prototype implementation allows the following user actions:

1. *Load ontology*: The first element of the portal is an open file dialog which enables the user to select an ontology file (in OWL or RDF). The ontology is loaded via RDFlib and converted into a dot file (Graphviz³⁵ file format).
2. *View graph*: The dot file is used to generate a graph which is converted into an image and loaded to the portal.
3. *Select topic*: Furthermore, the topics defined in the ontology are used to populate the selection field and can be selected by the user in order to retrieve time series.
4. *Discover time series*: Discovered time series are shown in the table which consists of the name, category, start time, end time, number of values, and a button to edit the time series.

4.5.2 Annotation

Time series can be annotated by using the “Edit” button in the discovery table of the portal start page. The annotation window which can be used for time series annotation is shown in Figure 4.10.

The annotation window provides the following means for annotation time series to the user:

- *User name*: Name of the author of the annotation.
- *Date*: Date of annotation.
- *Topic*: Topic of annotation which is a subtopic of the selected topic in the main menu of the portal.
- *Rating*: The user is able to rate the time series with a value between 0 and 100 by using a slider.
- *Description*: A detailed description of the annotation (in most of the cases describing the time series itself in more detail).

³⁵<http://www.graphviz.org/>

Semantic Time Series Annotation □ ×

User
Bojan Božić

Date
8/30/13

Topic
Quality ▼

Rating
● —————

Description
CO2 Concentration measured at
A2 Highway station Guntramsdorf.

Annotate

Figure 4.10: Window for annotation of semantic time series.

4.6 Future Improvements

Despite our efforts and implementations, the semantic time series processing framework has still room for improvement. For example, the reasoning process could be improved by the implementation of a dedicated reasoner which is able to reason over multiple ontologies and supports real-time reasoning of created annotations. Further, a semantic discovery component could be implemented to provide discovery of related resources by a general search through the Web and automatic harvesting of Web resources and RDF data. Last but not least, a lot of improvements could be implemented for the graphical elements of the framework. As an example, the graphical representation of the ontology could be improved to support navigation and direct selection of resources to be annotated.

Chapter 5

Community Building

Technically, the Community Building functionality is, as well as the Time Series Processing Language and the Semantic Framework, part of the whole framework. As it is an important part of the framework, an own prototype is dedicated to it.

The Community Building prototype evaluates the results of the Semantic Framework and builds user groups around different topics and resources. It is also responsible for the rating of resources and the correlation of a resource and a user group. It interacts with the Semantic Framework to get the results of semantic processing, and with a web portal to get ratings and to present the results graphically.

Nowadays, social communities play a very important role in our daily lives. They make it possible to connect and communicate with people who share someone's interests and domains, but also to build groups and to attract other people's attention to one's topics. Whereas this is widely spread for private use, there is no sufficient and satisfactory solution for scientific communities so far on the web. This is the gap which our language aims to fill by providing functionality to build social communities based on Semantic Web and Web 2.0 technologies.

Scientific communities are a special kind of social communities. It is a very complicated task to build a scientific community or scientific user group. This is mainly because of some special characteristics of scientific communities compared to general social communities, such as:

- *Networking*: The communication between scientists, and building of networks by meeting new scientists with common interests;
- *Document Sharing*: Working on publications online, and the ability to share libraries of publications and papers for a certain topic;

- *Correlation of Research Topics*: Linking between fields of research with certain commonalities, and therefore building new networks and groups of researchers;
- *Organization of Conferences*: Organising events, such as conferences with invitations for speakers, calls for papers, registrations, etc.

To show this concept, Figure 5.1 presents the approach of semantic tagging and building of interest groups. The input of the time series processor is a time series. Additionally, there are different communities which provide semantic tagging, and each of them has its own ontology. The time series processor uses the ontology together with community-specific tagging to produce interest groups with time series specific to each group. The result is a number of time series assigned to each group of scientists with domain-specific information, which is evaluated and relevant to a specific group.

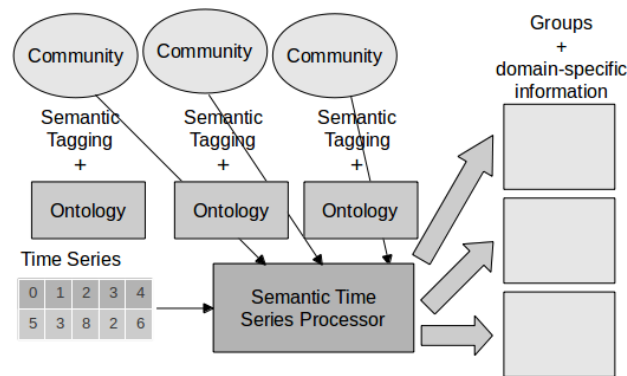


Figure 5.1: Semantic tagging and building of groups with common interests.

A closer look to the functionality of the language shows us how the community building is implemented. Regardless of the architecture used¹ there is somewhere an implementation of a web portlet, which provides standard social web functionality, like creating profiles for scientists, joining groups that have special interests, connecting to friends, etc. The portlet is connected to the time series processor in a manner that depends on the architecture. The processor has access to an RDF store or ontology store to save RDF triples

¹Be that a web service, a client-server architecture, or a stand-alone interpreter.

representing time series and their connections to a single scientist or group of scientists. Listing 5.1 Terse RDF Triple Language (Turtle)² code snippet shows such a triple.

```

1 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
2 @prefix cb:
3   <http://www.community-building.org/22-rdf-syntax-ns#> .
4
5 cb:time-series [cb:time-stamps "(0, 1, 2, 3, 4)"; cb:values "(4,
6   3, 5, 8, 1)"]
  cb:interests cb:semanticWebGroup .

```

Listing 5.1: Example for a triple.

In the code snippet, an `rdf` and `cb` prefix are defined (prefixes are much like namespaces in other languages). After that the code describes an RDF triple with the subject being a `cb:time-series` instance, the predicate (or relation between subject and object) being `cb:interests`, and the object being `cb:semanticWebGroup`, an instance of a group. Additionally, the subject gets initial data in square brackets for time stamps and values. The triple specifies that this particular time series interests the group Semantic Web Group.

The tags are provided by the scientists through the web portal. With this information (time series, RDF triple, and ontology) the language is able to extract relevant information (depending on tags the scientist provided) and deliver the according time series back to the portal and hence make it available to the scientist.

The following sections describe the integration of community building techniques into the Web portal of the framework.

5.1 Authentication

In order to access the Semantic Time Series Processing Portal we have implemented a login system based on the Vaadin³ framework. The login screen can be seen in Figure 5.2. To login to the portal, the user needs to provide an email address and a password, which she registered previously. This login data is checked with the database through the Vaadin framework.

The `LoginUI` class shown in Listing 5.2 represents the main entry point to the portal. It registers itself as a Vaadin Servlet and uses the predefined `stsp_portal` theme. Furthermore, it implements the following process:

²<https://http://www.w3.org/TeamSubmission/turtle/>

³<https://vaadin.com/>

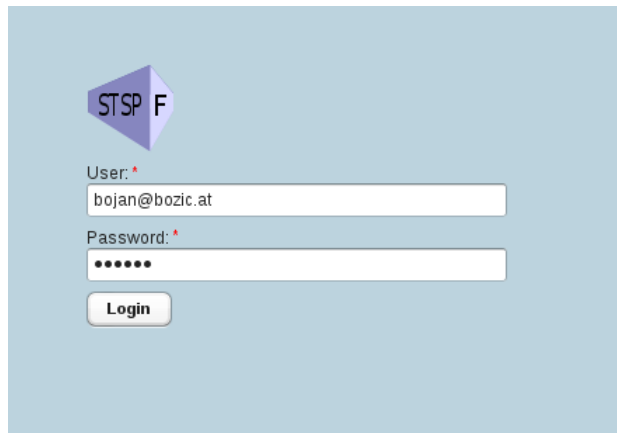


Figure 5.2: The login window of the Semantic Time Series Processing portlet.

1. Initialization of the navigator for the site navigation throughout the portal.
2. Registering of the `LoginView` (for the login process) and `MainView` (for other user interaction) to the navigator.
3. Checking if the user is already logged in.
4. Navigation to the login view if the user is not logged in yet.

```

1 @Theme("stsp_portal")
2 @Title("Semantic Time Series Processing Web Portal")
3 public class LoginUI extends UI {
4
5     @WebServlet(value = "/*", asyncSupported = true)
6     @VaadinServletConfiguration(productionMode = false, ui =
7         LoginUI.class)
8     public static class Servlet extends VaadinServlet {}
9
10    @Override
11    protected void init(VaadinRequest request) {
12        new Navigator(this, this);
13        getNavigator().addView(LoginView.NAME, LoginView.class);
14        getNavigator().addView(MainView.NAME, MainView.class);
15        getNavigator().addViewChangeListener(new ViewChangeListener
16            () {
17
18                @Override
19                public boolean beforeViewChange(ViewChangeEvent event) {
20                    boolean isLoggedIn = getSession().getAttribute("user")
21                        != null;

```

```


19         boolean isLoginView = event.getNewView() instanceof
LoginView;
20
21         if (!isLoggedIn && !isLoginView) {
22             getNavigator().navigateTo(LoginView.NAME);
23             return false;
24
25             } else if (isLoggedIn && isLoginView) {
26                 return false;
27             }
28
29         return true;
30     }
31 }
32 }
33 }

```


Listing 5.2: Implementation of the login user interface.

After the first login, the user settings screen (Figure 5.3) is presented to the user. The screen contains the name of the user which she used for registration as well as an optional picture. The other part of the screen is loaded dynamically and depends on the uploaded user domain ontology. The ontology defines not only the domain for a user (which matches in some parts the domain ontology of time series) but also the position of the user (in terms of scientific role), the institute, and department. These information is filled in the text fields of the user settings screen and can be edited by the user in order to be updated. Furthermore, groups of interest which are found in the domain ontology can be selected by the user. This triggers a search for related research topics which can then be selected in the twin column selection field. After editing all required information, the user can save the settings and start using the portal.

[Home](#) [User Settings](#) [Ontology](#) [Groups](#) [Related Users](#) [Time Series](#) [SPARQL Endpoint](#) [Logout](#)



Semantic Time Series Processing



Bojan Božić

Upload user domain ontology

[Browse...](#) [Upload](#)

Position:

Junior Scientist

Institute:

Austrian Institute of Technology

Department:

Safety & Security

Group of Interest:

Air

Water

Soil

Nuclear

Research Topics

Found topics:

Cryptosporidium

Dissolved Oxygen

>>

<<

Selected topics

Organic Pollutant

[Save](#)

Figure 5.3: User settings screen which is presented to the user after her first login.

5.2 Group Generation

Group Generation functionality provides a means of suggesting groups to a user who shares her interest regarding time series. The suggestions depend on the user's domain ontology and group users by common interests as defined in the ontology. Users with the same interest triple (e.g. '`stsp:bozicb stsp:has_interest stsp:Water`' and '`stsp:alanf stsp:has_interest stsp:Water`') are suggested to be in the same group. Therefore, the portal presents the user with groups retrieved by the according SPARQL query for `has_interest` and `is_in_group` in the 'Group' menu (see Figure 6.25).

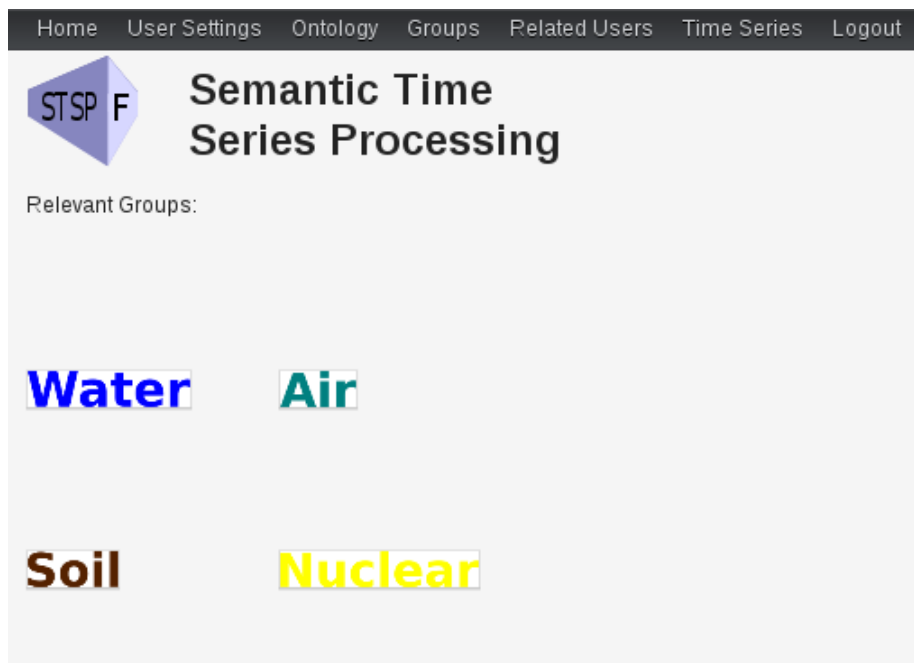


Figure 5.4: An example of group suggestions in the Groups menu.

Figure 5.4 shows a user who has 4 group suggestions (Water, Air, Soil, and Nuclear). The groups represent interests of users and contain not only users who have common interests, but also time series which correspond to those interests. The groups menu shows the logos or names of the groups. Details are only provided after the user clicks on one of the group logos.

The detailed view of a group is shown in Figure 5.5. This view presents on the one hand a table of users who are members of the group, and on the other hand, a table of relevant time series for the group.

The table of users defines the following data:

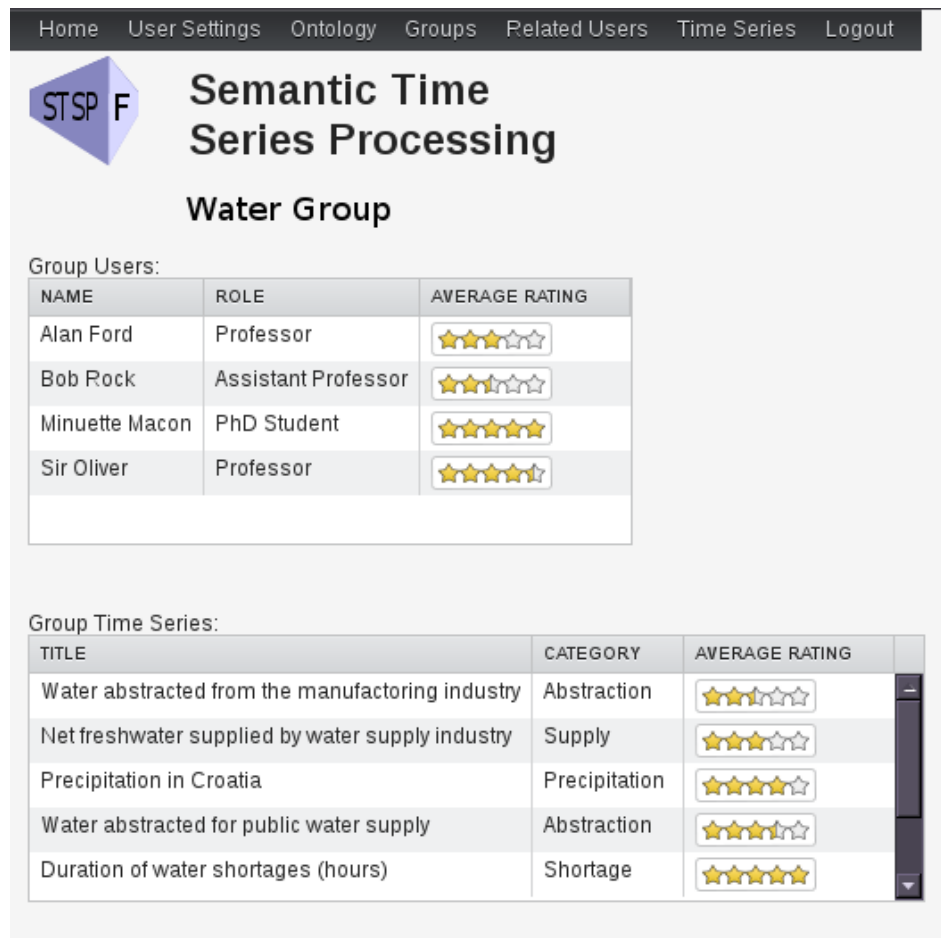


Figure 5.5: An example of a detailed view for a group (in this case the group ‘Water’).

- *User name*⁴: The full name of the group member.
- *Role*: The role of the group member within her organization (same as position).
- *Average rating*: The average user rating from other users of the group (defines the relevance and trustworthiness of the user from the view-point of the group).

An overview of the relevant time series information is presented in the time series table as follows:

⁴User names and pictures used for the examples are property of Magnus, Max Bunker Press, 1969, Source: <http://www.munix.it/1-10/1-10.html>

- *Title of the time series*: Descriptive name of the time series content.
- *Category of the time series*: Specific category for the time series within the topic or group (e.g. topic = water, category = precipitation).
- *Average rating*: The average rating from users of the group (defines the relevance of a time series to the group, as well as the quality of the time series data).

By selecting a user or time series from the table, the user is able to rate the item or provide additional annotations. This process is explained in detail in the ‘User and Time Series Rating’ section.

5.3 User and Time Series Rating

In order to find out which relevance certain users and time series in a group have, we introduce the concept of user and time series rating. For this task we use a 5-star-rating system enabling every user to rate another user or a time series from 1 to 5 stars and hence mark the relevance of the user or time series to the group. In addition to this, we also provide the possibility to annotate users and time series in the rating window.

Figure 5.6 shows such an annotation and rating window for a sample user. The window presents an image of the user, her name, position (role within her organization), and a list of groups the user is member of. Additionally, there is the possibility to rate the user and to provide a short annotation (comment about the user).

The annotation and rating window for time series is shown in Figure 5.7. It provides the title of the time series, the topic (which is a high level category, such as ‘Water’ or ‘Air’), the category (a subdomain of the topic), and a link to show the details (graph) of the time series. Furthermore, we have the same options for rating and annotation as we have already seen in the user example.

Finally, Figure 5.8 shows the window that appears after clicking the “Show Details” link.

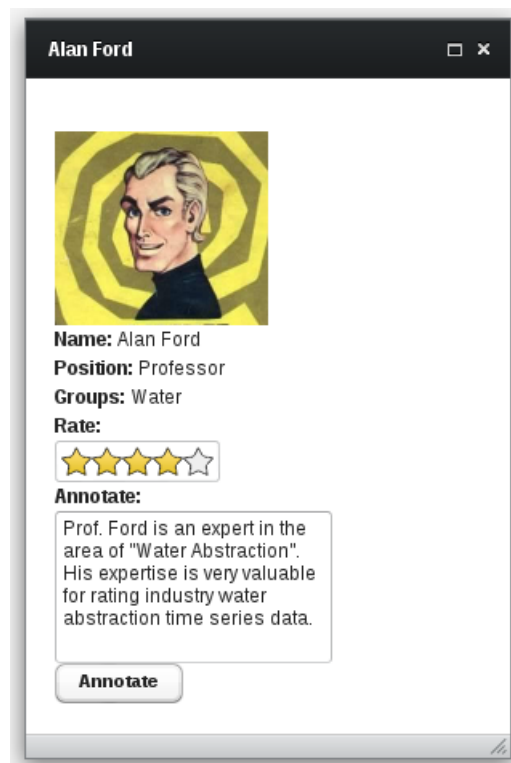


Figure 5.6: User annotation and rating window after selecting a user from the table in the group overview.

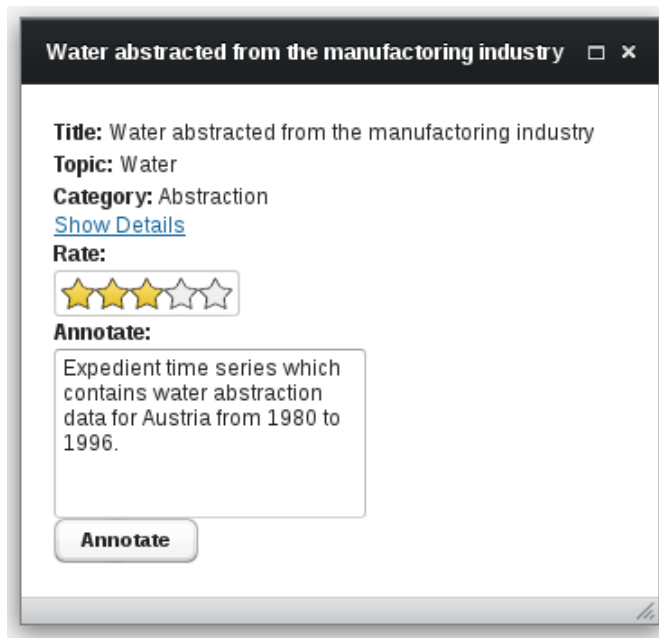


Figure 5.7: The annotation and rating window for time series.

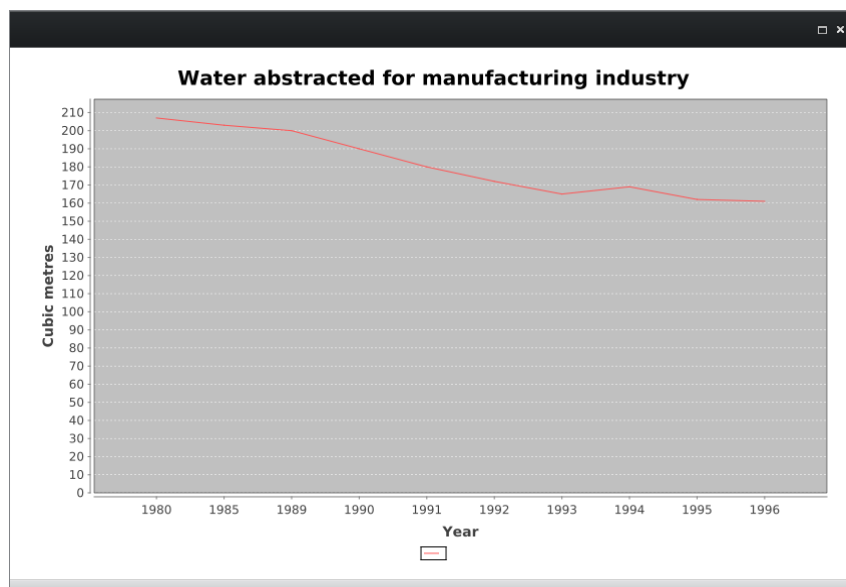


Figure 5.8: Overview of the time series data presented as a line chart.

5.4 Sharing of Time Series among Users and User Groups

After introducing the user interface elements, this chapter provides an insight on the conceptual backgrounds of the implementation. This means the relevant parts of the ontology, as well as important RDF triples, which make it possible to build the end user system.

To fulfill the task of time series sharing the following steps are necessary:

- *User definition*: definition of important details of a user, such as: name, position, interests, rating, etc., in order to find the right group (combined with other users with similar interests).
- *Time series definition*: definition of important information about a time series for assignment to the right group.
- *Rating and relevance definition*: ratings by other similar users for users and time series.
- *Unifying user and time series elements in a group*: collecting RDF triples for unifying users and time series in groups.

Therefore, a general ontology needs to be used in order to combine users and time series in the concept of a group. Listing 5.3 shows the most important classes of such an ontology as code snippets. The most important object and data properties for the connection of users and time series are: **according_to** correlates domains or groups to interests, **belongs_to** domains or groups to time series, **has_category** categories to time series, **has_domain** time series to domains, **has_group** users to groups, and **has_timeseries** domains or groups to time series. The most important and most frequently used classes are: **Domain**, **Group**, **Interest**, **TimeSeries**, and **User**.

```

1 :according_to rdf:type owl:ObjectProperty ;
2               rdfs:domain :Domain ,
3                   :Group ;
4               rdfs:range :Interest .
5
6 :belongs_to rdf:type owl:ObjectProperty ;
7             rdfs:range :Domain ,
8                 :Group ;
9             rdfs:domain :TimeSeries ;
10            owl:inverseOf :has_timeseries .
11
12 :has_category rdf:type owl:ObjectProperty ;
13             rdfs:range :Category ;

```

```

14         rdfs:domain :TimeSeries ;
15         rdfs:subPropertyOf :has_domain .
16
17 :has_domain rdf:type owl:ObjectProperty ;
18         rdfs:range :Domain ;
19         rdfs:domain :TimeSeries .
20
21 :has_group rdf:type owl:ObjectProperty ;
22         rdfs:range :Group ;
23         rdfs:domain :User .
24
25 :has_position rdf:type owl:ObjectProperty ;
26         rdfs:range :Position ;
27         rdfs:domain :User .
28
29 :has_timeseries rdf:type owl:ObjectProperty ;
30         rdfs:domain :Domain ,
31                 :Group ;
32         rdfs:range :TimeSeries .
33
34 :has_user rdf:type owl:ObjectProperty ;
35         rdfs:domain :Domain ,
36                 :Group ;
37         rdfs:range :User ;
38         owl:inverseOf :has_group .
39
40 :has_average_rating rdf:type owl:DatatypeProperty ;
41         rdfs:domain :TimeSeries ,
42                 :User ;
43         rdfs:range xsd:double .
44
45 :has_name rdf:type owl:DatatypeProperty ;
46         rdfs:domain :Category ,
47                 :Domain ,
48                 :User ;
49         rdfs:range xsd:string .
50
51 :has_title rdf:type owl:DatatypeProperty ;
52         rdfs:domain :Group ,
53                 :TimeSeries ;
54         rdfs:range xsd:string .
55
56 :Category rdf:type owl:Class ;
57         rdfs:subClassOf :Domain .
58
59 :Domain rdf:type owl:Class .
60 :Group rdf:type owl:Class .
61 :Interest rdf:type owl:Class .
62 :Position rdf:type owl:Class .

```

```

63 :TimeSeries rdf:type owl:Class .
64 :User rdf:type owl:Class .

```

Listing 5.3: Ontology snippets (classes and properties) for combining users and time series into groups.

Listing 5.4 shows the most important individuals from the ontology. The examples show how users are created and assigned to names, positions, ratings, and groups. As a first step some groups, categories, domains, and interests are created. Then users and time series are defined and assigned to the interests and domains. This offers us the connection between users and time series in order to combine users and time series in groups.

```

1 :abstraction rdf:type :Category ,
2               owl:NamedIndividual .
3
4 :air_domain rdf:type :Domain ,
5               owl:NamedIndividual .
6
7 :air_group rdf:type :Group ,
8               owl:NamedIndividual .
9
10 :air_interest rdf:type :Interest ,
11                  owl:NamedIndividual .
12
13 :alan_ford rdf:type :User ,
14              owl:NamedIndividual ;
15       :has_average_rating "3.0"^^xsd:double ;
16       :has_name "Alan Ford"^^xsd:string ;
17       :has_position :professor ;
18       :has_group :water_group .
19
20 :assistant_professor rdf:type :Position ,
21                           owl:NamedIndividual .
22
23 :bob_rock rdf:type :User ,
24              owl:NamedIndividual ;
25       :has_average_rating "2.5"^^xsd:double ;
26       :has_name "Bob Rock"^^xsd:string ;
27       :has_position :assistant_professor ;
28       :has_group :water_group .
29
30 :minuette_macon rdf:type :User ,
31                   owl:NamedIndividual ;
32       :has_average_rating "5.0"^^xsd:double ;
33       :has_name "Minuette Macon"^^xsd:string ;
34       :has_position :phd_student ;
35       :has_group :water_group .
36

```

```

37 :nuclear_domain rdf:type :Domain ,
38                      owl:NamedIndividual .
39
40 :nuclear_group rdf:type :Group ,
41                      owl:NamedIndividual .
42
43 :nuclear_interest rdf:type :Interest ,
44                      owl:NamedIndividual .
45
46 :phd_student rdf:type :Position ,
47                      owl:NamedIndividual .
48
49 :precipitation rdf:type :Category ,
50                      owl:NamedIndividual .
51
52 :professor rdf:type :Position ,
53                      owl:NamedIndividual .
54
55 :shortage rdf:type :Category ,
56                      owl:NamedIndividual .
57
58 :sir_oliver rdf:type :User ,
59                      owl:NamedIndividual ;
60      :has_average_rating "4.5"^^xsd:double ;
61      :has_name "Sir Oliver"^^xsd:string ;
62      :has_position :professor ;
63      :has_group :water_group .
64
65 :soil_domain rdf:type :Domain ,
66                      owl:NamedIndividual .
67
68 :soil_group rdf:type :Group ,
69                      owl:NamedIndividual .
70
71 :soil_interest rdf:type :Interest ,
72                      owl:NamedIndividual .
73
74 :supply rdf:type :Category ,
75                      owl:NamedIndividual .
76
77 :ts1 rdf:type :TimeSeries ,
78                      owl:NamedIndividual ;
79      :has_average_rating "2.5"^^xsd:double ;
80      :has_title
81      "Water abstracted for the manufacturing industry"
82      ^^xsd:string ;
83      :has_category :abstraction ;
84      :belongs_to :water_domain ,
85                  :water_group .

```

```

86
87 :ts2 rdf:type :TimeSeries ,
88         owl:NamedIndividual ;
89     :has_average_rating "3.0"^^xsd:double ;
90     :has_title
91     "Net freshwater supplied by water supply industry"
92     ^^xsd:string ;
93     :has_category :supply ;
94     :belongs_to :water_domain ,
95         :water_group .
96
97 :ts3 rdf:type :TimeSeries ,
98         owl:NamedIndividual ;
99     :has_average_rating "4.0"^^xsd:double ;
100    :has_title "Precipitation in Croatia"^^xsd:string ;
101    :has_category :precipitation ;
102    :belongs_to :water_domain ,
103        :water_group .
104
105 :ts4 rdf:type :TimeSeries ,
106         owl:NamedIndividual ;
107     :has_average_rating "3.5"^^xsd:double ;
108     :has_title
109     "Water abstracted for public water supply"^^xsd:string ;
110     :has_category :abstraction ;
111     :belongs_to :water_domain ,
112         :water_group .
113
114 :ts5 rdf:type :TimeSeries ,
115         owl:NamedIndividual ;
116     :has_average_rating "5.0"^^xsd:double ;
117     :has_title
118     "Duration of water shortages (hours)"^^xsd:string ;
119     :has_category :shortage ;
120     :belongs_to :water_domain ,
121         :water_group .
122
123 :water_domain rdf:type :Domain ,
124         owl:NamedIndividual .
125
126 :water_group rdf:type :Group ,
127         owl:NamedIndividual .
128
129 :water_interest rdf:type :Interest ,
130         owl:NamedIndividual .

```

Listing 5.4: Ontology snippets (individuals) for combining users and time series into groups.

The whole process of group definition is implemented through the follow-

ing steps:

- *Definition of users and time series:* RDF triples to define users and time series, and assign interests and domains to them.
- *Reasoning over the ontology:* usage of a reasoner to generate groups which combine users and time series and include ratings.
- *Retrieval of groups:* usage of SPARQL queries to retrieve users and time series for a specific group.
- *Presentation of groups:* the web portal can be used for the presentation of groups with their users and time series as well as further rating and annotation.

5.5 SPARQL Endpoint

An additional feature of the framework is a SPARQL endpoint for retrieving community-related data. The endpoint is implemented by using the OpenRDF Sesame⁵ framework and can be used to inspect retrieved individuals from the knowledge base.

Figure 5.9 shows a screenshot of the SPARQL endpoint. The web interface provides a possibility to the user to define a SPARQL query and retrieve the results in a list. The text area named “SPARQL query” can be used to enter the query with one or more variables. After formulating the query, the user can click the “Query” button in order to have the query executed on the semantic repository. The result of the query is displayed automatically in the list under the button as a list of URIs of the resulting instances.

```

1 File dataDir = new File("/home/bojan/stsp-repo/");
2 repo = new SailRepository(new NativeStore(dataDir));
3 try {
4     repo.initialize();
5     con = repo.getConnection();
6     File f = new File("/home/bojan/Dropbox/Code/semantic-time-
       series-processing/stsp.owl");
7     con.add(f, "http://www.semanticweb.org/bojan/ontologies
       /2013/10/stsp#", RDFFormat.TURTLE);
8 } catch (Exception e) {
9     e.printStackTrace();
10 }
11
12 try {

```

⁵<http://www.openrdf.org/>

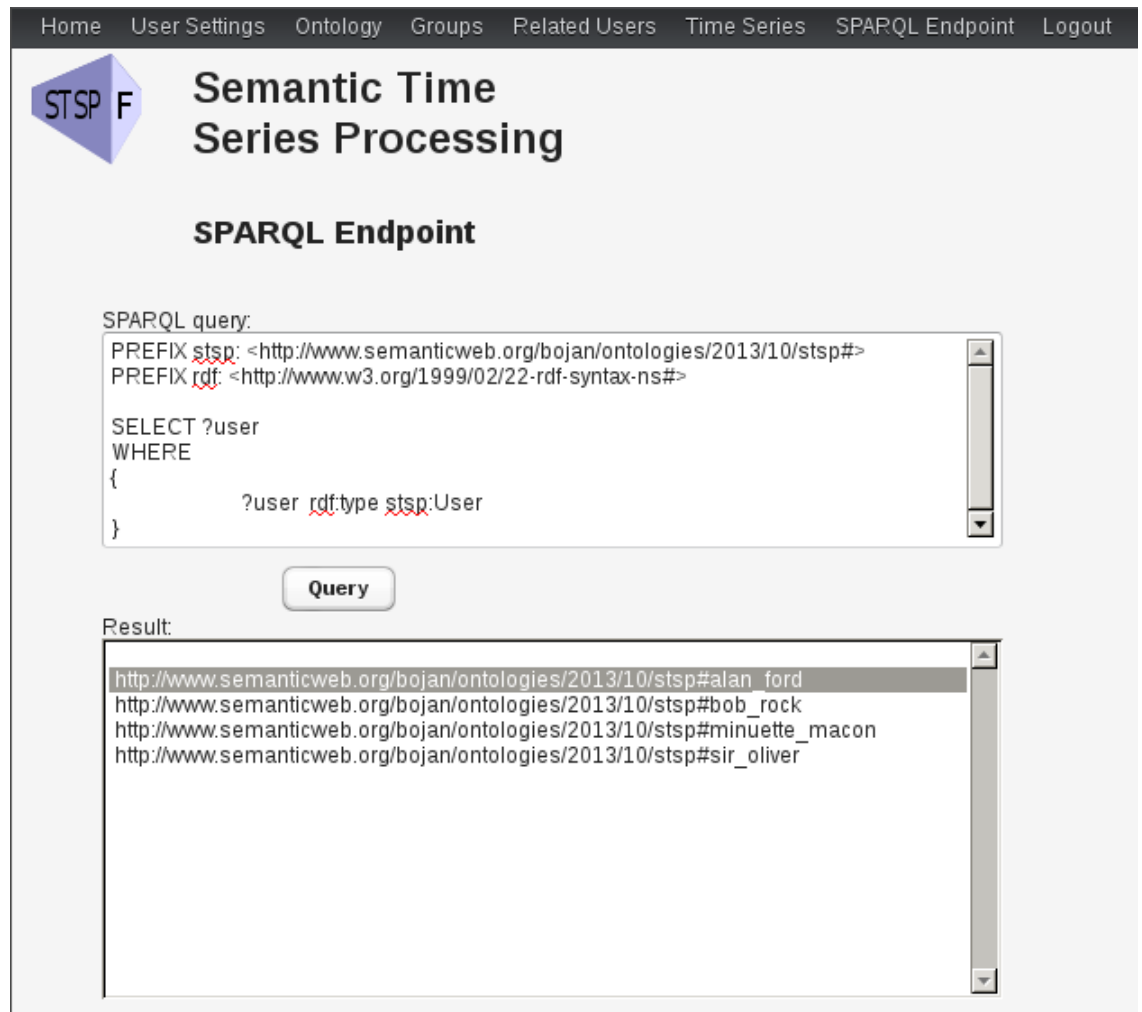


Figure 5.9: SPARQL endpoint of the Semantic Time Series Processing portal.

```

13 TupleQuery tupleQuery = con.prepareTupleQuery(QueryLanguage.
    SPARQL, taSPARQL.getValue());
14 TupleQueryResult result = tupleQuery.evaluate();
15 while(result.hasNext()) {
16     for(int i = 0; i < result.getBindingNames().size(); i++) {
17         lsResult.addItem(result.next().getValue(result.
            getBindingNames().get(i)).toString());
18     }
19 }
20 con.close();
21 repo.shutdown();
22 } catch(Exception e) {
23     e.printStackTrace();
24 }

```

Listing 5.5: Code snippet of the SPARQL endpoint implementation.

Listing 5.5 shows the main part of the SPARQL endpoint implementation. We use the directory “stsp-repo” to create a semantic repository by using the **SailRepository** class constructor. As a next step, we initialize the repository and retrieve a connection. To populate the repository we use our STSP ontology and add it to the repository by calling the **add** method on the connection object and defining the ontology file, the main namespace, and the RDF format. The query is prepared by calling the **prepareTupleQuery** method and providing the query language (which is SPARQL in our case) and the content of the according text area. The retrieved **TupleQuery** object is evaluated and the result consists of an object which contains the retrieved URIs. Finally, the **TupleQueryResult** object is iterated and the items are saved to the list by calling the **next** method as long as the result has further elements, and adding the elements to the list (considering the number of variables in the query as well - while loop for the results and for loop for the variables per result tuple).

5.6 Reasoning in Community Building

The community building part of the Semantic Time Series Processing framework supports (among general reasoning rules) the following reasoning workflows:

- *Assignment of users to groups:* Reasoning over related groups and assignment of users to highly related groups.
- *Suggestions of time series for users:* Reasoning over user groups and their time series, and suggestions to a user for a new time series relation.

- *Assignment of time series to groups:* Reasoning over groups and time series and assignment of time series to groups.
- *Suggestions of related users to users:* Reasoning over users, groups, and time series and suggestions for relations between users.

The following subsections describe the four reasoning scenarios in more detail through Description Logics (DL). Each scenario consists of a formal definition, a description of the scenario, and an example.

5.6.1 Assignment of Users to Groups

Our first reasoning example is the assignment of users to groups of interest. It is defined in terms of description logic in Equation 5.1.

$$hasUser.Group \equiv hasInterest.User \sqcap hasInterest.Group \quad (5.1)$$

The user is assigned to a group if the user and the group share the same interest (topic).

In order to automatically assign users to groups, there is a relation between the **Group** and the **User** class. The object property **hasInterest** has the domains **Group** and **User** and the range **Interest**. Therefore, through the definition of the reasoning rule in Equation 5.1 users are automatically assigned to groups with the same interest.

For example, the group **water_group** and the user **alan_ford** have the interest **water_interest**. This leads us to the conclusion that the user **alan_ford** belongs to the group **water_group** (see Listing 5.6).

```

1 : water_group rdf:type :Group .
2 : alan_ford  rdf:type :User   .
3
4 : water_group :hasInterest :water_interest .
5 : alan_ford  :hasInterest :water_interest .
6
7           |
8           v
9
10 : water_group :hasUser :alan_ford .
```

Listing 5.6: Reasoning example for assignment of users to groups in Turtle notation.

5.6.2 Suggestions of Time Series to Users

This workflow defines which time series are suggested to a user. The definition is shown in Equation 5.2.

$$\begin{aligned} SuggestedTimeSeries \equiv & \geq 3 \forall hasRating.TimeSeries \sqcap \\ & hasGroup.TimeSeries = hasGroup.User \end{aligned} \quad (5.2)$$

A time series is suggested to a user if the rating of the time series is greater than or equal to 3 stars and the group of the time series is equal to the group of the user.

Both, a **TimeSeries** and a **User** have the object property **hasGroup**. An individual of the type **TimeSeries** is at the same time an individual of the type **SuggestedTimeSeries** as well, if the rating is at least 3 stars (defined by the **hasRating** data property) and the **Group** is the same as the group of the user of a current session.

For example, the time series **ts1** has the rating 4 and belongs to the group **water_group**. The user **alan_ford** belongs to the group **water_group** as well. Therefore it can be inferred that the time series could be relevant for the user and gets the type **SuggestedTimeSeries** for the session (see Listing 5.7).

```

1 :ts1 rdf:type :TimeSeries .
2 :alan_ford rdf:type :User .
3 :water_group rdf:type :Group .
4
5 :ts1 :hasGroup :water_group .
6 :ts1 :hasRating "4"^^xsd:decimal .
7 :alan_ford :hasGroup :water_group .
8
9         |
10        v
11
12 :ts1 rdf:type :SuggestedTimeSeries .
13 :alan_ford :hasTSSuggestion :ts1 .

```

Listing 5.7: Reasoning example for suggestion of time series to users in Turtle notation.

5.6.3 Assignment of Time Series to Groups

The automatic assignment of time series to groups, which depends on the rating and the user, is presented in Equation 5.3.

$$\begin{aligned} hasGroup.(\geq 3 \forall hasRating.TimeSeries) \equiv \\ hasGroup.(rated.(User, TimeSeries) \sqcap \exists hasGroup.User) \end{aligned} \quad (5.3)$$

A group of a user is assigned to the time series if the rating of the time series is at least 3 stars and there exists a user who has rated the time series in the group.

The data property **hasRating** of the class **TimeSeries** needs to have a value of at least 3 for the time series to become member of a group automatically. Furthermore, there needs to be at least one individual of type **User** from the group who has rated the time series.

For example, the time series **ts2** has a rating of 4.5. The same time series is rated by the user **bob_rock** who belongs to the group **air_group**. This scenario leads us to the conclusion that the time series **ts2** should belong to the group **air_group** as well (see Listing 5.8).

```

1 :ts2 rdf:type :TimeSeries .
2 :bob_rock rdf:type :User .
3 :air_group rdf:type :Group .
4
5 :ts2 :hasRating "4.5"^^decimal .
6 :bob_rock :rated :ts2 .
7 :bob_rock :hasGroup :air_group .
8
9         |
10        v
11
12 :ts2 :hasGroup :air_group .

```

Listing 5.8: Reasoning example for assignment of time series to groups in Turtle notation.

5.6.4 Suggestions of Related Users to Users

Equation 5.4 shows the definition of suggestions for users with same interest.

$$SuggestedUser \equiv hasGroup.User \sqcap interestedIn.Interest \quad (5.4)$$

A user is suggested if she is in the same group and shares the same interest with the user of the session.

An individual is assigned the type **SuggestedUser** if she shares the range of her **hasGroup** and **interestedIn** object properties with the considered individual.

For example, the user **bob_rock** is member of the group **air_group** and is interested in **air_interest**. This is also the case for the user **alan_ford** (same interest and group). Therefore, the conclusion can be made that **alan_ford** is of type **SuggestedUser** for the user **bob_rock** (see Listing 5.9).

```

1 :bob_rock rdf:type :User .
2 :alan_ford rdf:type :User .
3 :air_group rdf:type :Group .
4 :air_interest rdf:type :Interest .
5
6 :bob_rock :hasGroup :air_group .
7 :bob_rock :hasInterest :air_interest .
8 :alan_ford :hasGroup :air_group .
9 :alan_ford :hasInterest :air_interest .
10
11      |
12      v
13
14 :alan_ford rdf:type :SuggestedUser .
15 :bob_rock :hasUserSuggestion :alan_ford .

```

Listing 5.9: Reasoning example for suggestion of related users to a user in Turtle notation.

The next chapter provides more details regarding ontology mapping and reasoning by validating the functionality in two validation scenarios.

Part III

Use Cases and Validation

Chapter 6

Validation of Semantic Time Series Processing

The approach taken in this thesis is best described by the scientific methodology used. The chosen methodology is prototyping, hence we build several prototypes to prove our hypotheses. However, central to our validation approach and thus to this chapter are two ontologies which have been built during the TaToo project¹.

The prototype provides time series processing, Semantic Web technologies, and community building functionality. Figure 6.1 presents a typical use case of the prototype in order to fulfill a common task.

In step 1 the user requests information for his area of interest by using the Web Portal. The portal forwards the request to the Semantic Processor in step 2, which does semantic processing with the user's data, meta data and request, and asks the Semantic Discovery component for an appropriate Web resource, based on the resulting semantic information (step 3). In step 4 and 5, the Semantic Discovery component finds the right Web resource and gives the information back to the Semantic Processor (step 6). The Semantic Processor checks the resources in the available databases, and eventually triggers updates (arrow). After that it delivers the information to the Web Portal (step 7), which presents the results to the user (step 8). Additionally, the Semantic Processor forwards meta data about the user and the resource to the Community Building component, which generates relational dependencies, updates its community affiliation data, and provides this information to the Web Portal (steps 9 and 10). The arrows show the tagging circle, in which the user and the Semantic Processor enrich meta data with manual and automatic annotations.

¹<http://www.tatoo-fp7.eu/tatooweb/>

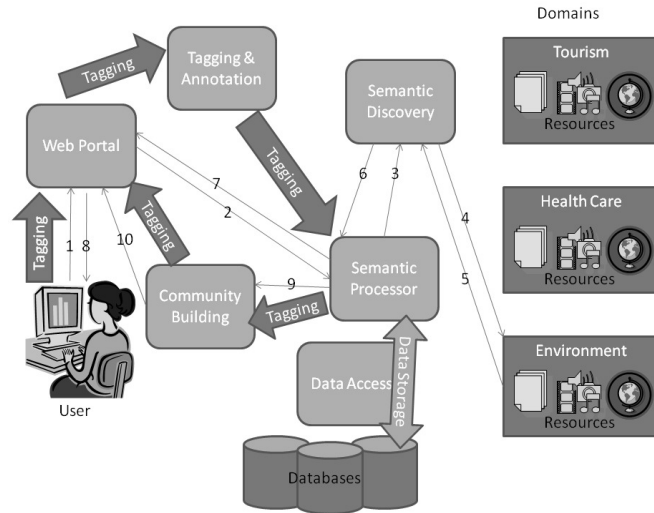


Figure 6.1: Use case of the prototype.

In our validation process, the prototype is used in two different validation scenarios (“Climate Change Twin Regions – Discovery Platform” and “Anthropogenic Impact and Global Climate Change”) which are described in detail in the following sections. But before we start with the description of the use cases, we need to define the most important terms used in the examples:

- *Topic*: The topic defines the interest of, e.g. a user. It is intentionally very abstract because it is meant to describe a large number of users or resources. The topic can be made more precise by the definition of a category. Examples for topics are: Water, Air, Soil, Nuclear, etc.
- *Category*: The category is a subclass of the topic which means that it defines an interest more precisely. Examples for categories are: Organic Pollutants, Cryptosporidium, Dissolved Oxygen, etc.
- *Group*: The group is independent of topics and categories (although a group can have an arbitrary number of topics and categories), but, for the ease of the reader, in some examples the topic is used as the name of a group as well.

- *Domain*: A domain is the concept for an area of interest for a specific user. The domain is defined by a domain ontology and describes all needed initial definitions for a user community.
- *Rating*: The rating is used to assign time series and users to groups. In order to be assigned to a group, a time series or user needs to have a rating of at least 3 stars (out of 5). This means that a user or time series is assigned to a group as soon as the rating exceeds 3, but also that the user or time series will be deleted from a group if the rating drops below 3. The number of groups a user or time series can be assigned to is not limited.

6.1 Climate Change Twin Regions – Discovery Platform

The “Climate Change Twin Regions – Discovery Platform” (short Climate Twins) has been developed by the Austrian Institute of Technology and used as a Validation Scenario in the TaToo FP7 project. One of the outputs of the TaToo project regarding the Climate Twins Validation Scenario was an ontology which describes the scenario and its meta-data. This ontology is used as an input for this thesis in order to show how the developed Ontology Mapping and Reasoning methods can be applied.

6.1.1 Introduction

The Climate Twins application has the goal to show the impact of climate change and allow adaptation for its users by finding model regions where the future climate of a certain Point-of-Interest (POI) is similar to the current climate of another location ([Ungar et al., 2011] and [Schimak et al., 2011]). These regions are called Climate Twins because they have a similar climate. The POI region which is subject to changes in the future can provide adaptation by knowing what the impacts of the future climate to the region will be. The tool for the search of Climate Twins regions is implemented as a web user interface which allows exploration of climate change effects on two maps. One map shows the source region (POI) and the other map shows the destination which is at the same time the region after a previously defined time period having a similar climate to the source region. The climatological similarity and its accuracy and validity depend on the used indicators as well as the defined similarity thresholds. The more indicators are used in the settings and the narrower the threshold ranges are defined, the less results

will be found which match the source region. The used indicators are daily mean temperature and precipitation, which are the most important indicators for the prediction of climate change. In the TaToo project, tools are used to improve the accuracy of the search for Climate Twins as well as the integration of further resources [Rizzoli et al., 2010]. This means that TaToo tools have enabled the application to add tags, annotate resources, reuse tags of other users, discover and retrieve further resources for the selected region (documents, web sites, web services, etc.).

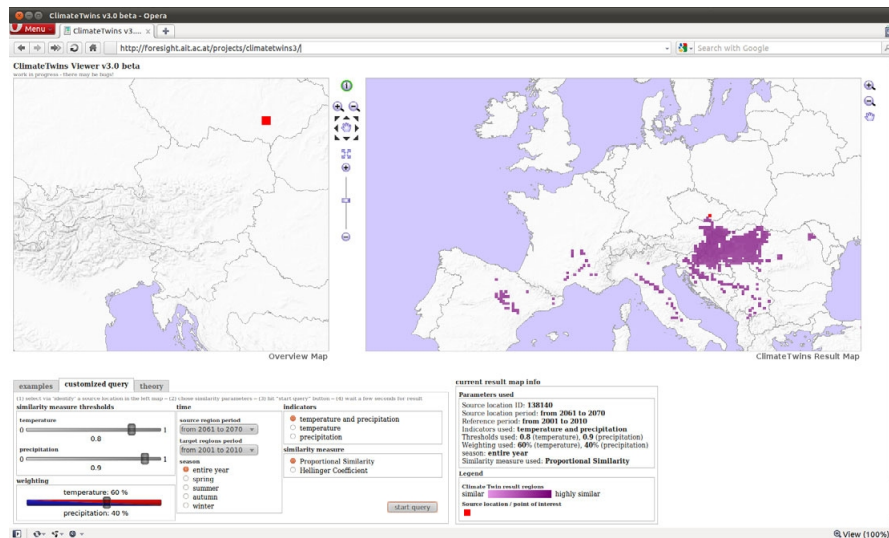


Figure 6.2: Screenshot of the Climate Twins application [Ungar et al., 2011].

Although similarity is based only on precipitation and temperature, there is a complex matching method which depends on quantification between data vectors and provides values which can be combined with other indicators as well [Božić et al., 2012]. This makes it also possible to differentiate between more or less similar regions as results of a query. Figure 6.2 shows a screenshot of the Climate Twins application. The left map shows the source of the calculation, which is a location in the Czech Republic. The red-marked region is the source location for the calculation. The map on the right side shows all twin regions which have a similar climate compared to the source region in a different time. Finally, the lower frame enables the user to set all required parameters for the calculation.

Figure 6.3 shows an enlarged view of the Climate Twins control panel. Here, the user is able to enter a customized query and to start it in order to get results presented on the map. On the left side the similarity measure thresholds and the weighting can be set by using sliders. Time and season

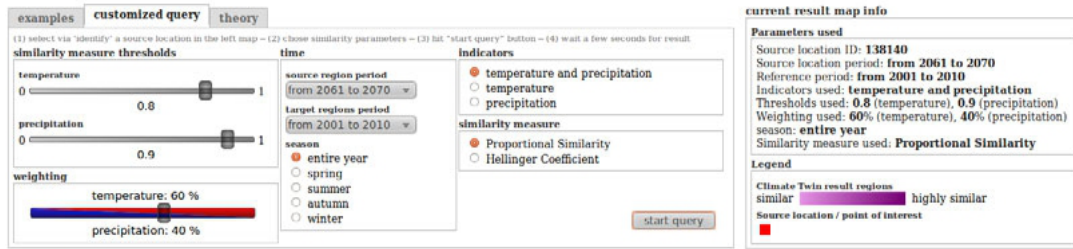


Figure 6.3: Enlarged screenshot of the Climate Twins control panel [Ungar et al., 2011].

can be selected as well together with indicators and the type of similarity measure. On the right side, the user is presented a box with information about the used parameters and a legend of the map.

6.1.2 Ontology

The ontology of the Validation Scenario 1 “Climate Twins” (AIT ontology) is a result of the TaToo project. It has been developed to enrich the data model of possible applications based on the Climate Twins technology, and represents single components of the field of application which are represented as classes of the ontology. The Semantic Web for Earth and Environmental Terminology (SWEET)² and the Clean Energy Info Portal (reegle)³ have been used as a basis for developing the “Climate Twins” domain ontology. In the following, the single first-level classes of the ontology are described (in alphabetical order) in detail by presenting the part of the graph which is related to the class (produced using OWLViz in Protégé) together with a textual description of the class itself.

The class **Building** (Figure 6.4) represents an architectural object which is related to documents (resources) describing it. It is subdivided into the two specialized classes **PassiveBuilding** and **GreenBuilding** which represent particularly energy-efficient forms of buildings.

A building has the **usesEnergy** property to define the kind of energy the building uses (solar, wind, etc.) and the **hasThermalProcess** property which defines the thermal process of a building.

The **Climate** class (Figure 6.5) encompasses the statistics of temperature, humidity, atmospheric pressure, wind, rainfall, atmospheric particle count,

²<http://sweet.jpl.nasa.gov/>

³<http://www.reegle.info/>

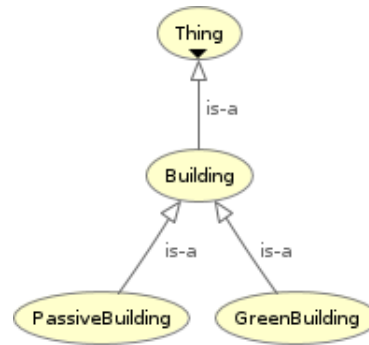


Figure 6.4: **Building** class of the AIT ontology.

and other meteorological elemental measurements in a given region over long periods of time. Climate can be contrasted to weather, which is the present condition of these same elements and their variations over shorter time periods. This class is the central class of the ontology and therefore related to all relevant objects for climate definition (most of all regarding precipitation and temperature in the first place, but also adaptation and mitigation for climate changes).



Figure 6.5: **Climate** class of the AIT ontology.

The class **ClimateAdaptation** (Figure 6.6) has been introduced in order to gather RDF triples specifically designed to represent arrangements regarding climate change. Once the user is aware of how the climate in a certain area will change during a time period, she needs to define which steps need to be taken in order to prepare (e.g. the architecture of a building) to respond to these changes.

The **ClimateMitigation** class (Figure 6.7) has a quite similar purpose to the **ClimateAdaptation** class. In fact, it reacts to climate change rather than trying to prepare adaptations before climate change happens. This can be done by, e.g. political decisions such as thresholds for emissions of dangerous materials, etc.

The class **ClimateModel** (Figure 6.8) defines the used model for climate

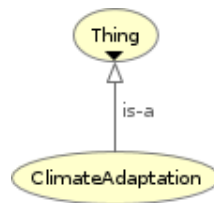


Figure 6.6: `ClimateAdaptation` class of the AIT ontology.

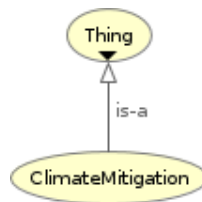


Figure 6.7: `ClimateMitigation` class of the AIT ontology.

predictions. In our case this is the COSMO-CLM⁴, a regional climate model which has been developed from a local model of the German Weather Service by the CLM (Climate Limited-area Modelling) community. This class has the following subclasses:

- **PhysicalApproximation**: represents the model for approximating the climate based on physical factors.
- **CTModel**: defines the Climate Twins model using proportional similarity and Hellinger coefficient.
- **Forecast**: a model used for making forecasts for a certain region.

The **Energy** class (Figure 6.9) describes all energy related phenomena. This concerns not only buildings, but also renewable energy sources which depend on the climate and can be planned by authorities based on the calculated climate change developments in a certain region. The **Energy** class has the following subclasses:

- **WindEnergy**: describes phenomena related to wind energy.
- **SolarEnergy**: describes phenomena related to solar energy.
 - **Sunlight**: subclass of **SolarEnergy** which can be used to define the amount of sunlight in a specific region.

⁴<http://www.clm-community.eu/index.php?menuid=17>

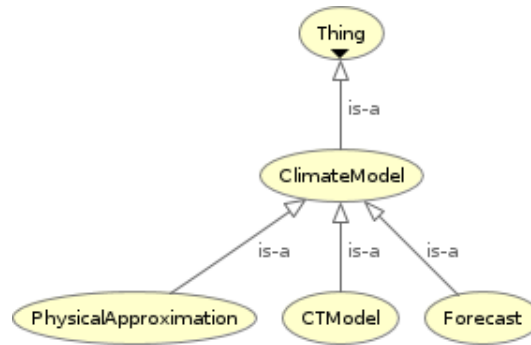


Figure 6.8: `ClimateModel` class of the AIT ontology.

The `Energy` class is included in the range of the `usesEnergy` property which specifies which kind of energy is used by a building.

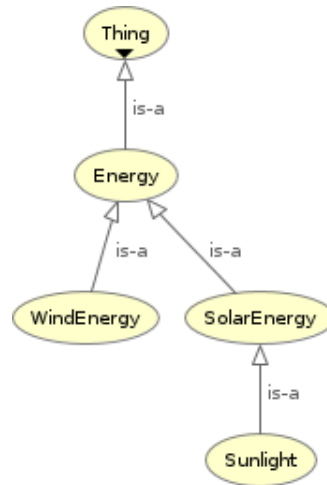


Figure 6.9: `Energy` class of the AIT ontology.

The class `MeasureType` (Figure 6.10) defines the type of measurement, for instance centimetre, Fahrenheit, etc.

`MeteorologicalPhenomena` (6.11) is a class for the representation of all phenomena definitions (e.g. snow, rain, etc.). It defines observations which are represented through time series and are relevant for prediction of climate change. The class `MeteorologicalPhenomena` has the following subclasses:

- **Cloud:** signals clouds as meteorological phenomenon.
- **Precipitation:** defines the main class for precipitation.

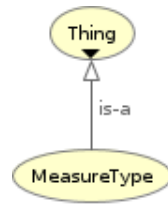


Figure 6.10: `MeasureType` class of the AIT ontology.

- **Rain**: subclass of precipitation defining rain.
- **Snow**: subclass of precipitation defining snow.
- **Wind**: defines wind as meteorological phenomenon.
 - **WindStorm**: subclass of wind (wind exceeding a certain speed).
- **Cyclone**: defines cyclone as meteorological phenomenon.
 - **Hurricane**: subclass of cyclone (more impact on the environment).

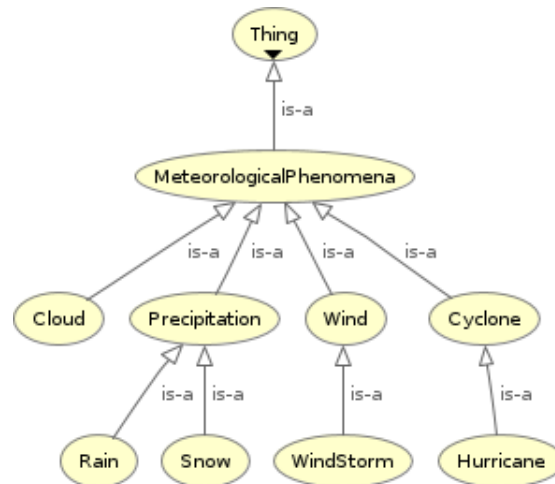


Figure 6.11: `MeteorologicalPhenomena` class of the AIT ontology.

The class `PrecipitationValueExpression` (Figure 6.12) represents detailed precipitation values in a dataset and can be used to filter regions based on precipitation (exact values or lower and upper thresholds). This class is the domain in the following properties:

- **hasExactPrecipitationValue**: defines an exact value of precipitation for filtering of regions.
- **hasPrecipitationIntervalMax**: maximum value of precipitation.
- **hasPrecipitationIntervalMin**: minimum value of precipitation.

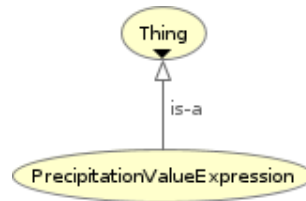


Figure 6.12: **PrecipitationValueExpression** class of the AIT ontology.

The class **Reliability** (Figure 6.13) defines how reliable the values are. The class is in the range of the property **producesReliability** which defines the reliability a climate model is able to produce.

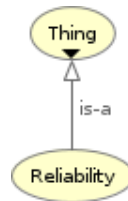


Figure 6.13: **Reliability** class of the AIT ontology.

The class **SpatialExpression** (Figure 6.14) is a superclass of any spatial object represented throughout the domain. Furthermore, it is the range of the property **hasSpatialExpression** which defines a spatial expression for a dataset. The subclass **CTAppGrids** defines which grids of the map are effected by the expression.

The **TemperatureExpression** class (Figure 6.15) is a class that represents temperature in detailed manner. It is part of the following properties:

- **hasTemperatureType**: defines the type of temperature of the measurement (what was measured).
- **hasExactTemperature**: defines the value for filtering regions based on temperature.

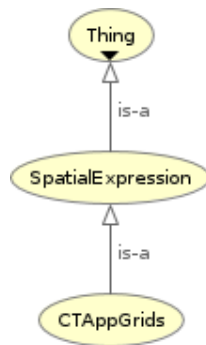


Figure 6.14: `SpatialExpression` class of the AIT ontology.

- `hasTemperatureIntervalMax`: defines the maximum temperature threshold.
- `hasTemperatureIntervalMin`: defines the minimum temperature threshold.

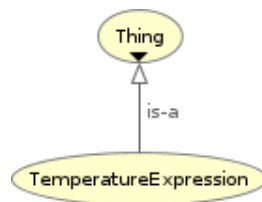


Figure 6.15: `TemperatureExpression` class of the AIT ontology.

The class `TemperatureType` (Figure 6.16) represents the source of temperature measurement. It defines the type of the temperature expression and individuals which can have abbreviations such as: “C” for “Celsius”, “F” for “Fahrenheit”, etc..

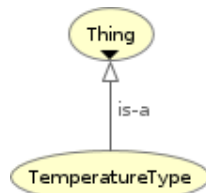


Figure 6.16: `TemperatureType` class of the AIT ontology.

The `TemporalExpression` class (Figure 6.17) is a superclass of any temporal object represented throughout the domain. Furthermore, it is the range of the property `hasTemporalExpression` which defines a temporal expression for a dataset.

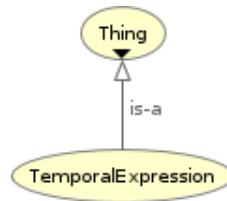


Figure 6.17: `TemporalExpression` class of the AIT ontology.

The class `ThermalProcess` (Figure 6.18) represents building insulation for annotation purposes. This class is the range of the `hasThermalProcess` property which defines the thermal process of a building. The following subclasses are defined for the class `ThermalProcess`:

- **AirConditioning:** thermal process for regulation of air temperature.
 - **Heating:** rising air temperature.
 - **Cooling:** decreasing air temperature.
- **SolarShading:** reducing warming through prevention of direct solar radiation.
- **BuildingInsulation:** isolation of buildings in order to reduce heating efforts or loss of warm air.

The `Weather` class (Figure 6.19) is meant to be used for general annotations where specific climate or other natural phenomena are not required.

Figure 6.20 shows a simplified overview of the most important classes of the ontology and their properties. As we have already described the class hierarchy of the ontology, we want to use the overview to describe the properties of the most important classes as well. A building has a thermal process which defines how the temperature inside the building is regulated (defined by the `hasThermalProcess` property) and it uses a certain type of energy (defined by the `usesEnergy` property). The climate model used for a certain scenario has a certain reliability (property `producesReliability`) and can have a proportional similarity or Hellinger coefficient (`hasProportionalSimilarity`).

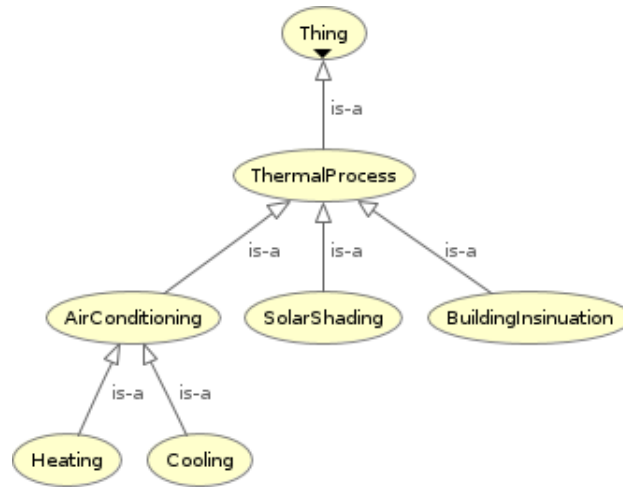


Figure 6.18: ThermalProcess class of the AIT ontology.

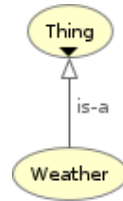


Figure 6.19: Weather class of the AIT ontology.

and `hasHellingerCoefficient`). The model can have an observed meteorological phenomenon (`observedMeteoPhenomena` property) and its subclass precipitation has a value (`hasPrecipitationValue`) and measure type (`hasMeasureType`). The precipitation value is defined by the precipitation value expression which has a minimum (`hasPrecipitationIntervalMin`) and a maximum (`hasPrecipitationIntervalMax`) for its interval as well as the exact value (`hasExactPrecipitationValue`). Furthermore, the temperature expression is defined by the type (`hasTemperatureType`), minimum (`hasTemperatureIntervalMin`) and maximum (`hasTemperatureIntervalMax`) thresholds, and exact temperature (`hasExactTemperature`).

6.1.3 Use Case

Our use case, which is based on the ontology from TaToo's Validation Scenario 1 "Climate Change Twin Regions – Discovery Platform", presents the generation of groups which unite users and time series sharing the same do-

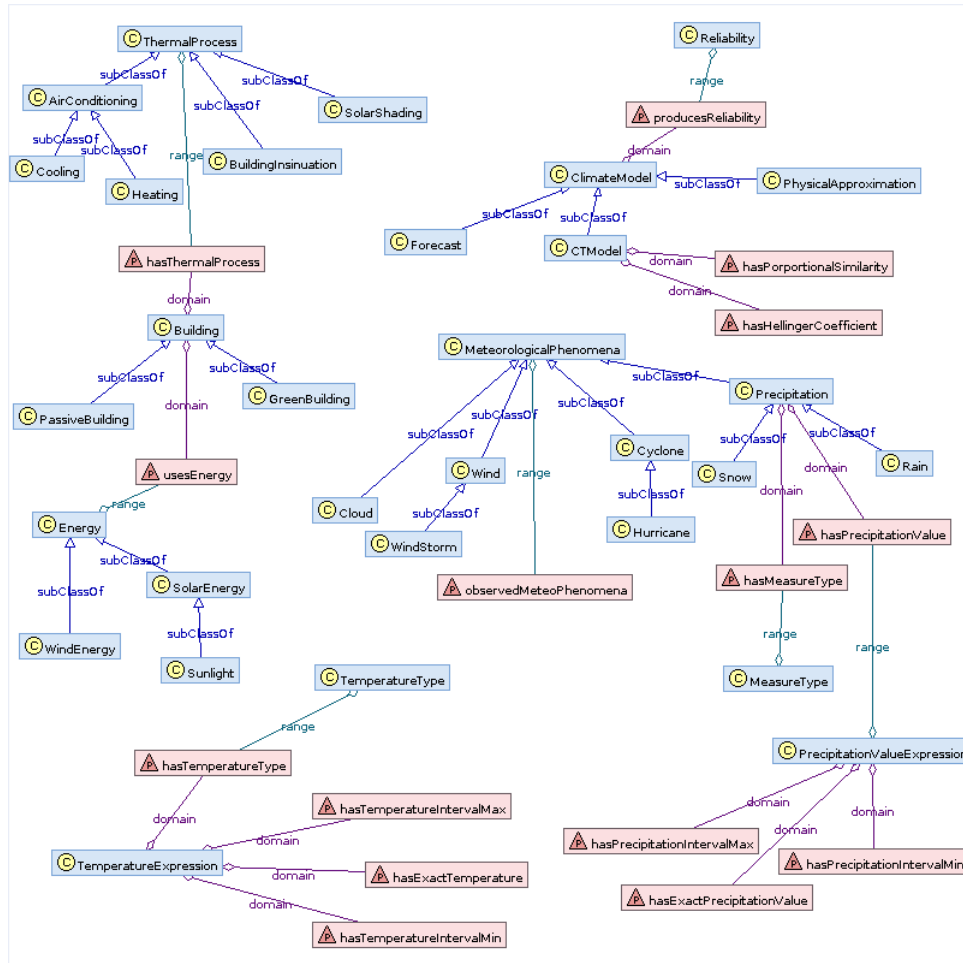


Figure 6.20: A simplified overview of the most important parts of the ontology.

main of interest. The following steps describe the workflow of our Climate Twins Use Case:

1. *Definition of users:* Creation of sample users from different fields (as described below) and different interests (required views on the data).
2. *Definition of time series:* Creation (import) of different time series from the climate area (related to prediction of climate change).
3. *Mapping of user (domain) and time series ontology:* Showcase of how the “Climate Twins” domain ontology is mapped to the bridge ontology.

4. *Reasoning examples*: Showcase of how new RDF triples are inferred as a result of mapping the domain ontology to the bridge.
5. *Group generation*: Generation of groups with time series and users sharing similar topics and interests.

Users

The most relevant user types in our Climate Twins Use Case are the following:

- *Politicians* and people working in public authorities who are interested in public issues linked to (the change of) climate conditions at a regional or local level. Such issues concern several policy areas like spatial planning, housing, agriculture and forestry, water and energy supply, etc.
- *Business managers* in industries which are climate-sensitive. This applies to all industries where renewable resources matter much in the production of their goods (e.g. food production, hydro energy). It also applies to industries where the climate is an important framework condition (e.g. tourism, construction).
- *Scientists* who work in fields of research related to the issues mentioned above and contribute through the results of their research to climate adaptations.
- *Non-professional users* who are only interested in information in trends and time series data of a certain field.

Example

Our example scenario covers 3 users with different backgrounds. The first user (see screenshot in Figure 6.21) is a politician. This user works for the ministry of environment and is interested in time series data about the climate change. Furthermore, the user loads the Climate Twins ontology as his domain ontology to the system, which enables him to set “Climate Change” as his topic of interest. This user needs to get data which supports his decisions to implement regulations and make investments in certain environmental fields.

The following RDF triples are most relevant for further processing to generate data of interest for the user (the classes and properties with the prefix *ct* - for climate twins - are defined in the domain ontology and the

The screenshot shows a web form titled "Create User" with a dark header bar containing window control icons. The form contains the following fields and controls:

- Load Image:** A text input field followed by "Browse..." and "Upload" buttons.
- Uploaded Image:** A placeholder area showing a portrait of a man in a suit.
- Name:** A text input field containing "Andrew Christmas".
- Position:** A dropdown menu with "Politician" selected.
- Institute:** A text input field containing "Austrian Government".
- Department:** A text input field containing "Ministry of Environment".
- Load Ontology:** A text input field containing "ClimateTwins_Domain_1.owl" followed by "Browse..." and "Upload" buttons.
- Topic:** A dropdown menu with "Climate Change" selected.
- Create:** A button at the bottom of the form.

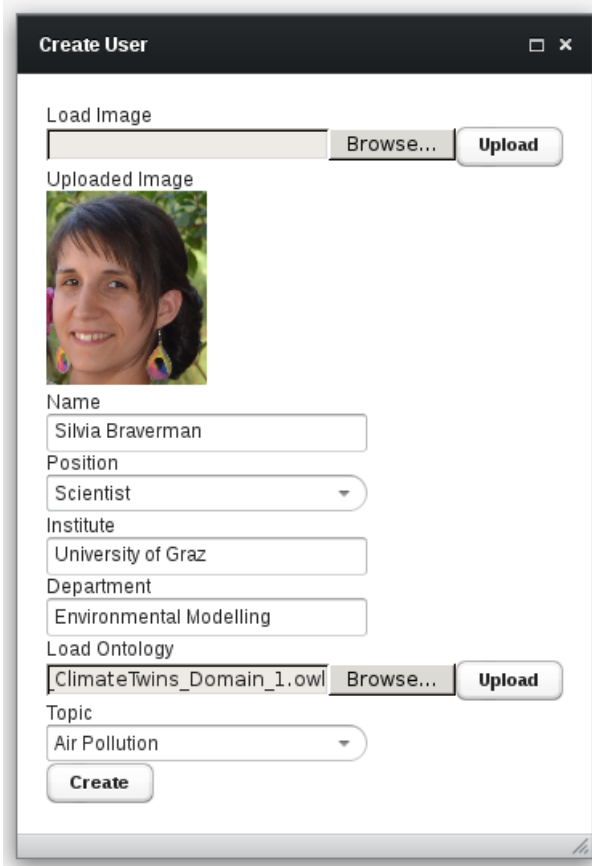
Figure 6.21: Sample politician user with Climate Twins ontology.

individuals – e.g. `:user1` – are defined by the Web portal, after creation of the user):

```
:user1 rdf:type ct:User .
:user1 ct:hasRole ct:Politician .
ct:Politician ct:hasEmployer ct:Government .
:user1 ct:worksFor ct:MinistryOfEnvironment .
:user1 ct:hasTopic ct:ClimateChange .
```

The second user (see screenshot in Figure 6.22) is a scientist and works in the Environmental Modelling department of the University of Graz. The user is interested in time series data with the topic of “Air Pollution” and has the Climate Twins ontology as her domain ontology as well. This user is in search for data for her research and therefore needs time series with a certain topic.

The following RDF triples are most relevant for the second user:



The screenshot shows a 'Create User' window with the following fields and values:

- Load Image:** An empty text box with 'Browse...' and 'Upload' buttons.
- Uploaded Image:** A small portrait photo of a woman.
- Name:** Text box containing 'Silvia Braverman'.
- Position:** Dropdown menu with 'Scientist' selected.
- Institute:** Text box containing 'University of Graz'.
- Department:** Text box containing 'Environmental Modelling'.
- Load Ontology:** Text box containing 'ClimateTwins_Domain_1.owl' with 'Browse...' and 'Upload' buttons.
- Topic:** Dropdown menu with 'Air Pollution' selected.
- Create:** A button at the bottom.

Figure 6.22: Sample scientist user with Climate Twins ontology.

```
:user2 rdf:type ct:User .
:user2 ct:hasRole ct:Scientist .
ct:Scientist ct:hasEmployer ct:University .
:user2 ct:worksFor ct:EnvironmentalModellingGroup .
:user2 ct:hasTopic ct:AirPollution .
```

The third user (see screenshot in Figure 6.23) is a business manager from the Austrian Tourist Agency and works in the Regional Development department. He is interested in time series with the “Climate Change” topic in order to plan the infrastructure for tourism in a certain region of interest (i.e. the region he is responsible for).

The following RDF triples are most relevant for the third user:

```
:user3 rdf:type ct:User .
:user3 ct:hasRole ct:BusinessManager .
```

The screenshot shows a 'Create User' window with the following fields and options:

- Load Image:** A text input field followed by 'Browse...' and 'Upload' buttons.
- Uploaded Image:** A small thumbnail image of a man.
- Name:** A text input field containing 'Robert Vanderburg'.
- Position:** A dropdown menu showing 'Business Manager'.
- Institute:** A text input field containing 'Austrian Tourist Agency'.
- Department:** A text input field containing 'Regional Development'.
- Load Ontology:** A text input field containing 'ClimateTwins_Domain_1.owl' followed by 'Browse...' and 'Upload' buttons.
- Topic:** A dropdown menu showing 'Climate Change'.
- Create:** A button at the bottom of the form.

Figure 6.23: Sample business manager user with Climate Twins ontology.


```
ct:BusinessManager ct:hasEmployer ct:Company .
:user3 ct:worksFor ct:RegionalDevelopment .
:user3 ct:hasTopic ct:ClimateChange .
```

Since these three users have the topics “Climate Change” and “Air Pollution” selected from their domain ontology, they get the list of time series presented in Figure 6.24 as a result.

The resulting time series are retrieved based on annotations from previous users or reasoning. In our case the time series are retrieved based on the annotation of the **Topic** subclasses **ClimateChange** and **AirPollution**. All time series with these topics are retrieved from the system. This is only the starting point of the scenario. To experience the real work in the scenario we need to have a look at the ontology mapping and reasoning processing steps of our workflow.

As a first step towards group generation, the relevant parts of the domain

Home	User Settings	Ontology	Groups	Related Users	Time Series	SPARQL Endpoint	Logout
------	---------------	----------	--------	---------------	-------------	-----------------	--------



Semantic Time Series Processing

Time Series found:

TITLE	CATEGORY	START	END	VALUE
CO2 emissions from electricity and heat production	Air Pollution	1960	2011	55
CO2 emissions from residential buildings and commercial and pu	Air Pollution	1960	2011	55
CO2 emissions from solid fuel consumption (kt)	Air Pollution	1960	2010	54
Electricity production from coal sources (% of total)	Climate Change	1960	2012	56
Electricity production from oil sources (% of total)	Climate Change	1960	2012	56
GHG net emissions/removals by LUCF (Mt of CO2 equivalent)	Air Pollution	1990	2009	23
Electricity production from renewable sources (kWh)	Climate Change	1960	2012	56
Electricity production from renewable sources	Climate Change	1960	2012	56
Electricity production from nuclear sources (% of total)	Climate Change	1960	2012	56
CO2 emissions from solid fuel consumption (% of total)	Air Pollution	1960	2010	54
CO2 emissions from gaseous fuel consumption (% of total)	Air Pollution	1960	2010	54
Droughts	Climate Change	2009	2009	4
Population in urban agglomerations of more than 1 million (% of tc	Climate Change	1960	2012	56
CO2 emissions from liquid fuel consumption (% of total)	Air Pollution	1960	2010	54
Electricity production (kWh)	Climate Change	1960	2012	56
Electricity production from renewable sources	Climate Change	1960	2012	56
CO2 emissions from liquid fuel consumption (kt)	Air Pollution	1960	2010	54
Nitrous oxide emissions (thousand metric tons of CO2 equivalent)	Air Pollution	1990	2010	8
GDP (current US\$)	Climate Change	1960	2012	56

Figure 6.24: A list of time series with the topics “Climate Change” and “Air Pollution”.

ontology are mapped to the bridge ontology of our system. Listing 6.1 shows the merged parts of the two ontologies, where the domain ontology parts are marked in blue and the bridge ontology parts in red:

```

1 (1)
2 ct:User owl:sameAs bridge:User .
3 ct:Topic owl:sameAs bridge:Topic .
4
5 (2)
6 ct:Government rdfs:subClassOf bridge:Institution .
7 ct:University rdfs:subClassOf bridge:Institution .
8 ct:Company rdfs:subClassOf bridge:Institution .
9
10 (3)

```

```

11 ct:Building rdfs:subClassOf bridge:Topic .
12 ct:Energy rdfs:subClassOf bridge:Topic .
13 ct:Weather rdfs:subClassOf bridge:Topic .
14
15 (4)
16 ct:Reliability rdfs:subClassOf bridge:Subject .
17 ct:ClimateAdaptation rdfs:subClassOf bridge:Subject .
18 ct:ClimateMitigation rdfs:subClassOf bridge:Subject .
19
20 (5)
21 ct:TimeSeries owl:sameAs bridge:TimeSeries .
22 ct:TemperatureExpression rdfs:subClassOf bridge:Property .
23 ct:PrecipitationValueExpression rdfs:subClassOf bridge:Property .
24 ct:SpatialExpression rdfs:subClassOf bridge:Property .
25 ct:TemporalExpression rdfs:subClassOf bridge:Property .

```

Listing 6.1: Mapping of domain ontology and bridge ontology.

In order to better explain the ontology mappings, we have split the mapped concepts in 5 parts (paragraphs). The first part defines which classes of the domain ontology are equal to which classes of the bridge ontology. Per definition of our bridge concept these are all classes with the same name (i.e. `User` and `Topic` in our example). This is the point where alignment between the bridge and the domain happens. After the statements of part (1) all individuals of `ct:User` become individuals of `bridge:User` (the same is valid for `ct:Topic` and `bridge:Topic`). This has the advantage that it does not matter whether a user has been defined in a domain ontology or in the bridge.

Part (2) defines subclasses for a bridge class. In this case the subclasses are `ct:Government`, `ct:University`, and `ct:Company` and the bridge class is `bridge:Institution`. This means that the three classes of the domain ontology become subclasses of the bridge class. Therefore, every property of the Institution defined in the bridge ontology is also valid for the special classes of the domain ontology which enables us to use new kinds of institutions when we define affiliations of users by using, e.g. the `:hasEmployer` property.

In parts (3) and (4) new subclasses for `bridge:Topic` and `bridge:Subject` are defined, which enables a user to be assigned to new topics (interests) and to create new types of subjects in order to add new resources to a topic.

Finally, part (5) defines the concept of a time series for the Climate Twins domain ontology. It states that `ct:TimeSeries` and `bridge:TimeSeries` are the same, and defines value expressions from the domain ontology as properties of a time series.

After the ontology mapping step, the domain and bridge ontologies can

be used as one single ontology and form the basis for the reasoning process. Therefore, the following rules, which have been defined previously in Description Logic, need to be applied by a reasoner (Pellet):

- Rule 1 - $topic : bridge:Topic \models \forall bridge:hasTopic.user1 \sqcup \forall bridge:hasTopic.user2 \sqcup \forall bridge:hasTopic.user3$ - Extraction of relevant topics for users.
- Rule 2 - $ts : bridge:TimeSeries \models \forall bridge:hasTopic.topic$ - Retrieval of time series with relevant topics and subtopics.
- Rule 3 - $bridge:hasTopic.user \equiv bridge:hasTopic.group \Rightarrow (user, group) : bridge:hasUser$ - Assignment of users to groups.
- Rule 4 - $bridge:hasTopic.ts \equiv bridge:hasTopic.group \Rightarrow (ts, group) : bridge:hasTimeSeries$ - Assignment of time series to groups.

Rule 1 defines the extraction of relevant topics from user definitions. This means that we look which topics the users who are observed are related to and collect them in a list. Of course, not only the topics themselves are taken into account, but also all subtopics (subclasses of the according topic class).

Rule 2 defines retrieval of all time series with relevant topics and subtopics. This is achieved by generation of SPARQL queries and the application of them to the knowledge base which in the next step retrieves all IDs of time series. These IDs can be used in order to retrieve the real time series from the time series database.

Rule 3 defines the creation of new groups based on extracted user topics and the assignment of the respective users to these groups. This is done by simply creating new instances of groups and extending them by triples with the properties **hasTopic** (for the topics) and **hasUser** for the users which need to be assigned.

Finally, in Rule 4, we also assign the right time series (with the same topic or subtopic) to the groups.

The application of the reasoning rules to our example use case (as described in the steps of the reasoning process) is shown in Listing 6.2. We have enumerated the single steps according to the previously described definitions.

```

1 (1)
2 : user1 ct : hasTopic ct : ClimateChange .
3 : user2 ct : hasTopic ct : AirPollution .
4 : user3 ct : hasTopic ct : ClimateChange .
5
6 |

```

```

7      v
8
9  ct:ClimateChange , ct:AirPollution
10
11  (2)
12  SELECT ?timeseries
13  WHERE {
14    ?timeseries bridge:hasTopic ct:ClimateChange ,
15                      ct:AirPollution .
16  }
17
18      |
19      v
20
21  ts1 , ts2 , ts3 , ...
22
23  (3)
24  :group1 rdf:type bridge:Group ;
25          bridge:hasTopic ct:ClimateChange ;
26          bridge:hasUser user1 ;
27          bridge:hasUser user2 .
28  :group2 rdf:type bridge:Group ;
29          bridge:hasTopic ct:AirPollution ;
30          bridge:hasUser user2 .
31
32  (4)
33  :group1 bridge:hasTimeSeries :ts1 ...
34  :group2 bridge:hasTimeSeries :ts2 ...

```

Listing 6.2: The reasoning process described for the example of the Climate Twins use case.

Another relevant part of the process is shifting users and time series between groups based on ratings and annotations. This means that we perform a periodic check of ratings and annotations of every user and time series. The check has the goal to evaluate whether a user or time series is in the right group. This is achieved by investigating which users have provided ratings and annotations for another user or time series. The critical point is reached when a user or time series receives a better rating and more annotations from users of another group, which initiates a shift of the element to the other group.

Figure 6.25 shows the generated groups with users and time series based on ontology mapping and reasoning. The “Climate Change” group is at the top and contains the two users and all time series which are assigned to the “Climate Change” topic and all its subtopics. The “Air Pollution” group shows tables with all users and time series assigned to the “Air Pollution” topic. Initially, all user and time series ratings are set to 2.5 stars in order

Climate Change - Group

Group Users:

NAME	ROLE	AVERAGE RATING
Andrew Christmas	Politician	
Robert Vanderburg	Business Manager	

Group Time Series:

TITLE	CATEGORY	AVERAGE RATING
Electricity production from coal sources (% of total)	Climate Change	
Electricity production from oil sources (% of total)	Climate Change	
Electricity production from renewable sources (kWh)	Climate Change	
Electricity production from renewable sources	Climate Change	
Electricity production from nuclear sources (% of total)	Climate Change	

Air Pollution - Group

Group Users:

NAME	ROLE	AVERAGE RATING
Silvia Braverman	Scientist	

Group Time Series:

TITLE	CATEGORY	AVERAGE RATING
CO2 emissions from electricity and heat production	Air Pollution	
CO2 emissions from solid fuel consumption (kt)	Air Pollution	
GHG net emissions/removals by LUCF (Mt of CO2 equivalent)	Air Pollution	
CO2 emissions from solid fuel consumption (% of total)	Air Pollution	
CO2 emissions from gaseous fuel consumption (% of total)	Air Pollution	

Figure 6.25: Screenshot of the Climate Change and Air Pollution groups.

to have an average rating at the beginning and to avoid movement between groups too early.

6.2 Anthropogenic Impact and Global Climate Change

The “Anthropogenic Impact and Global Climate Change” platform has been developed by the Masaryk University in Brno and used as a Validation Scenario in the TaToo FP7 project. One of the outputs of the TaToo project regarding this Validation Scenario was an ontology which describes the scenario and its meta-data. This ontology is used as an input for this thesis in order to show how the developed Ontology Mapping and Reasoning methods can be applied.

6.2.1 Introduction

In order to find which effects pollution has on the human health (i.e. anthropogenic impact), air pollution monitoring needs to be combined with epidemiological data [Rizzoli et al., 2010]. This is only possible when there are enriched capabilities of data discovery available. The TaToo project has the goal to provide tools for semantic discovery of resources and hence enable this combination as well as the enrichment with additional meta-data.

This is exactly the main focus of the validation scenario “Anthropogenic impact and global climate change”. In fact, it correlates environmental pollutants (POPs – Persistent Organic Pollutants) and their impact on the human health as well as the transport of environmental pollutants and global climate change⁵. The aim is to create a place for researchers, domain experts, and other decision makers who want to discover new resources and to access knowledge about the correlation in a user-friendly way. The discovery of resources is not fulfilled by standard search engines but rather by semantic discovery supported by TaToo tools. The basis for the search is a domain ontology which is provided by a user group and defines filters, a definition for an object of interest, and other relevant concepts for the domain. Furthermore, new relationships are discovered and created between different domains (e.g. environmental pollution and tumor epidemiology). This is also needed to enable the system to use resources from multiple domains.

The validation scenario consists of two web portals which use the developed domain ontology. The two portals are:

⁵<http://www.tatoo-fp7.eu/tatooweb-d7/about-tatoo/validation-scenarios>

*System for Visualizing of Oncological Data (SVOD)*⁶: This is a Web portal which provides tumor epidemiology data in the Czech Republic⁷. The data is free to all users and conforms to the open data initiative of the Czech Republic. It is based on the Czech National Cancer Registry⁸ which is managed by the Institute of Health Information and Statistics⁹. Epidemiological data in the portal is validated and ranges from 1977 - 2008 which is a unique data set in Europe.

Figure 6.26 shows a screenshot of the SVOD Web portal. The screenshot represents a selection window for epidemiological data of a human being. The user is able to select a region of the human body and browse for time series data about certain diseases of the observed area among the population of the Czech Republic.

*Global Environmental Assessment Information System (GENASIS)*¹⁰: This is a web portal which provides information support for the implementation of the Stockholm Convention¹¹ on Persistent Organic Pollutants (POPs)¹². The portal is developed in accordance with the objectives of a Single Information System of the Environment¹³ of the Ministry of Environment of the Czech Republic. It is connected to other data sources and this enables the assessment of anthropogenic impact on the environment, the ecology, and human health risks. GENASIS contains data collected by the Research Centre for Toxic Compounds in the Environment¹⁴ of the Masaryk University. The Web portal offers analytical tools with “statistical” program units for basic processing of measured environmental data.

Figure 6.27 shows a screenshot of the GENASIS Web portal. The time series view is selected presenting “Alpha-HCH (air active - gas phase) time series” data. The data ranges from 1994 to 2012 and shows a typical view of the GENASIS portal in its role as a data source of compounds time series data.

6.2.2 Ontology

The ontology of the Validation Scenario 2 “Anthropogenic Impact and Global Climate Change” (MU ontology) is a result of the TaToo project. It has been

⁶<http://www.svod.cz>

⁷<http://www.tatoo-fp7.eu/tatooweb-d7/about-tatoo/validation-scenarios>

⁸<http://www.linkos.cz>

⁹<http://www.uzis.cz>

¹⁰<http://www.genasis.cz>

¹¹<http://chm.pops.int>

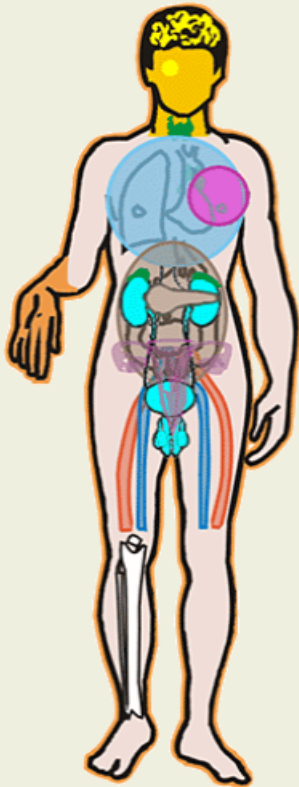
¹²<http://www.tatoo-fp7.eu/tatooweb-d7/about-tatoo/validation-scenarios>

¹³<http://www.mzp.cz>

¹⁴<http://www.recetox.muni.cz>

INCIDENCE A MORTALITA - vývoj v čase

Zvolte požadovanou diagnózu ?



**III. NÁDORY DÝCHACÍ SOUSTAVY A
NITROHRUDNÍCH ORGÁNŮ**

- [C30 - ZN nosní dutiny a středního ucha](#)
- [C31 - ZN vedlejších dutin](#)
- [C32 - ZN hrtanu](#)
- [C33 - ZN průdušnice - trachey](#)
- [C34 - ZN průdušky - bronchu a plice](#)
- [C33,C34 - ZN průdušnice, průdušky a plice](#)
- [C37 - ZN brzlíku - thymu](#)
- [C38 - ZN srdce, mezihrudí - mediastina a pohrudnice - pleury](#)
- [C39 - ZN jiných a nepřesně určených lokalizací v dých.soustavě a nitrohrud.orgánech](#)

[Přímo zobrazit všechny diagnostické skupiny](#)

Figure 6.26: Screenshot of the SVOD Web portal.

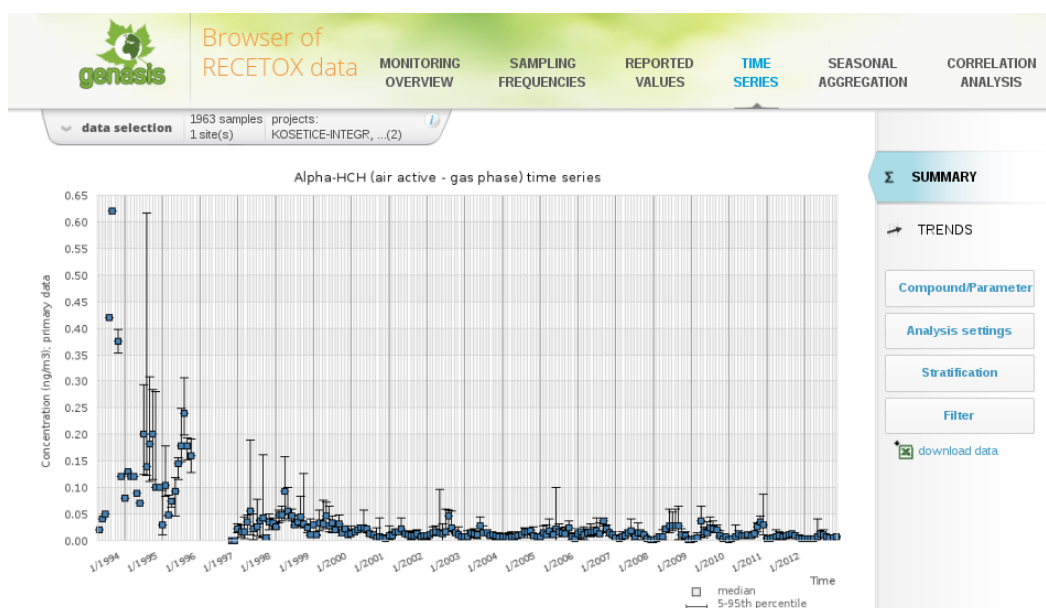
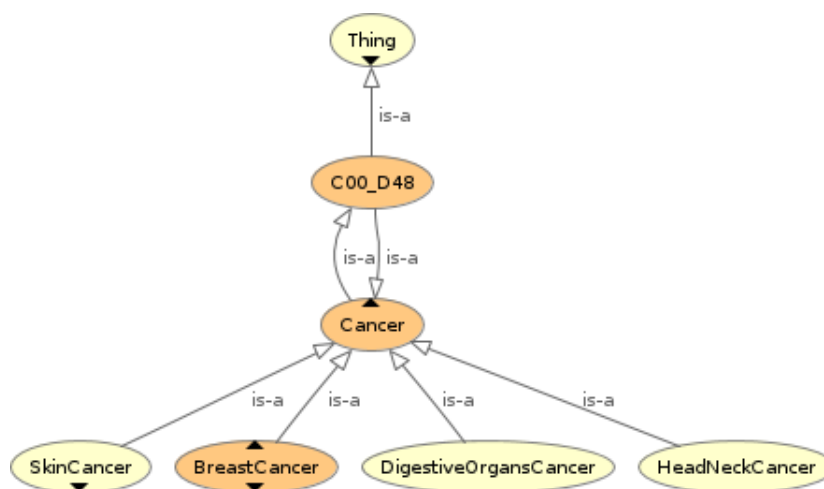
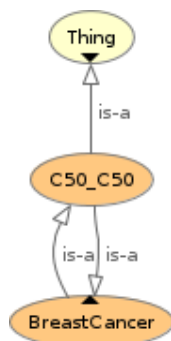


Figure 6.27: Screenshot of the GENASIS Web portal.

developed to enrich the data model of possible applications in the respective domain and represents single components of the field of application which are represented as classes of the ontology. The final version of the ontology uses the ICD-10 (International Classification of Diseases)¹⁵ class hierarchy and recommended POPs and Matrix taxonomy based on the Stockholm Convention. In the following, the single first-level classes of the ontology are described in detail by presenting the part of the graph which is related to the class (produced using OWLViz in Protégé) together with a textual description of the class itself.

The class **Cancer** (Figure 6.28) represents cancer diseases and their sub concepts. It inherits from the class **C00_D48** which represents general neoplasms. It has the following subclasses:

- **SkinCancer**: Malignant neoplasms of skin.
- **BreastCancer**: Malignant neoplasms of breast.
- **DigestiveOrgansCancer**: Malignant neoplasms of organs (stomach, colon, liver, etc.).
- **HeadNeckCancer**: Malignant neoplasms in the neck or head regions (e.g. brain tumors).

Figure 6.28: **Cancer** class of the MU ontology.Figure 6.29: **BreastCancer** class of the MU ontology.

As an example for a specialized cancer class, the class **BreastCancer** (Figure 6.29) is shown as a successor not only of the **Cancer** class, but also of the class **C50_C50** which stands for malignant neoplasm of breast (connective tissue).

The class **Compound** (Figure 6.30) represents a “chemical compound”. Chemical compound is a chemical substance consisting of two or more different chemical elements. These elements consist of one type of atom. Common examples of elements are iron, mercury, lead, etc. The subclasses of the class are:

- **POP**: Persistent Organic Pollutant is an organic compound that persists in the environment for a long time (e.g. several years, or even

¹⁵<http://www.who.int/classifications/icd/en/>

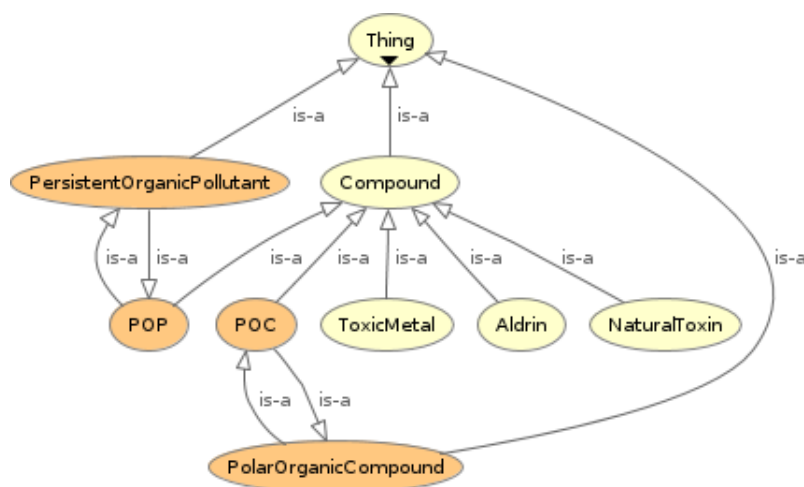


Figure 6.30: Compound class of the MU ontology.

decades), bioaccumulates, is toxic, and is subject to long-range transport. The class POP is defined to be the same as the class **Persistent OrganicPollutant**.

- **POC**: Organic compound with relevant polarity. The polarity directly depends on the electronegativity difference between atoms. The class POC is defined to be the same as the class **PolarOrganicCompound**.
- **ToxicMetal**: Metals which culminate a toxic effect on organisms and life in general. The poisonousness can be a result of forming poisonous soluble compounds.
- **Aldrin**: A pesticide which was used to treat seed and soil. It became notorious as a persistent organic pollutant.
- **NaturalToxin**: Chemicals which are produced by living organisms. This means that this kind of toxins are harmless to the organisms themselves but can have a negative impact on human health.

There is also a large number of subclasses of the presented classes which represent all the subcategories of the compounds. However, a detailed description of environmental compounds is not subject of this thesis.

The class **Data** (Figure 6.31) represents a data source which is a type of source for (mostly) digitized data: a database; a computer file; a data stream. Data from such sources is usually formatted and contains a certain amount of meta-information. It has the following subclasses:

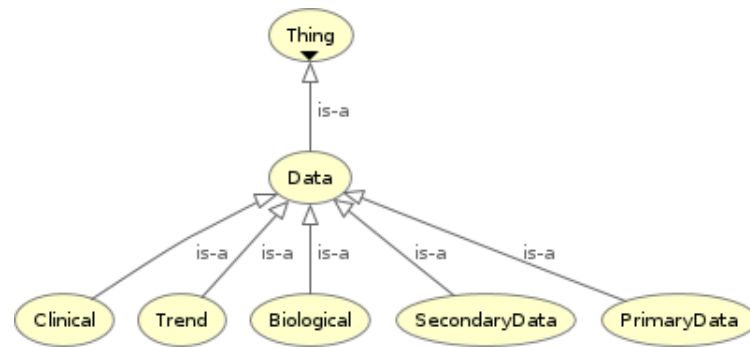


Figure 6.31: Data class of the MU ontology.

- **Clinical**: the result of clinical studies. Achieved through trials and collection of medical records, etc.
- **Trend**: data which shows certain trends and is therefore of special interest for the analysis.
- **Biological**: data collected from biological sources such as DNA, population, natural toxin concentration, etc.
- **SecondaryData**: is data which is not related to users of the system (respectively the web portals of the validation scenario), for example, data from open data sources of the Czech Republic's government.
- **PrimaryData**: in contrast to secondary data, this is personal data of the system's users, as well as data entered by users directly.

The **Data** class represents the measurements in a time series (the values of single time series slots) which need to be categorized in order to decide how they should be processed further. These classes are directly related to the **Property** class of the bridge ontology. The exact measurements which correspond to the type of data are therefore saved as values of the property.

The class **Disease** (Figure 6.32) represents several kinds of diseases which can be caused by elements of the class **Compound** and one of its subclasses. It is equivalent to the class **ICD10_Chapter**. The previously introduced class **Cancer** is a subclass of the class **Disease**.

The class **EpidemiologicalMeasures** (Figure 6.33) defines how the illness of the population should be measured (i.e. which factors contribute to negative impact on the human health). The following subclasses define the types of epidemiological measures:

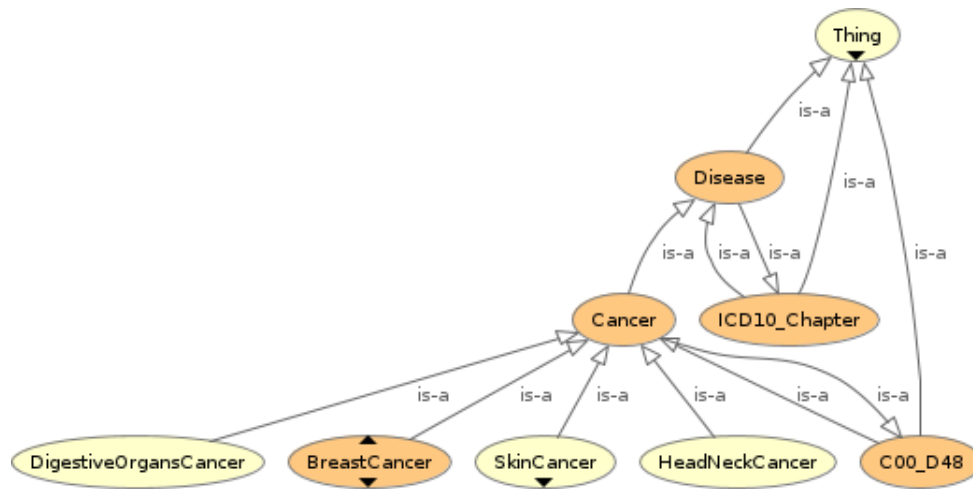


Figure 6.32: Disease class of the MU ontology.



Figure 6.33: EpidemiologicalMeasures class of the MU ontology.

1. **Incidence:** measures the risk of developing a new disease in a certain period of time.
2. **Mortality:** measures the number of deaths in a certain population.
3. **Prevalence:** proportion of a population which has developed a disease.

The **Matrix** class (Figure 6.34) represents a matrix - the media, of animated or unanimated nature, where POPs can accumulate. The possible media is defined as subclasses:

- **Foodwebs:** define feeding connections in an ecological community (also known as consumer-resource-system). POPs can be transported through feeding connections.

- **HumanMilk:** transportation of POPs through human breast milk.
- **Plants:** transportation through plants (seeds).
- **Sediment:** transportation through weathering and erosion.
- **HumanTissues:** transportation through interchange of tissue.
- **Air:** transportation through air.
- **Biofilm:** transportation through microorganisms.
- **Rainwater:** transportation through rain.
- **Soil:** transportation through dust.
- **SurfaceWater:** transportation through water (running water).
- **Animals:** transportation through animals (animal contact, not consumption as in foodwebs).
- **UndergroundWater:** transportation through subsoil water.

The class **ProjectType** (Figure 6.35) describes the type (kind) of a project together with its topic. The different kinds of a project are:

- **ShortTermMonitoring:** short observations of an area or population.
- **CaseStudy:** explanatory analysis of a population.
- **Screening:** identification of diseases developed by individuals through tests.
- **LongTermMonitoring:** long term observations of an area or population.

The **Risk** class (Figure 6.15) represents a possible risk in inter-domain perspective which can be an ecological risk (if it endangers the environment) or a human risk (if it directly endangers human beings).

Figure 6.37 shows a simplified overview of the MU ontology. Most of the defined classes are concentrated around the following “hot spots”:

- **Cancer:** This is a subclass of the class **Disease** and serves as the base class for several kinds of cancer.
- **Data:** Base class for several types of data, e.g. clinical, trend, biological, primary, and secondary.

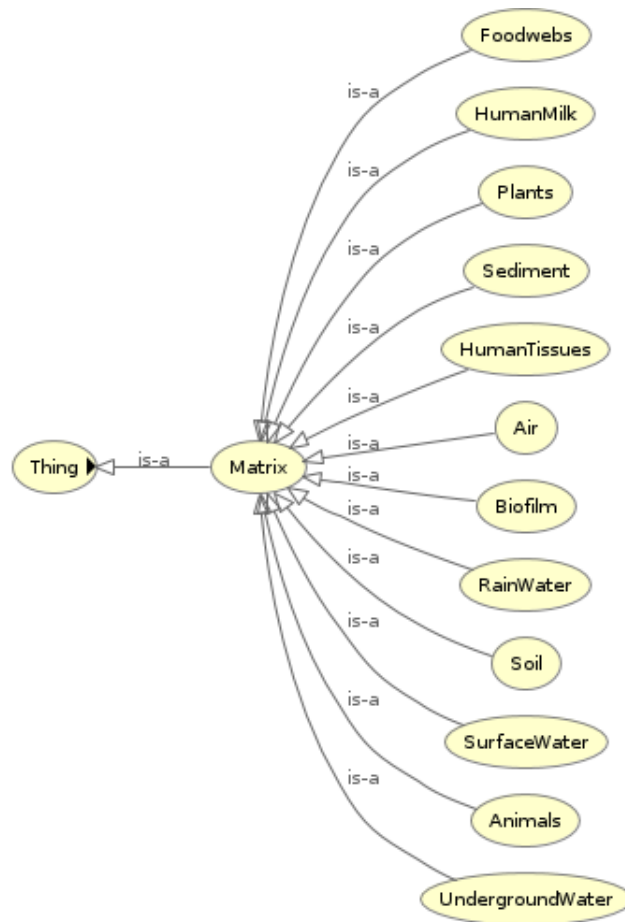


Figure 6.34: Matrix class of the MU ontology.

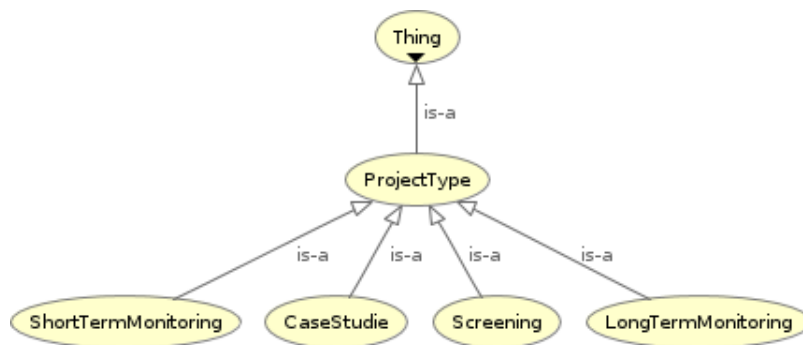


Figure 6.35: ProjectType class of the MU ontology.

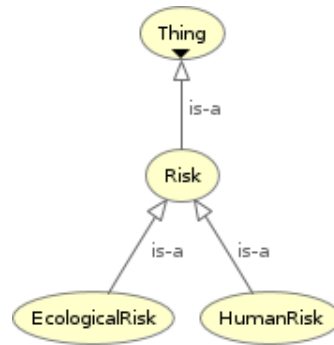


Figure 6.36: Risk class of the MU ontology.

- **Epidemiology**: Base class for mortality, incidence, and prevalence.
- **Risk**: Different kinds of risks, such as human and ecological risk.
- **Compound**: Defines the materials which are observed for having an impact on human health.
- **ProjectType**: Base class for different types of projects to observe impact on hazardous materials on the human health.
- **Matrix**: Defines different objects which can be found in the environment and contribute to the process.
- **hasAnnotation**: The most important property. Defines annotations for, e.g. diseases and compounds.

6.2.3 Use Case

Our second use case is based on the ontology from TaToo’s Validation Scenario “Anthropogenic Impact and Global Climate Change”, which demonstrates the selection of important parts of time series data for different kinds of users. The following steps describe the workflow of our Anthropogenic Impact and Global Climate Change Use Case:

1. *Creation of users with the MU domain ontology*: three different user types (specified below) are created using the MU domain ontology. The users have different backgrounds, are interested in different parts of the data, and need different views on their parts of a time series.

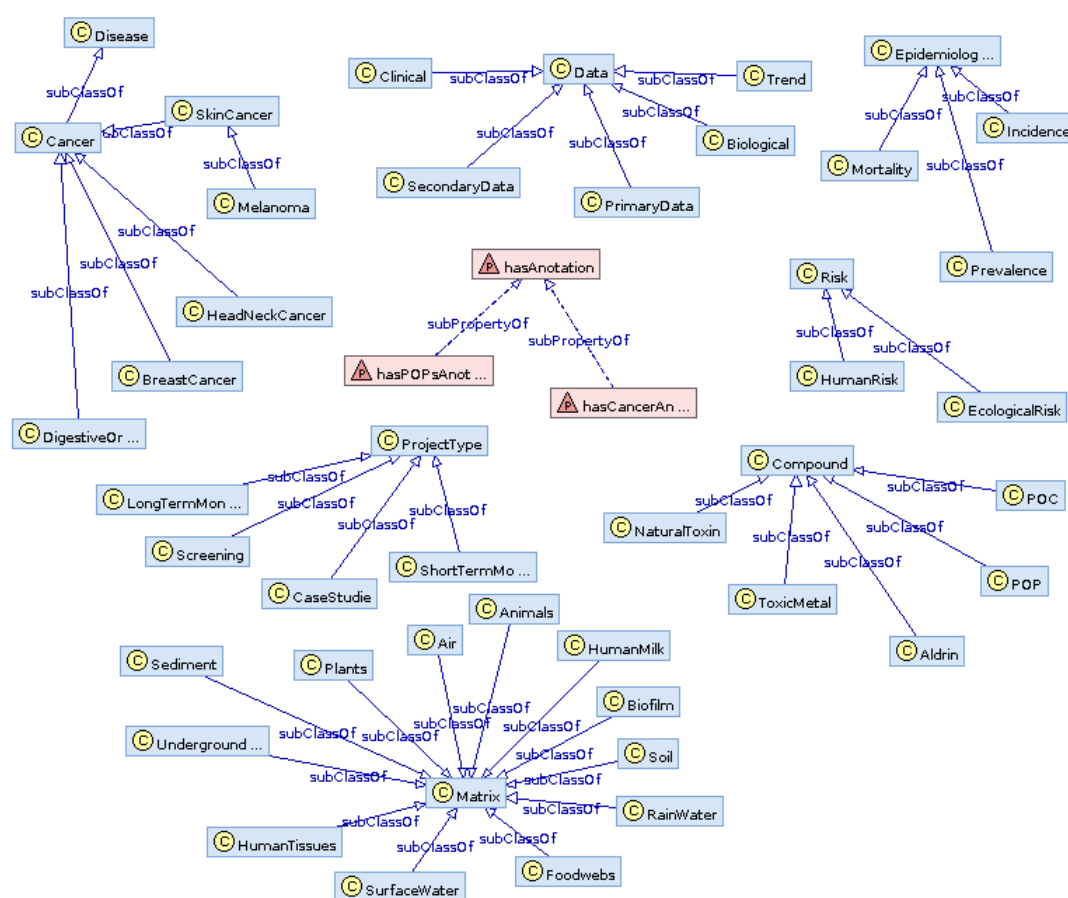


Figure 6.37: Simplified overview of the MU domain ontology.

2. *Retrieval of data from the according data sets:* The data of the use case is provided as a catalogue of resources in RDF/XML format¹⁶. Therefore, the data can be accessed by performing SPARQL queries and using the domain ontology in our Semantic Time Series Processing framework to provide the right view to the right user.
3. *Selection and filtering of data sets depending on user interest:* After the retrieval of resources related to a certain topic, the Web portal provides a customized view on relevant data parts to every user of the system. In order to achieve this, the data is filtered according to semantic time series expressions generated from the user inputs. Finally, the semantic time series processor is responsible to retrieve the filtered data from the knowledge base.

Users

The different kinds of users in this use case can be subdivided into the following three categories:

- *Scientific users:* scientific users are regular users with scientific background and assumed IT skills. They use the system to discover resources from both domains (POPs, health issues). They will be able to find resources, find similar resources (having already found some resource), compare the resources, and also to find connections between resources.
- *Domain experts:* the group of domain experts consists of users who have some additional functionality compared to scientific users. Domain experts can also evaluate resources and assign metadata to the resources. By the means of these functions they will contribute to the information enrichment process.
- *System administrators:* system administrators will be responsible for organizational and maintenance tasks in order to guarantee proper system functionality. This involves also user administration, system settings, problem solving, user support, etc.

Example

The first step in our use case is the creation of users from different groups (described previously) through our Web portal (see also screenshots of user

¹⁶<http://ontology.genasis.cz/www/>

creation in the Climate Twins use case). For demonstration purposes, we have created one user from each group, which is:

- *User1*: Role = “Scientific User”, domain ontology = “MU”, topic = “Epidemiology”.
- *User2*: Role = “Domain Expert”, domain ontology = “MU”, topic = “Czech Republic”.
- *User3*: System Administrator, domain ontology = “MU”, topic = “none”.

For user 1 and 2 the topics are used to retrieve data of interest. User 3 is presented with a number of administration portlets and is not considered any further in this use case, since she does not require any special views on time series data and does not participate in the semantic filtering process.

After creation of the users, our next step is to retrieve URIs of potentially relevant time series for each user. Listing 6.3 shows the SPARQL query which is executed for User1.

```

1 SELECT ?s
2 WHERE {
3   ?s <http://www.tatoo-fp7.eu/tatooweb/bridge\#Topic> ?topic .
4   FILTER(?topic = <http://ontology.genasis.cz\#epidemiology>)
5 }
```

Listing 6.3: SPARQL query for retrieval of resources for user 1 by using the selected topic.

This query retrieves URIs of resources (a part of them is shown in Listing 6.4) which can further be used to retrieve the time series stored in our time series data base.

```

1 http://www.svod.cz/report.php?lang=en&diag=C15
2 http://www.svod.cz/report.php?lang=en&diag=C60
3 http://www.svod.cz/report.php?lang=en&diag=C64
4 http://www.svod.cz/report.php?lang=en&diag=C25
5 http://www.svod.cz/report.php?lang=en&diag=C77
6 http://www.svod.cz/report.php?lang=en&diag=C40
7 http://www.svod.cz/report.php?lang=en&diag=C39
8 http://www.svod.cz/report.php?lang=en&diag=C04
9 http://www.svod.cz/report.php?lang=en&diag=C01
10 http://www.svod.cz/report.php?lang=en&diag=C53 ...
```

Listing 6.4: Part of the list of retrieved URIs for relevant time series.

The same can be done for User 2. The Time Series Processing component (as described in Chapter 3) has a data access component which interacts with

the time series database and uses the URIs as IDs to retrieve the data from the time series table by using simple SQL statements.

After the data is made available as a whole, a summary table of the retrieved time series is shown in the portal and the user can provide selections in order to filter relevant content of time series. An example of such a selection set is shown in the screenshot of Figure 6.38. The figure shows the time series filter selection window which is presented to a user for detailed selection of time series.

The screenshot provides the following information to the user (every part is separated by a line in the input window):

- *Name*: Shows the name of the currently selected time series.
- *Time Range*: Shows the range in which the time series provides values (depends also on the intervals of values).
- *Values*: Shows the number of values in the time series.
- *Select Time Range*: Enables the user to select a time range of interest. The date and time of the begin and end can be selected.
- *Select Property*: Enables the user to select the property of time series slots which is shown. In our case there are two properties which can be selected ('Incidence' and 'Mortality').
- *Select Minimum Rating*: Enables the user to select a minimum rating from other users of the same group for the time series parts. This means that only slots and slot ranges are retrieved which have at least the specified rating by users from the same group.
- *Show Annotations*: Provides a list of all available annotations of the time series provided from users of the same group. This is another option for the user to find out which parts are of particular relevance to her.

After receiving the user inputs from the time series filtering window, the framework transforms the selections into semantic time series processing expressions. The expressions for our example are shown in Listing 6.5.

```

1 Selection of the right time series range:
2 C77 < (tstart) .. (tend) > every 1 year
3
4     |
5     v
6
```


Time Series

Name:

Time trend of crude incidence and mortality

Time Range:

1977 - 2010

Values:

66

Select Time Range:

Start:

1/3/83

End:

11/19/02

Select Property:

Incidence

Select Minimum Rating:

★

★

★

★

★

[Show Annotations](#)

OK

Figure 6.38: Screenshot of filter settings provided by User 1.

```

7 Selection of the right property:
8 C77 < [n].incidence >
9
10      |
11      v
12
13 Selection of slot rankings (slots with lower rankings are set to
14 None):
15 C77 < [n] if [n].ranking > ranking otherwise None >
16
17      |
18      v
19
20 Retrieve all relevant annotations from the specified domain:
21 C77 < [n].getAnnotations(group:domain) >

```

Listing 6.5: Generated semantic time series processing expressions.

These expressions are used to query the time series repository (which remains in memory after collecting the time series from the database) and return resulting data to the user.

The final step is the visual presentation of the results to the user. Figure 6.39 shows how the results are presented.

The main part of the figure is the graph of the selected range of time series. This graph shows exactly the selected range in an appropriate interval and the selected properties. A property is represented as a line in the graph (and shown in the legend as well). The title of the graph is equal to the title of the time series. All available annotations about the selected range are shown in the right area of the window (user name and text of annotation). Additionally, at the bottom of the graph, there is an average rating representing the average user rating for this part of the time series, but also a date selection field to request further information about a certain time stamp of the time series data.

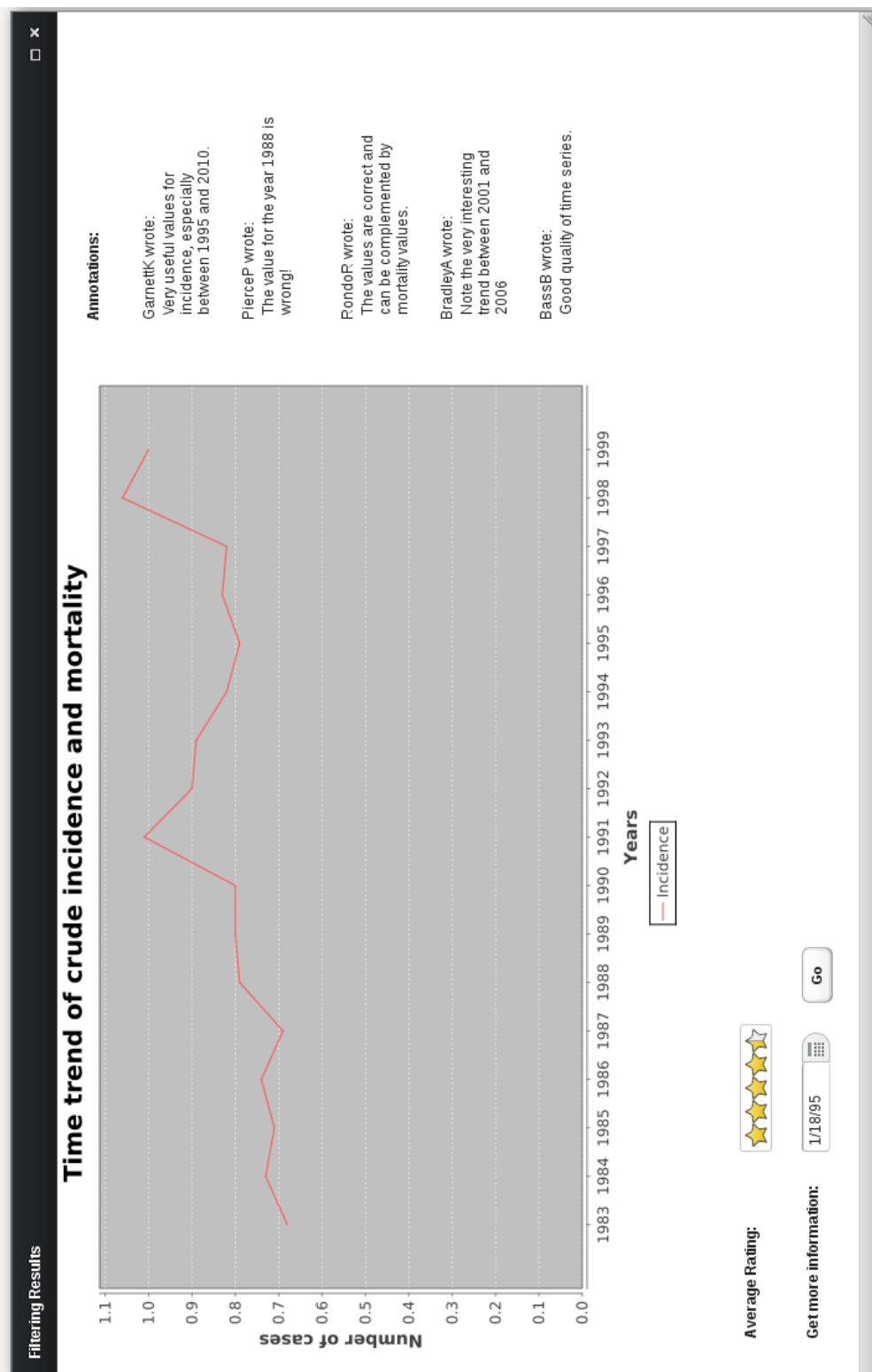


Figure 6.39: Screenshot of the filtering results as they are presented to the user.

Part IV

Conclusions

Chapter 7

Conclusions

In this thesis we present a multi-domain framework for community building based on data tagging. The framework is defined as a three-way solution which brings together the disciplines of time series processing, Semantic Web technologies, and community building.

Time series are broadly used by a lot of domain experts in different fields (e.g. financial experts interested in special parts of share value time series, environmental scientists interested in certain air quality measurements in time series, etc.). Our proposed framework solves the problems of (1) inflexible, tailor-made, and complex solutions which need to be implemented for every single domain in particular, (2) required effort by domain experts who need to manually select relevant content and are forced to spend a lot of time for analyzing tons of data from time series, and (3) lack of means to relate similar time series and users with common interests or need for shared (and collaboratively evaluated) relevance of data.

The following subsections describe the results of the thesis which contribute to solve the above-mentioned problems, future work in order to build a complete software solution, and new research topics which could be addressed using our proposed solutions.

7.1 Results and Contributions

The results of our work can be subdivided into the following three research fields:

- *Time Series Processing*: We provide a formal definition of a dedicated language for time series processing; the implementation of a runtime environment for the language including a lexer, parser, and interpreter

as well as modules for interoperability (Java via Jython and .NET via IronPython) and addition of customized functions.

- *Semantic Web*: For the Semantic Web research field, we offer a definition of Semantic Web extensions of the time series processing concepts. This includes a definition of semantic time series processing expressions as well as proposals for tackling the ontology mapping and reasoning problems.
- *Community Building*: Regarding community building we have presented our ideas for building groups of time series and users in order to satisfy the information need of domain experts and provide methods for sharing knowledge between users via time series data tagging.

For the *time series processing* field, we have designed a standardized language which is flexible enough to cope with many kinds of time series and take into account the diversity of clients for data stemming from time series. The language has been developed at the Austrian Institute of Technology and is already used in production systems. We used the concepts and provided a formal definition of the language together with the framework implementation to be used in different environments and for different users. We also assembled a complex set of possible time series processing expressions to show how the language can be used in different scenarios. Finally, we presented the results of our benchmark tests showing the performance of the language's interpreter and profiling, which identified bottlenecks and opportunities for performance improvements.

In the field of *Semantic Web technologies*, our aim was not to implement just another Semantic Web framework or to develop another ontology, but to introduce a new field of application for Semantic Web technologies. For the application of our semantic time series concepts, we propose a framework architecture with embedded time series processing. The architecture consists of a semantic repository which is responsible for the management of a domain ontology and new triples which are added later by users, other components, or the reasoner. Additionally, we provide results of benchmark tests for the semantic repository. Another part of the framework is the connector interface where we also offer a reference implementation in form of an RDFa parser. The RDFa parser collects triples which are defined in web sites by using RDFa tags in XHTML. Here, we also present our results in terms of harvesting web sites in order to show the performance of such a parser. The ontology mapping component is responsible for finding common items in domain ontologies, generation of a common time series processing vocabulary (bridge ontology), definition of mappings and relations between

ontologies, and generation of new relations for the resulting ontologies. This part is complemented by an analysis of techniques, methods, and tools for ontology mapping. This means that we analyzed the most common and popular ontology mapping concepts based on our previously defined criteria, and selected the most appropriate solution for our proposed framework architecture. After describing the selection process, we defined a bridge ontology for time series processing. The semantic processing component is then responsible for the management and execution of reasoning tasks. We decided not to implement an own reasoner, but to use an already existing one.

We identified 4 application areas as a contribution to the field of *community building*: networking and communication between users, resource (document) sharing and online collaboration, correlation of common topics between users and resources, and organization of common events and groups with common interests. The idea behind the concept is to develop domain ontologies for certain communities which are combined with time series and their data in order to generate groups where users and time series share common topics representing the interests of users from different communities. To demonstrate the workflow, we implemented a prototype portal which presents single use cases of the defined functionality. The portal implementation supports the following functionality: authentication, setup of a specific system environment for the user, identification of the user community by loading a domain ontology, group generation functionality, user and time series rating via an annotation window for user input, detailed view on time series as a line chart, reasoning to improve the process of group generation by inferring new RDF triples, and a SPARQL endpoint for querying the knowledge base. Reasoning is defined in the workflows of assignment of users to groups, suggestions of time series to users, assignment of time series to groups, and suggestions of related users to other users.

Having described our contributions to the different fields of research, in summary, our most significant contribution as a whole is the semantic time series processing framework architecture and prototype. The goal was to describe a framework which consists of modules for different functional blocks with the scope of bringing together time series processing and Semantic Web technologies in order to achieve a special way of community building. The leitmotif of this thesis is the description of the architecture of such a framework. Additionally, we described single components and functionalities through examples by using small prototype applications and demonstrations as well as sample code throughout the thesis.

In order to validate our results, we used real life ontologies which have been the result of the TaToo FP7 project. The validation scenarios “Climate Change Twin Regions - Discovery Platform” from the Austrian Institute of

Technology and “Anthropogenic Impact and Global Climate Change” from the Masaryk University in Brno provided domain ontologies for the validation. We defined two use cases, each for one of the validation scenarios, and used our concepts to prove the applicability of our approach.

The use case for the AIT validation scenario consisted of the definition of three different kinds of users (politician, scientist, and business manager) who all had different interests but the same domain ontology. In the second step, time series with defined topics from the climate area have been imported into the system. Then we have shown how the domain ontology and the time series ontologies were mapped using our bridge ontology. We also provided reasoning examples to describe how new RDF triples are inferred as a result of mapping the domain ontology and the bridge ontology. Finally, we presented the generation of groups with time series and users sharing similar topics and interests.

In the MU validation scenario use case, we created different (scenario specific) users who all use the MU domain ontology but are interested in different parts of time series data. We retrieved time series data by using the publicly available RDF catalogue from the Masaryk University by using SPARQL queries to access the right time series and the domain ontology to generate the right view to the data for every user. Finally, we offer selection and filtering of the data sets which depend on the interests of a user. This is done by using generated Semantic Time Series Processing expressions which are based on the user settings in the portal. We used them to address the semantic time series processor which, in return, provides us with filtered data from the knowledge base.

7.2 Future Work and Research

The research done in this thesis proposes solutions to the defined problems, but also leaves room for improvements of the implementation (since the implemented software is a prototype which can be seen as a proof-of-concept implementation) and opens the field for new, interesting research questions.

The most important improvements of the implementation, which would also be needed in order to make the framework ready for use in industry, are:

- *Integration of building blocks into one single semantic time series framework:* The single components which have been implemented during this thesis prove the applicability of our ideas as a functional software. However, this is not a complete framework since the implementation is still prototypical and should only be used for demonstration purposes. There is still the need for transformation of the building blocks into

components with well-defined interfaces and integration of the individual functional blocks into one single framework which can be used directly by an application. This is a pure engineering task and therefore not part of this thesis.

- *Development of a complete portal (Web portal with user portlets and stand-alone application):* Based on the framework, the next engineering step would be to build a complete Web portal for users. This portal should consist of different portlets customized for each possible type of user. The portal could be implemented in Vaadin¹ as the demos in this thesis, but also (preferably) in a more dynamic environment, e.g. Ruby on Rails², Grails³, or Django⁴. Anyway, it should cover the whole use case for domain experts from the beginning to the end. This would be a complete social networking platform for domain experts with the possibility to load ontologies, directly crawl time series databases, feature SPARQL endpoints, implement a large-scale knowledge base and databases for caching of time series data, take care of usability, performance, etc.

New challenging research questions are mainly seen in the fields of ontology integration, advanced reasoning, and usability studies for time series processing:

- *Integration of commonly used ontologies from other fields:* A major stepping stone for further research is the usage and integration of popular and broadly used general-purpose ontologies, e.g. Friend-of-a-Friend, Dublin Core, etc. in the bridge ontology. This would improve the expressiveness of the bridge ontology and the interoperability with domain ontologies which use concepts from these general-purpose ontologies as well. An easy approach would be to start by using Friend-of-a-Friend to improve the definition of users and Dublin Core for resources (starting with time series, but also documents and other resources). There is also a general definition of a Semantic Bridge Ontology (SBO), which has been developed in the Mafra⁵ project, that could be combined with our bridge ontology in order to improve interoperability. Anyway, there is sufficient room for further research in the field of interoperability of time series ontologies.

¹<https://vaadin.com/home>

²<http://rubyonrails.org/>

³<http://grails.org/>

⁴<https://www.djangoproject.com/>

⁵<http://mafra-toolkit.sourceforge.net/>

- *Advanced reasoning algorithms for time series processing:* There is a lot of research dedicated to reasoning in the area of Semantic Web. Recent research results presented new methods of reasoning and new approaches of dealing with the structure of OWL ontologies (e.g. justification, as described in Bail [2013]). Therefore, an interesting starting point for further research would be advanced reasoning techniques for semantic time series processing and handling complex time series processing ontologies.
- *Research about usability of portals for domain experts in time series processing:* Besides improved integration of time series ontologies and advanced methods for reasoning in semantic time series processing, the usability for domain experts in general is a topic in need of further research. For that purpose the organization of workshops with domain expert users from different sectors with different domain ontologies is required. Also, the definition and application of experiments based on different user groups would improve usability. Once the framework and platform are complete, experiments with domain experts from different fields as real users can be performed in order to determine usability and usefulness. This work would be an interesting field study for a social sciences research project.

Part V

Bibliography

Bibliography

- S. Agarwal, S. Handschuh, and S. Staab. Annotation, composition and invocation of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2:31–48, 2004.
- N. K. Ahmed, A. F. Atiya, N. El Gayar, and H. El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29:594–621, 2010.
- D. Allemang and J. Hendler. *Semantic Web for the Working Ontologist*. Morgan Kaufmann, Oxford, UK, 1st edition, 2008. ISBN 978-0-123-73556-0.
- J. M. Almendros-Jiménez. An RDF query language based on logic programming. *Electronic Notes in Theoretical Computer Science*, 200:67–85, 2008.
- I. Alon, M. Qi, and R. J. Sadowski. Forecasting aggregate retail sales: A comparison of artificial neural networks and traditional methods. *Journal of Retailing and Consumer Services*, 8(3):147–156, 2001.
- E. Alpaydin. *Introduction to Machine Learning*. MIT Press, 2nd edition, 2010. ISBN 978-0-262-01243-0.
- F. Amato, V. Casola, A. Gaglione, and A. Mazzeo. A semantic enriched data model for sensor network operability. *Simulation Modelling Practice and Theory*, (19):1745–1757, October 2010.
- A. Ankolekar, M. Krötzsch, T. Tran, and D. Vrandečić. The two cultures: Mashing up web 2.0 and the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, (6):70–75, November 2007.
- T. Apache Software Foundation. The jena ontology api. Online (<http://jena.sourceforge.net/ontology/>), July 2010.
- F. Baader. *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003.
- S. Bail. *The Justificatory Structure of OWL Ontologies*. PhD thesis, The University of Manchester, 2013.
- S. Battle, A. Bernstein, H. Boley, B. Grosz, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, D. McGuinness, et al. Semantic web services language (swsl). *W3C Member submission*, 9, 2005.

- R. Benjamins, J. Contreras, A. G. Pérez, H. Uszkoreit, T. Declerck, D. Fensel, Y. Ding, M. Wooldridge, and V. Tamma. Esperonto application: Service provision of semantic annotation, aggregation, indexing, and routing of textual, multimedia and multilingual web content. *Proc. of WIAMSI03*, 2003.
- T. Berners-Lee, J. Hendler, O. Lassila, et al. The semantic web. *Scientific American*, 284(5):28–37, 2001.
- B. Božić. Simulation and modeling of semantically enriched time series. In *Sustaining our Future: Understanding and Living with uncertainty*, MOD-SIM '11. Modelling and Simulation Society of Australia and New Zealand, 2011. ISBN 978-0-9872143-1-7.
- B. Božić and W. Winiwarter. Community building based on semantic time series. In *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services, iiWAS '12*, pages 213–222, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1306-3. doi: 10.1145/2428736.2428770. URL <http://doi.acm.org/10.1145/2428736.2428770>.
- B. Božić. A multi-domain framework for community building based on data tagging. In *International Semantic Web Conference (2)*, pages 441–444, 2012.
- B. Božić and W. Winiwarter. Ontology mapping and reasoning in semantic time series processing. In *iiWAS*, page 443, 2013a.
- B. Božić and W. Winiwarter. A showcase of semantic time series processing. *IJWIS*, 9(2):117–141, 2013b.
- B. Božić, J. Peters-Anders, and G. Schimak. Visualization and filtering of semantically enriched environmental time series. In *Proceedings of International Environmental Modelling and Software Society (iEMSs) 2012 International Congress on Environmental Modelling and Software Managing Resources of a Limited Planet, Sixth Biennial Meeting, Leipzig, Germany*, 2012.
- J. G. Breslin, S. Decker, A. Harth, and U. Bojars. Sioc: an approach to connect web-based communities. *International Journal of Web Based Communities*, 2(2):133–142, 2006.
- D. Brickley and R. V. Guha. Resource description framework (RDF) schema specification 1.0: W3C Candidate Recommendation 27 march 2000. 2000.

- D. Brickley and L. Miller. Foaf vocabulary specification 0.98. *Namespace Document*, 9, 2010.
- J. Broekstra and A. Kampman. SeRQL: A second generation RDF query language. In *Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, pages 13–14, 2003.
- J. L. Callen, C. C. Y. Kwan, P. C. Y. Yip, and Y. Yuan. Neural network forecasting of quarterly accounting earnings. *International Journal of Forecasting*, 12(4):475–482, 1996.
- R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168, Pittsburgh, 2006.
- V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice. Open knowledge base connectivity 2.0. *Artificial Intelligence Center of SRI International and Knowledge Systems Laboratory of Stanford University*, 1998.
- T.-T. Chuang and S. B. Yadav. The development of an adaptive decision support system. *Decision Support Systems*, 24:73–87, 1998.
- P. Cimiano and J. Völker. Text2onto. In *Natural Language Processing and Information Systems*, pages 227–238. Springer, 2005.
- D. Clinton et al. Opensearch 1.1 specification (draft 4). *Opensearch.org*.
- D. Connolly et al. Gleaning resource descriptions from dialects of languages (grddl). *W3C, W3C Recommendation*, 11, 2007.
- H. Cunningham, Y. Wilks, and R. J. Gaizauskas. GATE: A general architecture for text engineering. In *Proceedings of the 16th Conference on Computational Linguistics*, Copenhagen, Denmark, 1996.
- M. C. Daconta, L. J. Obrst, and K. T. Smith. *The Semantic Web*. Wiley, Indianapolis, US, 1st edition, 2003. ISBN 978-0-471-43257-9.
- J. Davies and R. Weeks. QuizRDF: Search technology for the semantic web. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*. IEEE, 2004.
- J. Davies, R. Studer, and P. Warren. *Semantic Web Technologies*. Wiley, Chichester, UK, 1st edition, 2000. ISBN 978-0-470-02596-3.

- J. Davies, M. Grobelnik, and D. Mladenić. *Semantic Knowledge Management*. Springer, Berlin, Germany, edition=1st, isbn=978-3-540-88844-4, 2009.
- J. De Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, E. Oren, et al. Web service modeling ontology (WSMO). *Interface*, 5:1, 2006a.
- J. De Bruijn, H. Lausen, A. Polleres, and D. Fensel. The web service modeling language WSMML: An overview. In *The Semantic Web: Research and Applications*, pages 590–604. Springer, 2006b.
- S. Decker, M. Erdmann, D. Fensel, and R. Studer. *Ontobroker: Ontology based access to distributed and semi-structured information*. Springer, 1999.
- S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, et al. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the 12th International Conference on World Wide Web*, pages 178–186. ACM, 2003.
- M. Dimitrov, A. Simov, V. Momtchev, and M. Konstantinov. WSMO studio – a semantic web services modelling environment for WSMO. In *The Semantic Web: Research and Applications*, pages 749–758. Springer, 2007.
- J. Domingue, D. Fensel, and R. González-Cabero. SOA4All, enabling the SOA revolution on a world wide scale. In *Semantic Computing, 2008 IEEE International Conference on*, pages 530–537. IEEE, 2008.
- M. Dzbor, J. Domingue, and E. Motta. Magpie – towards a semantic web browser. In *Proceedings of the 2nd International Semantic Web Conference*, Milton Keynes, UK, 2003.
- M. Ehrig and S. Staab. QOM – quick ontology mapping. In *The Semantic Web-ISWC 2004*, pages 683–697. Springer, 2004.
- N. B. Ellison, C. Steinfield, and C. Lampe. The benefits of facebook friends: Social capital and college students’ use of online social network sites. *Journal of Computer-Mediated Communication*, 12(4):1143–1168, 2007.
- J. Farrell and H. Lausen. Semantic annotations for WSDL and XML Schema. *W3c recommendation*, 28, 2007.
- B. Fazlollahi, M. A. Parikh, and S. Verma. Adaptive decision support systems. *Decision Support Systems*, 20:297–315, 1997.

- D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster. *Spinning the Semantic Web*. Massachusetts Institute of Technology, Cambridge, US, 1st edition, 2005. ISBN 978-0-262-06232-9.
- M. Fernández-López, A. Gómez-Pérez, and N. Juristo. Methontology: From ontological art towards ontological engineering. 1997.
- J. Figwer. A new method of random time-series simulation. *Simulation Practice and Theory*, 5:217–234, 1997.
- B. Fortuna, M. Grobelnik, and D. Mladenic. Ontogen: Semi-automatic ontology editor. In M. Smith and G. Salvendy, editors, *Human Interface and the Management of Information. Interacting in Information Environments*, volume 4558 of *Lecture Notes in Computer Science*, pages 309–318. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-73353-9. doi: 10.1007/978-3-540-73354-6_34. URL http://dx.doi.org/10.1007/978-3-540-73354-6_34.
- A. Funk, V. Tablan, K. Bontcheva, H. Cunningham, B. Davis, and S. Handschuh. Clone: Controlled language for ontology editing. In *The Semantic Web*, pages 142–155. Springer, 2007.
- A. Garcia-Castro, A. Labarga, L. Garcia, O. Giraldo, C. Montaña, and J. A. Bateman. Semantic web and social web heading towards living documents in the life sciences. *Web Semantics: Science, Services and Agents on the World Wide Web*, (8):155–162, April 2010.
- A. Geist, C. R. Brunschwig, F. Lachmayer, and G. Scheffbeck. *Structuring Legal Semantics*. Weblaw, Zurich, Switzerland, 1st edition, 2011. ISBN 978-3-905-74280-0.
- M. R. Genesereth, R. E. Fikes, et al. Knowledge interchange format-version 3.0: reference manual. 1992.
- N. Gibbins, S. Harris, and N. Shadbolt. Agent-based semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1: 141–154, 2004.
- C. Golbreich, S. Zhang, and O. Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4:181–195, 2006.
- W. W. Group et al. RDFa 1.1 primer. W3C working group note, 2012.

- T. Gruber. Collective knowledge systems: Where the social web meets the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, (6):4–13, December 2007.
- T. R. Gruber. *Ontolingua: A mechanism to support portable ontologies*. Stanford University, Knowledge Systems Laboratory, 1992.
- P. Haase, H. Lewen, R. Studer, D. T. Tran, M. Erdmann, M. d’Aquin, and E. Motta. The NeOn ontology engineering toolkit. In *WWW 2008 Developers Track*, April 2008.
- S. Handschuh, S. Staab, and F. Ciravegna. S-CREAM – semi-automatic creation of metadata. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 358–372. Springer, 2002.
- T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, New York, 2nd edition, 2009. ISBN 978-0-387-84857-0.
- T. Hill, M. O’Connor, and W. Remus. Neural network models for time series forecasts. *Management Science*, 42(7):1082–1092, 1996.
- S. Hinduja and J. W. Patchin. Personal information of adolescents on the internet: A quantitative content analysis of myspace. *Journal of Adolescence*, 31(1):125–146, 2008.
- P. Hitzler, M. Krötzsch, S. Rudolph, and Y. Sure. *Semantic Web*. Springer, Berlin, DE, 1st edition, 2008. ISBN 978-3-540-33993-9.
- I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosz, M. Dean, et al. Swrl: A semantic web rule language combining owl and ruleml. *W3C Member submission*, 21:79, 2004.
- B. Huberman, D. Romero, and F. Wu. Social networks that matter: Twitter under the microscope. *Available at SSRN 1313405*, 2008.
- C. Jenkins, M. Jackson, P. Burden, and J. Wallis. Automatic RDF metadata generation for resource discovery. *Computer Networks*, 31:1305–1320, 1999.
- S. Kairam, M. Brzozowski, D. Huffaker, and E. Chi. Talking in circles: Selective sharing in Google+. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1065–1074. ACM, 2012.

- A. Kalyanpur, J. Hendler, B. Parsia, and J. Golbeck. SMORE – semantic markup, ontology, and RDF editor. Technical report, DTIC Document, 2006a.
- A. Kalyanpur, B. Parsia, E. Sirin, B. C. Grau, and J. Hendler. Swoop: A web ontology editing browser. *Web Semantics: Science, Services and Agents on the World Wide Web*, 4(2):144–153, 2006b.
- M. Kifer. Rule interchange format: The framework. In *Web reasoning and rule systems*, pages 1–11. Springer, 2008.
- A. J. Kim. *Community Building on the Web*. Peachpit Press, Berkeley, US, 1st edition, 2000. ISBN 978-0-201-87484-6.
- A. Kiryakov, K. I. Simov, and M. Dimitrov. OntoMap: Portal for upper-level ontologies. In *Proceedings of the International Conference on Formal Ontology in Information Systems-Volume 2001*, pages 47–58. ACM, 2001.
- A. Kiryakov, D. Ognyanov, and D. Manov. OWLIM – a pragmatic semantic repository for OWL. In *Web Information Systems Engineering-WISE 2005 Workshops*, pages 182–192. Springer, 2005.
- P. Kogut and W. Holmes. AeroDAML: Applying information extraction to generate DAML annotations from web pages. In *First International Conference on Knowledge Capture*, 2001.
- M. Krötzsch, D. Vrandečić, and M. Völkel. Semantic mediawiki. In *The Semantic Web-ISWC 2006*, pages 935–942. Springer, 2006.
- J. Kunegis, A. Lommatzsch, and C. Bauckhage. The slashdot zoo: Mining a social network with negative edges. In *Proceedings of the 18th international conference on World wide web*, pages 741–750. ACM, 2009.
- M. Lehtokangas, J. Saarinen, K. Kaski, and P. Huuhtanen. A network of autoregressive processing units for time series modeling. *Applied Mathematics and Computation*, 75:151–165, 1996.
- T. Lukasiewicz and U. Straccia. Managing uncertainty and vagueness in description logics for the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6:291–308, 2008.
- D. L. McGuinness and F. van Harmelen. OWL web ontology language - overview. Online(<http://www.w3.org/TR/owl-features/>), February 2004.

- P. Mika. Flink: Semantic web technology for the extraction and analysis of social networks. *Web Semantics: Science, Services and Agents on the World Wide Web*, (3):211–223, May 2005a.
- P. Mika. Social networks and the semantic web: the next challenge. *IEEE Intell. Syst.*, 1(20):82–85, February 2005b.
- P. Mika and A. Gangemi. Descriptions of social relations. In *Proceedings of the First Workshop on Friend of a Friend*, 2004.
- A. Miles, B. Matthews, D. Beckett, D. Brickley, M. Wilson, and N. Rogers. Skos: A language to describe simple knowledge structures for the web. In *Proceedings of the XTech Conference*, Amsterdam, 2005.
- N. F. Noy and M. A. Musen. The prompt suite: interactive tools for ontology merging and mapping. *International Journal of Human-Computer Studies*, 59(6):983–1024, 2003.
- N. F. Noy and D. L. Rubin. Translating the foundational model of anatomy into OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6:133–136, 2007.
- P. F. Patel-Schneider and I. Horrocks. A comparison of two modelling paradigms in the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, (5):240–250, September 2007.
- E. Prud’hommeaux and A. Seaborne. SPARQL query language for RDF. <http://www.w3.org/TR/rdf-sparql-query/>, January 2008.
- D. Quan, D. Huynh, and D. R. Karger. Haystack: A platform for authoring end user semantic web applications. In *The Semantic Web – ISWC 2003*, pages 738–753. Springer, 2003.
- P. L. N. Raju. Spatial data analysis. *Satellite Remote Sensing and GIS Applications in Agricultural Meteorology*, pages 151–174, 2005.
- R. Rico-Martinez, J. S. Anderson, and I. G. Kevrekidis. Self-consistency in neural network-based NLPC analysis with applications to time-series processing. *Computers and Chemical Engineering*, 20:1089–1094, 1996.
- A. Rizzoli, G. Schimak, M. Donatelli, J. Hřebíček, G. Avellino, J. Mon, et al. Tatoo: tagging environmental resources on the web by semantic annotations. In *Proceedings of International Environmental Modelling and Software Society (iEMSs) 2010 International Congress on Environmental*

Modelling and Software Modelling for Environments Sake, Fifth Biennial Meeting, Ottawa, Canada, 2010.

- M. Sabou, C. Wroe, C. Goble, and H. Stuckenschmidt. Learning domain ontologies for semantic web service descriptions. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3:340–365, 2005.
- M. Schied, A. Kstlbacher, and C. Wolff. Connecting semantic mediawiki to different triple stores using rdf2go. In *5th Workshop on Semantic Wikis Linking Data and People (SemWiki2010)*, May 2010. URL <http://data.semanticweb.org/workshop/semwiki/2010/paper/main/18>.
- G. Schimak, B. Božić, A. Kaufmann, J. Peters-Anders, P. Dihé, and T. Pariente Lobo. The tatoo semantic case-requirements, impacts and applications. *Proceedings of the 25th EnviroInfo*, pages 832–847, 2011.
- P. Schmitz. Inducing ontology from flickr tags. In *Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland*, volume 50, 2006.
- T. Segaran, C. Evans, and J. Tylor. *Programming the Semantic Web*. O'Reilly, Sebastopol, CA, 1st edition, 2009. ISBN 978-0-596-15381-6.
- N. Shadbolt, N. Gibbins, H. Glaser, S. Harris, and M. Schraefel. CS AKTive Space, or how we learned to stop worrying and love the semantic web. *Intelligent Systems, IEEE*, 19(3):41–47, 2004.
- R. Sharda and R. B. Patil. Connectionist approach to time series prediction: An empirical test. *Journal of Intelligent Manufacturing*, 3(5):317–323, 1992.
- S. Shekarpour and S. D. Katebi. Modeling and evaluation of trust with an extension in semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8:26–36, 2010.
- R. H. Shumway and D. S. Stoffer. *Time Series Analysis and its Applications*. Springer, New York, US, 3rd edition, 2010. ISBN 978-1-441-97865-3.
- Stanford Center for Biomedical Informatics Research. The Protégé ontology editor and knowledge acquisition system. Online (<http://protege.stanford.edu/>), July 2010.
- G. Stumme, A. Hotho, and B. Berendt. Semantic web mining – state of the art and future directions. *Web Semantics: Science, Services and Agents on the World Wide Web*, (4):124–143, February 2006.

- Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, and D. Wenke. Ontoedit: Collaborative ontology development for the semantic web. In I. Horrocks and J. Hendler, editors, *The Semantic Web ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pages 221–235. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-43760-4. doi: 10.1007/3-540-48005-6_18. URL http://dx.doi.org/10.1007/3-540-48005-6_18.
- N. R. Swanson and H. White. A model-selection approach to assessing the information in the term structure using linear models and artificial neural networks. *Journal of Business and Economic Statistics*, 13(3):265–275, 1995.
- A. Telea, F. Frasincar, and G.-J. Houben. Visualisation of RDF(S) – based information. In *Information Visualization, 2003. IV 2003. Proceedings. Seventh International Conference on*, pages 294–299. IEEE, 2003.
- T. Tersvirta, D. van Dijk, and M. Medeiros. Linear models, smooth transition autoregressions, and neural networks for forecasting macroeconomic time series: A re-examination. Working Paper Series in Economics and Finance 561, Stockholm School of Economics, July 2004. URL <http://ideas.repec.org/p/hhs/hastef/0561.html>.
- C. Torrence and G. P. Compo. A practical guide to wavelet analysis. *Bulletin of the American Meteorological Society*, 79(1):61–78, 1998.
- J. Ungar, J. Peters-Anders, and W. Loibl. Climate twins - an attempt to quantify climatological similarities. In *ISESS 2011*, Brno (CZ), 06/2011 2011. TaToo Project, TaToo Project. URL <http://www.springerlink.com/content/14722218777588pj/>.
- M. Van Camp and P. Vauterin. Tsoft: graphical and interactive software for the analysis of time series and earth tides. *Computers and Geosciences*, (31):631–640, November 2004.
- F. van Harmelen, P. F. Patel-Schneider, and I. Horrocks. Reference description of the daml+ oil (march 2001) ontology markup language. *DAML+ OIL Document*, URL <http://www.daml.org/2000/12/reference.html>, 2001.
- J. Viinikka, H. Debar, L. Mé, A. Lehtikainen, and M. Tarvainen. Processing intrusion detection alert aggregates with time series modeling. *Information Fusion*, 10:312–324, 2009.

- Z. Wang, K. Wang, Y. Jin, and G. Qi. Ontomerge: A system for merging DL-Lite ontologies.
- S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin core metadata for resource discovery. *Internet Engineering Task Force RFC*, 2413:222, 1998.
- M. Weiten. OntoSTUDIO® as an ontology engineering environment. In *Semantic Knowledge Management*, pages 51–60. Springer, 2009.
- R. Wetzker, C. Zimmermann, and C. Bauckhage. Analyzing social bookmarking systems: A del.icio.us cookbook. In *Proceedings of the ECAI 2008 Mining Social Data Workshop*, pages 26–30, 2008.
- S.-T. Yuan and M.-Z. Huang. A study on time series pattern extraction and processing for competitive intelligence support. *Expert Systems with Applications*, (21):37–51, February 2001.
- G. P. Zhang, B. E. Patuwo, and M. Y. Hu. A simulation study of artificial neural networks for nonlinear time-series forecasting. *Computers and Operations Research*, 28:381–396, 2001.
- W. Zhang, Q. Cao, and M. J. Schniederjans. Neural network earnings per share forecasting models: A comparative analysis of alternative models. *Decision Sciences*, 35(2):205–237, 2004.
- L. Zhou, L. Ding, and T. Finin. How is the semantic web evolving? A dynamic social network perspective. *Computers in Human Behavior*, (27):1294–1302, August 2010.

Part VI

Appendix

Appendix A

Time Series Processing Language Specification

The following listing shows the specification of our time series processing language in Extended Backus-Naur Form (EBNF).

```
1 letter    ::= [a-zA-Z]
2 digit     ::= [0-9]
3
4 integer   ::= "-"? digit+
5 float     ::= digit+ "." digit+
6 number    ::= integer | float
7 string    ::= (letter | digit | "-")+
8
9 stmt      ::= ts_param_id_list pipe
10
11 pipe      ::= pipe "|" generator
12 pipe      ::= generator
13
14 generator ::= generator every_phrase
15 generator ::= "<<" expression_list ">>"
16
17 ts_param_id_list ::= formal_parameter
18 ts_param_id_list ::= ts_param_id_list , formal_parameter
19
20 formal_parameter ::= "@" letter (letter | digit | "-")*
21
22 expression_list ::= expression_list ";" assign_expression
23 expression_list ::= assign_expression
24
25 assign_expression ::= if_expression "=>" letter (letter | digit)
26                    *
27 assign_expression ::= letter (letter | digit)* "//.*//" interval
28 assign_expression ::= if_expression
```

```

28
29 if_expression ::= logic_expression "IF" if_expression "
    OTHERWISE" logic_expression
30 if_expression ::= logic_expression
31
32 logic_expression ::= logic_expression "AND" add_expression
33 logic_expression ::= logic_expression "OR" add_expression
34 logic_expression ::= logic_expression "XOR" add_expression
35 logic_expression ::= add_expression ">" add_expression
36 logic_expression ::= add_expression "<" add_expression
37 logic_expression ::= add_expression ">=" add_expression
38 logic_expression ::= add_expression "<=" add_expression
39 logic_expression ::= add_expression "==" add_expression
40 logic_expression ::= add_expression "!=" add_expression
41 logic_expression ::= add_expression
42
43 add_expression ::= add_expression "+" mult_expression
44 add_expression ::= add_expression "-" mult_expression
45 add_expression ::= mult_expression
46
47 mult_expression ::= mult_expression "/" expression
48 mult_expression ::= mult_expression "*" expression
49 mult_expression ::= expression
50
51 expression ::= expression "**" expression
52 expression ::= "-"? expression
53 expression ::= number
54 expression ::= string
55 expression ::= ts_slice
56 expression ::= "(" if_expression ")"
57 expression ::= formal_parameter "(" expression ")"
58 expression ::= "None"
59
60 ts_slice ::= property_access interval
61
62 property_access ::= letter (letter | digit | "_")*
63                  | letter (letter | digit | "_")* "." letter (
        letter | digit | "_")*
64
65 interval ::= numeric_interval
66            | logical_interval
67            | time_interval
68
69 numeric_interval ::= "[" integer "]"
70
71 logical_interval ::= "[" log_index_expression "]"
72 logical_interval ::= "[" log_index_expression ".."
        log_index_expression "]"
73

```

```

74 time_interval ::= bra time_index_expression ket
75 time_interval ::= bra time_index_expression "[" .. "]"
    time_index_expression ket
76
77 bra ::= "[" | "]"
78 ket ::= "]" | "["
79
80 log_index_expression ::= "i"
81     | "i" "+" integer
82     | "i" "-" integer
83
84 int_with_time ::= integer (ms | secs | sec | mins | min | hours
    | hour | days | day | weeks | week)
85
86 time_index_expression ::= "t"
87     | "t" "+" int_with_time
88     | "t" "-" int_with_time
89 every_phrase ::= "every" int_with_time
90 every_phrase ::= "every" int_with_time "@" int_with_time

```


Appendix B

RDFa Parsing Stylesheet

The following listing shows the XSL definition stylesheet for parsing RDFa web documents.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <stylesheet
4   xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="
5   1.0"
6   xmlns:h="http://www.w3.org/1999/xhtml"
7   xmlns="http://www.w3.org/1999/XSL/Transform"
8   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
9   xmlns:xalan="http://xml.apache.org/xalan"
10  xmlns:java="http://xml.apache.org/xalan/java"
11  exclude-result-prefixes="java" >
12
13 <output indent="yes" method="html" media-type="application/rdf+
14   xml" encoding="UTF-8" omit-xml-declaration="yes" />
15
16 <!-- base of the current HTML doc -->
17 <variable name='html_base' select="//*/h:head/h:base[position()
18   =1]/@href"/>
19
20 <!-- default HTML vocabulary namespace -->
21 <variable name='default_voc' select="'http://www.w3.org/1999/
22   xhtml/vocab#'" />
23
24 <!-- parser instance -->
25 <param name='parser' select="''"/>
26
27 <!-- url of the current XHTML page if provided by the XSLT
28   engine -->
29 <param name='url' select="''"/>
```

```

26 <!-- this contains the URL of the source document whether it was
    provided by the base or as a parameter e.g. http://example.
    org/bla/file.html-->
27 <variable name='this' >
28   <choose>
29     <when test="string-length($html_base)>0"><value-of select="
        $html_base"/></when>
30     <otherwise><value-of select="$url"/></otherwise>
31   </choose>
32 </variable>
33
34 <!-- this_location contains the location the source document e.g
    . http://example.org/bla/ -->
35 <variable name='this_location' >
36   <call-template name="get-location"><with-param name="url"
        select="$this"/></call-template>
37 </variable>
38
39 <!-- this_root contains the root location of the source document
    e.g. http://example.org/ -->
40 <variable name='this_root' >
41   <call-template name="get-root"><with-param name="url" select="
        $this"/></call-template>
42 </variable>
43
44
45 <!-- templates for parsing -----
    ----->
46
47 <!--Start the RDF generation-->
48 <template match="/">
49 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    >
50   <apply-templates mode="rdf2rdfxml" /> <!-- the mode is used
        to ease integration with other XSLT templates -->
51 </rdf:RDF>
52 </template>
53
54
55 <!-- match RDFa element -->
56 <template match="*[attribute::property or attribute::rel or
        attribute::rev or attribute::typeof]" mode="rdf2rdfxml" >
57
58   <!-- identify subject -->
59   <variable name="subject"><call-template name="subject"/></
        variable>
60
61
62   <!-- do we have object properties? -->

```

```

63 <if test="string-length(@rel)>0 or string-length(@rev)>0">
64   <variable name="object"> <!-- identify the object(s) -->
65     <choose>
66       <when test="@resource">
67         <call-template name="expand-curie-or-uri"><with-param
name="curie_or_uri" select="@resource"/></call-template>
68       </when>
69       <when test="@href">
70         <call-template name="expand-curie-or-uri"><with-param
name="curie_or_uri" select="@href"/></call-template>
71       </when>
72       <when test="descendant::*[attribute::about or attribute::
src or attribute::typeof or
73         attribute::href or attribute::resource or
74         attribute::rel or attribute::rev or attribute::property
]>">
75         <call-template name="recurse-objects"/>
76       </when>
77       <otherwise>
78         <call-template name="self-curie-or-uri"><with-param name
="node" select="."/></call-template>
79       </otherwise>
80     </choose>
81   </variable>
82
83 <call-template name="relrev">
84   <with-param name="subject" select="$subject"/>
85   <with-param name="object" select="$object"/>
86 </call-template>
87
88 </if>
89
90
91 <!-- do we have data properties ? -->
92 <if test="string-length(@property)>0">
93
94   <!-- identify language -->
95   <variable name="language" select="string(ancestor-or-self
::*/attribute::xml:lang[position()=1])"/>
96
97   <variable name="expended-pro"><call-template name="expand-
ns"><with-param name="qname" select="@property"/></call-
template></variable>
98
99   <choose>
100     <when test="@content"> <!-- there is a specific content
-->
101     <call-template name="property">
102       <with-param name="subject" select="$subject"/>

```

```

103         <with-param name="object" select="@content" />
104         <with-param name="datatype" >
105             <choose>
106                 <when test="@datatype='' or not(@datatype)"></when>
107 > <!-- enforcing plain literal -->
108                 <otherwise><call-template name="expand-ns"><with-
109 param name="qname" select="@datatype"/></call-template></
110 otherwise>
111             </choose>
112         </with-param>
113         <with-param name="predicate" select="@property"/>
114         <with-param name="attrib" select="'true'"/>
115         <with-param name="language" select="$language"/>
116         </call-template>
117     </when>
118     <when test="not(*)"> <!-- there no specific content but
119 there are no children elements in the content -->
120         <call-template name="property">
121             <with-param name="subject" select="$subject" />
122             <with-param name="object" select="." />
123             <with-param name="datatype">
124                 <choose>
125                     <when test="@datatype='' or not(@datatype)"></when>
126 > <!-- enforcing plain literal -->
127                     <otherwise><call-template name="expand-ns"><with-
128 param name="qname" select="@datatype"/></call-template></
129 otherwise>
130                 </choose>
131             </with-param>
132             <with-param name="predicate" select="@property"/>
133             <with-param name="attrib" select="'true'"/>
134             <with-param name="language" select="$language"/>
135             </call-template>
136         </when>
137         <otherwise> <!-- there is no specific content; we use the
138 value of element -->
139         <call-template name="property">
140             <with-param name="subject" select="$subject" />
141             <with-param name="object" select="." />
142             <with-param name="datatype">
143                 <choose>
144                     <when test="@datatype='' or not(@datatype)">http
145 ://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral</when> <!--
146 enforcing XML literal -->
147                     <otherwise><call-template name="expand-ns"><with-
148 param name="qname" select="@datatype"/></call-template></
149 otherwise>
150                 </choose>
151             </with-param>

```

```

140         <with-param name="predicate" select="@property"/>
141         <with-param name="attrib" select="'false'"/>
142         <with-param name="language" select="$language"/>
143     </call-template>
144 </otherwise>
145 </choose>
146 </if>
147
148 <!-- do we have classes ? -->
149 <if test="@typeof">
150     <call-template name="class">
151         <with-param name="resource"><call-template name="self-
152             curie-or-uri"><with-param name="node" select="."/></call-
153             template></with-param>
154         <with-param name="class" select="@typeof"/>
155     </call-template>
156 </if>
157
158 <apply-templates mode="rdf2rdfxml" />
159
160 </template>
161
162 <!-- named templates to process URIs and token lists - - - - -
163     - - - - -
164     -->
165
166 <!-- tokenize a string using space as a delimiter -->
167 <template name="tokenize">
168     <param name="string" />
169     <if test="string-length($string)>0">
170         <choose>
171             <when test="contains($string, ' ')">
172                 <value-of select="normalize-space(substring-before(
173                     $string, ' '))"/>
174                 <call-template name="tokenize"><with-param name="string"
175                     select="normalize-space(substring-after($string, ' '))"/></
176                     call-template>
177             </when>
178             <otherwise><value-of select="$string"/></otherwise>
179         </choose>
180     </if>
181 </template>
182
183 <!-- get file location from URL -->
184 <template name="get-location">
185     <param name="url" />
186     <if test="string-length($url)>0 and contains($url, '/')">

```

```

181     <value-of select="concat(substring-before($url,'/'),'/' )"/>
182     <call-template name="get-location"><with-param name="url"
183       select="substring-after($url,'/')" /></call-template>
184   </if>
185 </template>
186 <!-- get root location from URL -->
187 <template name="get-root">
188   <param name="url" />
189   <choose>
190     <when test="contains($url,'//')">
191       <value-of select="concat(substring-before($url,'//'),'/',
192         substring-before(substring-after($url,'/'),'/' )"/>
193     </when>
194     <otherwise>UNKNOWN ROOT</otherwise>
195   </choose>
196 </template>
197 <!-- return namespace of a qname -->
198 <template name="return-ns">
199   <param name="qname" />
200   <variable name="ns_prefix" select="substring-before($qname
201     ,':')"/>
202   <if test="string-length($ns_prefix)>0"> <!-- prefix must be
203     explicit -->
204     <variable name="name" select="substring-after($qname,':')"/>
205     <value-of select="ancestor-or-self::* / namespace::*[name()=
206       $ns_prefix][position()=1]"/>
207   </if>
208   <if test="string-length($ns_prefix)=0 and ancestor-or-self
209     ::* / namespace::*[name()=' '][position()=1]"> <!-- no prefix
210     -->
211     <variable name="name" select="substring-after($qname,':')"/>
212     <value-of select="ancestor-or-self::* / namespace::*[name()
213       =' '][position()=1]"/>
214   </if>
215 </template>
216 <!-- expand namespace of a qname -->
217 <template name="expand-ns">
218   <param name="qname" />
219   <variable name="ns_prefix" select="substring-before($qname
220     ,':')"/>
221   <if test="string-length($ns_prefix)>0"> <!-- prefix must be
222     explicit -->

```

```

217 <variable name="name" select="substring-after($qname,':')"
    />
218 <variable name="ns_uri" select="ancestor-or-self::* /
    namespace::*[name()=$ns_prefix][position()=1]" />
219 <value-of select="concat($ns_uri,$name)" />
220 </if>
221 <if test="string-length($ns_prefix)=0 and ancestor-or-self
    :::* / namespace::*[name()=''][position()=1]"> <!-- no prefix
    -->
222 <variable name="name" select="substring-after($qname,':')"
    />
223 <variable name="ns_uri" select="ancestor-or-self::* /
    namespace::*[name()=''][position()=1]" />
224 <value-of select="concat($ns_uri,$name)" />
225 </if>
226 </template>
227
228 <!-- determines the CURIE / URI of a node -->
229 <template name="self-curie-or-uri" >
230 <param name="node" />
231 <choose>
232 <when test="$node/attribute::about"> <!-- we have an about
    attribute to extend -->
233 <call-template name="expand-curie-or-uri"><with-param
    name="curie_or_uri" select="$node/attribute::about"/></call-
    template>
234 </when>
235 <when test="$node/attribute::src"> <!-- we have an src
    attribute to extend -->
236 <call-template name="expand-curie-or-uri"><with-param
    name="curie_or_uri" select="$node/attribute::src"/></call-
    template>
237 </when>
238 <when test="$node/attribute::resource and not($node/
    attribute::rel or $node/attribute::rev)"> <!-- enforcing the
    resource as subject if no rel or rev -->
239 <call-template name="expand-curie-or-uri"><with-param
    name="curie_or_uri" select="$node/attribute::resource"/></
    call-template>
240 </when>
241 <when test="$node/attribute::href and not($node/attribute::
    rel or $node/attribute::rev)"> <!-- enforcing the href as
    subject if no rel or rev -->
242 <call-template name="expand-curie-or-uri"><with-param
    name="curie_or_uri" select="$node/attribute::href"/></call-
    template>
243 </when>
244 <when test="$node/self::h:head or $node/self::h:body or
    $node/self::h:html"><value-of select="$this"/></when> <!--

```

```

enforcing the doc as subject —>
245   <when test="$node/attribute::id"> <!-- we have an id
attribute to extend —>
246     <value-of select="concat($this,'#',$node/attribute::id)"
/>
247   </when>
248   <otherwise>blank:node:<value-of select="generate-id($node)"
/></otherwise>
249 </choose>
250 </template>
251
252
253 <!-- expand CURIE / URI —>
254 <template name="expand-curie-or-uri" >
255   <param name="curie-or-uri" />
256   <choose>
257     <when test="starts-with($curie-or-uri,'[:'])"> <!-- we have
a CURIE blank node —>
258     <value-of select="concat('blank:node:',substring-after(
substring-before($curie-or-uri,']'),'[:'])"/>
259     </when>
260     <when test="starts-with($curie-or-uri,[''])"> <!-- we have a
CURIE between square brackets —>
261     <call-template name="expand-ns"><with-param name="qname"
select="substring-after(substring-before($curie-or-uri,']')
,[''])"/></call-template>
262     </when>
263     <when test="starts-with($curie-or-uri,'#')"> <!-- we have
an anchor —>
264     <value-of select="concat($this,$curie-or-uri)" />
265     </when>
266     <when test="string-length($curie-or-uri)=0"> <!-- empty
anchor means the document itself —>
267     <value-of select="$this" />
268     </when>
269     <when test="not(starts-with($curie-or-uri,['']) and
contains($curie-or-uri,':')"> <!-- it is a URI —>
270     <value-of select="$curie-or-uri" />
271     </when>
272     <when test="not(contains($curie-or-uri,'://')) and not(
starts-with($curie-or-uri,['/'])"> <!-- relative URL —>
273     <value-of select="concat($this_location,$curie-or-uri)" />
274     </when>
275     <when test="not(contains($curie-or-uri,'://')) and (starts-
with($curie-or-uri,['/'])"> <!-- URL from root domain —>
276     <value-of select="concat($this_root,substring-after(
$curie-or-uri,['/'])" />
277     </when>
278     <otherwise>UNKNOWN CURIE URI</otherwise>

```



```

279     </choose>
280 </template>
281
282 <!-- returns the first token in a list separated by spaces -->
283 <template name="get-first-token">
284     <param name="tokens" />
285 <if test="string-length($tokens)>0">
286     <choose>
287         <when test="contains($tokens, ' ')">
288             <value-of select="normalize-space(substring-before(
289                 $tokens, ' '))"/>
289         </when>
290         <otherwise><value-of select="$tokens" /></otherwise>
291     </choose>
292 </if>
293 </template>
294
295 <!-- returns the namespace for an object property -->
296 <template name="get-relrev-ns">
297     <param name="qname" />
298 <variable name="ns_prefix" select="substring-before(translate(
299     $qname, '[]', ' '), ':')"/>
300 <choose>
301     <when test="string-length($ns_prefix)>0">
302         <call-template name="return-ns"><with-param name="qname"
303             select="$qname"/></call-template>
304     </when>
305     <!-- returns default_voc if the predicate is a reserved
306         value -->
307     <otherwise>
308         <variable name="is-reserved"><call-template name="check-
309             reserved"><with-param name="nonprefixed"><call-template name=
310                 "no-leading-colon"><with-param name="name" select="$qname"
311                 /></call-template></with-param></call-template></variable>
312         <if test="$is-reserved='true'"><value-of select="
313             $default_voc" /></if>
314     </otherwise>
315 </choose>
316 </template>
317
318 <!-- returns the namespace for a data property -->
319 <template name="get-property-ns">
320     <param name="qname" />
321 <variable name="ns_prefix" select="substring-before(translate(
322     $qname, '[]', ' '), ':')"/>
323 <choose>
324     <when test="string-length($ns_prefix)>0">
325         <call-template name="return-ns"><with-param name="qname"
326             select="$qname"/></call-template>

```

```

318     </when>
319     <!-- returns default_voc otherwise -->
320     <otherwise><value-of select="$default_voc" /></otherwise>
321 </choose>
322 </template>
323
324 <!-- returns the qname for a predicate -->
325 <template name="get-predicate-name">
326     <param name="qname" />
327     <variable name="clean_name" select="translate($qname
328 , '[]', ' ')" />
329     <call-template name="no-leading-colon"><with-param name="
330 name" select="$clean_name"/></call-template>
331 </template>
332
333 <!-- no leading colon -->
334 <template name="no-leading-colon">
335     <param name="name" />
336 <choose>
337     <when test="starts-with($name, ': ')"> <!-- remove leading
338 colons -->
339     <value-of select="substring-after($name, ': ')" />
340     </when>
341     <otherwise><value-of select="$name" /></otherwise>
342 </choose>
343 </template>
344
345 <!-- check if a predicate is reserved -->
346 <template name="check-reserved">
347     <param name="nonprefixed" />
348     <choose>
349     <when test="$nonprefixed='alternate' or $nonprefixed='
350 appendix' or $nonprefixed='bookmark' or $nonprefixed='cite'">
351 true</when>
352     <when test="$nonprefixed='chapter' or $nonprefixed='contents
353 ' or $nonprefixed='copyright' or $nonprefixed='first'">true</
354 when>
355     <when test="$nonprefixed='glossary' or $nonprefixed='help'
356 or $nonprefixed='icon' or $nonprefixed='index'">true</when>
357     <when test="$nonprefixed='last' or $nonprefixed='license' or
358 $nonprefixed='meta' or $nonprefixed='next'">true</when>
359     <when test="$nonprefixed='p3pv1' or $nonprefixed='prev' or
360 $nonprefixed='role' or $nonprefixed='section'">true</when>
361     <when test="$nonprefixed='stylesheet' or $nonprefixed='
362 subsection' or $nonprefixed='start' or $nonprefixed='top'">
363 true</when>
364     <when test="$nonprefixed='up'">true</when>
365     <when test="$nonprefixed='made' or $nonprefixed='previous'
366 or $nonprefixed='search'">true</when> <!-- added because

```

```

they are frequent —>
354   <otherwise>false </otherwise>
355 </choose>
356 </template>
357
358 <!-- named templates to generate RDF - - - - -
      - - - - - —>
359
360 <template name="recursive-copy"> <!-- full copy —>
361   <copy><for-each select="node() | attribute::* "><call-template
      name="recursive-copy" /></for-each></copy>
362 </template>
363
364
365 <template name="subject"> <!-- determines current subject —>
366   <choose>
367
368     <!-- current node is a meta or a link in the head and with
      no about attribute —>
369     <when test="(self::h:link or self::h:meta) and ( ancestor::
      h:head ) and not(attribute::about)">
370       <value-of select="$this"/>
371     </when>
372
373     <!-- an attribute about was specified on the node —>
374     <when test="self::* / attribute::about">
375       <call-template name="expand-curie-or-uri"><with-param
      name="curie_or_uri" select="@about"/></call-template>
376     </when>
377
378     <!-- an attribute src was specified on the node —>
379     <when test="self::* / attribute::src">
380       <call-template name="expand-curie-or-uri"><with-param
      name="curie_or_uri" select="@src"/></call-template>
381     </when>
382
383
384     <!-- an attribute typeof was specified on the node —>
385     <when test="self::* / attribute::typeof">
386       <call-template name="self-curie-or-uri"><with-param name=
      "node" select="."/></call-template>
387     </when>
388
389     <!-- current node is a meta or a link in the body and with
      no about attribute —>
390     <when test="(self::h:link or self::h:meta) and not(
      ancestor::h:head ) and not(attribute::about)">
391       <call-template name="self-curie-or-uri"><with-param name="
      node" select="parent::*"/></call-template>

```

```

392     </when>
393
394     <!-- an about was specified on its parent or the parent had
395     a rel or a rev attribute but no href or an typeof. -->
396     <when test="ancestor::*[attribute::about or attribute::src
397     or attribute::typeof or attribute::resource or attribute::
398     href or attribute::rel or attribute::rev][position()=1]">
399     <variable name="selected_ancestor" select="ancestor::*[
400     attribute::about or attribute::src or attribute::typeof or
401     attribute::resource or attribute::href or attribute::rel or
402     attribute::rev][position()=1]" />
403     <choose>
404     <when test="$selected_ancestor[(attribute::rel or
405     attribute::rev) and not (attribute::resource or attribute::
406     href)]">
407     <value-of select="concat('blank:node:INSIDE_',generate
408     -id($selected_ancestor))" />
409     </when>
410     <when test="$selected_ancestor/attribute::about">
411     <call-template name="expand-curie-or-uri"><with-param
412     name="curie_or_uri" select="$selected_ancestor/attribute::
413     about"/></call-template>
414     </when>
415     <when test="$selected_ancestor/attribute::src">
416     <call-template name="expand-curie-or-uri"><with-param
417     name="curie_or_uri" select="$selected_ancestor/attribute::src
418     "/></call-template>
419     </when>
420     <when test="$selected_ancestor/attribute::resource">
421     <call-template name="expand-curie-or-uri"><with-param
422     name="curie_or_uri" select="$selected_ancestor/attribute::
423     resource"/></call-template>
424     </when>
425     <when test="$selected_ancestor/attribute::href">
426     <call-template name="expand-curie-or-uri"><with-param
427     name="curie_or_uri" select="$selected_ancestor/attribute::
428     href"/></call-template>
429     </when>
430     <otherwise>
431     <call-template name="self-curie-or-uri"><with-param
432     name="node" select="$selected_ancestor"/></call-template>
433     </otherwise>
434     </choose>
435     </when>
436
437     <otherwise> <!-- it must be about the current document -->
438     <value-of select="$this" />
439     </otherwise>
440

```

```

423     </choose>
424 </template>
425
426 <!-- recursive call for object(s) of object properties -->
427 <template name="recurse-objects" >
428   <xsl:for-each select="child::*">
429     <choose>
430       <when test="attribute::about or attribute::src"> <!-- there
         is a known resource -->
431       <call-template name="expand-curie-or-uri"><with-param name="
         curie_or_uri" select="attribute::about | attribute::src"/></
         call-template><text> </text>
432       </when>
433       <when test="(attribute::resource or attribute::href) and (
         not (attribute::rel or attribute::rev or attribute::property)
         )" > <!-- there is an incomplet triple -->
434       <call-template name="expand-curie-or-uri"><with-param name="
         curie_or_uri" select="attribute::resource | attribute::href"
         /></call-template><text> </text>
435       </when>
436       <when test="attribute::typeof and not (attribute::about)">
         <!-- there is an implicit resource -->
437       <call-template name="self-curie-or-uri"><with-param name="
         node" select="."/></call-template><text> </text>
438       </when>
439       <when test="attribute::rel or attribute::rev or attribute::
         property"> <!-- there is an implicit resource -->
440       <if test="not (preceding-sibling::*[attribute::rel or
         attribute::rev or attribute::property])"> <!-- generate the
         triple only once -->
441       <call-template name="subject"/><text> </text>
442       </if>
443       </when>
444       <otherwise> <!-- nothing at that level thus consider
         children -->
445       <call-template name="recurse-objects"/>
446       </otherwise>
447     </choose>
448   </xsl:for-each>
449 </template>
450
451 <!-- generate recursive call for multiple objects in rel or
         rev -->
452 <template name="relrev" >
453   <param name="subject" />
454   <param name="object" />
455
456   <!-- test for multiple predicates -->

```

```

457 <variable name="single-object"><call-template name="get-
first-token"><with-param name="tokens" select="$object"/></
call-template></variable>
458
459 <if test="string-length(@rel)>0">
460 <call-template name="relation">
461 <with-param name="subject" select="$subject" />
462 <with-param name="object" select="$single-object" />
463 <with-param name="predicate" select="@rel"/>
464 </call-template>
465 </if>
466
467 <if test="string-length(@rev)>0">
468 <call-template name="relation">
469 <with-param name="subject" select="$single-object" />
470 <with-param name="object" select="$subject" />
471 <with-param name="predicate" select="@rev"/>
472 </call-template>
473 </if>
474
475 <!-- recursive call for multiple predicates -->
476 <variable name="other-objects" select="normalize-space(
substring-after($object, ' '))" />
477 <if test="string-length($other-objects)>0">
478 <call-template name="relrev">
479 <with-param name="subject" select="$subject"/>
480 <with-param name="object" select="$other-objects"/>
481 </call-template>
482 </if>
483
484 </template>
485
486
487 <!-- generate an RDF statement for a relation -->
488 <template name="relation" >
489 <param name="subject" />
490 <param name="predicate" />
491 <param name="object" />
492
493 <!-- test for multiple predicates -->
494 <variable name="single-predicate"><call-template name="get-
first-token"><with-param name="tokens" select="$predicate"
/></call-template></variable>
495
496 <!-- get namespace of the predicate -->
497 <variable name="predicate-ns"><call-template name="get-
relrev-ns"><with-param name="qname" select="$single-predicate
"/></call-template></variable>
498

```

```

499     <!-- get name of the predicate -->
500     <variable name="predicate-name"><call-template name="get-
501     predicate-name"><with-param name="qname" select="$single-
502     predicate"/></call-template></variable>
503
504     <choose>
505     <when test="string-length($predicate-ns)>0"> <!-- there is
506     a known namespace for the predicate -->
507     <choose>
508     <when test="starts-with($subject, 'blank:node: ')"><
509     value-of select="java:eu.tatoofp7.corecomponents.
510     websiteconnector.RDFAParser.setSubjectNodeID($parser,
511     substring-after($subject, 'blank:node: '))"/></when>
512     <otherwise><value-of select="java:eu.tatoofp7.
513     corecomponents.websiteconnector.RDFAParser.setSubjectURI(
514     $parser, $subject)"/></otherwise>
515     </choose>
516     <!-- get full predicate -->
517     <variable name="expanded-predicate"><call-template name="
518     expand-ns"><with-param name="qname" select="$single-predicate
519     "/></call-template></variable>
520     <value-of select="java:eu.tatoofp7.corecomponents.
521     websiteconnector.RDFAParser.setPropertyURI($parser, $expanded-
522     predicate)"/>
523
524     <choose>
525     <when test="starts-with($object, 'blank:node: ')"><value
526     -of select="java:eu.tatoofp7.corecomponents.websiteconnector.
527     RDFAParser.setObjectNodeID($parser, substring-after($object, '
528     blank:node: '))"/></when>
529     <otherwise><value-of select="java:eu.tatoofp7.
530     corecomponents.websiteconnector.RDFAParser.setObjectURI(
531     $parser, $object)"/></otherwise>
532     </choose>
533     </when>
534     <otherwise> <!-- no namespace generate a comment for debug
535     -->
536     <xsl:comment>No namespace for the rel or rev value ;
537     could not produce the triple for: <value-of select="$subject"
538     /> - <value-of select="$single-predicate" /> - <value-of
539     select="$object" /></xsl:comment>
540     </otherwise>
541     </choose>
542
543     <value-of select="java:eu.tatoofp7.corecomponents.
544     websiteconnector.RDFAParser.flushObjectProperty($parser)"/>
545
546     <!-- recursive call for multiple predicates -->

```

```

525     <variable name="other-predicates" select="normalize-space(
substring-after($predicate, ' '))" />
526     <if test="string-length($other-predicates)>0">
527     <call-template name="relation">
528         <with-param name="subject" select="$subject"/>
529         <with-param name="predicate" select="$other-predicates"/>
530         <with-param name="object" select="$object"/>
531     </call-template>
532     </if>
533
534 </template>
535
536
537 <!-- generate an RDF statement for a property -->
538 <template name="property" >
539     <param name="subject" />
540     <param name="predicate" />
541     <param name="object" />
542     <param name="datatype" />
543     <param name="attrib" /> <!-- is the content from an
attribute ? true /false -->
544     <param name="language" />
545
546     <!-- test for multiple predicates -->
547     <variable name="single-predicate"><call-template name="get-
first-token"><with-param name="tokens" select="$predicate"
/></call-template></variable>
548
549     <!-- get namespace of the predicate -->
550     <variable name="predicate-ns"><call-template name="get-
property-ns"><with-param name="qname" select="$single-
predicate"/></call-template></variable>
551
552
553     <!-- get name of the predicate -->
554     <variable name="predicate-name"><call-template name="get-
predicate-name"><with-param name="qname" select="$single-
predicate"/></call-template></variable>
555
556     <choose>
557         <when test="string-length($predicate-ns)>0"> <!-- there is
a known namespace for the predicate -->
558
559             <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setPropertyURI($parser,$predicate
-name)"/>
560
561         <choose>

```



```

562     <when test="starts-with($subject,'blank:nod: ')"><
value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setSubjectNodeID($parser,
substring-after($subject,'blank:node: '))"/></when>
563     <otherwise><value-of select="java:eu.tatoofp7.
corecomponents.websiteconnector.RDFaParser.setSubjectURI(
$parser,$subject)"/></otherwise>
564     </choose>
565
566     <if test="string-length($language)>0"><value-of select="
java:eu.tatoofp7.corecomponents.websiteconnector.RDFaParser.
setLanguage($parser,$language)"/></if>
567
568     <choose>
569         <when test="$datatype='http://www.w3.org/1999/02/22-
rdf-syntax-ns#XMLLiteral'">
570             <choose>
571                 <when test="$attrib='true'"> <!-- content is in an
attribute -->
572                 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setDataType($parser,$datatype)"/>
573                 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setValue($parser,normalize-space(
string($object)))/>
574                 </when>
575                 <otherwise> <!-- content is in the element and may
include some tags -->
576                 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setDataType($parser,$datatype)"/>
577                 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setValue($parser,$object)"/>
578                 </otherwise>
579             </choose>
580         </when>
581         <when test="string-length($datatype)>0">
582             <!-- there is a datatype other than XMLLiteral -->
583             <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setDataType($parser,$datatype)"/>
584             <choose>
585                 <when test="$attrib='true'"> <!-- content is in an
attribute -->
586                 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setValue($parser,normalize-space(
string($object)))/>
587                 </when>
588                 <otherwise> <!-- content is in the text nodes of the
element -->
589                 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setValue($parser,$object)"/>

```

```

590         </otherwise>
591     </choose>
592     </when>
593     <otherwise> <!-- there is no datatype -->
594         <choose>
595             <when test="$attrib='true'"> <!-- content is in an
attribute -->
596                 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setValue($parser,normalize-space(
string($object)))/>
597             </when>
598             <otherwise> <!-- content is in the text nodes of the
element -->
599                 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.setValue($parser,$object)/>
600             </otherwise>
601         </choose>
602     </otherwise>
603 </choose>
604 </when>
605 <otherwise> <!-- generate a comment for debug -->
606     <xsl:comment>Could not produce the triple for: <value-of
select="$subject" /> - <value-of select="$single-predicate"
/> - <value-of select="$object" /></xsl:comment>
607 </otherwise>
608 </choose>
609
610 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFaParser.flushDataProperty($parser)/>
611
612 <!-- recursive call for multiple predicates -->
613 <variable name="other-predicates" select="normalize-space(
substring-after($predicate,' '))" />
614 <if test="string-length($other-predicates)>0">
615 <call-template name="property">
616     <with-param name="subject" select="$subject"/>
617     <with-param name="predicate" select="$other-predicates"/>
618     <with-param name="object" select="$object"/>
619     <with-param name="datatype" select="$datatype"/>
620     <with-param name="attrib" select="$attrib"/>
621     <with-param name="language" select="$language"/>
622 </call-template>
623 </if>
624
625 </template>
626
627 <!-- generate an RDF statement for a class -->
628 <template name="class">
629     <param name="resource" />

```

```

630 <param name="class" />
631
632 <!-- case multiple classes -->
633 <variable name="single-class"><call-template name="get-first
-token"><with-param name="tokens" select="$class"/></call-
template></variable>
634
635 <!-- get namespace of the class -->
636 <variable name="class-ns"><call-template name="return-ns"><
with-param name="qname" select="$single-class"/></call-
template></variable>
637
638 <if test="string-length($class-ns)>0"> <!-- we have a qname
for the class -->
639 <variable name="expended-class"><call-template name="
expand-ns"><with-param name="qname" select="$single-class"
/></call-template></variable>
640 <choose>
641 <when test="starts-with($resource, 'blank:node:')"><
value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFAParser.setSubjectNodeID($parser,
substring-after($resource, 'blank:node:'))"/></when>
642 <otherwise><value-of select="java:eu.tatoofp7.
corecomponents.websiteconnector.RDFAParser.setSubjectURI(
$parser, $resource)"/></otherwise>
643 </choose>
644 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFAParser.setPropertyURI($parser, 'http://
www.w3.org/1999/02/22-rdf-syntax-ns#type')"/>
645 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFAParser.setObjectURI($parser, $expended-
class)"/>
646 <value-of select="java:eu.tatoofp7.corecomponents.
websiteconnector.RDFAParser.flushObjectProperty($parser)"/>
647 </if>
648
649
650 <!-- recursive call for multiple classes -->
651 <variable name="other-classes" select="normalize-space(
substring-after($class, ' '))" />
652 <if test="string-length($other-classes)>0">
653 <call-template name="class">
654 <with-param name="resource" select="$resource"/>
655 <with-param name="class" select="$other-classes"/>
656 </call-template>
657 </if>
658
659 </template>
660

```

```
661
662 <!-- ignore the rest of the DOM -->
663 <template match="text()|@*|*" mode="rdf2rdfxml"><apply-templates
        mode="rdf2rdfxml" /></template>
664
665
666 </stylesheet>
```

Appendix C

Curriculum Vitae

Name: Bojan Božić
Address: Wulkahof 10/7, 7064 Oslip, Austria
Contact: bojan.bozic@ait.ac.at, +43 (0)664 2351 800



Education

- *Bachelor of Science in Engineering, Information and Communication Systems and Services*
University of Applied Sciences Technikum, Vienna, Austria, June 2007
Concentration: Telecommunication and Internet Technologies
- *Master of Science in Engineering, Software Engineering and Multimedia*
University of Applied Sciences Technikum, Vienna, Austria, June 2008
Concentration: Application Development
- *PhD, Doctoral Studies*
University of Vienna, Vienna, Austria, current
Concentration: Semantic Web

Experience

Software Development Engineer January 2005 - March 2009
Philips Speech Recognition Systems, Philips Austria, Vienna, Austria

- Development of different speech recognition editors (TX, Word, RTF)
- Development of a Java/COM and .NET adapter
- Leadership of a technical team

Scientist March 2009 - present
Safety and Security Department, Austrian Institute of Technology

- Development of a multi-domain language for time series processing in Python
- Development of a sensor web enablement framework
- Leadership of a technical team
- Leadership of a work package and scientific work in a European project in the field of Semantic Web

Scientific

Talks and Posters

- B. Božić
"The New Time Series Toolbox - Next Generation of Sensor Web and Time Series Processing";
Presentation: FOSS4G 2010 - Free Open Source Software for Geospatial, Barcelona; 06.09.2010 - 09.09.2010.
- B. Božić:
"The Time Series Toolbox";
Poster: European Geosciences Union General Assembly 2010, Vienna, Austria; 02.05.2010 - 07.05.2010.
- B. Božić:
"Time is on Your Side - Usage of Python in Time Series Processing";
Poster: PyCon 2010, Atlanta, USA; 19.02.2010 - 21.02.2010.
- T.Bleier, B. Božić:
"Time Series Toolbox";
Presentation: 52north SWE Workshop, Muenster, Germany; 10.11.2009 - 11.11.2009.

- T. Bleier, A. Bonitz, B. Božić, T. Ponweiser:
"The Time Series Toolbox";
 Presentation: FOSS4G 2009 - Free Open Source Software for Geospatial, Sydney, Australia; 21.10.2009.
- T. Bleier, B. Božić, A. Bonitz, R. Barta:
"Jumping Worlds with Python";
 Presentation: EuroPython 2009, Birmingham, UK; 30.06.2009.

Papers

- B. Božić, W. Winiwarter
"Tools for the Support of Semantic Search";
 Paper: in "Structuring Legal Semantics", published by Editions Weblaw, Bern, Switzerland, 2011, ISBN: 978-3-905742-80-0, S. 387 - 399.
- B. Božić
"Simulation and Modeling of Semantically Enriched Time Series";
 Paper: 19th International Congress on Modelling and Simulation, Perth, Australia; 12.12.2011 - 16.12.2011; in "Sustaining our Future: Understanding and Living with Uncertainty", Modelling and Simulation Society of Australia and New Zealand, (2012), ISBN: 978-09872143-1-7; 7 S.
- G. Schimak, B. Božić, A. Kaufmann, J. Peters-Anders, P. Dihé, A. Rizzoli, T. Lobo, G. Avellino
"The TaToo Semantic Case - Requirements, impacts and applications";
 Paper: EnviroInfo 2011, 25th International Conference Environmental Informatics, Ispra, Italy; 03.10.2011 - 05.10.2011; in "Innovations in Sharing Environmental Observations and Information", W. Pillmann, S. Schade, P. Smits (Hrg.); SHAKER Verlag, 2 (2011), ISBN: 978-3-8440-0451-9; S. 832-847.
- G. Schimak, P. Dihé, T. Pariente, G. Avellino, L. Petronzio, A. Rizzoli, S. Nešić, B. Božić
"Environmental Information Enrichment - The TaToo Approach";
 Paper: 19th International Congress on Modelling and Simulation, Perth, Australia; 12.12.2011 - 16.12.2011; in "Sustaining our Future: Understanding and Living with Uncertainty", Modelling and Simulation Society of Australia and New Zealand, (2012), ISBN: 978-09872143-1-7; 6 S.
- B. El-Gamil, W. Winiwarter, B. Božić, H. Wahl
"Deep web integrated systems: current achievements and open issues";

Paper: iiWAS2011 - Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services, Ho Chi Minh City, Vietnam; 05.12.2011 - 07.12.2011; in: "Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services", (2011), ISBN: 978-1-4503-0784-0; S. 447 - 450.

- B. Božić, J. Peters-Anders, G. Schimak
"Filtering of Semantically Enriched Environmental Time Series";
 Paper: International Congress on Environmental Modelling and Software, Leipzig; 01.07.2012 - 05.07.2012; in: "Proceedings of the sixth biennial meeting of the International Environmental Modelling and Software Society", Int. Environmental Modelling and Software Society, Leipzig (2012), ISBN: 978-88-9035-742-8; S. 2395 - 2402.
- B. Božić, W. Winiwarter
"Community Building Based on Semantic Time Series";
 Paper: International Conference on Information Integration and Web-based Applications and Services, Bali; 03.12.2012 - 05.12.2012; in: "Proceedings of the 14th International Conference on Information Integration and Web-based Applications and Services", Association for Computing Machinery, New York, (2012), ISBN: 978-1-4503-1306-3; S. 213 - 222.
- B. Božić
"A Multi-Domain Framework for Community Building Based on Data Tagging";
 Paper: International Semantic Web Conference, Boston; 11.11.2012 - 15.11.2012; in: "The Semantic Web - ISWC 2012", Springer, Heidelberg Berlin (2012), ISBN: 978-3-642-35172-3; S. 441 - 444.
- B. Božić, W. Winiwarter
"Ontology Mapping and Reasoning in Semantic Time Series Processing";
 Paper: International Conference on Information Integration and Web-based Applications and Services, Vienna; 02.12.2012 - 04.12.2012; in: "Proceedings of the 15th International Conference on Information Integration and Web-based Applications and Services", Association for Computing Machinery, New York, (2013); S. 443 - 452.

Lectures

- B. Božić:
"Software Engineering";
 Lecture: University of Applied Sciences Burgenland, Eisenstadt WS 2009/10, WS 2010/11, WS 2011/12, WS 2012/13, WS 2013/14.
- B. Božić:
"System Architectures, Nets, and Web Technologies";
 Exercise: University of Applied Sciences Burgenland, Eisenstadt SS 2010.
- B. Božić:
"Designing Software Systems";
 Lecture: University of Applied Sciences Burgenland, Eisenstadt WS 2010/11, WS 2011/12, WS 2012/13, WS 2013/14.
- B. Božić:
"Applied Project Software Engineering";
 Exercise: University of Applied Sciences Burgenland, Eisenstadt SS 2011, SS 2012.
- B. Božić:
"Applied Project System Architectures, Nets, and Web Technologies";
 Exercise: University of Applied Sciences Burgenland, Eisenstadt SS 2011, SS 2012.
- B. Božić:
"Distributed Systems";
 Exercise: University of Applied Sciences Burgenland, Eisenstadt SS 2012, SS 2014.

Books

- Bleier, T.; Božić, B.; Bumerl-Lexa, R.; Da Costa, A.; Costes, S.; Iosifescu, I.; Martin, O.; Frysinger, S.; Havlik, D.; Hilbring, D.; Jacques, P.; Klopfer, M.; Kunz, S.; Kutschera, P.; Lidstone, M.; Middleton, S.E.; Roberts, Z.; Sabeur, Z.; Schabauer, J.; Schlobinski, S.; Shu, T.; Simonis, I.; Stevenot, B.; Uslander, T.; Watson, K.; Wittamore, K. :
"SANY - An Open Service Architecture for Sensor Networks";
 Book: SANY Consortium, ISBN 978-3-00-028571-4; 2009.

Projects

- SANY - Sensors Anywhere - IST FP6 Integrated Project

- TaToo - Tagging Tool based on a Semantic Discovery Framework - FP7 Project
- Europeana Creative - creative re-use of cultural heritage metadata and content