Building Web Service Ontologies

Marta Sabou

SIKS Dissertation Series No. 2006-4.

The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Graduate School for Information and Knowledge Systems.

ISBN 90-9018400-7

VRIJE UNIVERSITEIT

Building Web Service Ontologies

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op donderdag 27 april 2006 om 13.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

Reka Marta Sabou

geboren te Turda, Roemenië

promotor:	prof.dr. F.A.H. van Harmelen
copromotor:	prof.dr. H. Stuckenschmidt

Preface

One of my favorite metaphors about life is the one that compares it to an ever more challenging expedition. According to this metaphor each challenge or task in our life is a mountain that we climb. At its time, each mountain seams the most important and the hardest to conquer, however, when we rich its top we realize that there are so many even higher mountains to climb ahead. This thesis is the result of a four years effort to climb probably the hardest mountain in my life so far. Reaching its peak has given me the possibility to engage in even higher challenges. There are many ahead. At the same time, looking back to all this journey I realize that I could have never completed it without the help of so many people. This is the time to thank them all!

My supervisors, Frank and Heiner, have undoubtedly played a central role in this four year apprenticeship in research. It is thanks to Frank that I have enrolled for a PhD position on the first place. After supervising my master's thesis, he considered that I had the potential to become a researcher and offered me a PhD position in his group (and convinced me to accept it). I am grateful that Frank and Heiner have supported me in pursuing a rather risky topic that was quite distant from their expertise. Even so, they have managed to steer my work in a successful direction. Besides technical content, by working with them, I had the opportunity to learn several of the skills that are essential for research, from writing papers to reviewing the work of fellow researchers. However, the hardest part of their role, much harder than overviewing the technical content or providing me with the actual tools to perform research, was that of "teaching" me the right attitude towards research. Through their passion for science, optimism and incredible dedication to their work they have inspired and motivated me every single day of these four years.

Because the topic of my thesis has been novel to the core expertise of the KR&R group, I have performed much of the work in cooperation with several researchers from different research groups. Already during my master thesis, and then all along these four years, I have often cooperated with the Dutch research company Aduna and its experienced staff (the late Jos van der Meer, Arjohn Kampman, Christiaan Fluit, Herko ter Horst, Jeen Broekstra, Jeroen Wester and Hilde Bleeker). My first papers were written in cooperation with the members of the IIDS group, Sander van Splunter and Frances Brazier, and with Debbie Richards from Macquarie University in Sydney who was visiting the group at that time. Most of the work presented in this thesis has been accomplished in the framework of the WonderWeb European project. This project gave me the opportunity to perform joint work with Daniel Oberle from the University of Karlsruhe, my colleague Peter Mika and Aldo Gangemi from ISTC Rome. I have also benefitted of important help in a crucial stage of my work from several researchers at the Univer-

siteit van Amsterdam, namely Maarten de Rijke, Gilad Mishne and Valentin Jijkoun. Yet another European project, Knowledge Web, gave me the opportunity to spend five very productive and motivating weeks in Sheffield with the GATE group (Kalina Bontcheva, Diana Maynard, Hamish Cunningham, Valentin Tablan, Niraj Aswani). The work done in Sheffield helped me to finalize my most important case study in cooperation with Carole Goble and Chris Wroe from the University of Manchester. I would like to acknowledge and thank the contribution that all these people have brought to the thesis.

Besides the many people with which I have closely cooperated, I have enjoyed the support of an even larger number of colleagues and friends. I owe special thanks to Mar Marcos, my first room mate who also became a dear friend. I remember with great pleasure the cooperation with Jeen Broekstra (my part-time roommate), Michel Klein and Jacco van Ossenbruggen from CWI on teaching the Web-based Knowledge Representation course at the VU. All other colleagues, both from the AI and the BI groups, (Annette, Borys, Gusz, Holger, Laura, Maksym, Mark, Peter, Radu, Stefen, Zharko, Zhisheng, Wouter) have contributed to a pleasant and motivating environment as well as several interesting lunch-time discussions (especially Ronny, Chide and Ziv). I owe thanks and an Italian dinner to Mark van Assen who has translated the summary of the thesis in Dutch. I would also like to thank the members of the reading committee for their insightful and constructive comments on this thesis. Thanks also to Harriett Cornish from KMi who helped me with the cover design. Finally, all my special thanks to my friends in Amsterdam - Alessandro, Cecile, Dirk, Federico, Peter, Raquel, Natalia and Yulia. I hope that distance will not break our friendships.

Even with the help of so many friends, colleagues and fellow researchers, I could have never completed this thesis without the help of my family. Its been thanks to their uncounted sacrifices and unlimited love that I had the opportunity to follow a higher education, to finalize my studies abroad and to finish this thesis. My uncle Zoltan from Australia provided the financial means that allowed me to go to university. But more than that, he has always been a raw model for daring and using ones talents to the maximum. It would take more pages than those of this thesis to describe all the hardships that my mother has endured through the last 18 years since I first left home in the interest of education. I want to thank her for all those years, for her immense love that helped her through them and for teaching me that humans can endure infinitely when they truly believe in something. My sister, also, has supported me in these years through her love and wisdom of life which compensates my own, often brittle, personality. Unfortunately, four of my family members, my three grandparents and my father, have not lived to rejoice with me on this day. Their dear memory and joy of life live on with me every day.

I am extremely fortunate that I could share the joy and the pain of the last three years with my dear husband, Giovanni. He has been by my side during hard times and taught me how to truly celebrate success. I am grateful for his deep love, endless optimism, jokes, his wonderfully interesting stories about history and politics that have turned me into a news-eater from a total ignorant, for teaching me obvious things about life that I have somehow ignored with all that schooling and for his cooking that forces me on a permanent diet. I also thank his family, Rossana, Massimo and Maurizio, for all their support and for regarding me as a true member of the *famiglia* even if my Italian is not yet perfect. *Grazie a tutti!*

Contents

Chapter 1

Introduction

The World Wide Web (Berners-Lee, 1999) granted uniform access to heterogeneous data sources residing anywhere on the Internet. However, the tremendous success of the Web lead to its explosive increase to levels where finding the right information becomes a non trivial task. As a response to these limitations, in the last few years the Web encountered two revolutionary changes which aim to transform it from a static document collection in an intelligent and dynamic data integration environment.

The first innovative technology, the ***Web service*** technology, allows uniform access via Web standards to software components residing on various platforms and written in different programming languages. As a result, software components providing a variety of functionalities (ranging from currency conversion to flight booking) are now accessible via the Web. Indeed, Web service technology has introduced a new abstraction layer over and a radically new architecture for software. From a business perspective, Web services often correspond to business services and thus the compositionality paradigm that underlies the Web service technology allows composing existing business services into new and more complex services.

The second novel Web technology, the ***Semantic Web*** (Berners-Lee et al., 2001), develops techniques for augmenting existing Web data with logics based formal descriptions of their meaning. In this thesis we are concerned with the annotation of semi-structured data. The semantic markup is machine processable and therefore facilitates access and integration of the vast amount of Web data. Core to the Semantic Web technology are *ontologies* (Gruber, 1993). An ontology is a formal representation of consensual world knowledge and it provides the elements for building semantic descriptions.

A major limitation of the Web services technology is that finding and composing services still requires manual effort. This becomes a serious burden with the increasing number of Web services (over 1000 services exist in bioinformatics alone (Lord et al., 2005)). To address this problem, semantic Web researchers proposed to augment Web services with a semantic description of their functionality in order to facilitate their discovery and integration. This technology, combining Web services and Semantic Web techniques, is referred to as ***semantic Web services*** (McIlraith et al., 2001).

Semantic Web service descriptions rely on two kinds of ontologies. First, a *generic Web service ontology* specifies the main aspects used to describe Web services regardless of the domain in which they operate. For example, such an ontology provides the vocabulary to describe Web services inputs, outputs and complex internal data flows. Second, a *domain ontology* provides concepts from the domain of the Web service to populate the generic description template built with the concepts of the generic ontology. These concepts denote entities in the domain of the Web service (e.g., *Food, Hotel*) as well as functionalities that can be performed by services in the given domain (e.g., *OrderFood, BookHotel*). We collectively refer to generic and domain ontologies employed for Web service descriptions as *Web service ontologies*.

A major **problem** of the semantic Web services technology, which we address in this thesis, is that **building Web service ontologies is a time consuming and complex task**. This constitutes a major bottleneck in the development of the semantic Web service technology. The ontology building process is different for generic and domain ontologies. A generic ontology is typically developed by a group of experts but intended for a large user base. A good example is OWL-S (Martin et al., 2003) which is developed by a committee of experts with less than 20 members but which is already used by many research projects which adopt it as a standard for structuring their Web service descriptions. Typically, experts reside all over the world and the development process is mainly conducted through virtual environments (i.e., email, phone conferences). Eventual feedback from the users influences the further development of the standard by the committee. In contrast with generic ontologies, there is little research focusing on building Web service domain ontologies (i.e., domain ontologies used in a Web service context). Domain experts must reuse an existing ontology or build their own ontology if an appropriate ontology does not exist already. However, despite their importance, few domain ontologies exist for describing Web services thus often requiring the building of a new ontology.

We are not aware of any research in the field of semantic Web services that aims to qualitatively improve Web service ontologies or to facilitate the process of building these ontologies. However, previous research performed in the context of software engineering and ontology engineering is relevant when considering the problem of building Web service ontologies.

The more generic problem of acquiring complex semantic descriptions for software components was experienced as difficult in software engineering. For example, a comprehensive survey of existing software libraries revealed that the difficulty of acquiring software semantics hampered the adoption of sophisticated solutions in industrial settings (Mili et al., 1998). Our literature study revealed a few approaches that have addressed the acquisition of software semantics. However, they are not explicitly tailored towards building ontologies.

There exists a plethora of work on ontology engineering that has a high applicability to the problem of building and enhancing Web service ontologies. First, the literature documents a number of methodologies that prescribe the main steps of an ontology building process (Gomez-Perez et al., 2003) (or knowledge engineering in general (Schreiber et al., 1999)). However, these traditional methodologies do not consider the particularities of distributed and decentralized settings where the ontology is shaped by the dynamic requirements of big or open user groups. An exception is the DILIGENT methodology (Vrandecic et al., 2005) which accurately captures the phases of such a process.

The second body of relevant work aims to qualitatively enhance existing ontologies. Methodologies relying of ontological notions, such as OntoClean (Guarino and Welty, 2004), validate the ontological adequacy of complex ontologies. Techniques, such as alignment to foundational ontologies, permit analyzing, disambiguating and enriching ontologies such as WordNet (Gangemi et al., 2002) or other industry ontologies (Borgo and Leitao, 2004). Prior to our work, no such methods were applied in the context of Web services, or more generically, software (except the work of G. Guizzardi on ontology-driven conceptual modeling which also included some aspects of software development (Guizzardi, 2005)).

Yet a third category of approaches, grouped under the term of *Ontology Learning*, facilitate (semi-)automatic acquisition of ontologies from unstructured, semi-structured or structured data sources (Maedche and Staab, 2001). A variety of ontology learning methods and tools have already been developed (Gomez-Perez and Manzano-Mancho, 2003) but none of these addresses the special context of Web services. While the field is slowly reaching maturity and it is increasingly applied in a variety of application domains, issues related to evaluation and usability still require investigation.

The work in this thesis addresses the problem of building Web service ontologies by adapting some of the existing techniques in the above mentioned fields to the particular context of Web services. Important contributions are brought to those fields as well.

In the rest of this chapter we state the research questions that we addressed in this thesis, we describe our contributions to the state of the art and we provide an overview of the thesis.

1.1 Research Questions

The central question investigated by this thesis is:

How to facilitate the process of building Web service ontologies and how to enhance the quality of these ontologies?

Our work was centered around the following concrete research questions:

Q1. What requirements should Web service ontologies fulfill?
A prerequisite for improving the state of the art in Web service ontology development is to have a good understanding of this process. In particular, it is important to identify the requirements that Web service ontologies should fulfill.

Q2. How to enhance the quality of generic Web service ontologies?
Since generic Web service ontologies are used as standards for creating semantic descriptions they should have a high quality. To ensure this we need ways to test if they fulfill the previously identified requirements, to identify problematic aspects or limitations and to provide appropriate solutions.

Q3. Is semi-automatic acquisition of Web service domain ontologies feasible?
The acquisition of Web service domain ontologies is a time consuming task whose automation is desirable. It is important to identify the requirements for an automatic acquisition solution in the context of Web services and the data sets that are typically available. Existing ontology learning methods have to be adapted to the particularities of the context, rigorously evaluated and incorporated in easy to use tools.

Our work on these research questions lead to several contributions, as summarized in the next section.

1.2 Contributions

The work presented in this thesis contributes to the development of Web service ontologies and, implicitly, to the realization of the semantic Web services idea. Our contributions fall in three major categories:

1. **Identifying requirements for Web service ontologies.**
   Despite the large body of work in the area of semantic Web services, few efforts are directed towards identifying requirements for Web service ontologies. We are only aware of the requirements stated in (Lara et al., 2003) and (Grosof et al., 2004). Based on our experiences when working with the evolving semantic Web services technology we identified a set of requirements that Web service ontologies should fulfil. While non-exclusive, our list of requirements complements the requirements brought forward by the community so far.

2. **Methods to enhance generic Web service ontologies. Enhancements to OWL-S.**
   A first important goal in the field of semantic Web services was the design of generic Web service ontologies. However, little attention has been given to the evaluation of the quality of these ontologies. We identified a set of methods to analyze generic Web service ontologies and to improve their quality. First, we recommend using the generic ontology to describe real life services in order to increase its modelling expressiveness. Second, we adapt the ontology for use in other, related domains in order to test the generality of the conceptualized knowledge. Third, we use alignment to a foundational ontology for disambiguating the meaning of the concepts proposed by the ontology and for increasing its axiomatization. Alignment of several ontologies to a foundational ontology allows their harmonization. We used these methods to analyze OWL-S, identify its limitations and propose solutions to overcome these limitations. Many of these observations and solutions are valuable findings for developing any other generic Web service ontology as well. From the perspective of foundational ontologies our contribution was an extension for using them in the context of software components.

3. **Learning Web service domain ontologies.**
   While domain ontologies play an important role when building semantic Web service descriptions, little research has concentrated on ways to automate their acquisition. No guidelines and no tools exist to support the acquisition of these ontologies. In this thesis we pioneered work that, on a long run, aims to achieve (semi-) automatic acquisition of Web service domain ontologies. We start by analyzing the problem of ontology learning in the context of Web services. We identify a set of characteristics that constrain the development of an automated solution and design a framework for ontology learning by taking into account these characteristics. We experiment with two different implementations of the framework and use an elaborate evaluation to determine which method provides a better performance. We implement both our methods in a prototype system and we use different means to make our tool more user friendly. With this work we also contributed to the field of ontology learning by showing that existing methods can be successfully adapted to the Web service context.

1.3 Structure of the Thesis

This thesis is structured in three main parts each covering one of the research questions.

Part I: Context and Related Work
In the first part we describe the context of our work, identify a set of requirements for Web service ontologies (as a response to the first research question) and survey related work.

Part II: Enhancing Generic Web Service Ontologies
In this part we address the second research question. We describe several methods that can be used to analyze and to enhance different aspects of generic Web service ontologies. We demonstrate the use of these methods in the context of OWL-S. We analyze whether OWL-S meets the requirements stated in Chapter 2 and offer (partial) solutions to enhance eventual limitations.

Part III: Learning Web Service Domain Ontologies
In the final part we investigate the third research question related to the semi-automatic acquisition of Web service domain ontologies. We analyze the requirements for such an automatic solution, then design, implement and evaluate two different learning methods.

The material of the thesis is distributed in individual chapters as follows:

Part I: Context and Related Work

In **Chapter 2** we introduce the semantic Web services technology and conclude on some requirements that Web service ontologies should fulfill.

In **Chapter 3** we overview related work and existing ontology engineering practices that can potentially be applied to solve the stated problem.

Part II: Enhancing Generic Web Service Ontologies

In **Chapter 4** we use OWL-S to describe a set of real life services. This allows us to conclude on the modelling expressiveness and usability of OWL-S.

In **Chapter 5** we investigate the adaptability of OWL-S by extending it to describe software modules managed by an application server middleware. Our analysis reveals that OWL-S is based on several valuable design principles that make it easy to reuse.

In **Chapter 6** we show that OWL-S is ambiguous and poorly axiomatized and suggest the enhancement of these aspects by alignment to the DOLCE foundational ontology. Our alignment methodology is generally valid for aligning any other generic Web service ontologies potentially contributing to a harmonization of these ontologies.

Part III: Learning Web Service Domain Ontologies

In **Chapter 7** we analyze the process of building domain ontologies in two application domains. Based on our analysis we establish a set of requirements that an ontology learning solution to this problem should fulfill. Then we present a framework that adapts ontology learning methods in the context of Web services. We describe two instantiations of the framework that rely on natural language processing techniques with different levels of complexity. The last part of this chapter provides some details on the implementation of our prototype system.

In **Chapter 8** we perform the evaluation of the framework. We provide generic considerations about the ontology learning evaluation methods and describe the evaluation metrics that we employ. We use these metrics to assess and compare the two instantiations of the framework on both available data sets.

Finally, in **Chapter 9** we discuss our conclusions and contributions and point out future work.

1.4 Publications

Several chapters of this thesis are based on previous publications:

- Chapter 2 on semantic Web services is based on "Stuckenschmidt, H., Sabou, M., and Klein, M. (2004a). Semantic Web Technology - Bringing Meaning to Distributed Systems. *IEEE Distributed Systems Online*" and two papers published in cooperation with members of the OWL-S committee: "Martin, D. et al. (2003) OWL-S 1.0 white paper[1]", and "Martin, D. et al. (2004). Bringing Semantics to Web Services: The OWL-S Approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA".

- Chapter 4 combines the material published in "Sabou, M., Richards, D., and van Splunter, S. (2003). An experience report on using DAML-S. In *Proceedings of the Workshop on E-Services and the Semantic Web, The 12th WWW Conference*, Budapest, Hungary" and "Richards, D. and Sabou, M. (2003). Semantic Markup for Semantic Web Tools: A DAML-S description of an RDF-Store. In *Proceedings of the Second International Semantic Web Conference*, volume 2870 of LNCS, pages 274 - 289, Sanibel Island, Florida, USA". This work was performed in cooperation with Debbie Richards, Sander van Splunter and Frances Brazier in the context of the Agent Factory project. Our work was also supported by the members of the Dutch company Aduna[2] by providing support while working with the Sesame ontology storage and query engine.

- Chapter 5 was published as "Sabou, M., Oberle, D., and Richards, D. (2004). Enhancing Application Servers with Semantics. In *Proceedings of the First Australian Workshop on Engineering Service-Oriented Systems (AWESOS)*, Melbourne, Australia" and is based on a cooperation with Daniel Oberle and Debbie Richards.

- The material in Chapter 6 is the result of joint work with Peter Mika, Daniel Oberle and Aldo Gangemi in the context of the WonderWeb project. This work was first published as "Mika, P., Sabou, M., Gangemi, A., and Oberle, D. (2004b). Foundations for DAML-S: Aligning DAML-S to DOLCE. *In First International Semantic Web Services Symposium (SWS2004), AAAI Spring Symposium Series*". Then, an extended version of this symposium paper was published as "Mika, P., Oberle, D.,

[1]`http://www.daml.org/services/owl-s/1.0/`
[2]`http://aduna.biz/`

Gangemi, A., and Sabou, M. (2004a). Foundations for Service Ontologies: Aligning OWL-S to DOLCE. In *Proceedings of the Thirteens International World Wide Web Conference (WWW2004).* ACM Press".

- Chapter 7 and 8 are based on material published in "Sabou, M. (2004b). From Software APIs to Web Service Ontologies: a Semi-Automatic Extraction Method. In *Proceedings of the Third International SemanticWeb Conference, ISWC,* Hiroshima, Japan", "Sabou, M., Wroe, C., Goble, C., and Mishne, G. (2005). Learning Domain Ontologies for Web Service Descriptions: an Experiment in Bioinformatics. In *Proceedings of the 14th International World Wide Web Conference,* Chiba, Japan" and "Sabou, M., Wroe, C., Goble, C., and Stuckenschmidt, H. (2005). Learning Domain Ontologies for Semantic Web Service Descriptions. *Journal of Web Semantics. 3(4).* ".

  Considerations about the evaluation of our methods (detailed in Chapter 8) were first published in "Sabou, M. (2004a). Extracting Ontologies from Software Documentation: a Semi-Automatic Method and its Evaluation. In *Proceedings of the ECAI-2004 Workshop on Ontology Learning and Population (ECAI-OLP),* Valencia, Spain" and refined in a follow-up book chapter - "Sabou, M. (2005a). Learning Web Service Ontologies: an Automatic Extraction Method and its Evaluation. In Buitelaar, P., Cimmiano, P., and Magnini, B., editors, *Ontology Learning and Population.* IOS Press".

  Ideas about possible extensions of the presented work were described in "Sabou, M. and Pan, J. (2005). Towards Improving Web Service Repositories through Semantic Web Techniques. In *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE) collocated with the 4th International Semantic Web Conference (ISWC 2005)*".

  The implementation details provided in chapter 7 describe work on using visualisation techniques to support ontology learning. Our experiences in this direction have been published in "Sabou, M. (2005b). Visual Support for Ontology Learning: an Experience Report. In *Proceedings of the 9th International Conference on Information Visualisation (IV05),* London, UK". The work on visualization was conducted in cooperation and with the support of Chritiaan Fluit from Aduna. This cooperation resulted in three book chapters published prior to our work on adapting the visualization to support ontology learning (Fluit et al., 2002, 2004, 2005).

Part I

Context and Related Work

Chapter 2

Semantic Web Services

In this chapter we provide a brief introduction to semantic Web services and the supporting Semantic Web and Web services technology. We give examples of the use of these technologies in a concrete example scenario showing how they enhance information access and integration. We conclude the chapter with a list of requirements that Web service ontologies should fulfill.

Parts of this chapter are based on the following co-authored materials: an article in an online journal (Stuckenschmidt et al., 2004), the white paper for OWL-S 1.0 (Martin et al., 2003) and a workshop paper about the use of OWL-S (Martin et al., 2004).

2.1 The Changing World Wide Web

The World Wide Web and the novel technologies that extend it (i.e., Semantic Web, Web Services and Semantic Web Services), address data access and integration issues. In this chapter we briefly introduce these technologies showing the advantages each of them brings in an **example scenario**. The scenario consists of the task of *finding a Medicare*[1] *certified pharmacy which is in a mile range from a location identified by zip code 19901*.

The World Wide Web emerged as a solution to the information management problem that the CERN[2] research institute was facing in the late nineties (Berners-Lee, 1989). The data-intensive research activity performed at CERN produced an increasing amount of digital data (about projects, researchers, software etc.) which was distributed on a network of computers running different operating systems and using particular (often incompatible) data formats. Finding and integrating this distributed and heterogeneous data was increasingly difficult. The Web was a distributed hypertext system that allowed access to heterogeneous data sources by using a set of standards for describing data (HTML), localizing documents on a network (URI) and accessing them (HTTP). Due to the simplicity of HTML, which allowed creation of Web pages with a minimum effort, the Web became a tremendous success in a relatively short time. While the first prototype of the Web was written in 1990, the Web expended so fast that in 1994 the W3C (World

[1]Medicare is an American health insurance program.

[2]`http://www.cern.ch`

Wide Web Consortium) was created[3]. Since then the number of Web sites and servers increased at a rapid pace to over 60 million web sites in March 2005[4].

Example scenario. Before the Web, information needed for performing our example task (e.g., list of pharmacies with their details) was available either in printed format or in digital format isolated on different computers. With the Web all this information is available for search to anyone. A vast range of medical information is available online from disease and drug descriptions to homepages of medical institutions (e.g., hospitals, pharmacies, health insurance programs). All it takes is looking up the Web page of some pharmacies and estimating the distance between two zip codes.

Problems of the World Wide Web. With the accelerated growth of the Web, finding the right information becomes increasingly difficult even with support from search engines such as Google. Current search technology relies on keyword based search. These techniques provide a relatively *high recall* (as all Web sites that mention a given keyword are retrieved) but a low semantic recall (as pages about the desired topic but not containing the keywords are ignored). Their *precision* is low because only few of the retrieved pages contain the information that the user needs. As a result, complex queries such as "*Medicare certified pharmacy one mile from 19901*" are hard to answer because (1) such specific information is often not made available as such on the Web or, (2) if it is made available it can take different syntactic forms. A more generic query, "*Medicare certified pharmacy*", retrieves more than half a million hits, where these keywords appear spread within each document. Trying to refine this query by enforcing that the three keywords appear in this order, results in two hits — and again none of them answers our quest. The major cause of these inefficiencies of keyword based search is that keywords are treated as strings rather than meaningful entities. This technique cannot deal with the complexity of human language and phenomena such as synonymy, polysemy etc.

The Changing World Wide Web. As a response to these limitations, the Web encountered two revolutionary changes as depicted in Figure 2.1. First, the Semantic Web community aims at a **semantic** extension of the current **syntactic** Web by augmenting existing Web information with logics based formal descriptions of their meaning. This semantic markup would be machine processable and therefore allow easier access and integration of the vast amount of the available information than what can be achieved with keyword based search. Second, the Web is changing from a collection of **static** Web pages to a **dynamic** environment with the advent of the Web services technology that makes software components accessible via Web protocols. The semantic Web services technology developed at the cross road of these two technologies by applying Semantic Web techniques to Web services. In the rest of this chapter we briefly present these three technologies.

2.2 Semantic Web

The goal of the Semantic Web is to solve the current limitations of the Web by augmenting Web information with a formal (i.e., machine processable) representation of its meaning. A direct benefit of this machine processable semantics would be the enhance-

[3]See `http://www.w3.org/2004/Talks/Styles/w3c10/images/timeline.pdf` for a timeline on the Web's development.

[4]`http://news.netcraft.com/`

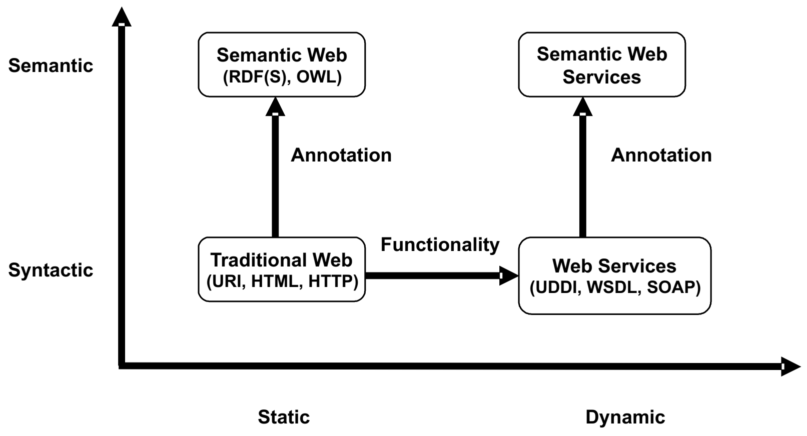

Semantic
Syntactic
Semantic Web (RDF(S), OWL)
Semantic Web Services
Annotation
Annotation
Traditional Web (URI, HTML, HTTP)
Functionality
Web Services (UDDI, WSDL, SOAP)
Static
Dynamic

Figure 2.1: The changing Web.

ment and automation of several information management tasks, such as search or data integration. The idea of applying knowledge representation and reasoning techniques in the context of the Web has been investigated from the mid-nineties notably by work on SHOE (Simple HTML Ontology Extensions) (Luke et al., 1996) and Ontobroker (Fensel et al., 1998). The Semantic Web term was clearly associated to this line of research in 2001 when it was defined as:

"The Semantic Web is an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." (Berners-Lee et al., 2001)

There have been several different approaches to realizing the Semantic Web. These approaches can be distinguished by the type of data sources that they consider for semantic description as well as the richness of the semantic annotation. A first approach, reflecting the principles of GOFAI (Good Old Fashioned AI), aims at extensively capturing the meaning of textual data sources in rich semantic descriptions. Unfortunately, research in the Information Retrieval community has already shown that formally representing information contained in textual documents is a bad idea, that content simply cannot be recoded in a general way. Indeed, the success of this community relied on developing more lightweight techniques that do not require a complete understanding of the document content. The focus of the semantic Web community has gradually shifted from this initial approach.

An alternative to the GOFAI approach is that of decreasing the complexity of semantic annotations added to textual documents. This can be achieved by using semantic Web languages that have a low semantic expressiveness (such as XML). This kind of lightweight annotations have been successfully used in the natural language processing field to capture several aspects of the analyzed corpora. Also, looking to the evolution of the Semantic Web, many of the first applications using Semantic Web technology have mainly relied on limited semantic information and used semantic languages with low expressivity (XML, RDF(S)).

The current focus of Semantic Web research is more and more directed towards supporting intelligent data exchange. In this case the information that is being annotated is not unstructured text but rather semi-structured information available from databases or exchanged between Web services. Annotations are also extended to several different information artifacts, such as images, music or Web services. The role of the semantic annotations is to support merging, integrating and exchanging data between applications/organizations. While this approach is considerably different from the original GOFAI view on the Semantic Web, it brings enormous challenges and promises immediate practical benefits. The content of this thesis is in line with this third view on the Semantic Web as our goal is not so much to access textual documents but to facilitate the annotation of Web services and to allow intelligent data manipulation.

The *"well-defined meaning"* of information is provided by semantic descriptions, often referred to as *metadata* (i.e., data about data). There are two characteristics specific to semantic metadata. First, metadata describes information in terms of a domain vocabulary whose meaning is specified by a formal domain model (ontology). Second, metadata is expressed in a representation language that can be parsed and interpreted by machines. We briefly describe ontologies and ontology languages in what follows.

2.2.1 Ontologies

The term *ontology*, originating from Philosophy as detailed in (Kivela and Hyvonen, 2002) and (Smith and Welty, 2001), was adopted by AI researchers to describe formal domain models. Several ontology definitions were provided in the last decades. The most frequently cited definition is that given by Gruber in 1993 according to which an ontology is "an **explicit specification** of a **conceptualization**". In other words, an ontology is a domain model (*conceptualization*) which is *explicitly* described (*specified*). Later, in 1997 Borst defines an ontology as a "**formal** specification of a **shared** conceptualization" (Borst, 1997). This definition requires, in addition to Gruber's definition, that the conceptualization should express a *shared* view between several parties, a consensus rather than an individual view. Also, this conceptualization should be expressed in a machine readable format (*formal*). In 1998, Studer et al. (Studer et al., 1998) merge these two definitions stating that:

"An ontology is a formal, explicit specification of a shared conceptualization."

As consensual domain models, the primary role of ontologies is to enhance communication between humans (e.g., establishing a shared vocabulary, explaining the meaning of the shared terms to reach consensus). As formal models, ontologies represent knowledge in a computer processable format thus enhancing communication between humans and computer programs or two computer programs. Therefore, ontologies are investigated by several research fields in the context of diverse application areas. Already in 1998, Guarino offers an extensive list of references to research fields that have recognized by that time the importance of ontologies (Guarino, 1998), ranging from knowledge engineering to information retrieval and integration. Four years later, McGuinness reports on the use of ontologies in several Web related tasks such as Web site organization, navigational support, browsing and searching (McGuinness, 2002).

Example Scenario. To demonstrate the benefits of using ontological descriptions for our task we built an ontology that specifies a set of concepts related to health care providers (See Figure 2.2). **Concepts** such as *HealthCareProvider*, *HealthProgram* and *Pharmacy* denote the main entities in the domain to be conceptualized. A set of relations

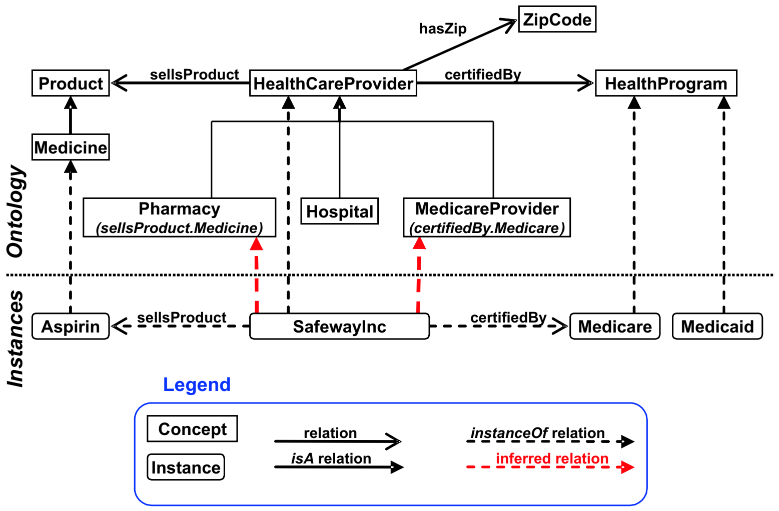


Figure 2.2: Example ontology and ontology instances.

can be established between concepts. An important relation is the *isA* relation which indicates subsumption between two concepts. The example ontology specifies three kinds of *HealthCareProvider* concepts, i.e. three subclasses. Several restrictions can be imposed on properties, property restriction being a frequently used mechanism for defining subclasses. For example, a *Pharmacy* is any *HealthCareProvider* which sells *Medicine*s (a *Hospital* might differ from a *Pharmacy* in the kind of offered products or services). Several pharmacy Web pages can be semantically described with the concepts defined by this ontology. For example, the data instances in Figure 2.2 describe that SafeWayInc is a Medicare certified *HealthCareProvider* which sells aspirin (which is a *Medicine*).

Semantic descriptions enhance information access and integrating. First, several different syntactic forms will be encoded by the same semantic information thus overcoming the limitations of the keyword based search. Second, based on the ontology that explains the relations between concepts in a machine understandable, formal way, a computer can perform reasoning. For example, it can automatically deduce that SafewayInc can be classified both as a *Pharmacy* (because it fulfills the restriction that it sells medicines) and a *MedicareProvider*, because it is certified by this program. Thus semantic data helps to enhance and automate our task.

Even if Studer's definition (Studer et al., 1998) states the major ontology characteristics, considerable variations exist along the dimensions defined by these characteristics. Semantic Web technology, and implicitly our work, relies on ontologies with different *levels of detail* and *generality* of the captured conceptualization. We briefly discuss these characteristics for the purpose of classifying the ontologies used in the thesis. Note that we adopt an intuitive rather than a rigorous attitude in this discussion.

While there is a general agreement that one of the major characteristics of an ontology is the ***level of generality*** of the specified conceptualization, there has been much debate on defining different categories of generality (Guarino, 1997; van Heijst et al., 1997; Guarino, 1998; Studer et al., 1998). It is not our purpose to debate the differences between these views but we rather adopt three intuitive classes of generality.

Foundational (or top-level) ontologies are conceptualizations that contain specifications of domain and problem independent concepts and relations (such as space, time, matter) based on formal principles derived from linguistics, philosophy, and mathematics. In this thesis we use of the DOLCE foundational ontology. Other examples of top-level ontologies, all discussed and compared in (Borgo et al., 2002), are: the Suggested Upper Merged Ontology (SUMO) (Pease et al., 2002), OpenCyc[5] and the Basic Formal Ontology (BFO) (Smith, 2003).

Generic ontologies contain generic knowledge about a certain domain such as medicine, biology, mathematics or Web services. These domain concepts are often specified in terms of top-level concepts thus inheriting the general theories behind the top-level concepts. In this thesis we used the OWL-S (Martin et al., 2003) ontology which specifies a set of concepts that allow describing Web services.

Domain ontologies have the lowest reusability and are specific to a particular domain.

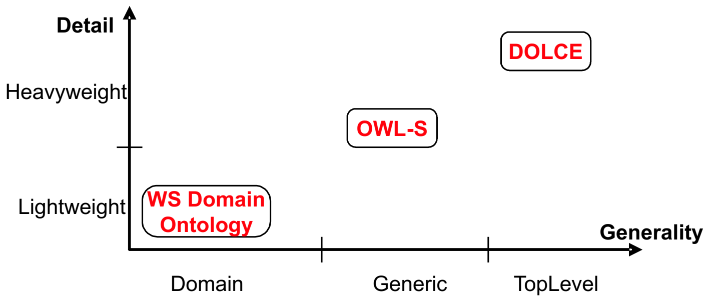


Figure 2.3: Example ontologies classified according to the two dimensions.

A second classification criterion is the ***level of detail*** of the specification (Guarino, 1997; McGuinness, 2002; Uschold and Jasper, 1999). In this thesis we adopt the terminology introduced in (Corcho et al., 2003) to distinguish between weak and rich domain theories. *Lightweight* ontologies are domain models that include a taxonomic hierarchy as well as properties between concepts. *Heavyweight* ontologies contain axioms and constraints as well. Note that the distinction between lightweight and heavyweight ontologies is rather blurred as these are intuitive rather than fixed measures.

In this thesis we worked with several types of ontologies, as depicted in Figure 2.3:

- **lightweight, domain ontologies** that are automatically learned;
- a **heavyweight, generic ontology**, OWL-S, which contains a couple of restrictions but which is still relatively poor from an ontological perspective;
- a **heavyweight, foundational ontology**, DOLCE.

[5]`http://www.opencyc.org`

2.2.2 Ontology Languages for the Semantic Web

Ontologies are explicitly specified in a formal language. The work performed in this thesis relied on the RDF(S) and OWL languages.

RDF(S). RDF and RDF Schema[6] represented the first step towards a Web based ontology language. RDF is a data model allowing to describe resources on the Web. RDF Schema is based on RDF and allows the definition of basic ontology elements such as classes and their hierarchy, properties with their domain, range and hierarchy (McBride, 2004). As such RDF(S) is well suited for expressing lightweight ontologies. Several tools were developed for RDF(S) and a considerable number of Semantic Web applications used this language before a more expressive ontology language was developed.

```
<owl:Class rdf:ID="MedicareProvider">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="certifiedBy"/>
      </owl:onProperty>
      <owl:hasValue>
        <HealthProgram rdf:ID="Medicare"/>
      </owl:hasValue>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:ID="HealthCareProvider"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="HealthProgram"/>
```

OWL. The example above shows the OWL declaration of the *MedicareProvider* concept as a Medicare certified *HealthCareProvider*[7]. The Web Ontology Language (OWL)[8] is a more expressive ontology language than RDF(S). The basis for developing OWL was the DAML+OIL language which originated by merging two language proposals that aimed at overcoming the expressivity limitations of RDF(S): DAML-ONT[9] and OIL[10] (Horrocks et al., 2003). OWL enhances the expressivity of RDF(S) providing means to describe relations between classes (e.g., disjointness, union, intersection), cardinality and value restrictions on properties (e.g., cardinality, universal and existential quantifiers), property characteristics (e.g., transitivity, symmetry), equality etc.. OWL provides the constructs to encode ontological knowledge (such as in the previous subsection) using the XML based syntax.

2.3 Web Services

Besides its spectacular growth, the Web becomes more dynamic with the advent of the Web service technology. A Web service (WS) is a (self-contained) software component

[6] `http://www.w3.org/RDF/`

[7] This serialization was generated by Protege's OWL Plugin, `http://protege.stanford.edu/plugins/owl/`.

[8] Web-Ont Working Group Web site: `http://www.w3.org/2001/sw/WebOnt/`.

[9] `http://www.daml.org/2000/10/daml-ont.html`

[10] `http://www.ontoknowledge.org/oil/`

that allows access to its functionality via a Web interface. WSs communicate by employing established protocols for message transport and encoding. Indeed, the W3C Web Services Architecture Working Group defines a Web service as:

"*a software application identified by an URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols.*" (W3C, 2002)

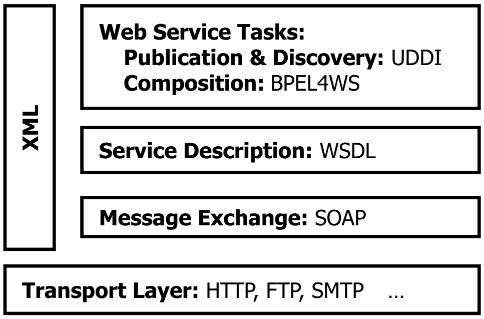

XML
Web Service Tasks:
Publication & Discovery: UDDI
Composition: BPEL4WS
Service Description: WSDL
Message Exchange: SOAP
Transport Layer: HTTP, FTP, SMTP ...

Figure 2.4: Overview of Web Service Standards.

Web service technology has introduced a new abstraction layer over and a radically new architecture for software. Indeed, the innovative vision is that by employing a set of XML standards to define and describe Web service functionalities, several tasks such as discovery and composition of these services can be facilitated (or even automated) to some extent. Web service technology also aims to facilitate the interaction between different Web services (i.e., software programs) by enforcing the use of XML standards for data exchange. Note, that any kind of data can be exchanged between Web services (e.g., semi-structured, textual, structured) as long as it is embedded in an XML based messaging protocol.

Figure 2.4 (adapted from (de Aalst, 2003)) shows the main Web service technology standards, all based on XML. A Web service interface is described using the Web Service Description Language[11] (WSDL). Web services exchange messages encoded in the SOAP[12] (Simple Object Access Protocol) messaging framework and transported over HTTP or other Internet protocols. Several tasks can be performed with Web services. A typical Web service life-cycle envisions the following scenario. A service provider *publishes* the WSDL description of his service in UDDI[13], a registry that permits Universal Description Discovery and Integration of Web services. Subsequently, service requesters can inspect UDDI and *locate/discover* Web services that are of interest. Using the information provided by the WSDL description they can directly *invoke* the corresponding Web service. Further, several Web services can be composed to achieve a more complex functionality. Such compositions of services can be specified using BPEL4WS[14] (Busi-

[11]`http://www.w3.org/TR/wsdl`
[12]`http://www.w3.org/TR/soap/`
[13]`http://www.uddi.org/`
[14]`ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf`

ness Process Execution Language for Web Services). By relying on these standards, Web services hide any implementation details therefore increasing cross-language and cross-platform interoperability.

Example Scenario. Many Web services allow access to large databases permitting controlled access to information that might not be explicitly stated on Web pages. For example, the pharmacy finding process could be repeated for *any* zip code (or for any health care provider) relying on the output of one or more Web services and not on data provided by static Web pages. A good Web service to generalize this task is the MedicareSupplier Web service[15] which can retrieve details of Medicare suppliers given a zip code, a city name or the types of supplies provided.

```
Service(
   PortType:MedicareSupplierSoap (
     op:GetSupplierByZipCode(
           IMsg(zip), OMsg(SupplierDataLists))
     op:GetSupplierByCity(
           IMsg(City), OMsg(SupplierDataLists))
     op:GetSupplierBySupplyType(
           IMsg(description), OMsg(SupplierDataLists))
     )
)
```

The example above shows a schematic representation of the WSDL file associated to the MedicareSupplier service[16]. WSDL has considerable support from industry and increasing tool-support (WSDL generators, editors). As an XML-based language, it is machine processable, being a structured and standardized way to describe web-interfaces of services. In WSDL a service is seen as a collection of network endpoints which operate on messages. The example service provides one **port**: *MedicareSupplierSoap*. This port groups together three **operations** that return lists of Medicare suppliers and their details given a zip code (for *GetSupplierByZipCode*), a city name (for *GetSupplierByCity*) or the description of the supplied material (for *GetSupplierBySupplyType*). Each operation has an input (IMsg) and an output (OMsg) message. A **message** has a name and a set of **parts** of certain type. Parts represent input/output parameters depending if they are declared in the input or the output message. For brevity, the example above does not state the name of the message only the name of its parts. The type of the parts can be any XMLSchema data type or a previously defined complex type (the type of the parts is also omitted by the schematic description). A WSDL document has two major parts. First, the ***abstract interface*** of the service specifies the data types, messages and portTypes with the corresponding operations (which refer to previously defined messages). Second, an ***implementation part*** binds the abstract interface to concrete network protocols and message formats (SOAP, HTTP).

Limitations of the Web Service Technology. SOAP, WSDL, UDDI, and BPEL4WS are the standard combination of technology to build a Web service application. However, they fail to achieve the goals of automation and interoperability because they require humans in the loop (Lassila, 2002). Indeed, WSDL specifies the functionality of the service only at a syntactic level. While these descriptions can be automatically parsed and

[15]The WSDL description of this service is available at `http://www.webservicex.net/medicareSupplier.wsdl`.

[16]We use a schematic representation of WSDL files because the XML-based syntax of WSDL makes it too verbose to be reproduced here.

invoked by machines, the interpretation of their meaning is left for a human programmer. To support reliable, large-scale interconnectivity of Web services by software, computer-processable semantics are needed, which include the properties, capabilities, interfaces, and effects of the service (de Aalst, 2003).

2.4 Semantic Web Services

The Semantic Web community addressed the limitations of current Web service technology by augmenting the service descriptions with a semantic layer in order to achieve their automatic discovery, composition, monitoring and execution (McIlraith et al., 2001; Martin et al., 2004; Stuckenschmidt et al., 2004). The automation of these tasks is highly desirable and, as a result, several research projects adopted semantic Web service technology in different application domains (e.g., bioinformatics grid (Wroe et al., 2004), Problem Solving Methods (Motta et al., 2003)).

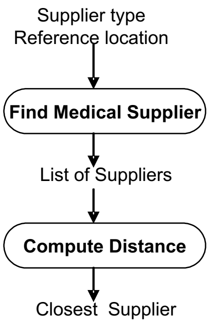


Figure 2.5: Workflow for finding the closest medical supplier.

Example scenario. The example task is a specialization of the more generic task of finding the closest medical provider (or, of the even more generic task of finding a provider in any domain). One of the strategies for performing this generic task is to 1) retrieve the details of all medical providers (of a certain type) and then selecting the closest by 2) computing the distance between the location of the provider and a reference location. This workflow can be schematically represented as in Figure 2.5. For our example task it is enough if we populate this workflow with the MedicareSupplier service and a Web service that calculates distance between zip codes.

Semantic Web service technology aims to automate performing such tasks based on the semantic description of Web services. Using these descriptions the right services can be selected and combined in a way that would solve the task at hand. There are two major approaches to Web service composition (ten Teije et al., 2004). First, given the specification of a start and final state, pre/post condition reasoning is performed to select and combine the right services. Second, using the parametric design paradigm, generic task workflows are formally specified and than populated with the right Web services depending on the task at hand.

A common characteristic of all emerging frameworks for semantic Web service descriptions (OWL-S (Martin et al., 2003), WSMO[17], IRS (Motta et al., 2003) - see overview and comparison in (Cabral et al., 2004)) is that they combine two kinds of ontologies to obtain a service description. First, a *generic Web service ontology*, such as OWL-S, specifies generic Web service concepts (e.g., *Input*, *Output*) and prescribes the backbone of the semantic Web service description. Second, a *domain ontology* specifies knowledge in the domain of the Web service, such as types of service parameters (e.g., *City*) and functionalities (e.g., *FindMedicalSupplier*), that fills in this generic framework. We discuss these two kinds of ontologies in the next two subsections.

2.4.1 Generic Web Service Ontologies

Two generic ontologies for Web service descriptions are under development. First, DAML-S is an ontology that permits describing several aspects of a Web service. DAML-S was translated from DAML to OWL and renamed to OWL-S. The second, more recent, initiative is WSMO (Web Service Modelling Ontology) which, even if has overlaps with OWL-S, is based on different principles and brings several additions to OWL-S (see (Lara et al., 2005) for a detailed analysis of these two ontologies). The research presented in this thesis was performed in the transition period from DAML-S to OWL-S. In this period the WSMO initiative had not yet started. In this thesis we only consider OWL-S. Although we believe that many of our results, particulary those presented in the third part of the thesis, are equally applicable to WSMO. We will elaborate on the generality of our work in Section 9.1.4.

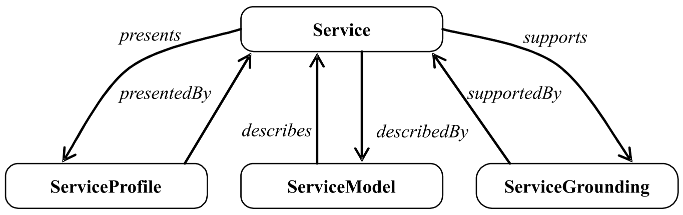


Figure 2.6: The OWL-S Service Ontology. (Note that the arrows in this picture are directed according to the OWL-S model even if their direction might seam counterintuitive.)

The OWL-S ontology is conceptually divided into four sub-ontologies for specifying *what a service does* (Profile[18]), *how the service works* (Process[19]) and *how the service is implemented* (Grounding[20]). A fourth ontology (Service[21]) contains the *Service* concept which links together a *ServiceProfile*, a *ServiceModel* and a *ServiceGrounding* concept (see Figure 2.6). The *Service presents* a *ServiceProfile*, is *described by* a *ServiceModel* and *supports* a *ServiceGrounding*. These three concepts are all further specialized in the Profile, Process and Grounding ontologies respectively. In the rest of this subsection, we explain all the three parts of OWL-S by exemplifying their use for describing our

[17]`http://www.wsmo.org/`
[18]`http://www.daml.org/services/owl-s/1.0/Profile.owl`
[19]`http://www.daml.org/services/owl-s/1.0/Process.owl`
[20]`http://www.daml.org/services/owl-s/1.0/Grounding.owl`
[21]`http://www.daml.org/services/owl-s/1.0/Service.owl`

example service. We also introduce the schematic service representations that will be used through this thesis.

1. The Profile Ontology specifies the functionality offered by the service (e.g., *GetMedicalSupplier*), the semantic type of the inputs and outputs (e.g., *City, Medicare-Supplier*), the details of the service provider and several service parameters, such as quality rating or geographic radius. This description is used for discovering the service. The central concept of this ontology, *Profile*, is a subclass of *ServiceProfile*.

In the schematic representation of semantic Web service descriptions used throughout this thesis, for each *Profile* instance we depict the process it describes (indicated by the *hasProc* relation) and its functional characteristics (Inputs, Outputs, Preconditions, Effects - from now referred to as IOPE's) together with their type. In the example below, the *MedicareSupplier* service presents three profiles (i.e., it offers three distinct functionalities). Each Profile has a semantic type described by one of the functionality concepts *FindMedicareSupplierByZip*, *FindMedicareSupplierByCity* or *FindMedicareSupplierBy-Supply*. Each Profile describes a Process (later specified in the Process Model - *P1*, *P2*, *P3*). Finally, all Profiles return an output which was described with the *SupplierDetails* concept. The input type varies for each Profile: *ZipCode*, *City* or *SupplyType*. Note that this description was constructed using concepts defined in the Web service domain ontology presented in Section 2.4.2.

```
Service MedicareSupplier:
 *Profile:FindMedicareSupplierByZip (hasProc P1)
           (I(ZipCode), O(SupplierDetails))
 *Profile:FindMedicareSupplierByCity (hasProc P2)
           (I(City), O(SupplierDetails))
 *Profile:FindMedicareSupplierBySupply (hasProc P3)
           (I(SupplyType), O(SupplierDetails))
 *ProcessModel: ...
 *WSDLGrounding: ...
```

2. The Process ontology. Many complex services consist of smaller services executed in a certain order. For example, buying a book at Amazon.com involves using a browsing service (which selects the book) and a paying service. OWL-S allows describing such internal process models. These are useful for several purposes. First, one can check that the business process of the offering service is appropriate (e.g., product selection should always happen before payment). Second, one can monitor the execution stage of a service. Third, these process models can be used to automatically compose Web services. A *ServiceModel* concept describes the internal working of the service and it is further specialized as a *ProcessModel* concept in the Process ontology. A *Process-Model* has a single *Process* which can be atomic, simple or composite (composed from atomic processes through various control constructs). Each Process has a set of IOPE's. In our notation, for each service we represent its *ProcessModel* with its *Process*. For each *Process* we depict its type, the involved control constructs, the IOPE's and their types. The *MedicareSupplier* service allows a choice from its three *AtomicProcesses* (corresponding to the three Profiles), therefore its *ProcessModel* consists of a *CompositeProcess* modelled with the *Choice* control construct.

```
Service MedicareSupplier:
 *Profile:...
```

```
*ProcessModel:
   CompositeProcess: MedicareProcess:Choice
   {
    AtomicProcess:P1 (I(ZipCode),O(SupplierDetails))
    AtomicProcess:P2 (I(City),O(SupplierDetails))
    AtomicProcess:P3 (I(SupplyType),O(SupplierDetails))
    }
*WSDLGrounding: ...
```

Profile to Process Bridge. A *Profile* contains several links to the *Process*. Figure 4.2 shows these links, where terms in bold-face belong to the *Profile* ontology and the rest to the *Process* ontology. Firstly, a *Profile* states the *Process* it describes through the unique property `has_process`. Secondly, the *Input*, *Outputs*, *Preconditions* and *Effects* (from now on IOPE) of the *Profile* correspond (in some degree) to the IOPEs of the *Process*. Understanding this correspondence is not so trivial given the fact that the IOPE's play different roles for the *Profile* and for the *Process*. In the *Profile* ontology they are treated equally as parameters of the *Profile* (they are subproperties of the *profile:parameter* property). In the *Process* ontology only *Inputs* and *Outputs* are regarded as subproperties of the *process:parameter* property. The Preconditions and Effects are just simple properties of the *Process*. While technically the IOPEs are properties both for *Profile* and *Process*, the fact that they are treated differently at a conceptual level is misleading. The link between the IOPE's in the Profile and Process part of the DAML-S descriptions is created by the *refersTo* property which has as domain *ParameterDescription* and ranges over the *process:paramater*.

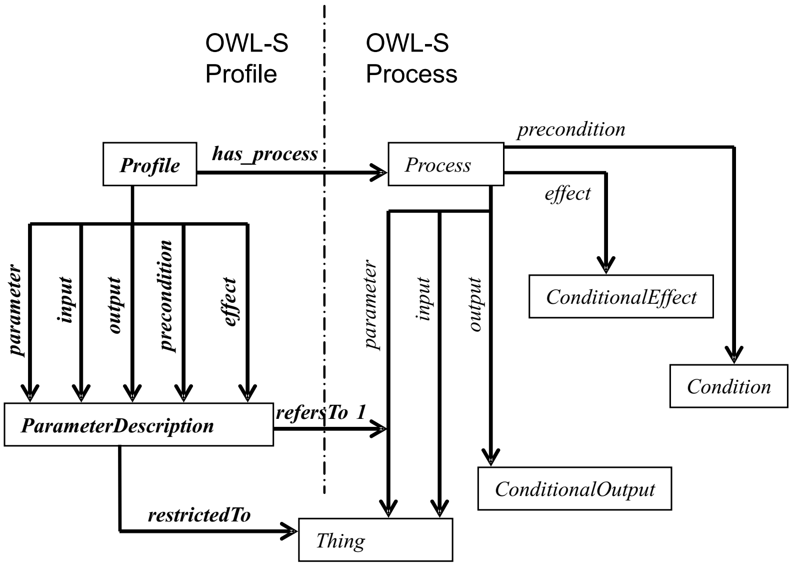


Figure 2.7: Profile to Process bridge.

3. The Grounding ontology provides the vocabulary to link the conceptual description of the service, specified by the Profile and Process, to actual implementation details, such as message exchange formats and network protocols. The grounding to a WSDL description is performed according to three rules:

R1. Each *AtomicProcess* corresponds to one WSDL operation.

R2. As a consequence of the first rule, each input of an *AtomicProcess* is mapped to a corresponding message-part in the input message of the WSDL operation. Similarly for outputs, each output of an *AtomicProcess* is mapped to a corresponding message-part in the output message of the WSDL operation.

R3. The type of each WSDL message part can be specified in terms of a OWL-S parameter (i.e., an XML Schema data type or a OWL concept).

The Grounding ontology specializes the *ServiceGrounding* as a *WSDLGrounding* which contains a set of *WsdlAtomicProcessGrounding* elements, each grounding one of the atomic processes specified in the *ProcessModel*. In our abstract notation, we depict each atomic process grounding by showing the link between the atomic process and the corresponding WSDL element. The *MedicareSupplier* service has three atomic process groundings for each processes of the *ProcessModel*.

```
Service MedicareSupplier:
 *Profile:...
 *ProcessModel:...
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (P1->op:GetSupplierByZipCode)
    WsdlAtomicProcessGrounding: Gr2 (P2->op:GetSupplierByCity)
    WsdlAtomicProcessGrounding: Gr3 (P3->op:GetSupplierBySupplyType)
```

We finish this subsection by describing a set of **design principles** underlying OWL-S that we identified during our use of this ontology.

1. **Semantic vs. Syntactic descriptions.** OWL-S differentiates between the semantic and syntactic aspects of the described entity. The *Profile* and *Process* ontologies allow for a semantic description of the Web service while the WSDL description encodes its syntactic aspects (such as the names of the operations and their parameters). The *Grounding* ontology provides a mapping between the semantic and the syntactic parts of a description facilitating flexible associations between them. For example, a certain semantic description can be mapped to several syntactic descriptions if the same semantic functionality is accessible in different ways. The other way around, a certain syntactic description can be mapped to different conceptual interpretations offering different views of the same service.

2. **Generic vs. Domain knowledge.** OWL-S offers a core set of primitives to specify any type of Web service. These descriptions can be enriched with domain knowledge specified in a separate domain ontology. This modelling choice allows using the core set of primitives across several domains.

3. **Modularity.** Another feature of OWL-S is the partitioning of the description over several concepts. The best demonstration for this is the way the different aspects of a description are partitioned in three concepts. As a result a *Service* instance will

relate to three instances each of them containing a particular aspect of the service. There are several advantages of this modular modelling. First, since the description is split up over several instances it is easy to reuse certain parts. For example, one can reuse the *Profile* description of a certain service. Second, service specification becomes flexible as it is possible to specify only the part that is relevant for the service (e.g., if it has no implementation one does not need the *ServiceModel* and the *ServiceGrounding*). Finally, any OWL-S description is easy to extend by specializing the OWL-S concepts.

2.4.2 Web Service Domain Ontologies

Externally defined knowledge plays a major role in each OWL-S description. OWL-S offers a generic framework to describe a service, but to make it truly useful, domain knowledge is required. For example, domain knowledge is used to define the type of functionality the service offers as well as the types of its parameters.

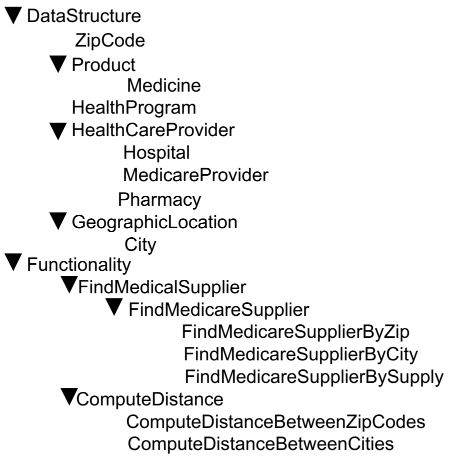


Figure 2.8: Web service domain ontology.

Figure 2.8 depicts the hierarchical structure of the domain ontology used to describe the example Web service. Note that it specifies a DataStructure hierarchy (consisting of the concepts discussed in subsection 2.2.1) and a Functionality hierarchy. The Functionality hierarchy contains a classification of service capabilities. Two generic classes of service capabilities are shown here, one for finding a medical supplier and the other for calculating distances between two locations. Each of these generic categories has more specialized capabilities either by restricting the type of the output parameters (e.g., find Medicare providers) or the input parameters (e.g., *ZipCode, City, SupplyType*).

The complexity of the reasoning tasks that can be performed with semantic Web service descriptions is conditioned by several factors. First, all Web services in a domain should use concepts from the same (or a small number of coupled) domain ontology in their descriptions. Otherwise the issue of ontology mapping has to be solved which is a difficult problem in itself. This requires that domain ontologies should be *broad* enough to provide the needed concepts by any Web service in a certain domain. Second, the richness of the available knowledge is crucial for performing complex reasoning.

Example scenario. By using the semantic Web service descriptions presented above, the example task can be generalized and automated. The right services needed to perform the task can be selected automatically from a collection of services. Semantic metadata allows a flexible selection that can retrieve services that partially match a request but are still potentially interesting. For example, a service which finds details of medical suppliers will be considered a match for a request for services that retrieve details of Medicare suppliers, if the used Web service domain ontology specifies that a *MedicareSupplier* is a type of *MedicalSupplier*. Note that this matchmaking is superior to the keyword based search offered by UDDI. The composition of several services in a more complex service can also be automated. Finally, after being discovered and composed based on their semantic descriptions, the services can be invoked to solve the task at hand.

The material of this chapter so far has briefly described and exemplified the technologies that should be understood for further following this thesis. In the next section we conclude on and explain a number of requirements that ontologies used for semantic Web service descriptions should fulfill. These requirements are then investigated throughout the thesis.

2.5 Requirements for Web Service Ontologies

The goal of this thesis is to enhance Web service ontologies and the process of building them. To achieve this goal we need to understand the requirements that these ontologies should fulfill. Unfortunately, little work was directed towards defining such requirements. We are only aware of work described in (Lara et al., 2003) and (Grosof et al., 2004). However, these papers only list a set of functional requirements (i.e., Web service activities that should be supported) and ignore overall, qualitative requirements that the ontologies should fulfill. We now state a set of such requirements based on our experience when working with semantic Web services. Note that these requirements aim to complement the set of requirements identified by the community so far.

2.5.1 Requirements for Generic Web Service Ontologies

The role of generic Web service ontologies is to capture the domain independent aspects of Web services. These ontologies should be richly axiomatized, heavyweight ontologies that would facilitate creating formal descriptions to be reasoned upon with automatic means. Generic ontologies are designed and maintained by a committee of experts which usually cooperate through virtual means. The user base of these ontologies is considerably larger than the committee which maintains them. The observations of the users that are brought to the knowledge of the committee are analyzed, and if it is the case, incorporated in the next versions of the standards. We consider that (at least) the following requirements[22] should be fulfilled by generic Web service ontologies:

[22]We thank John Domingue for his observations that lead to clarifying the descriptions of these requirements.

Modelling Expressiveness. Generic Web service ontologies are used to model a wide range of services from different domains. Therefore, a requirement for these ontologies is that they should scale to cover a large variety of heterogeneous services by allowing the modelling of several different types of services.

Clear Semantics and Rich Formalization. Semantic Web service descriptions aim to enable complex (reasoning-based) tasks involving multiple software agents. It is therefore important that the intended meaning of these descriptions can be interpreted in an unambiguous way by all agents. A prerequisite for this goal is that the semantics of the generic Web service ontologies is clearly and formally specified. Indeed, a rich formalization is crucial both for clearly expressing the semantics of the descriptions and for facilitating complex reasoning tasks with them.

Adaptability. Another important requirement is that models created using the generic ontologies should be easily reusable (adaptable) for other Web services. Even more, the generic knowledge captured by a generic Web service ontology should be reusable to describe other kinds of software entities as well. This would facilitate an easy transition from traditional software to Web services.

Harmonization with other standards. While several evolving generic Web service ontologies (e.g., OWL-S, WSMO, IRS-II) have the same objective they employ different ontology frameworks for specifying services. To this point there is no harmonization between these proposed standards meaning that services described according to different frameworks cannot be used together. Lack of harmonization also hampers performing any formal comparison between these different ontologies. We think that on a long term a harmonization of these proposed standards would be beneficial. If their maturity level does not allow for a harmonization, it should be attempted to make the proposed standards at least interoperable. Given the fact that semantic Web service technology aims to complement and enhance current technology, it is also important to be aware of the correspondence between generic Web service ontologies and other industrial Web service standards (e.g., WSDL, OASIS).

Usability. The success of a generic ontology is not only conditioned by its quality but also by its early adoption by the community. Several factors can contribute to this early adoption. For example, supporting material such as usage examples and modelling guidelines ensure that the ontology and its use are easy to understand. Further, tool support is important to facilitate the use of the ontology. Finally, maintainability of the created semantic models is another facet of usability. We acknowledge that there is a natural tension between the requirements of modelling expressiveness and minimal ontological commitment and usability. In order to be adopted by a wide community there will be pressure on any standard to be minimal. Additionally, to be useable the defined concepts and relations should not be over complex.

In the first part of this thesis we inspect OWL-S (which was the first initiative towards establishing a generic Web service ontology) from different perspectives and determine whether these requirements are met. We also offer several solutions for enhancing the problematic aspects that we identify.

2.5.2 Requirements for Web Service Domain Ontologies

Web service domain ontologies contain domain specific knowledge which is used to complete the generic descriptions. Usually, domain ontologies are lightweight specifying the important concepts and functionality types in a domain (note, however, that more complex applications might require more complex domain ontologies as it was experienced in the DIP project[23]). Unlike generic ontologies, domain ontologies employed in Web service descriptions are usually developed by a single domain expert. Often, the designer of the domain ontology also uses this ontology to describe his services. We identified the following requirements for Web service domain ontologies.

Broad domain coverage. A desirable feature for domain ontologies is a broad domain coverage. These ontologies should contain most of the concepts that describe a certain domain so that many Web services in that domain can be described according to the same ontology. However, this characteristic is hard to achieve due to several factors (all of these factors were observed in our case studies in Chapter 7). Note that the broad domain coverage requirement does not also require that the ontology should be heavyweight (though, a richly formalized ontology is always desirable). Indeed, an ontology containing a broad taxonomy of terms but no axioms (as is the case for many domain ontologies used for Web service descriptions) has a broad domain coverage but it is still a lightweight ontology.

Scalability. The number of existing Web services per domain is ever increasing. Several domains witnessed a rapid increase to several hundreds of services (e.g., over 1000 in bioinformatics (Lord et al., 2005)). With the wide adoption of this technology, Web service repositories are changing on a daily basis being enriched with several, often dissimilar services. This situation requires that domain ontologies should scale to cover large, dynamically changing collections of dissimilar services. Naturally, achieving this goal requires some level of automation as the ontology building process needs to be repeated regularly over large collections of services.

In the second part of this thesis we describe our work on semi-automatically acquiring Web service domain ontologies which aims to support domain engineers to build domain ontologies for large and dynamically changing Web service collections.

2.6 Summary

In this chapter we briefly described three novel Web technologies and showed how they enhance information access and integration. The Web made a vast amount of digital data available to search for everyone. However, current keyword based search techniques often fail to identify the right pages for a query. Also, they are limited to the information that is explicitly stated on static Web pages without being able to perform simple data processing. These limitations are addressed by two technologies. First, the formal description of the meaning of Web pages (created with Semantic Web technology) are machine processable and interpretable thus enhancing several search and data integration tasks compared to keyword search. Second, Web services make the processing service

[23]`http://dip.semanticweb.org/`

of computer programs dynamically accessible via Web interfaces. As a consequence, a more generic version of the same task can be performed. However, discovery and integration of these Web services still needs to be done manually. Semantic Web services overcome this limitation by semantically describing the capabilities of Web services. Generic task models can be created which can be automatically filled in at run time with the right Web services.

Based on our description of the semantic Web services technology, we identified a set of requirements that Web service ontologies should fulfill. These observations constitute an extended problem statement for our thesis since our goal is to enhance the quality and to facilitate the building process of Web service ontologies. In the first part of the thesis, we describe our efforts to enhance the OWL-S generic Web service ontology. Then, in the second part, we detail our work on automatic learning of Web service domain ontologies.

Chapter 3

Related Work

In this chapter we identify and briefly overview related work. Because the goal of this thesis is to facilitate building ontologies for Web services, work on acquiring software semantics in general, and Web service semantics in particular, is relevant. Nevertheless, none of these fields has been concerned with the acquisition of ontologies that describe software. In our work we employ two different ontology engineering methods. On one hand, we use methods relying on generic ontological principles to improve the quality of OWL-S. Such methods were successfully used in other contexts than that of software description. On the other hand, we adapt ontology learning to support quick acquisition of Web service domain ontologies.

3.1 Introduction

The acquisition of semantic markup is considered a fundamental problem in reaching the vision of the Semantic Web (Uschold, 2001; Hendler, 2001). Because semantic markup relies on using concepts provided by ontologies, a subproblem is that of acquiring ontologies in the first place. While ontologies have been developed for several decades now, the Semantic Web raises new challenges for developing them, such as larger scale and increased change rate (van Harmelen, 2002).

In this thesis we investigate the ontology acquisition problem in a special context, that of Web services. An obviously related body of work can be found in software engineering in the area of acquiring software semantics (see Section 3.2).

Several methods exist for dealing with ontology acquisition. First, methodologies exist that prescribe the major steps in the process of building ontologies. However, we do not aim at establishing such a methodology here and therefore we do not discuss this topic in this chapter. Second, a suit of methodologies relying on extensive research of fundamental philosophical, linguistic and mathematical principles have been used to enhance the quality of (heavyweight) ontologies. These methods can detect, explain and correct frequent mistakes and faults in ontologies by relying on rigorously studied ontological principles. We describe these methods in Section 3.3 . Finally, the ontology learning field aims to automatically derive (lightweight) ontologies from existing data sources. These methods have a high relevance for our third research question and they are discussed in Section 3.4. We point out a set of problematic issues within the ontology learning field in Section 3.4.4 . These issues are addressed in our work as described in the rest of the thesis.

3.2 Acquisition of Software Semantics

The use of semantic descriptions of software components was explored in software engineering to support tasks such as program understanding (Biggerstaff et al., 1993) or building software reuse libraries (Mili et al., 1998). However, the acquisition of semantic descriptions was always a critical stage and hampered the development of semantic based techniques. For example, as concluded by a major survey of software reuse libraries (Mili et al., 1998), even if many sophisticated approaches exist to build and exploit software reuse libraries, *"the practice is characterized by the use of ad-hoc, low-tech methods"* that were more practical to use than semantically sophisticated methods.

Several methods emerged that tried to acquire some of the semantics automatically. An approach for automatically building a software library is presented by (Maarek et al., 1991) and implemented in the GURU tool. The approach analyzes natural language documentation of software, which, according to the authors, is richer in functional information than source code or explanatory comments found in the code. The approach uses a combination of information retrieval and clustering methods. In a first stage, a *profile* is built for each analyzed document containing all the relevant terms that define the document. In a second stage, these profiles are clustered using an unsupervised hierarchical agglomerative clustering (HAC). The resulting hierarchy is a binary tree. The nodes in this hierarchy are not named because it is impossible to automatically predict the right name for them. The automatically built library proved to perform 15% better (in terms of precision) when compared to the InfoExplorer system, a commercial tool that relies on a lot of manually encoded knowledge about the same set of software collection. Later, this technique was combined with code analysis methods (Helm and Maarek, 1991). Finally, an attempt to build a term hierarchy is reported by (Mili et al., 1997). They use IR techniques limited to the noun phrases in the corpus and then rely on the co-occurrence of terms in some documents to deduce semantic relations between them.

Note that none of these approaches actually builds taxonomic structures. It is remarkable though that the derived semantics are evaluated in the context of a concrete application – a topic which is currently under development in the ontology learning field (see Chapter 8).

A subtopic of acquiring semantics for software is the **acquisition of Web service semantics**. Indeed, the creation of semantic Web service descriptions is a time consuming and complex task whose automation is desirable, as signaled by many researchers in this field, for example (Wroe et al., 2004). This task can be broken down in two smaller tasks. First, acquiring a suitable Web service domain ontology is a prerequisite when creating semantic Web service descriptions. This is required to create a shared terminology for the semantic service descriptions. Second, the actual process of annotating Web services with the concepts provided by the domain ontology (i.e., creating their semantic descriptions) has to be performed.

To our knowledge, two research teams concentrate on the second task, that of Web service annotation. Hess and Kushmerick (Hess and Kushmerick, 2003, 2004; Hess et al., 2004) employ Naive Bayes and SVM machine learning methods to classify WSDL files (or Web forms) in manually defined task hierarchies. Patil et al. (Patil et al., 2004) employ graph similarity techniques to select a relevant domain ontology for a WSDL file from a collection of ontologies. Then they annotate the elements of the WSDL file with the concepts of the selected ontology. Both teams use existing domain ontologies and acknowledge that their work was hampered by the lack of such ontologies. The work presented in the second part of this thesis is complementary, since we address the acquisition of Web service domain ontologies.

3.3 Formal Ontology based Methods

To be successfully used within information systems, ontologies should provide clear semantics for their concepts and a rich formalization of these semantics. These requirements are particulary important in the area of semantic Web services whose ontological descriptions should be used for reasoning tasks. However, few existing ontologies fulfill these requirements. In this section we overview work which shows that highly general ontological notions can be successfully employed to enhance the quality of ontologies.

OntoClean (Guarino and Welty, 2004) is a methodology for validating taxonomic relationships and relies on the generic ontological notions of essence, identity and unity. Using OntoClean it is possible to detect and correct common modelling mistakes such as the confusion of subsumption with instantiation, or the confusion of subsumption and meronymy (partOf) relations.

More generically, one can rely on a more complex set of generic principles to enhance existing ontologies. Several projects have proved that it is beneficial to align existing ontologies to foundational ontologies, such as DOLCE, that capture generically valid ontological notions drawn from philosophy, linguistics and mathematics. For example, in (Gangemi et al., 2002) the authors describe the use of the DOLCE ontology and the OntoClean methodology in assessing the quality of WordNet's top level category of nouns. They perform a *direct mapping* of WordNet concepts to concepts in DOLCE and they apply the rigidity properties introduced by OntoClean. Using these methods several problematic issues of WordNet were revealed. Confusion was detected between several different ontological elements: between concepts and individuals, between object and meta level entities (notably, the Abstraction_1 synset includes both object level concepts - *Set*, *Time*, *Space* - and meta-level concepts - *Attribute*, *Relation*). They also discovered a mixture in the level of generality of sibling synsets. The mapping resulted in making several concrete suggestions to enhance WordNet. Another example of using alignment, is provided in (Borgo and Leitao, 2004) where DOLCE is used in the process of building a manufacturing ontology.

An alignment of software related ontologies has not been performed. We pioneered this line of work by performing an alignment of OWL-S to DOLCE. This alignment allowed us to validate and improve the quality of OWL-S (see Chapter 6).

3.4 Ontology Learning

The aim of ontology learning methods is to (semi-)automatically derive conceptual structures from different types of unstructured, semi-structured and fully structured data. Work that fits into this category was performed in numerous disciplines - computational linguistics, machine learning, databases, software engineering (Maedche and Staab, 2001). However, the term of ontology learning has been only recently coined as a response to the growing need to automate ontology acquisition in the context of the Semantic Web research.

This multidisciplinary field encompasses a variety of approaches. We are aware of at least two extensive overviews of existing approaches and tools for ontology learning (Maedche and Staab, 2001), (Gomez-Perez and Manzano-Mancho, 2003). Therefore, in this chapter we only provide the background necessary to motivate our choices and position our work in the landscape of this research field.

3.4.1 Ontology Learning Approaches

An important characteristic of ontology learning approaches is the input data that they rely on. This has a strong influence on the ontology learning method that is used. We shortly describe ontology learning approaches grouped based on the data structures that they use. For this we rely on the classification provided by (Gomez-Perez and Manzano-Mancho, 2003) and extend it with work published after the survey was finalized.

The majority of existing approaches learn an ontology from *textual sources*. A sub-category of text based approaches are approaches that process machine readable dictionaries (MRD's) to derive some conceptual structures (Jannink and Wiederhold, 1999), (Rigau, 1998). A recent trend in ontology learning is to leverage on the vast amount of (redundant) data available on the Web. Pioneer in this area is the work of Cimiano who learns subsumption hierarchies (and also instanceOf relations) by querying the data available on the Web (Cimiano et al., 2004a,c).

Besides this large category of text based methods, there are a few methods developed for learning ontologies from knowledge bases (Suryanto and Compton, 2001), semi-structured schemata (Delteil et al., 2001) and relational schemata (Rubin et al., 2002).

When deciding what data sources were appropriate for learning Web service ontologies we considered several possibilities. First, resources connected to the underlying implementation of the service might provide useful knowledge about the functionality of the Web service since Web services are simply Web accessible software products. Such are the source code, its textual documentation or existing UML diagrams. Second, one could use Web service specific data sources such as associated WSDL files or activity logs. One major criterion when choosing a data source as basis for our ontology learning method was its availability. We observed that Web services are almost always accompanied by a short textual description of their functionality which helps a user to quickly understand what the service does. Such descriptions exist in on-line Web service repositories such as XMethods[1]. Similarly, javadoc style method descriptions also offer such short descriptions of functionality (software API documentation being more frequently available than source code). Therefore, we experimented with **textual** Web service functionality descriptions attached to Web services as a whole or extracted from the underlying software documentation (as detailed in Chapters 7 and 8).

[1]`http://www.xmethods.net/`

3.4.2 Methods for Text Based Ontology Learning

A variety of methods have been developed for learning ontologies from textual sources. Most significantly, these methods rely on pattern based techniques, use association rules (Maedche and Staab., 2000) or employ clustering methods (Faure and Nedellec, 1998). The approach we adopted for learning Web service domain ontologies is motivated by the observations that textual sources attached to Web services (1) contain valuable information for building ontologies and that (2) they use natural language in a specific way. In fact, such texts belong to what is defined as a sublanguage in (Grishman and Kittredge, 1986) - a specialized form of natural language (see Chapter 7 for details). Given these characteristics of our texts we chose to rely on pattern based techniques in our ontology learning.

Pattern based techniques are widely used in several natural language processing applications. Notably they have been used to derive semantic relations from large corpora. A pioneer in this direction of research was the work of Hearst which introduced the idea of learning hyponymy relations using lexico-syntactic patterns (Hearst, 1992). Lexico-syntactic patterns are defined on both lexical and basic syntactic information (POS tags). As such, they allow extracting relations after shallow text processing only. For example, the hyponymy relationship suggested by *Bruises, wounds, broken bones or other injuries* could be extracted using the *NP, NP*, or other NP* pattern (Hearst, 1992). As a follow up of this work, Charniak developed a set of lexico-syntactic patterns that identify meronymy (partOf) relations (Berland and Charniak, 1999). In both cases, the identified semantic relations were used to enlarge WordNet.

Naturally, such patterns have a clear relevance for ontology learning. Indeed, Hearst-style patterns are used in the work of Cimiano (Cimiano et al., 2004c) and in the CAMELEON tool which incorporates over 150 generic patterns for the French language (Seguela and Aussenac-Gilles, 1999; Aussenac-Gilles, 2005). While such generic patterns work well in general corpora they often fail in small or domain specific corpora. In these cases domain-tailored patterns provide a better performance (Aussenac-Gilles, 2005). Besides using domain tailored patterns, one can also enlarge the extraction corpora. For example, World Wide Web data can be used for pattern based learning (Cimiano et al., 2004a). In our work and in several ontology learning approaches pattern based extraction is just a first step in a more complex process (Reinberger et al., 2004; Faure and Nedellec, 1998; Buitelaar et al., 2004b). In these cases patterns identify potentially interesting terms in the corpus and then the next processing steps derive relevant semantic structures from these terms.

3.4.3 Ontology Learning Tools

A major survey on ontology learning, performed before our research started, identified 18 tools for learning ontologies from textual sources (Gomez-Perez and Manzano-Mancho, 2003). These tools implement a variety of methods such as clustering techniques (ASIUM (Faure and Nedellec, 1998), the Mo'K Workbench (Bisson et al., 2000)) or pattern based techniques (CAMELEON (Seguela and Aussenac-Gilles, 1999)). Since our goal was to reuse existing ontology learning methods in the context of Web services, we wanted to experiment with these tools. However, only 3 of these 18 tools were available for download at the beginning of our research. The most useful was TextToOnto.

TextToOnto[2] (Maedche and Staab, 2004) is an ontology learning workbench offering a suite of algorithms that can be combined to build an ontology. In particular, the workbench offers modules for identifying important concepts, learning taxonomic relations and extracting generic binary relations using association rules. The learned ontology is visually presented using a TouchGraph[3] based schema visualization technique. By using TextToOnto we were able to experiment with several basic algorithms. However, the particularities of our corpus lead to a suboptimal performance of the implemented methods. One of the major problems was that our corpus employed a sublanguage specific to software documentation which is different from generic texts. This finding prompted us to implement our own, modified versions of these algorithms. The source code of TextToOnto, as well as support from its developers, provided valuable help in starting up our research.

During our research, two other tools were developed. Unlike their predecessors, these tools aim to make ontology learning easier to use and to adapt for novice users.

OntoLT[4] (Buitelaar et al., 2004b) is a Protege[5] (Noy et al., 2001) plugin that allows quick extraction of domain ontologies through a set of mapping rules defined in terms of XPATH queries on XML-based linguistic annotations. Currently the tool comes with two readily defined mappings. The first one allows learning taxonomic relations based on noun phrase information. The second mapping extracts slots. According to this mapping a predicate becomes a slot having as domain its subject and as range its object. Other mappings can be built depending on the needs of the user. The extracted ontology is presented in the Protege environment. Each learned structure (concept or slot) contains detailed information regarding its provenance (i.e., the parts of the corpus where it was identified and the rule which led to its identification).

Text2Onto[6] (Cimiano and Voelker, 2005) is the successor of TextToOnto offering an extended set of ontology learning algorithms. Text2Onto brings several enhancements to its predecessor. First, it relies on a *Probabilistic Ontology Model (POM)* which allows combining the results of different processing algorithms and representing probabilities for the learned structures. Second, based on this probabilistic model, an improved user interface is built. Learned constructs are listed ordered by their probability of being correct or represented using a graph-based visualisation component. Another advantage of the POM is the traceability of the extraction: each learned structure is stored with a set of pointers to parts of the corpus from where it was extracted. Finally, Text2Onto implements a *data driven change discovery* mechanism which allows updating the ontology model based on the changes in the underlying data without having to perform the whole extraction from scratch. This grants efficiency and allows the user to trace the evolution of the ontology based on the changes in the underlying data.

3.4.4 Major Issues in Ontology Learning

With the growth of the Semantic Web, ontologies are needed in various domains. This increases the demand for ontology learning solutions to be used in new, different domains.

[2]`http://sourceforge.net/projects/texttoonto/`
[3]`http://www.touchgraph.com/`
[4]`http://olp.dfki.de/OntoLT/OntoLT.htm`
[5]`http://protege.stanford.edu/`
[6]`http://ontoware.org/projects/text2onto/`

For example, the context of Web services is a new application domain for ontology learning. These specialized domain have several characteristics:

Corpora with special characteristics. The corpora that describe the knowledge available in a specialized domain often exhibit rather strong characteristics. Most often, they use natural language in a specific way, a sublanguage. This is also the case in the domain of Web services, see Subsection 7.3.2.

Clear requirements for the output of the learning method. In a restricted domain it is often clear what kind of knowledge is needed, what should be extracted from the corpus. This knowledge is sometimes more specialized than the semantic structure extracted by generic algorithms. For example, in the case of Web services, we know that not only domain concepts but also functionality hierarchies need to be learned (see Subsection 7.3.3). This is a novelty because existing algorithms do not consider verbs (which usually denote functionality) as candidates for concepts.

Non NLP-expert developers and users. Finally, domain experts which use the ontology learning tools or which need to develop them for a domain are seldom experts in ontology engineering or in NLP.

These characteristics broaden the field of research on ontology learning with topics such as:

Adaptability. The special characteristics of the analyzed corpora in restricted domains, as well as the clear requirements for the knowledge to be extracted, require the adaptation of existing ontology learning methods. This need for adaptation is already addressed by OntoLT which offers a generic framework for defining pattern based extraction rules on specialized domains. We believe that, in order to meet the requirements of non expert ontology learning solution developers, not only toolkits of easily adaptable and combinable methods are needed but also guidelines of how to implement a domain tailored ontology learning solution. We discuss how we adapted ontology learning techniques to the context of Web services in Chapter 7.

Evaluation. Evaluation remains an important issue for ontology learning in general. Given the fact that different methodologies are developed it is hard to provide joint evaluation metrics or benchmarks. This is even more problematic in the case of specialized domains where no corpora are collected, it is not clear which Gold Standards should be used and there is no baseline for comparison. An overview of existing evaluation practices which served as a basis for developing our own evaluation scheme is described in Chapter 8.

Usability. Ontology learning is essentially a knowledge transfer processes from domain knowledge as perceived by humans to formal models that can be reasoned upon by computers. Ontology learning methods involve multiple stages. Topically, they extract *terms* from a *document* corpus, then combine and transform them into *ontology elements* (concepts). It is therefore essential, both during the development and the use of a ontology learning tool, that the knowledge structures involved in these processes as well as their relations are easily understood by human users. Therefore, it is important to enhance the usability of ontology learning tools. We briefly present our efforts in this direction in Chapter 7.

3.5 Summary

In this chapter we provided a brief overview of research fields that are relevant for our work.

In the field of software engineering, we described efforts for automatically acquiring some semantic description of software components. However, to our knowledge, no efforts have addressed building an ontology about a set of software components. Automating such a task is especially important in the context of semantic Web services where domain ontologies play a major role.

In the field of ontological engineering we looked both at methods based on ontological principles which aim to enhance the quality of existing ontologies, and to ontology learning methods that aim to bootstrap ontology learning by automatically learning lightweight ontologies. The use of ontological methods has a direct applicability for enhancing the quality of generic Web service ontologies. Indeed, these methods have not been applied yet in the domain of software descriptions. Ontology learning can be employed to support the building of domain ontologies. For this, ontology learning methods have to be adapted to the context of Web services as discussed in the second part of the thesis.

Part II

Enhancing Generic Web Service Ontologies

Chapter 4

Improving DAML-S

In Chapter 2 we stated that the wide scale adoption of a generic Web service ontology is conditioned by its ***expressiveness*** (i.e., that it allows modelling a wide range of services) and ***usability*** (i.e., modelling guidelines and examples should be provided for learning the ontology). In this chapter we investigate DAML-S from these two perspectives by reporting on our experiences when modelling a set of increasingly complex services. Our conclusion is that DAML-S has both a low expressiveness and usability. The contribution of our work is to enhance both aspects of DAML-S. First, we identify a set of modelling limitations and offer solutions for eliminating them (thus extending the expressiveness of the ontology). Second, we offer a set of modelling examples and guidelines to help the take-up of the ontology by the community.

The material presented in this chapter was published as part of a paper in the workshop on E-Services and the Semantic Web held in conjunction with the Twelfth International World Wide Web Conference (Sabou et al., 2003) and a paper at the Second International Semantic Web Conference (Richards and Sabou, 2003).

4.1 Introduction

A generic Web service ontology should be used to describe any Web services, independently of their domain. Two factors are key to the wide-scale adoption of such an ontology. First, it should be *expressive* enough to allow the modelling of a wide range of services. Second, examples of marked up services and modelling guidelines should exist to support the easy learning of this ontology.

At the start of our research, DAML-S was weakly documented (low *usability*). Indeed, even though DAML-S was growing into a *de facto* standard for semantic Web service markup, we only found few complete service descriptions and even less papers discussing technical issues about the markup process. Our preliminary literature study yielded four types of reported usage of DAML-S. First, within the DAML-S coalition two complete, fictitious examples (on the DAML-S site) and two concrete applications (Paolucci et al., 2002) were provided. Second, several projects used only certain parts of the DAML-S ontology, e.g., matchmaking research focused on the Profile ontology (Cardoso and Sheth, 2002; Somacher et al., 2002). Third, we found work which extended parts of DAML-S: (Brison et al., 2002) enriched the Process ontology, (Wroe et al., 2003) extended the Profile ontology with bio-informatics related properties, (Lopes et al., 2002) extended the specification of conditions. Finally, some papers mentioned the use of complete DAML-S as is, but gave no details about their experiences. Common to all the above referenced papers is that none of them described the process of writing the DAML-S markup. We were also concerned that very little of the DAML-S markup we found pointed to actual services. In this conditions, it was a considerable effort to get started with using DAML-S since example markup and guidelines about modelling practices were almost completely absent.

We sought to fill this gap by providing a set of complete, real Web service descriptions and sharing our modelling experiences. The result of this exercise was twofold. First, we concluded that DAML-S[1] had a limited **expressiveness** as we encountered difficulties in modelling even simple Web services. Our observations (and proposed solutions) were brought to the knowledge of the DAML-S committee and partly implemented. Second, our marked up services[2] as well as considerations about their markup were made accessible to the research community in order to ease the adoption of this new ontology.

We start this chapter by describing the application which provided the set of example services (Section 4.2). Then we describe our modelling experiences with three increasingly complex services in Sections 4.3 to 4.5. Finally, we conclude in Section 4.6 by summarizing and discussing a set of emerging, noteworthy issues.

4.2 The Web Services

We described a set of services developed in the context the SW@VU[3] project that uses Semantic Web technology to build a portal of scientific publications from BibTex files. Table 4.1 summarizes these services and Figure 4.1 depicts a typical invocation pattern.

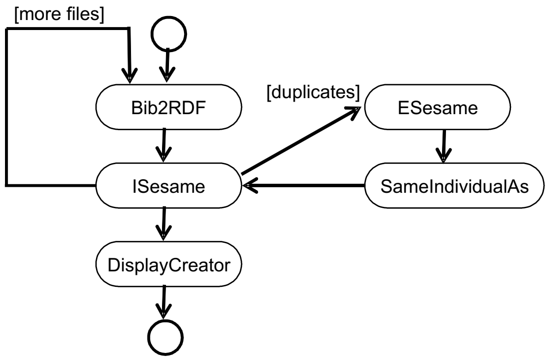


Figure 4.1: Web services used for building bibliography portals and their invocation workflow.

The main invocation steps of these services are:

[1]Versions v0.7 and v0.9.

[2]All services are available at `http://www.cs.vu.nl/~marta/services`.

[3]`http://www.cs.vu.nl/~mcaklein/SW@VU/`

Step 1: Data Translation. First, each available BibTex file is converted to RDF using the *Bib2Rdf* service, then saved in a web-accessible RDF(S) repository and query engine, *Sesame*[4] (Broekstra et al., 2002), by the *ISESAME* service.

Step 2: Duplicate Detection. The merger of all available data often results in redundancies as different owners of the bibliographies use syntactically different resources to denote the same author. We used `daml:sameIndividualAs` to encode these redundancies and extended Sesame's reasoning capabilities to interpret this new tag. The *SIA(SameIndividualAs)* service provides automated support for the task of finding the resources referring to the same person. Using machine learning techniques on the RDF(S) source, SIA extracts the resources which might refer to the same person and returns tuples of similar resources. Therefore the second step is to extract all the data from Sesame with the *ESESAME* service, send it to SIA, obtain the redundancy file and save it back to Sesame using again the *ISESAME* service.

Step 3: Display Creation. Finally, a portal creator software creates the portals of publications by querying Sesame.

Web Service	Functionality
Bib2Rdf (B2R)	A file conversion service that takes a file in BibTEX format and outputs a file in RDF format.
ISESAME	A file import service that takes a file in RDF format and adds it to a specified public or private repository in SESAME.
ESESAME	A file export service that extracts data from a specified public or private repository in SESAME and outputs the data in RDF, n3 or ntriples formats.
SameIndividualAs (SIA)	A utility service that reads in an RDF file and adds the $daml : sameIndividualAs$ tag to duplicate names.
PortalCreator (AI-DIS)	A service which takes the contents of a SESAME repository and displays it in a Web portal.

Table 4.1: Overview of the used services and their functionality.

These Web services can be combined differently depending on the available input data and the user's requirements. For example, if the data files are already provided as RDF there is no need to perform the Bib2Rdf translation. Or, if they are in different formats a corresponding translator should be invoked. Further, if the data originates from a single data source the duplicate detection step is not needed anymore. Or, if the data is already available in a Sesame repository no storage should be performed.

Semantic Web service technology aims at automating such adaptive service composition by using reasoning over service and requirements descriptions. We wished to use

[4]`http://www.openrdf.org/`

this technology in order to automate the portal creation task. For this we semantically annotated our Web services using DAML-S. These descriptions were reasoned upon by the Agent Factory design service (Brazier and Wijngaards, 2001) in order to prescribe an optimal composition depending on the input and output requirements. For details on the use of the Agent Factory for this task, the interested reader is referred to (Richards et al., 2003) and (van Splunter et al., 2003). We further report on our experiences with creating the DAML-S markup for three of these services: Bib2Rdf, SIA and Sesame.

4.3 Modelling a Simple Service - Bib2Rdf

The first service that we described using DAML-S was Bib2Rdf[5]. Bib2Rdf is a simple service: it transforms a BibTex file into a RDF representation. The service takes as input the URL of a BibTex file and returns the RDF encoding of this data. Despite the simplicity of this service, modelling it using DAML-S lead us to discover a severe inconsistency between the *ServiceProfile* and the *ServiceModel* parts of the description.

In Section 2.4.1 we described the mapping between the parameters of the *Service-Profile* and the *ServiceModel*. Several aspects of this mapping can be critiqued. First, one would expect that *Profile* parameters of a certain type can only refer to *Process* parameters of the same type. However, this is not enforced. With the current specification one can easily make a link between parameters of different types (e.g., *profile:input* and *process:output*). Second, because the *Process* ontology does not model *Preconditions* and *Effects* as subproperties of *process:paramater*, it is inconsistent to use entities of this type as values for *profile:refersTo*. Therefore, *Preconditions* and *Effects* defined in the *Profile* cannot refer to the corresponding *Preconditions* and *Effects* of the *Process* (since they are not included in the range of *refersTo*).

The DAML-S coalition acknowledges the possibility of inconsistencies between *Profile* and *Process* and that they will be discovered at some point (DAML Services Coalition, 2002). Since matchmaking is based on the *Profile* description, the break may occur during (attempted) usage of the service. The rationale for this design decision is not clear. We conclude that this link between the IOPE's at two distinct levels of specification should be corrected and made more explicit.

Solution. A solution to this issue (included in OWL-S v.1.0 and OWL-S v.1.1.B)[6] was developed during the author's participation in the coalition's work. The new solution offers a simplified modelling (see a schematic representation in Figure 4.2):

The schema for describing IOPEs is only offered by the Process ontology. The Profile ontology references these IOPE definitions through a set of properties. The rationale behind this modelling is that (1) both the Profile and Process parts of the description refer to the same *Input*, *(Conditional)Output*, *Precondition* and *(Conditional)Effect*[7] instances and that (2) the set of IOPEs declared in the Profile is always a subset of those declared in the Process. Through this solution we avoid duplicating the definition of IOPE instances in both models. Also, since both models will point to the same instances of IOPEs the problem of correctly mapping between a set of Profile IOPEs and a set of Process IOPEs is circumvented.

[5]`http://www.cs.vu.nl/~marta/services/Bib2Rdf/Bib2RdfService.daml`

[6]More recent releases of OWL-S are still based on this solution but extended it with several new notions.

[7]Conditional means that the existence of the output/effect depends on a condition being fulfilled.

The Process ontology models IOPEs as concepts rather than properties. Both *Profile* and *Process* instances refer to IOPEs through a set of properties. Note that the unintuitive initial modelling where IOPEs from the *Profile* were linked to IOPEs in the *Process* through a property (*profile:refersTo*) that ranged over properties (the IOPEs in the Process were properties) is now eliminated. The conceptual difference between *Inputs* and *Outputs* on one hand and *Preconditions* and *Effects* on the other was maintained in the current modelling as well. This distinction results from two different views that one can have about the functionality of a service. On one hand, the functionality of a service can be regarded as an information transformation process which is defined by *Input* and *Output* parameters. On the other hand, to describe the world state change that is provoked by the execution of a service it is useful to define *Preconditions* (world states that have to be fulfilled before execution) and *Effects* (world states after the execution of the service). Following this rationale, *Inputs* and *Outputs* are grouped under the *Parameter* concept.

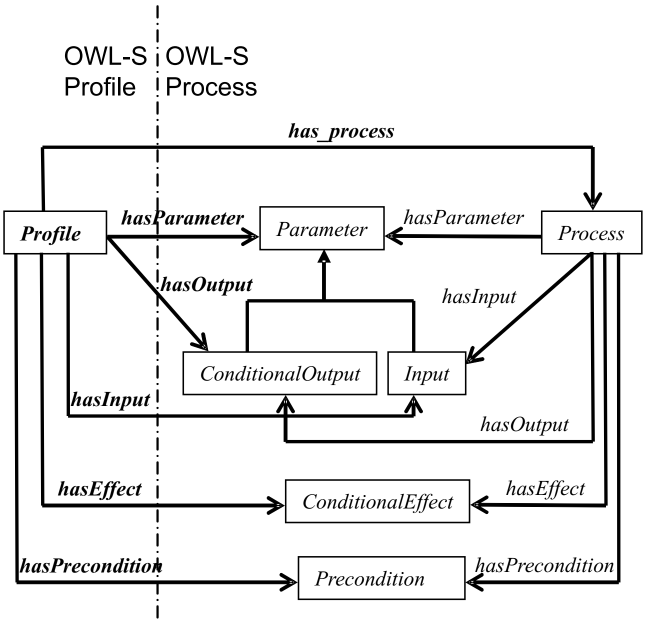


Figure 4.2: The new way to model the Profile to Process bridge.

4.4 Modelling a Service with Multiple Interfaces - SIA

The SIA (SameIndividualAs) service is essentially as simple as the Bib2Rdf service: it acts upon an RDF file, determines resources that possibly point to the same physical person and returns an RDF file with tuples of equivalent resources. The only element of complexity is that this service can acquire the RDF source in multiple ways: (1) by reading it from a URL, (2) by accepting the data itself as a string and (3) by reading the data from a Sesame repository, given the name of the repository and the log-in information.

Intuitively, this situation is similar to a specific kind of **ad-hoc polymorphism**, that of *overloading*: a certain method allows different signatures, but essentially it executes the same function. Indeed, the SIA service performs its function (duplicate detection) independently of the set of parameters with which it is invoked.

The available documentation about DAML-S provided little guidance on how to model this situation. Some coalition members have considered the problem of supporting multiple interfaces and offered a solution for the situation where the number of arguments are the same and in the same order but where the data types may differ, i.e. type polymorphism (Ankolenkar et al., 2002). The offered solution is to define a higher level concept that covers all possible alternative data types. However, this approach does not address our problem where we have a different number of arguments.

One simple solution to our problem would be to treat each alternative interface as a separate service. We rejected this solution for conceptual, practical and reuse reasons. At a conceptual level, we are describing one and the same service. If we made them separate services, DAML-S did not provide any way of identifying that they were in fact the same service. Knowing that a service and/or its description is related or in fact identical may be important when it comes to choosing services. From a practical point of view, marking up a service is time-consuming enough without having to perform the activity for every possible interface. From a reuse point of view, we wanted to share and reuse as much as possible between these alternative ways of accessing the service.

In order to provide the semantics that would allow more intelligent matchmaking and to handle interfaces with different data types and number of parameters, we tried a number of alternatives. The first variant (**SIA1**) was based on a **top-down design** starting with a service model and working down to the service grounding. Due to the problems we faced in SIA1, the second variant (**SIA2**) used a **bottom-up approach** starting with the WSDL definition. SIA2 clarified the DAML-S view of a service as being primarily defined by its IOs, rather than its effects. We developed **SIA3** to support the new view of our **service as a composite process** involving data readers and translators, rather than an atomic process. However, these first three descriptions were not valid solutions as they were either conceptually wrong or impossible to specify using DAML-S and WSDL. Our **final design** (**SIA4**) was a compromise that provided a valid solution but which did not completely represent our conceptual model of the service. It also involved significant repetition of descriptions. These variants are presented next in more detail, along with discussion of our rationale, choices and conclusions during our design. A schematic description of each variant is provided to clarify the discussion.

4.4.1 Top-Down Design of SIA1

Because the effect of the service is not altered by the way in which the RDF file is provided, our first intuition was to model a single service and to make the differentiation between the three ways of accessing it at the grounding level. We adopted this modelling in **SIA1**[8]. At the Profile (*Pr1*) / Process (*P1*) level we described the service as accepting an *RdfFile* and producing another *RdfFile*. The WSDL representation of the service consists of a port with three operations (*op1, op2, op3*) which differ through their input

[8]`http://www.cs.vu.nl/~marta/services/sia/Sia1Service.daml`

messages. The first expects a *url*, the second a *string* and the third receives four Sesame related parameters (*server url, password, login, repository name*).

```
*Service SIA1:
 *Profile:Pr1 (hasProc=P1)(I(RdfFile), O(RdfFile))
 *ProcessModel:
    AtomicProcess:P1(I(RdfFile),O(RdfFile))
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (P1->op1)
    WsdlAtomicProcessGrounding: Gr2 (P1->op2)
    WsdlAtomicProcessGrounding: Gr3 (P1->op3)   !!!
 *WSDL:
  Service(PortType:SIA(
    op1 (IMsg(url), OMsg(stream))
    op2 (IMsg(string), OMsg(stream))
    op3 (IMsg(url,pse,ln,rep), OMsg(stream))))
```

Specifying the first two groundings of *P1* to *op1* and *op2* was easy. However, for the third grounding we realized that it was impossible to build a one-to-one mapping between the single input of process *P1* and the four parameters of the WSDL operation, *op3*, because the DAML-S/WSDL mapping states that "*each atomic process input and output corresponds to a WSDL message part*". The DAML-S coalition acknowledges that this could be a possible limitation, but do not give a concrete example of a problematic scenario. This example shows that the assumption prohibits modelling of ad-hoc polymorphism. Therefore we encourage its revision.

Partial Solution. The new version of DAML-S (v.0.9) is less restrictive, but nevertheless does not solve this issue. There is a possibility of defining XSLT transformations between the *Process* parameters and the WSDL message parts. In case of inputs, multiple input parameters can be mapped to a single WSDL message part. Still, we cannot solve our particular problem where we need to map one input parameter *(RdfFile)* to multiple message parts (*url, pse, ln, rep*).

4.4.2 Bottom-Up Modelling of SIA2

After SIA1, we changed our strategy to bottom-up modelling in **SIA2**[9]. Based on the structure of the WSDL file, we modelled two *AtomicProcesses*: one with a single input (*P1*), just like before, and one with four parameters (*P2*) needed when a Sesame repository is used as the data source. With this model we excluded any grounding problem, but new issues emerged.

The first issue relates to whether the distribution of these two Processes should be within one or two *ServiceModel* instances. Previously we decided to model a single service. A service can have at most one *ServiceModel* (maxCardinality(`describedBy`)=1). Therefore, both our processes have to be part of a single *ServiceModel*. However, the *ServiceModel* can only accommodate a single *Process* (of type *Atomic, Simple* or *Composite*) because the `process:has_process` property has an exact cardinality of 1. This requires the combination of the two atomic processes into a composite one. The `process:Choice` control construct is closest to our needs: it expresses that only one process can be chosen for execution. We modelled a composite process(*CP*) as a *Choice* between the two atomic processes and included it in a single *ServiceModel* entity.

[9]`http://www.cs.vu.nl/~marta/services/sia/Sia2Service.daml`

The second issue relates to the *Profile*. Keeping the *Profile* as in the previous example is technically correct: we can link the input and the output of the Profile to the IO's of the *Process P1*, while the parameters of *P2* are not referenced from the *Profile*. However, this means that when advertised, the service does not expose its ability to read data directly from Sesame. Adding the other four parameters to the profile is not a solution either because we cannot specify how these parameters relate. This would be misleading at matchmaking. We decided to create two *Profile* instances: *Pr1* generically describes the IO's and maps to *P1*, while *Pr2* reflects the parameters of *P2*. The service has three groundings, two for the generic *Process P1* and one for the Sesame *Process P2*.

```
*Service SIA2:
 *Profile:Pr1 (hasProc=P1)(I(RdfFile),O(RdfFile))
 *Profile:Pr2 (hasProc=P2)(I(server),I(url),I(psw),
                           I(ln),O(RdfFile))
 *ProcessModel:
    CompositeProcess: CP:Choice
    {   AtomicProcess:P1(I(RdfFile),O(RdfFile))
        AtomicProcess:P2(I(server),I(url),I(psw),
                        I(ln),O(RdfFile))}
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (P1->op1)
    WsdlAtomicProcessGrounding: Gr2 (P1->op2)
    WsdlAtomicProcessGrounding: Gr3 (P2->op3)
 *WSDL: same as SIA1
```

This design was based on the constraints imposed by the DAML-S model. First, there should be a one-to-one correspondence between the IO(PE)'s of all modelling levels - *Profile*, *Process*, *Grounding* - even if directly not imposed but for the sake of consistency. Second, what defines a service is not its effect but its signature. Therefore our approach for SIA1 (and *P1* for SIA2) was conceptually inconsistent with the DAML-S view: we should provide a different Profile for each of the different ways to access the service. These Profiles provide the meaning of the IOPE's.

With this new view of what it means to define a service we considered another model based on the observation that our service is a combination of four processes: a translator process and three different data acquisition processes.

4.4.3 Composite Process Approach in SIA3

We modelled the **SIA3**[10] as having a complex *ProcessModel*. We consider that the service offers three *CompositeProcesses* (*CP1*, *CP2* and *CP3*) combined in a global *CompositeProcess* (*CP*). Each of these processes is a sequence of a *DataReader* (*DR*) and the *Translator* (*T1*) process itself. The service has three *Profiles* (*Pr1, Pr2, Pr3*) each describing one of the three composite processes. The same WSDL file can be used.

```
Service SIA3:
 *Profile:Pr1 (hasProc=CP1)(I(url),O(RdfFile))
 *Profile:Pr2 (hasProc=CP2)(I(RdfStream),O(RdfFile))
 *Profile:Pr3 (hasProc=CP3)(I(server),I(url),I(psw),
                            I(ln),O(RdfFile))
 *ProcessModel
    CompositeProcess: CP:Choice
```

[10]`http://www.cs.vu.nl/~marta/services/sia/Sia3Service.daml`

```
    {
     CompositeProcess:CP1: Sequence
        {
         AtomicProcess:DR1(I(url),O(RdfFile))
         AtomicProcess:T1(I(RdfFile),O(RdfFile))}
     CompositeProcess:CP2: Sequence
        {
         AtomicProcess:DR2(I(RdfStream),O(RdfFile))
         AtomicProcess:T1(I(RdfFile),O(RdfFile))}
     CompositeProcess:CP3: Sequence
        {
         AtomicProcess:DR3(I(server),I(url),I(psw),
                           I(ln),O(RdfFile))
         AtomicProcess:T1(I(RdfFile),O(RdfFile))}
     }
*WSDLGrounding:
   WsdlAtomicProcessGrounding: Gr1 (CP1->op1)   !!!
   WsdlAtomicProcessGrounding: Gr2 (CP2->op2)   !!!
   WsdlAtomicProcessGrounding: Gr3 (CP3->op3)   !!!
*WSDL: same as SIA1
```

We encountered a new grounding problem. Conceptually each composite process corresponds to a WSDL operation: the inputs of the *DataReader* are the same as the inputs of the WSDL operation, and the output of the *Translator* corresponds to the output of the WSDL operation. However, the first assumption of the existing mapping states that "*a single atomic process corresponds to a single WSDL operation*". Therefore we cannot perform the mapping of a composite process to a WSDL operation. The DAML-S coalition acknowledges that this could be a possible limitation, but they are skeptic about "*the importance of relaxing this assumption*" (DAML Services Coalition, 2002). This example shows that the assumption prohibits modelling of a complex internal structure if it is not directly reflected in the Web interface of the service. We see this as a serious limitation to conceptual modelling.

Alternative solution. Members of the DAML-S coalition indicated an alternative solution to the issue of mapping composite processes to WSDL operations. This mapping can be done by using the *SimpleProcess* concept which mediates between a *CompositeProcess* to an *AtomicProcess*. More precisely, given a *CompositeProcess*, it has to be correlated with a *SimpleProcess* instance using the *collapsesTo* property. Then, the *SimpleProcess* instance is linked via the *realizedBy* property to an *AtomicProcess* which can be grounded to the corresponding WSDL operation. Unfortunately, the existing material about DAML-S at the time of our investigations did not contain any guidelines (or explanations) that would reveal the role of the *SimpleProcess* construct for this particular problem.

4.4.4 Final Model in SIA4

Based on all these experiences and aware of the limitations of DAML-S, our final model is reflected in **SIA4**[11]. We model a single service with three different profiles (*Pr1, Pr2, Pr3*) each describing one of the three different functionalities of the service. The service has a single *ProcessModel* containing a composite process (*CP*). This indicates that the service can perform one of three possible atomic processes (*P1, P2, P3*). We map each

[11]`http://www.cs.vu.nl/~marta/services/sia/Sia4Service.daml`

of these atomic processes to one WSDL operation. This is essentially the same as SIA3, however we must give up the conceptual complex internal model of processes so that we are able to map them to WSDL operations. This description is also similar to SIA2 but reflects that we are dealing with three different signatures, not just two.

```
Service SIA4:
 *Profile:Pr1 (hasProc=P1)(I(url),O(RdfFile))
 *Profile:Pr2 (hasProc=P2)(I(RdfStream),O(RdfFile))
 *Profile:Pr3 (hasProc=P3)(I(server),I(url),I(psw),
                          I(ln),O(RdfFile))
 *ProcessModel:
    CompositeProcess: CP:Choice
    {
    AtomicProcess:P1(I(RdfStream),O(RdfFile))}
    AtomicProcess:P2(I(RdfFile),O(RdfFile))}
    AtomicProcess:P3(I(server),I(url),I(psw),
                    I(ln),O(RdfFile))}
    }
 *WSDLGrounding:
    WsdlAtomicProcessGrounding: Gr1 (P1->op1)
    WsdlAtomicProcessGrounding: Gr2 (P2->op2)
    WsdlAtomicProcessGrounding: Gr3 (P3->op3)
 *WSDL: same as SIA1
```

Concluding Section 4.4, we are not completely satisfied with the final model as we are unable to model the *P1, P2* and *P3* processes as composite processes which we feel better reflects the structure of the service. Also, we are concerned with the amount of effort involved in providing (and maintaining) this rather large set of descriptions. While at the *Service* superclass level we only have one definition, which is appropriate conceptually, this document is the smallest of all. The amount of repetition that exists in the three Profile and Grounding documents is considerable. Of course, cut and pasting will reduce the initial effort and defining concepts in one document and pointing to them in the other two can reduce maintenance overheads. However, the verbosity of this approach seems somewhat excessive. Our greatest concern with this final compromise is that we are unsure whether this model would be consistent with the model developed by another team to represent this same service. We feel that if semantics are to be added in a meaningful and useful way, greater direction and precision should be provided by the ontology so that this uncertainty is minimal.

4.5 Modelling a Complex Service - Sesame

The most complex service used by the SW@VU project is Sesame. This service presents modelling issues which are not addressed yet in any research paper but which are likely to surface when modelling services that expose the functionality of Semantic Web tools.

DAML-S was used to describe services in a variety of application domains. The first described services were fictitious (e.g., the examples offered on the DAML-S site), just like the services used for demonstrating solutions for complex tasks such as composition or matchmaking ((Sheshagiri et al., 2003), (Laukkanen and Helin, 2003), (Lei and Horrocks, 2003)). However, recently different communities have started to experiment with describing their own real-life services, such as the merging of agent technology with DAML-S ((Gibbins et al., 2003), (Richards et al., 2003)). Important developments are reported in the field of bioinformatics as well (Wroe et al., 2003). Also, wishing to describe mathematical services, (Barbera-Medina et al., 2003) concludes that DAML-S offers a good base. Last but not least, e-commerce services are described: a currency converter and a description of Amazon[12]. Surprisingly however, no Semantic Web tools were described semantically even if several Semantic Web services exist. Such are ontology storage facilities (Sesame), validators[13] or annotation tools[14].

We address this lack by using the entire DAML-S ontology and WSDL to describe a typical Semantic Web tool, Sesame [15]. We found that Sesame exhibits a set of characteristics which were not previously discussed for any particular group of services and therefore made its modelling non-trivial. First, it has a modular architecture composed of independent components which can be used in arbitrary combinations to fulfill certain tasks. In contrast, Web services described to-date, even if modular, have a well-defined work-flow model and their components are not usable stand-alone. Second, it uses complex data types requiring a shift from the current practice of perceiving inputs and outputs as atomic values. Third, there are several constraints between its *Input/Output* parameters.

Naturally, the question arises: *Is DAML-S expressive enough to model these characteristics?* This part of the chapter attempts to answer this question since it is likely that such characteristics exist for other (SW) tools as well. Therefore the problems that we encountered are likely to be typical of the problems other people will encounter with publishing their tools as a Web service. Thus we provide our solutions and recommendations in terms of usage guidelines and extensions to DAML-S.

4.5.1 Description

Sesame is an RDF(S) repository and query engine. As such it can provide valuable support for ontology-based applications. It can be used either as a part of an application or it can be accessed via a web-interface. Currently both HTTP and SOAP communication protocols are available. Sesame's HTTP interface[16] makes available six different functionalities at six different URLs. We shortly present these functionalities including in brackets the URL extension where they are published. [17]

A Sesame server contains a set of password-protected repositories. So called public repositories are accessible without any login information. Through the server's HTTP interface one can request a list of all repositories which are available for a specified login, including the public ones (*listRepositories*). As a storage facility, Sesame offers functionality to upload data and to interact with it. The data upload functionality adds data with a specified URL or sent as a string (*addData*). The content of a repository can be deleted completely (*clearRepository*). Further, deletion at the statement level is also supported (*removeStatements*). The user can retrieve the whole content of the repository (*extractRDF*) or alternatively only the schema information, only the instance information or both. A more refined data extraction method, the query method (*performRQLQuery* and

[12]`http://www.daml.org/services/`.

[13]DAML Validator at `http://www.daml.org/validator/`.

[14]Annotea - `http://annotest.w3.org/annotations`.

[15]The description is available at `http://www.cs.vu.nl/~marta/services/sesame/owl/`.

[16]See detailed description at `http://www.openrdf.org/doc/sesame/users/ch08.html`.

[17]A base URL of the form "http://HOSTNAME/SESAMEDIR/servlets" proceeds each extension - capitalized words depend on the server installation.

performRDQLQuery), transforms Sesame into a query engine. There is no predefined way in which these functionalities should be used. In fact, they are highly self-contained and can be used stand-alone rather than in combination with other functionalities.

4.5.2 Modelling Requirements

We expect that our service descriptions will be used by intelligent agents which will reason with the provided semantic data. Also, for operational level integration all needed technical information must be properly captured. This requirement is fulfilled by DAML-S which allows both conceptual and syntactic level specifications (through WSDL).

At the conceptual level the ***semantics of the offered functionality*** has to be specified. There are multiple ways to do this. First, one can specify a service by *describing its parameters*. This issue is treated throughout this part of the chapter but mostly in section 4.5.4. Second, the meaning of a service depends on its relation to other services. It is desirable to describe how these components relate to each other so that an intelligent service could determine usage patterns for fulfilling certain tasks. One dependency between services is that of *composition*, when a certain service is composed out of multiple other services. Since this is the case of Sesame we investigate this issue in section 4.5.3.

Another issue is ***linking between the semantic and syntactic descriptions*** of a service. It is desirable that a single conceptual description can be mapped to descriptions of different technical implementations of the same service. In Sesame's case, it should be possible to ground its semantic description to the syntactic descriptions of all its different interfaces (HTTP, SOAP, RMI). We only experimented with linking to a WSDL description of an HTTP interface, however, we encountered problems as those discussed in Section 4.4.1. Still related to grounding, technical descriptions used in combination with semantic ones should undergo minor changes so that they are still usable by tools which do not require semantic data. More on this in Section 4.5.4.

4.5.3 Specifying Service Semantics

Splitting up Sesame. In order to satisfy the various needs of service providers, DAML-S does not impose a particular modelling style leaving a gap between conceptual considerations and the actual modelling. This happens with the very notion of service. Conceptually, they define Web services as *"Web sites that [...] allow one to effect some action or change in the world"*[18] (DAML Services Coalition, 2002). Further they differentiate between simple and complex services. A simple service invokes a single Web-accessible program and there is no interaction with the user during its execution (e.g., request-response services). A complex service is composed of multiple simple services.

In the light of these definitions Sesame and each of its functionalities are Web services, where the functionalities are simple services and Sesame is a complex service. Deciding on the actual modelling was not so trivial. We present two of a number of alternatives we tried, underlining their benefits and disadvantages.

1. First modelling alternative. We first adopted the modelling style used in the existing DAML-S examples (see a schematic representation in Figure 4.3). Accordingly,

[18]Note that this definition of a Web service is broader than that given by the Web services community which considers a Web service a Web accessible program.

we created a single *Service* instance, *Sesame*. The component functionalities were modelled as *AtomicProcesses (*P1, ...P6*)* within the *ProcessModel (*P*)*. Because, there is no fixed workflow between the components of Sesame, i.e., the six functionalities can be used in arbitrary ways to solve tasks, the workflow model that was closest to our needs was *process:Choice*. This expresses little about the relations between the components and thus its semantic value is weak. We tried to compensate this at the *Profile* (*Pr*) level where we associated multiple *Profile* instances with the service, one for each functionality *(Pr1, ...Pr6)* and one for the global functionality of Sesame (*P* as an *OntologyStore*). Finally, the *Grounding (G)* of Sesame contains six *AtomicProcessGrounding* entities *(g1, ..g6)*, one for each internal process and its corresponding WSDL operation *(o1, ...o6)*.

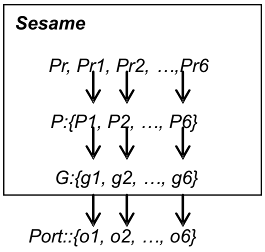


Figure 4.3: Traditional service modelling for Sesame.

This approach has the following limitations. First, several of the simple services offered alternative interfaces and implementations. Because only the *Service* concept can have different groundings (and not atomic processes), a different implementation of an atomic service results in a different grounding for the whole service. Therefore, we face a combinatorial explosion of groundings for the general service whenever a component offers a new implementation. Second, at the *Profile* level, there is no indication of the relation between the seven services. Third, even if all services are visible at matchmaking time (since their *Profile* is explicitly mentioned), one needs to interpret the full description of the complex service when using just a simple service.

Note that this style of description works fine for Web services composed of simple services which (1) do not have meaning outside the service and (2) are used according to a fixed workflow. In such cases a single Profile suffices to describe the global task that their execution can achieve. There is no need to advertise all of them since they cannot be used stand-alone anyway. However, none of these assumptions apply to Sesame: its functionalities can be used in any combination and also just alone.

2. Second modelling alternative. Due to these considerations we decided to model each functionality as a separate *Service (in services S1 to S6)*, as represented in Figure 4.4. Two of the limitations of the previous model are solved: (1) the overhead of declaring a new implementation of a simple service is reduced to declaring a single atomic grounding and (2) when using a service only its own description needs to be interpreted. However, the issue of expressing how these services inter-relate is still open. Optimally we would like to express that sets of these services can be used upon a certain Sesame server. Depending on the exposed services, different Sesame servers can have different types: some can offer only storage functionality (*OntologyStore* - *Sesame2*), some offer

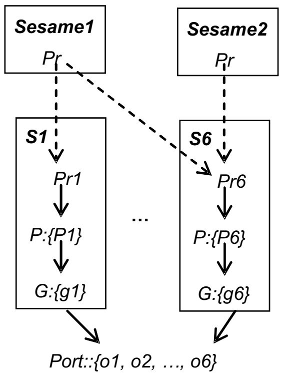


Figure 4.4: Proposed modelling for Sesame.

querying facilities (*QueryEngine*), some offer both (*OntologyStore and QueryEngine - Sesame1*). We want to express composition at the *Profile* rather than the *Process* level. We implemented this by building a domain ontology[19] in which we enrich some of the DAML-S concepts. We relied of the assumptions that a widely used domain ontology (1) should reflect terms accepted by a large community while (2) being easy to extend with new knowledge and (3) straightforward to integrate in the service descriptions.

In our domain ontology we make a conceptual difference between tools and their functionalities and define tools in terms of the functionalities that they provide. In terms of modelling this involves two inter-related constructions. First, we specialize the DAML-S *profile:Profile* into *SWTool* and *Functionality* (see Figure 4.5). The subclasses of the first concept reflect the main tool categories described by a thorough survey of existing Semantic Web tools (Gomez-Perez, 2002). The second concept is the superclass for all functionalities. For now, we declared the functionality types relevant for Sesame and grouped them under more generic concepts. For example, we consider *RetrieveOntology* and *AskQuery* as methods for data retrieval. Their superclass, *RetrieveData*, can be extended with other methods specific to other tools which fulfill the same function.

Second, we declared the new property, (*hasFunctionality*), which allows us to specify the kind of functionalities that a certain tool offers (depicted as part of the DomainOntology in Figure 4.6). This construction allows us to define *SWTools* by imposing constraints on the type of functionalities they have. For example, an *OntologyStore* might allow a functionality of type *StoreData* and *must* offer a *RetrieveOntology* functionality. Also, a *QueryEngine* must provide an *AskQuery* functionality. Further, the ontology contains a set of terms needed to describe Sesame, such as *Repository, User, Password*.

We fulfilled the requirements for a widely used domain ontology. First, we used concepts from the largest survey about Semantic Web tools within the SW community and extended the concepts of DAML-S, which is growing into a standard. Second, our model is easy to extend: new functionality types can be added and new tool types can be defined using existing or newly added functionality. The third issue, that of easy use for

[19]`http://www.cs.vu.nl/~marta/services/swTools.owl`

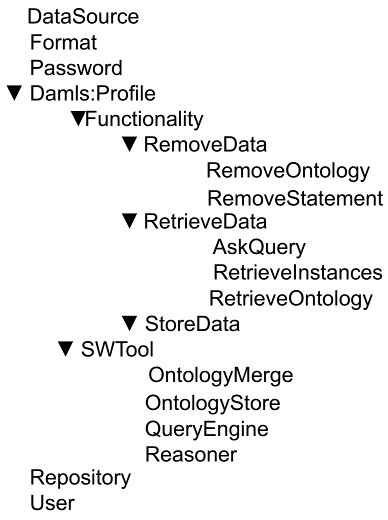


Figure 4.5: The domain ontology.

service description is demonstrated next.

Using Domain Knowledge in DAML-S descriptions. Domain concepts are used for several reasons in a DAML-S description. First, they express the meaning of the offered functionality at the *Profile* level. This section is devoted to this first issue. Second, domain knowledge can be used to describe the IOPE's both at the *Process* and the *Profile* level. Finally, during the grounding process, WSDL descriptions are enriched with domain knowledge (as shown in Section 4.5.4).

One indicates the overall functionality of a service by declaring its *Profile* as being of a type documented in the domain ontology. As discussed previously, when modelling tools that offer complex functionalities we model both the tools and their functionalities as services. We indicate their type of being a tool or just a functionality by using appropriate types for their profiles. Figure 4.6 demonstrates this process. *Sesame1* is a *Service* instance and its *Profile*, *Sesame1Prof* is of the type *OntologyStore* (a kind of *SWTool* profile). Similarly, each individual functionality is declared as a service (according to our modelling decision). For example, for the *AddData* functionality, we declared a *Service* instance, *S1AddData*, and associated with it a *Profile* instance of type *StoreData* (which is a kind of *Functionality* profile).

By declaring Sesame's *Profile* of type *OntologyStore* we inherit the *hasFunctionality* property defined for any *SWTool*. At the instance level, we use this property to associate *Sesame1Prof* with *S1Prof*. This indicates that the functionality offered by the *Sesame1* service relies on the functionality of the *S1AddData* service.

It is easy to define different Sesame instances with different combinations of these functionalities. This is useful in practical scenarios when different user groups can access different functionality. For example, a company which stores its data in Sesame may wish to make it available both for its customers and employees. Customers are only

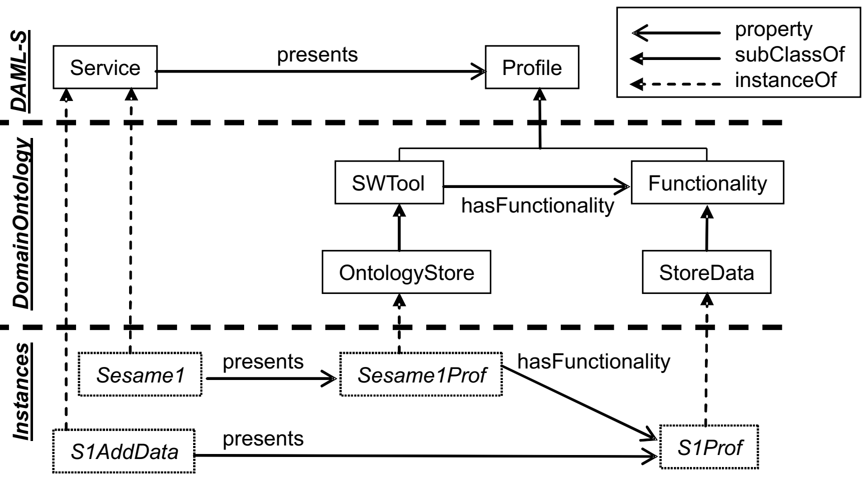


Figure 4.6: Use of domain knowledge in the *Profile*.

allowed to read the stored data. Therefore, that particular service instance of Sesame only advertises a data reading capability. On the contrary, employees can both add and read data, therefore their service instance points to both functionalities.

We managed to express composition (interdependency) at the *Profile* rather than just at the *Process* level, as originally possible with DAML-S. At matchmaking time, if someone wants to use Sesame he gets a list of all needed service profiles and can discover the individual services associated to those profiles.

4.5.4 Input/Output Specification

We have previously been concerned about the service as a whole. We continue with details about how the parameters of the service can be specified. The first part of this section proposes a more flexible modelling of input parameters so that inputs depending on a condition can be specified as well. In the second part we propose an alternative to complex data type specification.

Conditional inputs. DAML-S does not cover cases when inputs depend on a certain condition. Currently one can only indicate whether an input is mandatory or optional by relying on the DAML+OIL cardinality restriction mechanism. For each mandatory input a cardinality restriction must enforce its existence. However, we encountered situations when an input is only required in combination with another input. For example, when specifying the log-in information, a *password* is only mandatory when the *user* parameter is supplied. Further, in case of the *addData* service if the actual data is supplied, rather than a *URL*, then a *baseURL* must be specified.

Solution. A solution that is the most consistent with the current version of DAML-S is to extend the *Process* ontology to support conditional inputs in the same way that outputs and effects may be conditional (see Figure 4.7). Following the general template

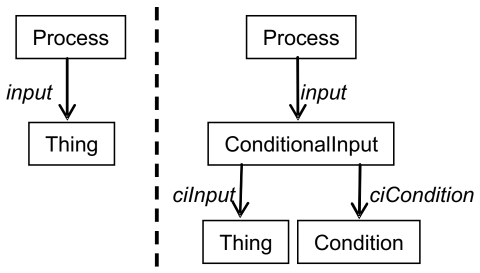


Figure 4.7: Current and proposed modelling of inputs.

of defining conditional elements, we propose defining the *ConditionalInput* class which simply bundles a *Condition* and an input, using two properties: the condition of the conditional input (*ciCondition*) and the input of the conditional input (*ciInput*). A conditional input is an input that only occurs when a condition is true.

This modelling allows specifying a wider range of inputs than currently possible. First, inputs depending on factors external to the process can be described. For example, one can specify that customers from outside the USA do not need to provide the zipcode input. In this case the customers geographic location conditions whether the zipcode input is needed or not. Second, we can capture situations exemplified before where an input is conditioned by the existence of another input, or generally a condition that involves other inputs. Third, we propose an alternative way to specify mandatory inputs by conditional inputs with a condition that is always “True”. Finally, optional inputs can be modelled as *UnconditionalInputs*, i.e., inputs that do not depend on any condition. This list is not complete, however, we think that these cases are likely to appear in other tools (and services) as well. Note that we did not deal with how conditions should be specified. In fact, this is an ongoing research topic for the whole community.

Complex data types: XML Schema or DAML+OIL? To support automatic discovery and operational integration of Web services, their markup should provide information about (1) the semantics of their methods and parameters as well as (2) their syntactic signature (e.g., method names, data format of parameters). The issue of how one can best specify complex data types (both at a syntactic and semantic level) has a major importance for Sesame where four methods provide complex output data types. As a concrete example, consider the simplest functionality of Sesame: *requestRepository*. It’s output conforms to the DTD specified below.

```
<!ELEMENT repositorylist (repository)*>
<!ELEMENT repository(title)>
<!ELEMENT title          (#PCDATA)>
<!ATTLIST repository
         id       ID           #REQUIRED
         readable (true|false) #REQUIRED
         writable (true|false) #REQUIRED>
```

Such complex data formats are defined in the “types” section of a WSDL document. For maximum platform independence XML Schema (XSD) is used to specify the format of complex data types. The syntactic information provided by XSD allows other tools to easily parse any output that conforms to this DTD, therefore allowing an easy integration between Web services. However, this information has no semantics.

The DAML-S solution to this is to replace the syntactic XSD definitions with semantic definitions written in DAML+OIL, since WSDL allows using any XML-based type definition language in its “types” section. The claim is that this would use DAML+OIL’s rich data typing feature[20] (Ankolekar et al., 2002a).

We felt that this recommendation had limitations when applied to the complex types in Sesame. First, we found it difficult to express the complex syntax of the data type solely with DAML+OIL elements. We even considered using an OWL-based type definition as it provides more advanced ways for data typing than its ancestor. Even so we did not achieve our goal. This is understandable since DAML+OIL (and OWL) is an ontology language and as such it delegates the data type representation to XSD.

Second, a WSDL document using purely DAML+OIL based data types would be useless for those users of the service which do not understand DAML+OIL. Ideally, an existing WSDL description should undergo minor changes when enabled for use with semantic technology to ensure its backward compatibility with traditional applications. We expect that many tools would use the XSD definitions of Sesame’s WSDL description and only few would understand DAML+OIL. Therefore, we want to extend XSD types rather than to replace them.

Third, for some applications, not all parts of a complex data type were interesting semantically. For example, the syntax of the previous output is very complex, but semantically we are not interested in all its parts. We want to specify that objects of type *Repository* are returned but we do not want to specify further details.

Solution. With these considerations in mind, we used an alternative for complex type definition. We used XML Schema to specify the syntax of the output, just like for a traditional WSDL description. To add semantics to this type we augmented its components with references to corresponding DAML+OIL concepts. The *xsd:annotation* tag has exactly this function. We wrote the following XSD definition and augmented it with concepts defined in the domain ontology.

```
<xsd:element name="repositorylist">
  <xsd:complexType>
    <xsd:element name="repository">
      <xsd:annotation>
        <xsd:documentation>do:Repository</xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
        <xsd:attribute name="readable" type="xsd:boolean" use="required"/>
        <xsd:attribute name="writeable" type="xsd:boolean" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:complexType>
</xsd:element>
```

[20]The issue of converting between XML messages and an ontology have been addressed by WSMO/IRS-III by the *lifting* and *lowering* mechanisms.

This method satisfies all of our requirements. Using XML Schema to define complex types is straightforward since the language was designed for this purpose. We can easily add semantics to any parts of the description in such a way that it remains usable for non-semantic based applications as well.

4.6 Conclusions

Based on our experiences when describing these three services we distilled some general observations about DAML-S.

Positive Aspects of DAML-S. The strength of DAML-S is that it goes beyond syntactic description of a service by providing a semantic description. Semantics allow reasoning about a service and move us towards the ultimate goal of dynamic service discovery and usage. The DAML-S upper ontologies provide semantics for high level concepts concerning Web services. Using these concepts and a set of domain relevant concepts offered by a *domain ontology*, meaningful descriptions of services can be achieved. This is particularly valuable for matchmaking where the requestor may use alternative terms. The domain ontology can use the *SameClassAs* relation to identify synonyms and the *SubClassOf* relation to identify hypernyms and hyponyms. This supports matchmaking where different levels of abstraction are used between the requestor and provider. The other key strength of DAML-S is that it links to an industry standard, namely WSDL. In this way, it fulfills its role as a link between the Semantic Web community and industry.

Negative Aspects of DAML-S. Besides the above mentioned positive aspects of DAML-S we also encountered a set of shortcomings.

A) Imprecise conceptual model. While it is commendable that DAML-S seeks to provide flexibility and thus has not fully defined a number of its concepts, this flexibility comes at the expense of clarity. The result is that DAML-S has an imprecisely underlying conceptual model. We base this statement on the following facts:

Different models exist within DAML-S. The three parts of DAML-S employ different metaphors to describe services. At the Profile level a service has four types of parameters: IOPEs. At the Process level IOs and PEs are treated conceptually differently as they emerge from two different views of a service (see Section 4.3). The conceptual gap is even wider when a DAML-S model is aligned to a WSDL model which defines services as collections of ports (see Section 4.4). These alternative conceptual models make specification of services difficult and mapping between models almost impossible. Even within DAML-S the different models can lead to inconsistencies in the specification (see Section 4.3).

The mappings between parts of DAML-S are unclear. First, the mapping between the Profile and Process lead to possible inconsistencies in the final service description (Sections 4.3). Second, the mapping to WSDL limits the expressiveness of DAML-S (Section 4.4). Just by modelling a simple service we conflicted with two out of the three mapping rules. This forced us to revise our models so that a grounding was possible at the expense of giving up polymorphism (SIA1) or an accurate specification of a complex internal structure (SIA3).

No correspondence to software engineering (SE) concepts is established. Many of the intended users of DAML-S are software engineers. We consider that reference to and support for SE concepts, perhaps in the form of concept mappings, would ease the understanding of DAML-S. While WSDL intuitively models different interfaces as PortTypes and allows grouping operations in ports (as the methods of an interface), it seems that DAML-S only considers the very simple function metaphor (methods). More complex concepts such as polymorphism or re-use are not supported. We think that a SE model could both disambiguate some of the concepts and give a shared framework for DAML-S and WSDL.

The imprecise conceptual model reduces the modelling power of DAML-S:

Multiple modelling possibilities exist. The complexity of DAML-S modelling increases with the complexity of the modelled service. Because there is no clear view of what services are, one can produce a variety of models (as we demonstrated for two of our services, SIA in Section 4.4 and Sesame in Section 4.5). Note that the nature of ontology based modelling is also partly the cause of having multiple modelling possibilities.

Ad-hoc polymorphism cannot be modelled. Ad-hoc polymorphism is not just a term used in software engineering, but a mechanism equally valid in an e-commerce scenario. For example, any e-commerce site would optimally allow multiple ways of paying, e.g., by credit card or by bank transfer. Basically, the same effect would be achieved (i.e., paying for an item) even if the money would be obtained from different sources. From a SE point of view this is only a method so it should be modelled as a single service. However, several unclarities arise about how to model these different facets of the same service in DAML-S (see Section 4.4). It is not only a question to conform to the limitations of the ontology but also to know if our model is conceptually correct.

B) Difficult to learn. One of our major comments (and worries) is that it was difficult to get started with writing DAML-S. The previously mentioned lack of conceptual model played a fair role in this. Other inhibiting factors were:

Limited tool support. At the time of our research there was no tool support for writing DAML-S markup. When using simple text editors, the task of building complex descriptions consisting of three interlinked parts become complex, error-prone and frustrating. This situation is changed nowadays given the plethora of semantic Web service description editors such as ODE-SWS (Gomez-Perez et al., 2004), or the OWL-S Editor (Elenius et al., 2005).

Few examples and guidelines are available. The DAML-S site provided two complex examples. However, these examples were artificially created to fit the ontology rather then being real-life examples. As a consequence, they ignore situations that arise in the case of real life services. Further, the lack of examples was augmented by the lack of guidelines about modelling certain situations.

Knowledge of DAML/WSDL/SOAP is required. The pre-requisites to start writing complete DAML-S descriptions are rather high: one has to know DAML,WSDL and SOAP. For users who are only partly familiar with these techniques it is a considerable burden to learn all of them in sufficient depth.

4.7 Summary

In this chapter we investigated the expressiveness and usability of DAML-S by employing it to model three real life services.

Part A of our conclusions, referring to the imprecise conceptual model of DAML-S and its modelling limitations, indicate that the ontology has a **low expressiveness**. The contribution of this chapter is to point out some concrete limitations of DAML-S and to provide solutions. In short, these limitations and the corresponding solutions are:

1. The mapping between the Profile and the Process lead to inconsistent descriptions. As a solution, both DAML-S ontologies were redesigned (Section 4.3);

2. The mapping to WSDL does not permit modelling ad-hoc polymorphism and complex internal structures. The problem was partially addressed with a new grounding model (Section 4.4).

3. It is not obvious how to model complex services that combine several self-contained services. We provide our solution, based on the use of domain ontologies, in Section 4.5.3.

4. Conditional inputs are not supported by DAML-S. We present a rationale for them and a possible modelling in Section 4.5.4.

5. Finally, we argue for performing minimal changes to the described WSDL files in terms of the added semantic information (Section 4.5.4).

Part B of our conclusions indicates that DAML-S has a **low usability**. Our contribution to this point is to provide our example marked-up services for the community. Also, we identify a set of commonly occurring modelling situations, we document them and offer our modelling solutions (as possible guidelines).

The findings of this chapter were derived while using DAML-S, however, some of the general lessons that we learned are applicable to other Web service ontologies as well. First, the modelling situations that we documented should be covered by other ontologies as well. Second, we believe that it is crucial to apply a new ontology to real life services in order to assure its expressiveness. In the case of DAML-S, using it to describe only three real life services lead to discovering a set of noteworthy issues which suggested important improvements. Finally, usage examples and guidelines, as well as tools, are important for ensuring the adoption of the ontology.

In the next chapters we continue to analyze DAML-S from the perspective of its adaptability to other domains and ontological correctness.

Chapter 5

Adapting OWL-S to Generic Software Entities

After concluding on the expressivity and usability of OWL-S in the previous chapter, in this chapter we investigate its adaptability to describe other kinds of software entities than Web services. The motivation to perform this analysis is twofold. First, we wish to establish whether OWL-S is generic in a way that makes it reusable in similar domains. This would prove a quality asset since knowledge reuse is one of the goals of ontology engineering. Second, if OWL-S can be adapted to semantically describe generic software entities then these entities can be easily exposed as semantically described Web services thus facilitating the uptake of Semantic Web services technology. To perform our analysis we use OWL-S as a basis for developing an ontology that would support a certain type of middleware system, i.e., an application server. We begin by identifying the aspects of application servers that benefit from semantic technology and than show how OWL-S was reused for building an ontology to support application servers.

The material presented in this chapter has been published at the First Australian Workshop on Engineering Service-Oriented Systems (AWESOS)((Sabou et al., 2004)).

5.1 Introduction

As a generic Web service ontology, OWL-S captures knowledge for describing a particular type of software entities, Web services. Typically, such generic knowledge structures should be easily adaptable in related domains. Indeed, facilitating knowledge reuse is one of the goals behind building ontologies. Therefore, one indication of the quality of OWL-S would be its reuse in other contexts than Web services.

There is also a practical motivation behind reusing OWL-S for describing other types of software entities and for supporting other tasks than those specific to Web services. We envision that in a short time many different software entities will be made available as Web services. For example, components that are now managed by middleware systems or device controllers. If this entities are described in an ontology that has been derived from OWL-S then at the time of their exposure as Web services it will be straightforward to derive their semantic descriptions. This will foster the adoption of Semantic Web services technology. The benefits of using semantic descriptions can be already explored in the original setting where the components are used (for example, in this chapter we show how such semantic descriptions automate several middleware specific tasks).

In this chapter we are interested in the degree to which OWL-S is adaptable in the context of middleware systems. Generally speaking, middleware systems facilitate multi-tier application development in distributed information systems. Common middleware solutions focus on software entities that provide an application programmer's interface (API), also called *software modules*. Like current Web service technology, middleware relies on syntactic standards (e.g., IDL (Object Modelling Group, 2002)) for describing the software modules. However, these syntactic standards have a limited expressivity.

We perform our analysis on the adaptability of OWL-S in the context of a concrete application server: the *Application Server for the Semantic Web (ASSW)* (Oberle et al., 2004c). *Application Servers* are component-based products that provide functionality for security and state maintenance, along with data access and persistence for the development of Web applications. Despite the comprehensive functionality of application servers, realizing a complex distributed system remains all but an easy task. For instance, managing component dependencies, versions, and licenses is a typical problem. In Microsoft environments, this is often referred to as "DLL Hell". Developers are also confronted with an ever-increasing repository of programming libraries for standard tasks such as IO, networking, database access, or the handling of XML. It would be desirable to assist the developer in using these resources.

This chapter is structured as follows. In Section 5.2 we provide the motivation from a middleware perspective for developing an ontology to support frequent middleware tasks. Then, in Section 5.3 we present such an ontology that was built by adapting OWL-S. We report on the status of embedding the newly built ontology into a middleware server in Section 5.4. We present related work in Section 6.2 then conclude in Section 5.6.

5.2 Motivation

In this section we introduce the Application Server for the Semantic Web middleware (Section 5.2.1) and describe a set of usage scenarios which require semantic descriptions of software modules (Section 5.2.2). These scenarios lead to a set of requirements (Section 5.2.3) which guided us in developing our ontology as presented in Section 5.3.

5.2.1 Application Server for the Semantic Web

Integration of existing software modules is an important issue for the Semantic Web since complex applications require more than a single software module. Ideally the developer of such a system wants to easily combine different — preferably existing — software modules. So far, however, such integration had to be done ad-hoc, generating a one-off endeavour, with little possibilities for reuse and future extensibility of individual modules or the overall system. The *Application Server for the Semantic Web (ASSW)* (Oberle et al., 2004c) addresses this issue by facilitating reuse of existing modules (e.g., ontology stores, editors, and inference engines) and, thus, the development and maintenance of comprehensive Semantic Web applications. ASSW combines means to coordinate the information flow between modules, to define dependencies, to broadcast events between different modules and to translate between Semantic Web data formats.

The architecture of the application server relies on the Microkernel and component approach. The Microkernel offers a minimal functionality of managing, i.e., starting, stopping and initializing components. Existing software modules have to be made deployable[1] in order to be managed by the Microkernel. This process adds a wrapper around a software module and transforms it into a *Component*. At run-time, the application server may host several components with the same application interface (API). In order to distinguish between components that are of direct interest to the developer and components providing functionality for the Application Server itself (e.g., connectors or the registry), we call the first *Functional Components* and the latter *System Components*.

Client software hard-codes the use of a certain API. In order to facilitate the working with a deployed component, a client software can use so-called surrogates which are client-side objects that reveal the same API like a particular component and relay communication to them[2]. Thus, the client is relieved from handling network protocols and middleware idiosyncrasies. At run-time, the application server allows the client to chose which component the surrogate should relay communication to.

The Application Server for the Semantic Web is based on the design and development of existing application servers, applying and augmenting their underlying concepts for use in the Semantic Web. Even more, we wish to use semantic technology within the server itself in several scenarios as discussed in the next subsection.

5.2.2 Scenarios

In order to tackle the problems of managing and locating components, we propose to introduce a formal conceptualization of these software related issues. Using this formalism allows representing common knowledge about the domain, such as the fact that component dependencies are transitive. The use of formal descriptions of software modules has several benefits. Already the development of an ontology is beneficial to gain conceptual agreement between the Application Server and the developer. For example, the ontology formalizes relationships between the internal components of the server. Further, a multitude of software tools can leverage the knowledge specified by means of a reasoning engine. Even more, semantic descriptions of software modules can improve many of the frequently occurring scenarios within an application server. The scenarios listed below apply to any application server but we detail them in the ASSW setting.

S1. Implementation details. Libraries often depend on other libraries and a certain archive can contain several libraries at once. Given this information, a system could assist the developer in locating all the required libraries. Furthermore, the user might be notified when two libraries require different versions of a certain third component. For instance, the various early versions of XML parsers cause a lot of trouble. The system only runs if the libraries are included in the path in a certain order so that the class loader picks up the latest version. We envision to reason with this kind of data in order to make an educated suggestion in these situations.

S2. Component Discovery. At run-time, a client can dynamically decide to which component its surrogate should relay communication. For example, an ontology editor

[1]We use the word deployment as the process of registering, possibly initializing and starting a component to the Microkernel.

[2]Similar to stubs in CORBA.

might use a specific ontology store, however, there might be several of them deployed in the Application Server. At this moment information other than functionality is important, most prominently certain properties of a component, e.g., if an RDF store component is capable of transactions. The registry, a system component and simple ontology store, holds semantic descriptions of all deployed components and can be queried accordingly.

S3. API Discovery. When programming its application, a developer might want to find a certain API in order to be able to program against it. Preferably, he will want to perform this search at a semantic level, i.e., specify high level details of API functionalities. For example, a developer of client software might need an API that provides ontology *storing* and *retrieving* functionality. As there can be several related and overlapping APIs, the system should recommend the best fitting one. To support such semantic search we suggest describing existing functionalities (i.e., those offered by the methods of the APIs) in terms of the concepts contained by a common service taxonomy.

S4. Classification of APIs. A developer might want to determine the type of a new API based on the type of its offered functionality (i.e., its methods). For example, an API offering ontology storage and inferencing capabilities will be of both *StoreAPI* and *InferenceAPI* types.

S5. Publishing Web services. The developer might want to use the functionality hosted in components as Web services. A Web service connector may publish components' methods correspondingly. Development toolkits usually provide the functionality for creating stubs and skeletons or for automatically generating interface descriptions. For example, the java2wsdl tool generates WSDL definitions from java based service implementations. With representation languages like OWL-S, tool support for these new languages is needed. Whereas WSDL tools can obtain almost all of the required input directly from the source code, more powerful languages require additional measures. Using the internal semantic description of APIs, it will be easier to generate their corresponding OWL-S description.

5.2.3 Requirements

The scenarios discussed above lead us to a set of requirements which served as design principles for the ontology presented in section 5.3.

R1. Module Implementation and Functionality Syntax. The ontology should contain means to describe the implementation and invocation details of software modules which will be used to support implementation tasks (scenario S1).

R2. Module Characteristics and Functionality Semantics. The ontology should contain means to give high level descriptions of software modules, e.g. their types such as ontology stores and reasoners, as well as their characteristics, providers etc. This description supports component and API discovery (scenarios S2 and S3).

R3. Reusability and Sharing. Semantic descriptions of software modules should be reusable. Therefore, easy coupling of syntactic and semantic description should be supported. Another aspect of reuse is that the ontology that we design for use within the application server should incorporate existing efforts (such as standards and technologies) that are already used by communities.

R4. Domain Independence. The ontology should be reusable over a wider range of domains (not just in our Semantic Web domain), therefore we should separate generic and domain specific concepts.

These requirements guided us in designing the ontology we present next.

5.3 Ontology Design

In this section we present the ontology (shown in Figure 5.2) built by adapting OWL-S for supporting the ASSW application server. We start by providing a comparative overview of the ASSW ontology and OWL-S in Section 5.3.1. In Section 5.3.2 we present each sub-ontology in detail and show how each of them fulfills a certain requirement. A concrete example of a module description showing how all these ontologies are used at instantiation level is presented in Section 5.4.1.

5.3.1 Overview

The design principles of OWL-S described in Section 2.4.1 underpin our work, as comparatively depicted in Figure 5.1. These principles influenced the kinds of sub-ontologies and their relations. The following discussion gives the rationale of our design decisions.

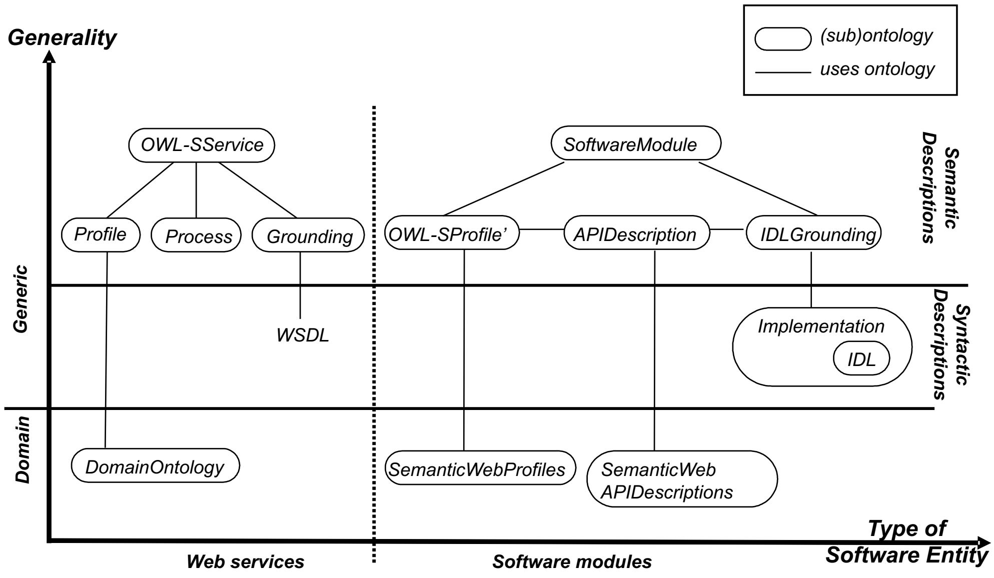


Figure 5.1: A comparative overview of OWL-S and the ontology for software modules.

1. Semantic vs. Syntactic descriptions. We adopted the separation between semantic and syntactic descriptions, therefore complying with requirements R1 and R2. A number of our ontologies allow semantic description and others are used for syntactic descriptions. A mapping exists between the description of both aspects. However, given the different types of entities we want to describe, we modified some of the OWL-S ontologies as follows:

- we kept the OWL-S *Profile* ontology for specifying semantic information about the described components. Also, we extended it with a few concepts for describing the functionality of APIs (and their methods) at the conceptual level. This was necessary because the *Profile* ontology's constructs for specifying functional descriptions were too shallow. These extensions are grouped in a small ontology called *API Description*.

- we did not use the *Process* ontology because our previous analysis (detailed in Chapter 4) yielded that Semantic Web tools usually offer a set of simpler functionalities, but there is no predefined way of invoking them that could be captured in a certain dataflow. Should the type of described components change, our modularly-designed ontology can easily be extended with the Process ontology.

- we defined our own language for describing APIs syntactically since WSDL is designed for specifying network endpoints. For this purpose, we formalized a subset of IDL (Interface Description Language (Object Modelling Group, 2002)) terms in the *IDL* ontology.

- as a consequence of the changes above, we could not reuse the existing OWL-S *Grounding*, rather we wrote our own grounding ontology (*IDLGrounding*) which allows mappings between the conceptual description of the APIs (in the *Profile*) and their syntactic specification (*IDL*).

2. Generic vs. Domain knowledge. We use both generic and domain ontologies in order to fulfill our desiderata expressed by R4. OWL-S (and WSDL) are at a generic knowledge level. The same is true for our extensions of OWL-S, as shown in Figure 5.1. For our specific goals we built two domain ontologies in the area of the Semantic Web. The first one specifies the type of existent Semantic Web software modules at a coarse level. The second one describes the functionality of Semantic Web specific APIs at a more fine grained level (i.e., in terms of methods and their parameters). Naturally, these ontologies can be easily replaced depending on the application domain, for example bio-informatics.

3. Modularity. Modularity enables easy reuse of specifications and extensibility of the ontology. An important issue is the size of the reusable parts. For example, because a *Profile* instance contains a lot of information, which is often specific such as the contact information of the providers, it is less likely that this instance will be reused by any other service description. Therefore a coarser granularity (less information per concept) increases the chance of reusability. We reused this principle by identifying related content, relating it to a central concept and grouping everything in small ontologies which can be re-used as sub-ontologies. We describe the process of isolating reusable knowledge in the following subsection, where we present a short overview of each ontology.

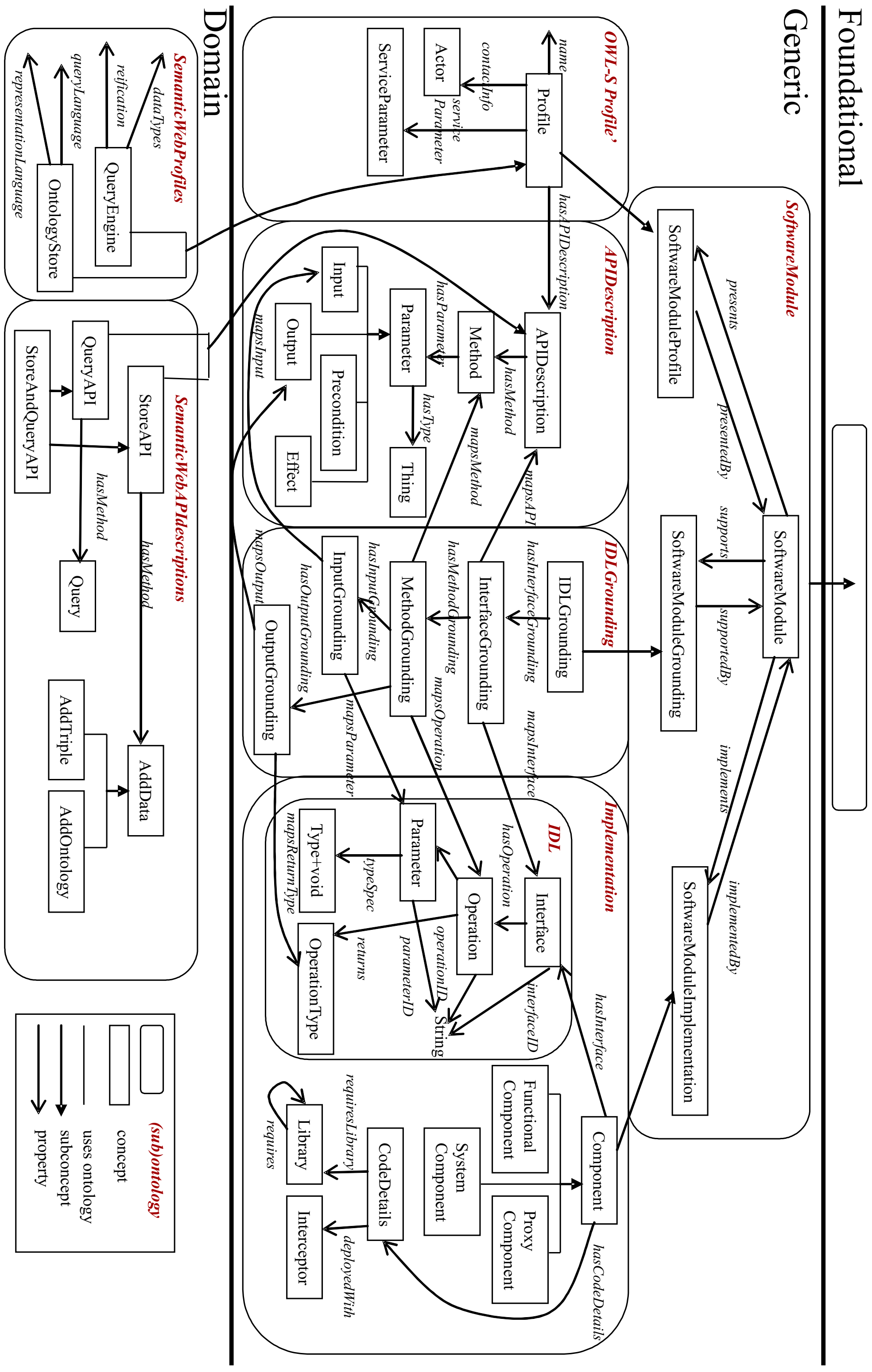


Figure 5.2: The ASSW ontology.

5.3.2 The sub-ontologies

Besides reusing the design principles underlying OWL-S we were also guided in our design by the requirements put forward in Section 5.2.3. In this section we briefly describe each of our sub-ontologies and the relations that exist between them (a graphical overview of all the ontologies is shown in Figure 5.2). For each ontology that we discuss we indicate the scenarios (as presented in Section 5.2.2) that it supports as well as the requirements (stated in Section 5.2.3) that it fulfills. Table 5.1 shows the relations between our requirements and the sub-ontologies, confirming the major influence that these requirements had on our design.

Requirement \ Sub-ontology	**R1. Syntax**	**R2. Semantics**	**R3. Reuse and Sharing**	**R4. Domain Independence**
Software Module			X	
OWL-S Profile’		X	X	
API Description		X		
Implementation	X			
IDL	X		X	
IDL Grounding		X	X	
Domain Ontologies				X

Table 5.1: Dependencies between requirements and sub-ontologies.

Software Module ontology

This ontology is similar to the OWL-S *Service* ontology and thus responds to our requirement R3 of sharing and reuse of existing standards. The ontology, depicted in Figure 5.3, contains the main concept and the top concept for each type of description, ensuring a coarse-grained modularity for the whole description. We performed some changes:

- we renamed the *Service* concept to *SoftwareModule*, as such entities are the focus of our descriptions. Accordingly we renamed the *ServiceProfile* and *ServiceGrounding* concepts to *SoftwareModuleProfile* and *SoftwareModuleGrounding*.
- we excluded the *ServiceModel* concept, since, as stated in Section 5.3.1, we are not interested in the internal working of the modules.
- we added a *SoftwareModuleImplementation* concept that groups together implementation details described in the *Implementation* ontology.

The three concepts that describe a *SoftwareModule* can be specified using the corresponding ontologies as described next.

OWL-S Profile’ (extension)

We use the OWL-S *Profile* ontology to specify the particular characteristics of a *SoftwareModule* such as the contact information of the providers and certain parameters. For example, an ontology store would have a service parameter specifying the representation

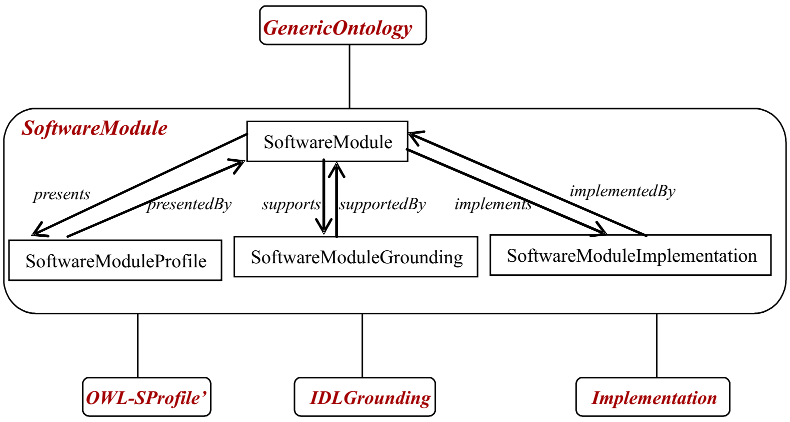


Figure 5.3: The *SoftwareModule* ontology.

language used. Therefore, our *Profile* describes the component as a whole. Information of this type might be used during component discovery at run-time and corresponds to our requirement R2 of providing generic, high level characteristics of the described modules. Some examples of such parameters are provided in Section 5.3.2.

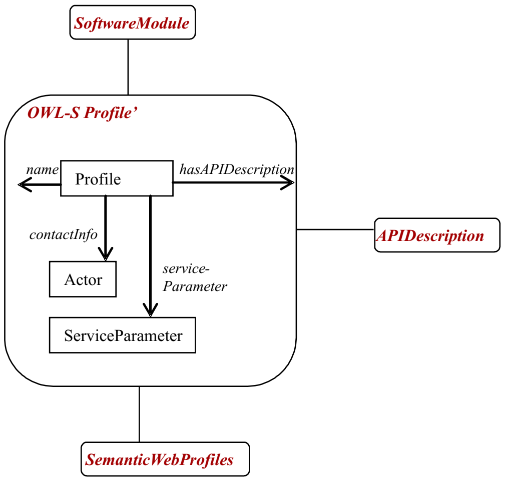


Figure 5.4: The *OWL-SProfile'* ontology.

We found that the current functional description specification of OWL-S is focused on expressing a single functionality, while we want to describe several functionalities offered by a software module, which correspond to (a set of) methods in the API[3]. Because

[3]Note that in the previous chapter we encountered the same weakness of the Profile model where we wished to model that a service relies on different other services.

of that, we added a new property to *Profile* (see Figure 5.4), namely *hasAPIDescription*, which ranges over the *APIDescription* concept that groups the information used to describe an API and is separated in a small ontology (*APIDescription*). We separated this content in a small ontology because we expect that many modules will be able to reuse such functionality descriptions (much more than the contact information of the providers). The *Profile* was kept also to support sharing and reuse of existing standards (requirement R3).

API Description

The *API Description* ontology offers a framework for semantically describing the functionality offered by methods of APIs (e.g., *AddData*, *RemoveData*) and accordingly several types of APIs (e.g., *StoreAPI, InferenceAPI*). As such, it complements the OWL-S *Profile* for our purposes.

The ontology's central concept, called *APIDescription*, can have multiple *hasMethod* properties for instances of type *Method* (see Figure 5.5). Furthermore, each instance of *Method* has a set of parameters such as inputs, outputs, preconditions and effects. Each parameter features a *hasType* property which points to a concept in a domain ontology. Types of *Methods* and *APIDescriptions* are specialized in terms of domain concepts. This kind of information is used to perform the task of discovering available APIs according to their offered functionality (methods), to classify new APIs (and methods) and to derive OWL-S descriptions for the corresponding Web services (scenarios S3, S4 and S5). Requirement R2 motivates such semantic functionality descriptions.

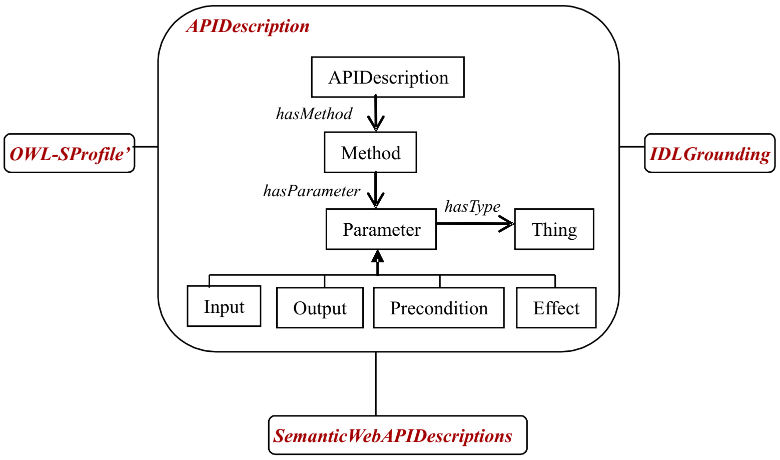


Figure 5.5: The *APIDescription* ontology.

Implementation

The *Implementation* ontology, depicted in Figure 5.6, contains implementation level details of a module and thus responds to requirement R1. There are two aspects of the implementation:

- *CodeDetails* describe characteristics of the code, such as the class that implements the code, the required archives or the version of the code. All these aspects are modelled as properties of the *CodeDetails* concept. Note that these characteristics are specific for a certain implementation and therefore not reusable. They are used during automatic deployment of the components (Scenario S1).
- the signature of the interface. The name of the methods and their parameters are modelled using the ontology presented next (*IDL*).

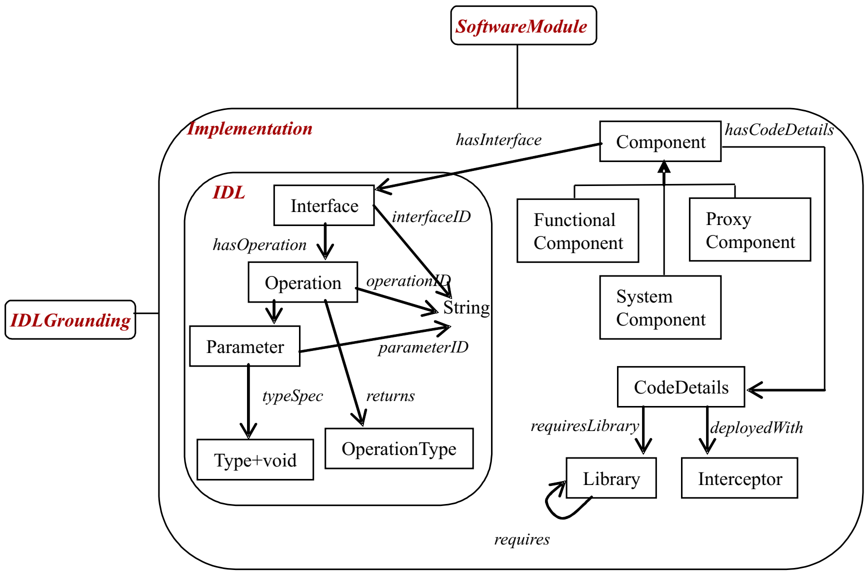


Figure 5.6: The *Implementation* ontology.

The main concept, *Component* (which is a subclass of *SoftwareModuleImplementation*) bundles an instance of *CodeDetails* and an instance of *Interface* (the class which describes the signature of the API).

IDL

We formalized a small subset of the IDL (Interface Description Language (Object Modelling Group, 2002)) specification into an ontology that allows describing signatures of interfaces. The *Interface* concept corresponds to a described interface. It features the property *hasOperation* which points to an *Operation* instance. Each *Operation* can have a set of (input) parameters of a certain type. Also each *Operation* returns an *OperationType* (which can also be void). *Interfaces*, *Operations* and *Parameters* have identifiers (which correspond to the names by which they are used in the code). This allows us to specify all the invocation and implementation details needed for automatic invocation of the methods (see requirement R1) using a widely used industry standard (therefore complying to R3).

IDL Grounding

The *IDL Grounding* ontology (see Figure 5.7) provides a mapping between the *API-Description* and the *Interface* description. The mapping is straightforward: the concepts *InterfaceGrounding*, *MethodGrounding*, *InputGrounding* and *OutputGrounding* map between respective concepts from the *API Description* and *Implementation* sub-ontologies.

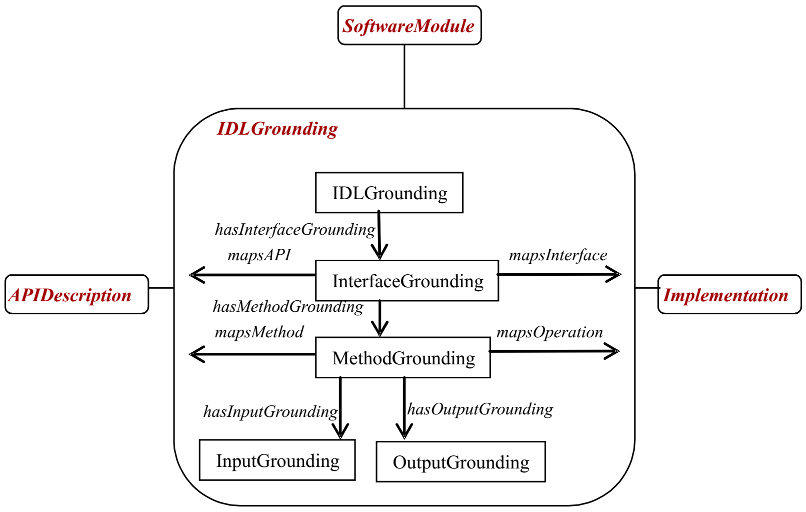


Figure 5.7: The *IDLGrounding* ontology.

We acknowledge the possibility of redundancy in our approach (given that both the *IDL* and the *API Description* ontologies look similar) but easy reuse and flexible coupling (see R2 and R3) were a higher design goal in this work. Namely, a certain concept level description can be grounded to many different interfaces that may look technically different, i.e. there might be other signatures.

Domain Ontologies

We built two domain ontologies that specialize parts of the generic ontology presented above. By isolating domain knowledge in separate sub-ontologies, we conform to the OWL-S design principles and implicitly requirement R4 (Domain Independence).

The first otology (*SemanticWebProfiles*) generically describes Semantic Web software modules. We based our ontology on the outcome of an extensive survey (Gomez-Perez, 2002) in this domain. The survey distinguishes several categories of software modules (ontology building modules, ontology evaluation modules etc.) and for each category proposes a set of characteristics. These characteristics are used in the survey as a framework for comparing the actual modules which are presented.

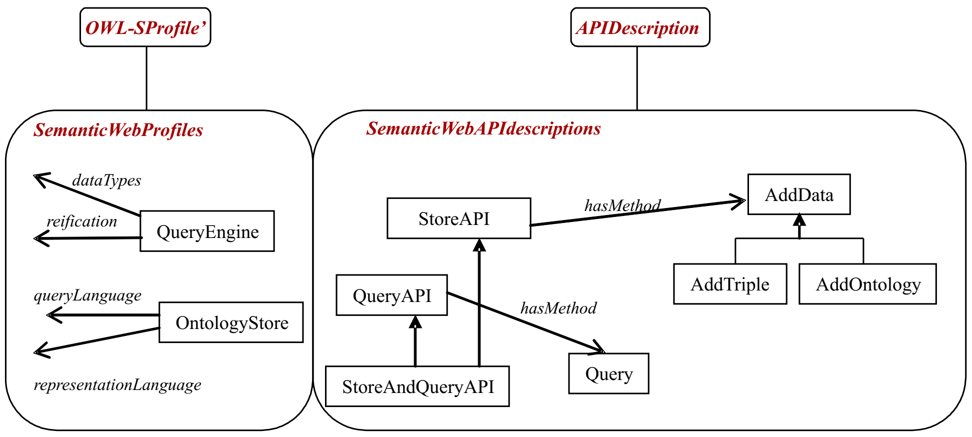


Figure 5.8: The domain ontologies.

We transformed this information in a domain ontology as follows. We built a taxonomy of categories according to the document. Each category became a subclass of *Profile*. The characteristics of each category were modelled as sub-properties of the OWL-S *serviceParameter*. For example, we created the *OntologyStore* category and added properties such as *queryLanguage*, *representationLanguage* as suggested by the survey. The additional properties (e.g., *QueryLanguage*) are all specializations of *serviceParameter*. We concluded that it was easy to extend OWL-S Profile (the *serviceParameter* property) for modelling the information in the survey. Also, this will allow easy addition of extra knowledge in the future, since the survey only offers a non-exhaustive, reduced set of characteristics.

The second ontology (*SemanticWebAPIDescription*) describes Semantic Web specific functionalities. It contains a set of API and functionality types (methods) which are generally offered. For example, we declared a *StoreAPI* concept, which denotes APIs for storing engines, and defined it as providing an *AddData* method (for adding data into the store) and a *Query* method (for querying the data in the store). Note that by combining simple APIs one can create complex ones. For example, a *StoreAndQueryAPI* will be obtained by inheriting methods both from a *StoreAPI* and a *QueryAPI*. Further, within a type of API, specializations can be created by declaring extra methods specializing the existing ones. The schema of this ontology is provided by the *API Description* ontology, where APIs are of type *APIDescription* and their functionalities (such as *AddData*) are of type *Method*. We believe that such an ontology will allow performing a flexible search over the existing APIs. Note that when building these domain ontologies we relied on the ontology that we built for describing Sesame (see Section 4.5.3).

5.4 Ontology Deployment

The ASSW ontology is incorporated in the application server[4]. Semantic descriptions of the registered components are stored in a central repository, called registry. These descriptions are then used to support a set of application server specific tasks. In this section we exemplify an actual description of a software component (in Section 5.4.1) and explain how semantic descriptions are used within the application server (Section 5.4.2).

5.4.1 An Example Component Description

We exemplify the use of the above described ASSW ontology for describing three ontology stores that use RDF(S) as their representation language. For more examples on describing software modules with this ontology the reader is referred to (Oberle et al., 2004b). We assume that all ontology stores are deployed to ASSW as functional components:

KAON RDF Main Memory is an ontology store that is transient, implementing the RDF API as used in the Karlsruhe Ontology and Semantic Web Toolsuite (KAON).

KAON RDF Server is an ontology store that implements the same API as above, however, it applies a database system for actual storage.

Sesame is an ontology store that implements its own API.

For the sake of brevity we only illustrate the description of two methods for each API (see below). Note that we omit fully qualified classnames to improve the readability. Also, we provide a complete semantic description only for the first component. For the other two components we only highlight how they differ from the first one.

```
KAON RDF API:
    void add(Statement statement)
    Model find(Resource subject, Resource predicate, RDFNode object)

Sesame API:
    int addDataFromUrl(String dataURL, String baseURL)
    String[][] evalRqlQuery(String query)
```

Describing KAON RDF Main Memory. We declare *KAONRDFMainMemory* as a *SoftwareModule* that links to three instances (*MM_Profile*, *KAONRDFAPIGrounding*, *MM_Impl*) containing different aspects of the description:

```
    <softwareModule:SoftwareModule rdf:ID="KAONRDFMainMemory">
        <damlservice:presents rdf:resource="#MM_Profile"/>
        <damlservice:supports rdf:resource="#KAONRDFAPIGrounding"/>
        <softwareModule:implements rdf:resource="#MM_Impl"/>
    </softwareModule:softwareModule>
```

From the profile point of view (*MM_Profile*) *KAONRDFMainMemory* is an *OntologyStore*, therefore we can describe all the associated properties which we declared in the Semantic Web Profiles domain ontology. The *Profile* also includes contact information and a pointer to the *APIDescription* (*RDFAPI*) instance.

[4]The implementation,“KAON SERVER”, is available at `http://kaon.semanticweb.org`

```
<swProfiles:OntologyStore rdf:ID="MM_Profile">
    <damlservice:presentedBy rdf:resource="#KAONRDFMainMemory"/>
    <damlprofile:serviceName> KAONOntologyStore</damlprofile:serviceName>
    <swProfiles:platform rdf:resource="swProfiles#Any"/>
    <swProfiles:ontologyLanguage rdf:resource="swProfiles#RDF"/>
    <owlsprofile':hasAPIDescription rdf:resource="#RDFAPI"/>
</swProfiles:OntologyStore>
```

Note that all the above information is specific to *KAONRDFMainMemory*. However, the *APIDescription* (*RDFAPI*) instance can be reused by other modules as well. In terms of our domain ontology the *RDFAPI* is a *StoreAndQueryAPI* since it offers both adding data (i.e., the *add* method) and querying the repository (i.e., the *find* method). The description of the API contains the declaration of the two methods (one of type *AddTriple*, the other of type *Query*) and their parameter, as follows:

```
<swApis:StoreAndQueryAPI rdf:ID="RDFAPI">

    <apiDescr:hasMethod>
        <swApis:AddTriple rdf:ID="RDFAPI_StoreTriple">
            <apiDescr:hasParameter>
                <apiDescr:Input rdf:ID="StatementForStore">
                    <apiDescr:hasType rdf:resource="swProfiles#OntologyStatement"/>
                </apiDescr:Input>
            </apiDescr:hasParameter>

            <apiDescr:hasParameter>
                <apiDescr:Output rdf:ID="NoOutput">
                    <apiDescr:hasType rdf:resource="swProfiles#NoOutput"/>
                </apiDescr:Output>
            </apiDescr:hasParameter>
        </swApis:AddTriple>
    </apiDescr:hasMethod>

    <apiDescr:hasMethod>
        <swApis:Query rdf:ID="RDFAPI_Query">
        ...
        </swApis:Query>
    </apiDescr:hasMethod>

</swApis:StoreAndQueryAPI>
```

Furthermore, we declare the technical details of the module (*MM_Impl*).

```
<impl:FunctionalComponent rdf:ID="MM_Impl">
    <impl:hasCodeDetails>
        <impl:CodeDetails rdf:ID="MM_CodeDetails">
            <impl:requiresLibrary rdf:resource="#someLibraryDecl"/>
            ...
        </impl:CodeDetails>
    </impl:hasCodeDetails>
    <impl:hasInterface rdf:resource="#KAONRDFInterface"/>
</impl:FunctionalComponent>
```

The syntactic description for the API (*KAONRDFInterface*) is shown below. A grounding instance (*KAONRDFAPIGrounding* - not presented here) establishes the correspondence between the conceptual *RDFAPI* description and the *KAONRDFInterface* description of the technical implementation.

```
<idl:Interface rdf:ID="KAONRDFInterface">
```

```
        <idl:interfaceIdentfier>
            edu.unika.aifb.rdf.api.model
        </idl:interfaceIdentfier>

        <idl:hasOperation>
            <idl:Operation rdf:ID="addOp">
                <idl:operationIdentifier>add</idl:operationIdentifier>
                <idl:hasParameter>
                    <idl:Parameter rdf:ID="statement">
                        <idl:parameterIdentifier>statement</idl:parameterIdentifier>
                        <idl:typeSpecification>
                            edu.unika.aifb.rdf.api.model.Statement
                        </idl:typeSpecification>
                    </idl:Parameter>
                </idl:hasParameter>

                <idl:returns>
                    <idl:OperationType rdf:ID="response">
                        <idl:hasType>void</idl:hasType>
                    </idl:OperationType>
                </idl:returns>
            </idl:Operation>
        </idl:hasOperation>

        <idl:hasOperation>
            <idl:Operation rdf:ID="queryOp">
                ...
            </idl:Operation>
        </idl:hasOperation>

    </idl:Interface>
```

Describing KAON RDF Server. The definition of the KAON RDF Server is very similar to that of KAON RDF Main Memory since they provide the same API. We provide the same type of descriptions and we can reuse the *RDFAPI* (the semantic description of the API), *KAONRDFInterface* (the syntactic description) and *KAONRDFAPI-Grounding* (the grounding between these two aspects). Only the *Profile* description has to be modified in order to describe that this server uses database storage.

Describing Sesame. When describing Sesame we cannot reuse any API related descriptions from the previous descriptions. We only present here the semantic description of Sesame's functionality (its *APIDescription*). Sesame provides other functionalities than the previous two modules, i.e., *AddOntology* and *Query*.

```
    <swApis:StoreAndQueryAPI rdf:ID="SesameAPI">
        <apiDescr:hasMethod>
            <swApis:AddOntology rdf:ID="SesameAPI_LoadOntology">

                <apiDescr:hasParameter>
                    <apiDescr:Input rdf:ID="OntologyForLoad">
                        <apiDescr:hasType rdf:resource="swProfiles#Ontology"/>
                    </apiDescr:Input>
                </apiDescr:hasParameter>

                <apiDescr:hasParameter>
                    <apiDescr:Output rdf:ID="LoadedStatements">
                        <apiDescr:hasType rdf:resource="swProfiles#LoadedStatements"
                </apiDescr:Output>
            </apiDescr:hasParameter>
        </swApis:AddOntology>
    </apiDescr:hasMethod>

    <apiDescr:hasMethod>
        <swApis:Query rdf:ID="SesameAPI_Query">
        ...
        </swApis:Query>
    </apiDescr:hasMethod>

</swApis:StoreAndQueryAPI>
```

5.4.2 Using Component Descriptions

Currently, semantic descriptions support two main scenarios as introduced in Section 5.2.2. First, "implementation details" (scenario S1) such as loading components are carried out automatically by using (1) the implementation details in each description (according to the *Implementation* ontology) and (2) the reasoning capabilities of the server. For example, the transitive closure of all required libraries by a certain component can be deduced. This and other uses of the semantic descriptions are presented in (Oberle et al., 2004a).

Second, "component discovery" (scenario S2) is supported at run-time. Therefore, the client can (1) query the registry for available components, (2) chose a component from the returned list (based on its properties) and (3) use that component. For example, OilEd (Bechhofer et al., 2001), an ontology editor acting as a client, can query for existing ontology stores or reasoners, and then select and interact with any of the available components (as described in more detail in (Oberle et al., 2004b)). Note that this situation is superior to traditional application servers where the relation between different clients has to be hard-coded.

5.5 Related Work

Classical Software Reuse Systems are comparable to our work in that they also need to describe software modules appropriately for efficient and precise retrieval. Techniques like the faceted classification (Diaz, 1991) are limited to the representation of the provider's features. Analogical software reuse (Massonet and van Lamsweerde, 1997) shares a representation of modules that is based on functionalities achieved by the software, roles and conditions. Zaremsky and Wing (Zaremski and Wing, 1997) describe a specification language and matching mechanism for software modules. They allow for multiple degrees of matching but consider only syntactic information. UPML, the Unified Problem-solving Method Development Language (Fensel et al., 1999), has been developed to describe and implement intelligent broker architectures and components to facilitate semi-automatic reuse and adaptation. It is a framework for developing knowledge-intensive reasoning systems based on libraries of generic problem-solving components that are represented by inputs, outputs, preconditions and effects of tasks. Note that these efforts either describe very different kinds of components or concentrate solely on syntactic or semantic descriptions without blending them together.

Another body of related work are adaptations of OWL-S to particular domains. For example, (Wroe et al., 2003) uses an extension to OWL-S for describing Web services in the bio-informatics domain. OWL-S is enriched with speech-acts when describing agent based Web services in (Gibbins et al., 2003). However, none of them actually considers software description at the API level.

IDL is augmented with concepts specified in Description Logics by (Borgida and Devanbu, 1999). More specifically they consider adding the following kinds of information to an IDL interface: a) data invariants (particularly useful for database-like integrity constraints, b) procedure pre- and post-conditions, c) object behavior models of dynamics. Thus, in addition to checking for exact matching of the information between client and server, the formal assertions and Description Logic (DL) reasoning allow them to perform additional tasks. Among them, (Borgida and Devanbu, 1999) list *Compatibility testing of the specifications*, *Local consistency checking*, *More thorough treatment of exceptions* and *Variability in services provided*. However, this approach just augments the syntactic part of an API's description. It does not deal with semantic information about a method's functionality like our approach.

5.6 Summary

In this chapter we investigated the adaptability of OWL-S in the context of a particular application server which supports development of complex Semantic Web applications. As a proof of concept we built an ontology that adapts OWL-S to meet the requirements of the application server. The newly built ontology has already been incorporated in the server and used for automating implementation tasks and component discovery (scenarios S1 and S2).

Our conclusion is that OWL-S incorporates several valuable design principles that make it easily adaptable for describing other types of software entities than Web services. We believe that adaptability is an important features of generic ontologies and it should characterize other ontologies used for Web service descriptions as well.

Chapter 6

Aligning OWL-S to a Foundational Ontology

In previous chapters we analyzed the expressivity, usability and adaptability of OWL-S. In this chapter we analyze OWL-S from an ontological perspective. We identify its problematic aspects and suggest enhancements through alignment to a foundational ontology. Another contribution of our work is the Core Ontology of Services that tries to fill the conceptual gap between the foundational ontology and OWL-S. This ontology can be reused to align other Web service description languages as well, thus contributing to their harmonization.

The material of this chapter has been first published at the AAAI Spring Symposium entitled the First International Semantic Web Services Symposium (Mika et al., 2004b), then an extended version was published at the Thirteenth International World Wide Web Conference, 2004 (Mika et al., 2004a).

6.1 Introduction

Clarity in semantics and a rich formalization of this semantics are important requirements for ontologies designed to be deployed in large-scale, open, distributed systems such as the envisioned Semantic Web. This is due to the fact that ontologies should facilitate mutual understanding, either for enabling effective cooperation between multiple artificial agents, or for establishing consensus in a mixed society where artificial agents cooperate with human beings. Foundational ontologies fulfill those requirements because they serve as a starting point for building new domain and application ontologies, provide a reference point for easy and rigorous comparisons among different ontological approaches and create a framework for analyzing, harmonizing and integrating existing ontologies and metadata.

Clarity in semantics together with a rich formalization are especially important for ontologies describing Web services (such as OWL-S) because they enable complex tasks involving multiple agents. Our contribution to the development of OWL-S is to identify some of its problematic aspects and to suggest enhancements through alignment to a foundational ontology. We found that OWL-S suffers conceptual ambiguity, lacks concise axiomatization, is designed too loosely and offers an overly narrow view on Web Services.

Through our alignment, we discovered possible enhancements to these problematic aspects of the ontology. We present these findings for the benefit of the designers and users of OWL-S. Furthermore, a Core Ontology of Services is developed as a middle layer which can also be used for aligning other (Web) service description languages. Lastly, we note that the contribution of our work is not only limited to the concrete results reported in this chapter, but also consists of (1) examples of the benefits of alignment to foundational ontologies and (2) a description of the alignment method itself.

This chapter is structured as follows. We begin with related work in Section 6.2. In Section 6.3 we identify and explain several problematic aspects of OWL-S. In Section 6.4 we present the main body of work, the alignment of OWL-S to the DOLCE foundational ontology. In Section 6.5 we detail our suggested improvements to the problematic aspects introduced before. We conclude in Section 6.6.

6.2 Related Work

Previous efforts responded to some of the problems of OWL-S. We briefly discuss the two initiatives we are aware of by describing their motivation, the parts of OWL-S they focus on, the techniques they use as well as some initial results (when available).

The first initiative (Narayanan and McIlraith, 2003) is motivated by the need of formal semantics to describe, simulate, automatically compose, test and verify Web service compositions. It focuses solely on the OWL-S *ServiceModel* which provides all the constructs for specifying composition. The authors establish a situation calculus semantics for the main elements in the OWL-S *ServiceModel* (e.g., atomic and composite processes, conditional effects and outputs), then translate it to the operational semantics provided by Petri Nets. This knowledge representation formalism has a rich theoretical and tool support for the various composition tasks. Indeed, this semantics allowed to re-use an existing simulation and modelling environment. Further, the authors were able to identify more tractable subsets of OWL-S (less expressive but more efficient analysis for verification, composition and model checking).

The second effort (Ankolekar et al., 2002b) also focuses only on the OWL-S *Service-Model* and proposes a concurrent operational semantics that incorporates subtype polymorphism. The motivation for this work is to provide an initial reference semantics that would discover any possible ambiguity in the developed language. It would also serve for developing techniques for automated verification of OWL-S models. Finally, if other Web standards would provide a similar semantics it would be much easier to compare them and to understand their strengths and weaknesses. The authors of both efforts mutually acknowledge the similarity between the two proposed semantics, except some minor details discussed in (Ankolekar et al., 2002b).

Besides aiming at increased formal axiomatization, we wish to explain the OWL-S concepts in terms of a foundational ontology which reflects several generally accepted theories from linguistics, philosophy, cognitive sciences etc. We show that this "ontological" analysis of OWL-S also brings to surface several irregularities in the model (just like the reference semantics promises to do). Further, one of the long term benefits of alignment is that it allows a comparison between several aligned ontologies (a goal also stated in (Ankolekar et al., 2002b)). We extend our analysis to the entire OWL-S model. From a methodological perspective, the previous approaches provide independent reconstructions of OWL-S, while, through alignment, we embed the OWL-S model in the larger context offered by the foundational ontology. Therefore we can deduce, for example, that OWL-S does not address the difference between a real life object (e.g., book) and its representational counterpart in an information system (e.g., ISBN number), an important ontological distinction. Finally, the semantics established by previous work are not reflected in the current OWL formalization of the model. In our case, the model inherits the axiomatization available for the OWL-DL version of DOLCE.

6.3 Problematic Aspects of OWL-S

In this section we identify and illustrate some of the problematic aspects of understanding OWL-S from a foundational perspective. We revisit most of them when discussing some of our suggestions for improvements in Section 6.5. We also relate these issues to the question of ontology quality.

Ontology quality is the topic of (Borgo et al., 2002), which provides (among others) three criteria for evaluation: *extensional coverage* (concerning the amount of entities that are supposed to be described by an ontological theory), *intensional coverage* (concerning what kinds of entities are described by an ontological theory), and *precision* (concerning what axioms are required to describe just the models the ontology designer intends to cover). According to these criteria, a good ontology should approximate the domain of discourse that is supposed to be described, it should have a signature that maps all the kinds of entities intended by the designer, and it should axiomatize the predicates in order to: 1) catch all the intended models, and 2) exclude the unintended ones.

Below we introduce four problems encountered in OWL-S. The first one (conceptual ambiguity) features both insufficient intensional coverage and overprecision. The second and the third (poor axiomatization and loose design) are cases of insufficient precision. In the third problem, the weakness is mainly inherited by limitations of OWL expressivity. The fourth (narrow scope) is a case of both extensional and intensional coverage.

6.3.1 Conceptual Ambiguity

Since there is no clear conceptual framework behind OWL-S (as we concluded in Section 4.6), it is often difficult for users to understand the intended meaning of some concepts, the relationship between these concepts as well as how they relate to the modelled services. Many concepts are still being clarified both within the OWL-S coalition and in public mailing lists. In addition, the Web Services Architecture (WSA) Working Group of the W3C introduced an OWL ontology of Web service concepts that seems to be independent of OWL-S[1]. This probably leads to the necessity of an alignment between the two ontologies, which needs an explanation of the respective assumptions.

Conceptual ambiguity affects particularly the upper level of OWL-S shown in Figure 2.6. The notion of a service is introduced in (Martin et al., 2003) as follows: *"By 'service' we mean Web sites that do not merely provide static information but allow one to effect some action or change in the world, such as the sale of a product or the control*

[1]`http://www.w3.org/2004/02/wsa/`

of a physical device". Later, we read that *"any Web-accessible program/sensor/device that is declared as a service will be regarded as a service"*.

However, neither of these definitions are operationalized as neither the concept of a "Web site" nor the "Web" appears in the ontology. Instead, the notion of a service is characterized solely by its relationship to a number of *ServiceProfiles*, at most one *ServiceModel* and any number of *ServiceGroundings*, which is not sufficient to understand the concept of *Service* considered by OWL-S.

We note that the term Web service and closely related terms (e-Service, Service, etc.) also suffer from overloading. In our search for possible formalizations, we found a variety of definitions emphasizing different aspects of a service (Gangemi et al., 2003): offering functionality (usefulness for a particular task), interoperability using standards or providing an interface to an existing system. We also refer the reader to (Baida et al., 2004), which compares and contrasts the definitions used in the business literature, in software engineering and in information sciences.

6.3.2 Poor Axiomatization

OWL-S descriptions should be machine processable. Hence, it is important that each concept is characterized by a rich axiomatization in order to support meaningful inferences. In general, we believe that the level of commitment in OWL-S will need to be raised if it shall support the complex reasoning tasks put forward by the coalition.

Unlike the issue mentioned in the previous section, poor axiomatization reflects the lesser problem when the definition of concepts is clear, but axiomatization in the ontology itself needs improvement. In many respects, OWL-S shows the characteristics of a typical application ontology: there is no firm concept or relation hierarchy (most concepts and relations are direct subconcepts of the top level concept or relation) and several relations take *owl:Thing* as their domain or range.

We propose that by adding foundations to OWL-S, the level of axiomatization can be increased. Alignment to a foundational ontology means relating the concepts and relations of an ontology to the basic categories of human cognition investigated by philosophy, linguistics or psychology. This approach has the advantage that restrictions on the level of common sense are inherited by the concepts in the application ontology. This prompts the ontology engineer to sharpen his notions with respect to the distinctions made in the foundational ontology. It also promotes reuse by highlighting commonalities, which especially helps to reduce the proliferation of relations - a typical phenomenon for application ontologies.

Alignment to a well-modularized foundational ontology also allows to selectively import theories from the ontology such as mereology, time theory etc. We will demonstrate this in Section 6.5 when aligning the control constructs of OWL-S to the Ontology of Plans which is one of the basic extensions of the DOLCE foundational ontology.

6.3.3 Loose Design

A further problematic aspect of OWL-S from an ontologist's point of view is its entangled design. At the heart of this problem lies the purpose of OWL-S in providing descriptions of various views on Web services required to support a number of different service related tasks (discovery, composition, invocation). Besides the functional dimension, Web service descriptions should be contextualized to represent various points of view on a service, possibly with different granularity.[2] Most of these views, however, are overlapping in that they concern some of the same attributes of a service.

A straightforward modularization in such cases results in an entangled ontology, where the placement of certain knowledge becomes arbitrary and intensive mapping is required between modules. This phenomenon is well described in object-oriented design, where the notion of *aspects* (Elrad et al., 2001) was recently proposed to encapsulate concerns that cross-cut the concept hierarchy of a software.

A case in point is the application of attribute binding in OWL-S. The construct of attribute binding is necessary in OWL-S to express, for example, that the output of one process is the input for another process or that the output of a composite process is the same as the output of one of its subprocesses. In programming, such equivalences are expressed by the use of *variables*. Variables are governed by the rules of *scoping*, which define the boundaries of commitment.

Since OWL lacks the notion of variables, argument binding is expressed by explicit value maps. As shown in Figure 6.1, the value map has the form of a *List*, attached to a *ProcessComponent*. This *List* should contain instances of the *ValueOf* concept as members[3]. Each *ValueOf* concept should point to a single relation of a single concept by using *theParameter* and *atProcess* relations[4]. For example, in case of two processes A and B where process B takes the output of process A as an input, the list would have two *ValueOf* members, one related to concept A and the output relation, while the other would be related to concept B and its input relation.

The reader may also note that the intended meaning of the entire construct, namely that all 'sensible' instantiations of the process should respect the equivalences expressed in the value map, is not encoded in the axiomatization. This is explained by the lack of expressivity of the Description Logic used.

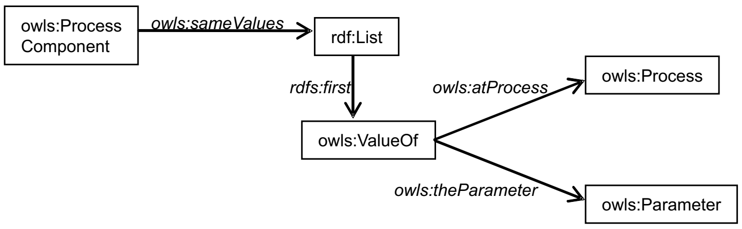


Figure 6.1: The representation of attribute binding in OWL-S.

Besides a tedious representation, an unfortunate consequence of the present solution is that we can only guess about the scope of the commitment represented by the value map. OWL-S seems to suggest attaching the value map to the process whose sub-processes are involved in the value map. As argued above, however, there could be

[2]The OWL-S specification mentions the ability to use the *Profile* for providing such views. However, no actual constructs are provided to map them to possible service executions or to each other.

[3]However, this is not enforced. There's also no explanation given why an ordered collection is used, i.e. what the ordering means.

[4]The cardinality restrictions are missing from the formalization.

multiple value map restrictions on the inputs/outputs of a process resulting from service composition (expanding/collapsing processes). Taking the current OWL-S proposal, it is unclear how one could approach such a situation.

6.3.4 Narrow Scope

Typically, service descriptions cross the boundary between an information system (with objects such as a record about a book) and the external world (with objects such as the physical book). The reason is that the Web services are only a part of the overall service to which a value is attributed by the requester. We believe that this phenomenon will characterize most real world services, where users are paying not simply for their information being recorded and manipulated, but for the overall process, which includes actual changes and effects in the real world, such as a book being delivered. Therefore, the scope of OWL-S needs to be extended to represent real world services that naturally cross the lines between information systems and the physical world.

While OWL-S acknowledges this aspect of services, it is unclear how a distinction could be made between the objects and events within an information system (regarding data and the manipulation of data) and the real world objects and events external to such a system. Using a foundational ontology, however, it is possible and even required for the creator of a description to make such distinctions, because they fundamentally affect the ontological nature of the objects and events concerned. We return to this issue in Section 6.5.4.

Besides its insufficient intensional coverage, the OWL-S core also shows an over-commitment in precision: the top *Service* concept is related to the *ServiceModel* concept with a cardinality 1:1. This means that for each *Service*, only one *ServiceModel* is expected to hold. This prevents us to consider alternative *ServiceModels*, or to evaluate the relationship between a *ServiceModel* required by a customer's guideline, or by a legal regulation, and the one underlying the provider's system, for instance.

A further contribution of our work is to extend OWL-S with relationships for mapping between service descriptions and the elements of actual service executions, which are not yet covered by OWL-S. These relationships will be directly inherited from the Descriptions & Situations ontology, another module of DOLCE, which is introduced in Section 6.4.2.

6.4 Alignment

In this Section we describe the alignment of OWL-S to the DOLCE foundational ontology (explained in Section 6.4.1). DOLCE is extended by an ontology of Descriptions & Situations further detailed in Section 6.4.2. As the conceptual gap between OWL-S and Descriptions & Situations is too large, we constructed a Core Ontology of Services (Section 6.4.3). Section 6.4.4 depicts how OWL-S concepts are to be expressed by using the Core Ontology of Services. We give a short summary of this alignment methodology in 6.4.5.

6.4.1 DOLCE

The role of foundational ontologies is to serve as a starting point for building new ontologies, to provide a reference point for easy and rigorous comparisons among different ontological approaches, and to create a foundational framework for analyzing, harmonizing and integrating existing ontologies and metadata standards. They are conceptualizations that contain specifications of domain independent concepts and relations based on formal principles derived from linguistics, philosophy, and mathematics.

DOLCE, a Descriptive Ontology for Linguistic and Cognitive Engineering, belongs to the WonderWeb project Foundational Ontology Library (WFOL) and is designed to be minimal in that it includes only the most reusable and widely applicable upper-level categories, rigorous in terms of axiomatization and extensively researched and documented (Gangemi et al., 2002; Masolo et al., 2003).

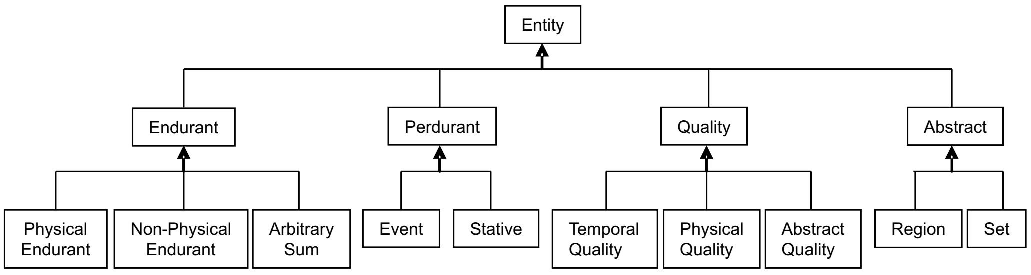


Figure 6.2: The top-level taxonomy of DOLCE.

The upper part of DOLCE's taxonomy is sketched in Figure 6.2. DOLCE is based on a fundamental distinction between enduring and perduring entities. The main relation between *Endurants* (i.e., objects or substances) and *Perdurants* (i.e., events or processes) is that of participation: an endurant "lives" in time by participating in a perdurant. For example, a person, which is an endurant, may participate in a discussion, which is a perdurant. A person's life is also a perdurant, in which a person participates throughout its duration. *Qualities* can be seen as the basic entities we can perceive or measure: shapes, colors, sizes, sounds, smells, as well as weights, lengths or electrical charges. Spatial and temporal qualities encode the spatio-temporal attributes of objects or events. Finally, *Abstracts* do not have spatial or temporal qualities, and they are not qualities themselves, e.g., (quality) regions or sets. In particular, regions are used to encode the measurement of qualities as conventionalized in some metric or conceptual space.

DOLCE is axiomatized in a modal logic (S5), but it is maintained also in other languages, used according to the particular trade-off between expressivity and computational complexity that is required by a certain application. For example, the KIF version is suited for most detailed meaning negotiations and for machine-readability of the complete axiomatization. The Loom (MacGregor, 1991) version has been used until recently to support ontology-driven industrial applications that required both high expressivity and classification services; in this version, some modal and temporal axioms have been removed or transformed, in order to take advantage of the Loom variety of description logic (which is incomplete, but desirable in certain settings). The OWL-DL (McGuinness and van Harmelen, 2004) version is currently maintained for Semantic Web applications. It probably provides the best scaled version due to the completeness of OWL-DL[5]. The strategy applied in porting DOLCE to different languages is quite liberal, and consists in finding the most appropriate naming policy and constructs that sound natural within the best modelling practices for a certain language, provided that the subsumption hierarchy and the axioms have an accurate mapping to the reference S5 version.

Although out of the scope of this chapter, we should mention that DOLCE has been chosen as basis for several reasons, some due to its internal structure (rich axiomatization, explicit construction principles, careful reference to interdisciplinary literature, common sense-orientedness, etc.), others due to its modular nature. In fact, being part of the WonderWeb Foundational Ontology Library, DOLCE will be mapped to other foundational ontologies (possibly more suitable for certain applications), and will be extended with many modules covering different domains (e.g., legal and biomedical), problems (e.g., planning, contexts), and lexical resources (e.g., WordNet-like lexica). These features (internal consistency and external openness) make DOLCE specially suited for our needs.

6.4.2 Descriptions & Situations

While modelling physical objects or events in DOLCE is quite straightforward, intuition comes to odds when we want to model non-physical objects such as social institutions, plans, organizations, regulations, roles or parameters. This difficulty is due to the fact that the intended meaning of non physical objects results from statements, i.e., their meaning emerges only in the combination of other entities. For example, a norm, a plan, or a social role are usually represented as a set of statements and not as a concept. On the other hand, non physical objects may change and be manipulated similar to physical entities, and are often treated as first-order objects. That means an ontology should account for such objects by modelling the context or frame of reference on which they depend. The representation of context is a common problem in many realistic domains from technology and society (such as law or finance) which are full of non physical objects.

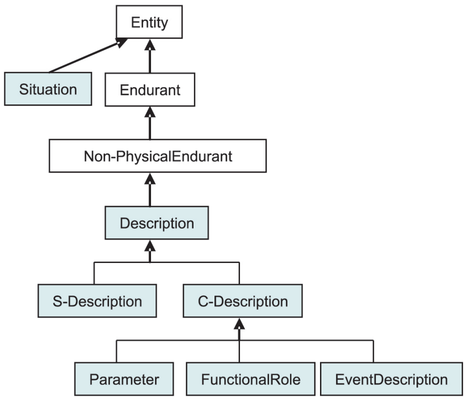


Figure 6.3: Aligning D & S to DOLCE.

[5]The OWL-DL version of DOLCE can be found at `http://www.loa-cnr.it/DOLCE.html`

In order to respond to those modelling requirements the Descriptions & Situations (D & S) (Gangemi et al., 2003) module of DOLCE was developed. D & S results to be a theory of ontological contexts because it is capable of describing various notions of context or frame of reference (non physical situations, topics, plans, beliefs, etc.) as entities. It features a philosophically concise axiomatization.

As depicted in Figure 6.3, D & S introduces a new category *Situation* that reifies a state of affairs and involves entities of the ground ontology (in our case DOLCE). A *Situation satisfies* a *Situation Description (S-Description)*, which is aligned as a dolce:Non-Physical Endurant and is composed of descriptive entities (*C-Descriptions*), i.e., *Parameters*, *Functional Roles* and *EventDescriptions*. Axioms enforce that each descriptive component links to a certain category of DOLCE: *Parameters* are *valued-by Regions*, *Functional Roles* are *played-by Endurants* and *EventDescriptions satisfies Perdurants* (see Figure 6.4).

S-Descriptions can be used to model contexts, for example, a murder (*Situation*[6]) that has been reported by a witness (*FunctionalRole*), which is played-by a person (*Endurant*), in a testimony (*S-Description*). The same situation may be interpreted according to other, alternative descriptions. This captures that multiple overlapping (or alternative) contexts may match the same world or model, and that such contexts can have systematic relations among their elements.

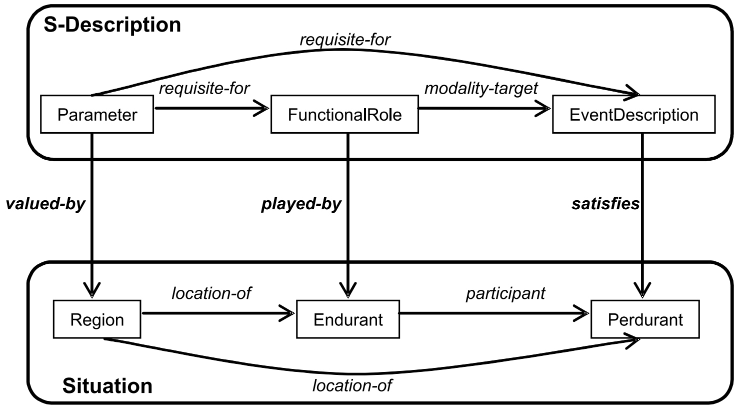


Figure 6.4: Descriptions and Situations.

D & S shows its practical value when applied as an *ontology design pattern* for (re)structuring application ontologies that require contextualization. As we will see in the remainder of this section, this is the case when describing (Web) services.

6.4.3 A Core Ontology of Services

The descriptions of services show a clear contextual nature and are to be modelled as *Situation Descriptions* in the sense of DOLCE and Descriptions & Situations.[7] One may

[6]As mentioned above, a *Situation* refers to an arbitrary entity in DOLCE's domain.

[7]In the following, we will refer to DOLCE with its basic extensions, i.e. D & S, Ontology of Plans, etc., as DOLCE+.

only have to consider the number of different views that may exist on a service: the view of a service provider, that of the service requester or the legal view of a contract etc. The concepts used to formulate any given view are clearly separate from the actual objects they act upon and often independent from the concepts appearing in other views.

Different views on the service need not be equally detailed either. For example, commercial advertisements typically feature only selected characteristics of a service. The various views also naturally focus on different aspects of a service, which means that the descriptions may only be partially mapped to each other.

Instead of directly aligning OWL-S to Descriptions & Situations, we developed a Core Ontology of Services (COS) and aligned the OWL-S sources to this ontology. This two-stage alignment is a common technique when the conceptual gap between the source ontologies and the foundational ontology is large. The Core Ontology of Services also features a concise axiomatization and can be reused in other scenarios (e.g., purely commercial services). We depict it in Figure 6.5 showing how the concepts of COS relate to DOLCE and how the D&S pattern was implemented.

At the description level, we consider five frequently occurring descriptions of a service (specializations of *S-Description*), where each represents a separate viewpoint:

1. **Service Offering (Description).** The service offering is the viewpoint of the legal entity providing the service. Much like commercial advertisements, the service offering may not describe entirely how the service will be carried out. This can also be considered as a proposal for a contract (agreement) for a service.

2. **Service Request (Description).** This is the counterpart of the offering in that it comprises the expectations of the requester of the service. Requirements are often flexible, concerning only a subset of the tasks, roles and parameters of service activities (but might also contain others).

3. **Service Agreement (Description).** Once an agreement is reached between the provider and the requester of the service, their joint understanding regarding the service may be described in a service agreement. Agreement means an understanding of the service as providing some value to the requester, which may or may not be the same as the originally offered functionality of the service.

4. **Service Assessment (Description).** Typically, when an agreement is reached measures are taken to monitor, assess and control the execution of the service provided. Assessment concerns matching the service activities against the agreement.[8] Service assessment may be executed by a third party and may also involve aspects not even mentioned in the above three descriptions, e.g. the cleanness of a hotel room may be checked by looking for dust on the TV sets.

5. **Service Norms Description.** This is a description of the social conventions regarding the execution of a service, whether a written code of practice (ISO) or unwritten norm. This view is the basis for legal action once a service deviates from the norms in ways not foreseen in the agreement.

[8]In an ideal world such a function would be meaningless. In reality, contracts are incomplete, since it is difficult to imagine all possible outcomes flowing from the agreement. Also, violations and the resulting penalties are often accepted rather than adhering to the contract (a kind of control strategy).

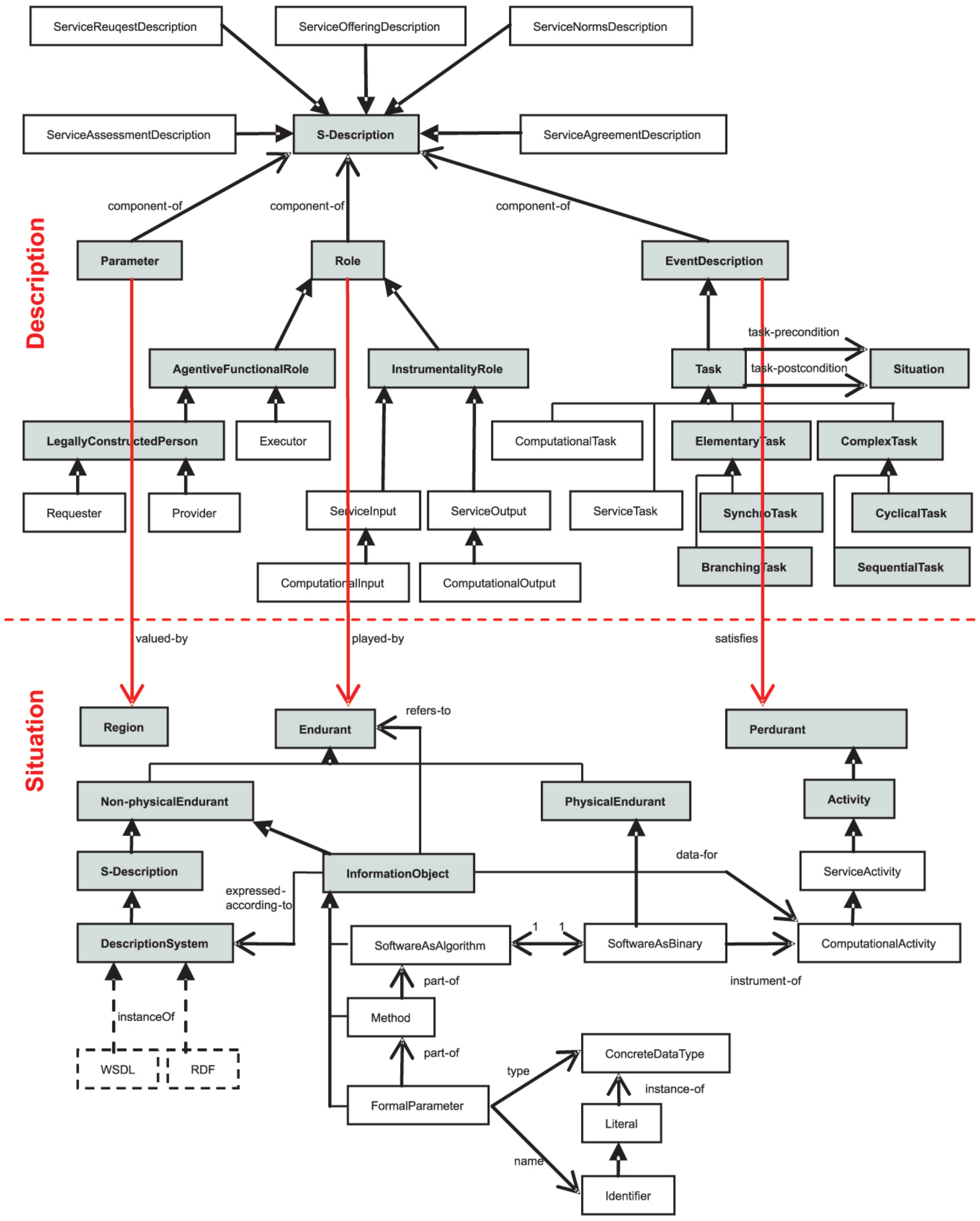


Figure 6.5: The Core Ontology of Services aligned to DOLCE/D&S. DOLCE/D&S concepts have a blue (dark) background. Note also how the D&S design pattern has been implemented.

We also introduce specializations of *EventDescription*, such as *Task*, *ServiceTask* and *ComputationalTask* - see Figure 6.5. This allows us to model activities in an information system and in the real world. Axioms ensure that *ServiceTasks* only describe*ServiceActivities* and that *ComputationalTasks* only sequence *Computational Activities*. The activities are new kinds of perdurants especially introduced here (see the lower part of Figure 6.5). A *ComputationalActivity* is a special kind of *ServiceActivity* which has only information objects or binary software as participants (*ComputationalActivity* is the activity produced by running the software). An example of a *ServiceActivity* would be flying Joe, a particular passenger, to his destination. An example of a *ComputationalActivity* would be the execution of the procedure that reserves a particular seat for a particular passenger.

The chief difference between tasks and activities is that of between a plan and a particular execution of the plan: a plan describes possible activities, constraining the way they should occur in order to achieve the plan's goals. Examples of *ComputationalTask* are the reservation of a flight and the collection of payment, as described in an information system, even if they may be implemented in a number of ways. A *ServiceTask* can be flying the passenger (some passenger, not a particular one) to some destination. Again, this may be carried out in several ways.

Our Core Ontology of Services may optionally take advantage of a number of concepts from the Ontology of Plans which is another module for DOLCE+. It allows the division of tasks into elementary and complex and the construction of complex tasks from elementary ones among other features.

Further axioms also ensure that only *InformationObjects* (a newly introduced *NonPhysicalEndurant*) participate in *ComputationalActivities*. *InformationObject* is a non-physical endurant in DOLCE, which may be expressed according to a *DescriptionSystem* such as RDF or WSDL. *SoftwareAsAlgorithm* is an *InformationObject*, while *SoftwareAsBinary* represents its physical counterpart (more specifically, *SoftwareAsBinary* is said to be the instrument of a *ComputationalActivity*, while *InformationObjects* are *data-for* the *ComputationalActivity*).

The Core Ontology of Services also models frequently occurring *FunctionalRoles* (see Figure 6.5). The *Requester* and *Provider* of a service are conceived as *LegallyConstructedPerson*s, an agentive legal role in DOLCE, while the *Executor* of a service is considered an agentive functional role without a legal nature. Examples are a passenger role (requester of the booking service) and the role of the travel agency (provider of the service).

Another group of roles is played by the instruments used in services. These include *(Computational) Inputs* and *Outputs*, formalized as *InstrumentalityRoles*. Our comprehensive axiomatization requires that, e.g., a *ComputationalInput* is only played by an *InformationObject*.

6.4.4 Aligning OWL-S to the Core Ontology of Services

In the following we describe the alignment of OWL-S to the Core Ontology of Services and our experiences with the process.[9]

[9]In the following concepts printed in italics are part of the OWL-S namespace, except when indicated otherwise.

The process of alignment proved its value early on by allowing us to quickly separate concepts of the ontology that had no clear and unique ontological interpretation with respect to the basic categories of DOLCE. For example, the *ValueOf* concept, which seems to be introduced for technical reasons (see Section 6.3.3). Similar arguments hold for the *ConditionalEffect* class, which models a ternary relationship between a process, a precondition and an effect. Much like *ValueOf*, this class is introduced for representation purposes, its real semantics are not captured by the ontology[10]. Similarly, the distinctions between *ServiceProfile* and *Profile* as well as *ServiceModel* and *ProcessModel* are introduced in OWL-S to provide flexibility in modelling, rather than representing conceptual differences. Although the definition of *Service* is ambiguous even in the natural text description of OWL-S, for the sake of argument we considered *Service* as a *ServiceOfferingDescription*, which has the *ServiceProfile* and *ServiceModel* (also a *ServiceOfferingDescriptions*) as parts. Note that the *ServiceProfile* just expands the *ServiceModel* by process descriptions. In our opinion there is no need to separate both and, e.g., have parameters like *serviceName* only in the *Profile* and not in the *ProcessModel*. However, our intention was to just align OWL-S rather than reorganizing it.

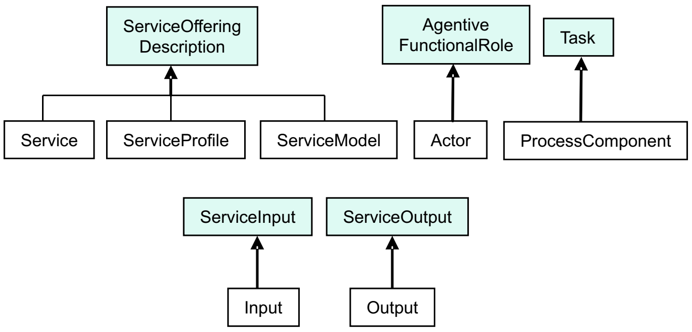


Figure 6.6: Aligning OWL-S to the Core Ontology of Services.

Relations like *serviceName* or *textDescription* regard the profile as a whole and are thus aligned with *ServiceOfferingDescription* as domain and literal as range. The notion of *Actor* in the *ServiceProfile* is aligned as an *AgentiveFunctionalRole* like depicted in Figure 6.6. The *ProcessComponent* concept was aligned to the *Task* concept of the Core Ontology of Services, while the individual control constructs were mapped to task components included from the Ontology of Plans. For example, the *Repeat-until* control construct was aligned to the *Cycle-UntilTask* concept, a kind of *CyclicalTask* with an exit condition and/or repetition interval. As another example, the *If-then-else* construct maps to the notion of an *AlternateTask*, a case-task with exactly two branches (both are not shown in Figure 6.6). Note that there is a difference, however, between the *ControlConstructs* of OWL-S and the *Task* types of the Core Ontology, because task types, like all other tasks, sequence activities themselves (branching and synchro tasks, in particular, sequence planning activities).

[10]This leaves open to interpretation, for example, the case when multiple conditional effects are given for a process.

The disambiguation of *Inputs, Outputs, Preconditions* and *Effects* (IOPE) was relatively straightforward using the Core Ontology. *Input* and *Output* are aligned to *ServiceInput* and *ServiceOutput*, respectively. On the other hand, the notions of *Precondition* and *Effect* are inherited from the Ontology of Plans where they are modelled as *Situations* and linked to their respective tasks using the *task-precondition* and *task-postcondition* relationships.[11] *ConditionalOutputs* and *ConditionalEffects* are modelled using the Case-Task construct.

We omitted the alignment of the grounding ontology for WSDL (Christensen et al., 2003) because it was not the focus of our work. Nevertheless, the notion of software tool is present in the Core Ontology of Services as an *InformationObject* that can be expressed according to any number of description systems.[12] WSDL could be such a description system and modelled to the extent required to express groundings.[13]

6.4.5 Summary

The ontology stack in Figure 6.7 summarizes our alignment effort. We used DOLCE as foundational ontology (6.4.1), extended it by the Descriptions & Situations module (6.4.2), defined our Core Ontology of Services (6.4.3), which was used to align OWL-S (6.4.4). Note that this methodology of alignment could be used to align and compare other service description efforts as well, e.g. the Web Services Architecture (WSA) or the ontology used within the Application Server for the Semantic Web (both alignments are detailed in (Gangemi et al., 2003)). Specialized domain and application ontologies of service descriptions such as (Richards and Sabou, 2003) are formulated according to one of these generic service ontologies.

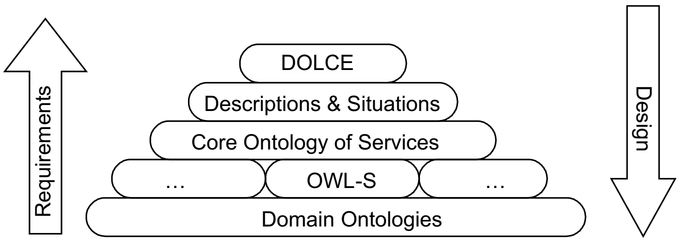


Figure 6.7: The stack of ontologies used in the alignment process.

Our method was a combination of a bottom-up and a top-down approach. On the one hand, ontologies in the lower layers provided representation requirements for the higher layers, which abstracted their concepts and relationships. On the other hand, the upper layers provided design guidelines to the lower layers. This also meant that although our

[11]For preconditions, this means that $task - precondition(p, c) \Leftrightarrow Process(p) \wedge PreCondition(pc) \wedge Condition(c) \wedge hasPrecondition(p, pc) \wedge preCondition(pc, c)$.

[12]A more refined representation we considered was to model *Software* as an S-Description, in the sense of an abstract algorithm.

[13]The Core Ontology of Services and the OWL-S alignment are available for download at `http://www.cs.vu.nl/~pmika/research/www2004/`.

goal was to preserve the structure of OWL-S as much as possible, our method suggested a rearrangement of the ontology based on the backbone provided by the D & S ontology.

6.5 Suggestions for Improvement

In this section we present suggestions for improvement of the problematic aspects of OWL-S, discussed in Section 6.3. Note that each subsection corresponds to the one introduced in Section 6.3.

6.5.1 Conceptual Disambiguations

The alignment to a foundational ontology helped us in understanding and crystalizing several concepts of OWL-S. As an example, ontological analysis explained the difference between an information object, its application domain counterpart and the role it plays in an information system (see also Section 6.5.4). This indicated possible enhanced modelling: since the same information object is modelled both in the *ServiceProfile* and *ServiceModel*, it is more logical to consider a single instance playing multiple roles. This improvement is already implemented by the OWL-S coalition as described in Section 4.3.

In our Core Ontology of Services, we went further to separate the functionality, process and software aspects of a service loaded onto the single concept of *Service* in OWL-S. It replaces *Service* with the concept of different kinds of *ServiceDescriptions*, which are *S-Descriptions* (a context in D & S) that envision a process as well as certain roles related to the individual tasks of the process. Inputs, outputs and abstract tools used to carry out a certain task are examples of roles. In case of information services, inputs and outputs are played by information objects and tools are played by particular software implementations. While this definition of a *ServiceDescription* may not be the only one, the fact that it is formulated according to a foundational ontology allows to compare it to alternative definitions and foster discussion on alternative conceptualizations of a (semantic) Web service.

6.5.2 Increased Axiomatization

A key advantage of the alignment to a foundational ontology is that it prompts the engineer to take a stance with respect to the principles established by the foundational ontology. What is typically gained is an increased understanding of one's own ontology and a richer axiomatization through ties to the foundational ontology. DOLCE mitigates the danger of overcommitment in this process (importing theories that are not used or not shared by the engineer) by extensive modularization along world views (3D, 4D, etc.) and domains (law, finance, etc.).

As an example, in the Core Ontology of Services we made use of an Ontology of Plans which includes subtypes of the generic Task concept for a detailed modelling of plans or process models. These constructs are directly comparable to the control constructs of OWL-S, but provide a higher level of axiomatization. An example of such types is DOLCE's *Synchro-Task* whose OWL definition is depicted in Appendix A. It matches the concept of "join" in the "Split-Join" control construct from OWL-S. A synchronization task is typically used to bind the execution of a "planning" activity rather than of a domain activity, since the referred activity is supposed to re-synchronize a process when it waits for the execution of two or more concurrent (or partly concurrent) activities.

Higher axiomatization is partly possible by the natural linkage to the Ontology of Time, another module for DOLCE, for describing (constraints on) temporal relations between process elements when they are executions of a plan. OWL-S would also need such an Ontology of Time and then it would be natural to adopt or reference an existing ontology instead of creating an ontology from scratch.

The Ontology of Plans also allowed to align relations such as *owl-s:components*, which is used to relate control constructs to their components. In OWL-S this relation is described merely as a subrelation of *owl:Property* with a domain of *ControlConstruct*. In our work, we aligned this relation to the *temporary-component* relation in DOLCE. The latter has a firm foundation as a subrelation of the more basic *component* (*functional proper-part*) mereological relation and *partly-compresent-with* temporally indexing relation, both characterized with formal restrictions on its application to other basic concepts, such as *Object*, *Description*, *Event*, etc.

6.5.3 Improved Design

In our work we propose to complement modularization in OWL-S with contextualization as a design pattern. Contextualization allows us to move from a monolithic process description of a service to the representation of different, possibly conflicting views with various granularity. The Descriptions & Situations ontology provides us the basic primitives of context modelling such as the notion of roles, which allows us to talk of inputs and outputs on the abstract level, i.e., independent of the objects that play such roles.

Using this pattern results in a more intuitive representation of attribute binding, with clearly defined semantics and scoping provided by Descriptions&Situations. Inputs and outputs can be modelled as *FunctionalRoles* (more precisely: *InstrumentalityRoles*), which serve as variables in our ontology. A single endurant — for example, a physical book — can play multiple roles within the same or different descriptions and thus it is natural to express that the given book is output with respect to one process, but input to another. Moreover, it is easier to represent the requirement that the input of a process *has to be* played by the same instance as the output of another process by putting constraints on the *objects* (and not the process or task) which play these roles (however, the expressivity required is the same and therefore goes beyond the power of OWL).

Besides a more intuitive representation, *FunctionalRoles* as components have an explicit scope, namely the *S-Descriptions* they belong to. Although not addressed in the present work, clearly defined limits in scope are necessary to describe semantic relationships among (service) descriptions, for example, to talk of conflicts between descriptions.

6.5.4 Wider scope

As we have seen before, Web services exist on the boundary of the world inside an information system and the external world. Except for the rare case of a pure information service, Web services carry out operations to *support* a real world service. Functionality, which is an essential property of a service, then arises from the entire process that comprises computational as well as real wold activities.

Web service descriptions are thus necessarily descriptions of two parallel worlds. In information systems, the world consist of software manipulating (representations of) information objects. While computational processes are running, in the real world books are being delivered to their destinations.

The connection between these worlds is that some of the information objects are representations of real world objects. Also, computational activities comprise part of the service execution in the real world. For example, an order needs to be entered by the Web agent into an information system, so that the warehouse knows which books to deliver to a given address.

The distinction between information objects, events and physical ones is not explicitly made in OWL-S. [14] Nevertheless, we believe that this distinction is important for disambiguating the nature of services in an open environment such as the Semantic Web.[15] In our work this separation naturally follows from the use of the DOLCE+ foundational ontology, where the distinction is an important part of the characterization of concepts. In particular, it makes possible to be more precise about the kinds of relationships that can occur among objects or between objects and events.

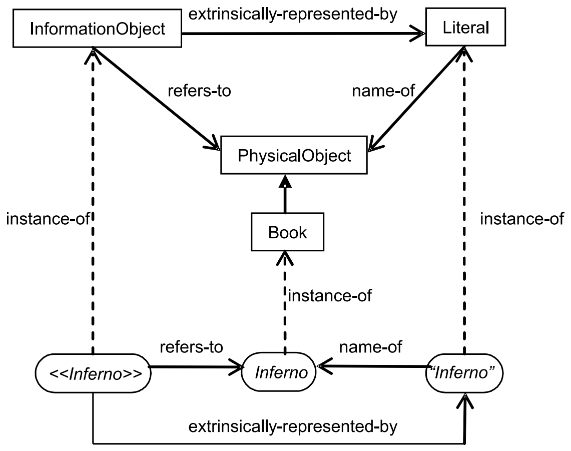


Figure 6.8: The relation between *InformationObjects* and *PhysicalObjects* in DOLCE.

For example, using DOLCE+ we can distinguish between a physical object (such as a book), an information object (such as the name of a book) and a representation of such information using a particular description system (e.g., a string encoding). The relations provided by DOLCE are shown in Figure 6.8.

The reader may note that by building on the Descriptions & Situations ontology design pattern, our work naturally extends OWL-S with the representation of service situ-

[14]Based on the examples so far, one may conjecture that OWL-S inputs and outputs concern physical objects relating to information objects such as message parts in WSDL through grounding.

[15]In fact, the lack of this distinction stands behind the emergence of the 'Semantic Web identity crisis' that results from the ambiguous use of identifiers in Semantic Web ontology languages such as RDF (Pepper and Schwab, 2003). In practice a URI can be used to reference a document on the Web, to reference (a fragment of) a document containing some definition of a concept or to represent a concept (without any intended reference to an actual location on the Web). Unfortunately, no standard scheme exists to distinguish between the three kinds of identifiers even though they need to be resolved in different ways.

ations. Service situations in our work correspond to a possible executions of a service. The description of service executions is already considered by the OWL-S coalition for the purposes of service execution monitoring. We believe that this direction should also be pursued by OWL-S as service requests are often formulated in terms of actual values of input/output parameters (or relatively narrow sets of parameter values). For example, customers of bookshops often have a clear idea of which book they want to buy or at least what kind of book it is. One could imagine an intelligent matching engine that in such case returns only services that offer a particular book or a category of books, instead of returning all known book selling services.

6.6 Conclusion

Our exercise of giving an ontological foundation to OWL-S is useful both for better understanding OWL-S and enriching it with additional formal semantics. We see the presented results as an example for the benefits of alignment to foundational ontologies as our methodology is applicable also to other standards. As a matter of fact, our Core Ontology of Services can be applied as a framework for harmonizing the ongoing efforts to characterize Web services, because it does not commit to a specific software design reference framework, and it is based on a generic, social notion of service. For example, the ontology of the Web Services Architecture (WSA) Working Group of the W3C, as well as other interesting methods for Web service deployment, such as problem-solving methods (Motta et al., 2003), can be interpreted (aligned, harmonized, or made interoperable) according to our reusable ontological components.

The alignment of OWL-S to the Core Ontology of Services also means that Web services described in OWL-S are automatically aligned to DOLCE. Such descriptions can be further enriched by adding DOLCE-based semantics (for example, spatio-temporal relations) to the domain concepts involved. We imagine this would allow a sufficiently sophisticated matching or composition engine to reason with the additional semantics in order to provide more targeted matches as a result. However, building such a tool is beyond the scope of our work.

One of the difficulties we encountered with our method of ontology alignment was that it required us to understand to some extent the principles of the foundational ontology. These principles stem from other sciences (philosophy, psychology, semiotics, communication theory etc.), which means that a (re)engineering of this kind requires a considerable intellectual investment from the knowledge engineer at the moment. We think, however, that this investment, materialized in the Core Ontology of Services, will pay off whenever new (Web) service ontologies are to be aligned or when a Web service ontology should communicate with domain ontologies or with workflow ontologies of the service actors, or even in the matching and composition of services that have overlapping domains or tasks. Such a pay off also shows why we have not just taken a reusable ontology, but a foundational one. A reusable ontology could have been used to carry out an analysis of OWL-S, but in order to gain access to conceptual alignment with other service, domain or task ontologies' reusability is not enough: we need an appropriate and flexible foundational ontology.

6.7 Summary

In this chapter we finalize our analysis of OWL-S by inspecting it from an in-depth, ontological perspective. We conclude that OWL-S exhibits severe problems such as ambiguity, poor axiomatization, loose design and narrow scope, thus failing to fulfill the **clear semantics and rich formalization** requirement. Nevertheless, many of these negative aspects can be solved by aligning OWL-S to a richly formalized and extensively researched foundational ontology. We use a stack of ontologies for the alignment made up of DOLCE, Descriptions & Situations as well as the Core Ontology of Services. Note that the alignment is not dependent on DOLCE, because Descriptions & Situations may be aligned to any foundational ontology. Parts of the service description that deal with service quality and assessment are left for future work.

The alignment to a foundational ontology is a time consuming and intellectually demanding activity. However, once the conceptual gap between the foundational ontology and the domain was bridged by the development of a core ontology (in our case the Core Ontology of Services), the alignment of other standards in the domain (e.g., WSMO, IRS) can be performed much easier thus reaching a cross-standard harmonization.

This chapter finalizes our analysis of the OWL-S generic Web service ontology. In the next part of the thesis we focus on solving the problem of acquiring Web service domain ontologies.

Part III

Learning Web Service Domain Ontologies

Chapter 7

A Framework for Learning Web Service Domain Ontologies

Generic and domain ontologies are equally important when building semantic Web service descriptions. In this part of the thesis we turn our attention to Web service domain ontologies. One of the major issues related to domain ontologies is that acquiring broad coverage domain ontologies is difficult and costly. In this chapter we describe some of the factors that hamper building high quality domain ontologies and conclude on some requirements for an automatic solution to this problem. Further, we present an ontology learning framework that addresses these requirements. Note that the novelty of our work is *not* in the used natural language processing methods but rather in the way they are put together in this generic framework specialized for the context of Web services. We end the chapter by providing some details about the implementation of our framework as a prototype system.

The material presented in this chapter is synthesized from several different publications. It is based on a paper published at the Third International Semantic Web Conference (Sabou, 2004b), a paper presented at the 14th International World Wide Web Conference (Sabou et al., 2005a), an article in the Journal of Web Semantics (Sabou et al., 2005b) and a workshop paper published in conjunction with the Fourth International Semantic Web Conference (Sabou and Pan, 2005). The content of the last section is a summary of an article published at the 9th International Conference on Information Visualization (Sabou, 2005b) and is based on previous work on visualization published in three book chapters (Fluit et al., 2002, 2004, 2005).

7.1 Introduction

An important role in semantic Web service descriptions is played by Web service specific domain ontologies. Despite their importance, few domain ontologies for Web service descriptions exist and building them is a challenging task. In this part of the thesis we address the problem of (semi-)automatically learning Web service domain ontologies. We report on the first stage of this work in which we aim to get a better understanding of the ontology learning task in the context of Web services and to identify potentially feasible technologies that could be used. Early in our work we learned that the context of Web services raises several issues that constrain the development of an ontology learning solution. We designed a framework for performing ontology learning in the context of Web services which addresses these issues in two ways. First, it exploits the particularities of Web service documentations to extract information used for ontology building. In particular, the sublanguage characteristics of these texts lead to the identification of a set of heuristics. These heuristics are implemented as pattern based extraction rules defined on top of linguistic information. Second, the learned ontologies are suited for Web service descriptions as they contain both static and procedural knowledge.

We implemented two learning methods that follow the basic principles of the framework but use different linguistic knowledge. The first method uses basic Part-of-Speech (POS) information and was developed and tested in the context of the WonderWeb[1] project (Sabou, 2004b). The second method uses deeper dependency parsing techniques to acquire linguistic knowledge. It was designed and tested on data sets provided by the *my*Grid project[2] (Sabou et al., 2005a). In this chapter we present the framework and both methods. In the next chapter we present a comparative evaluation of the two methods in the context of both projects.

This chapter is structured as follows. We start by analyzing the factors that hamper building Web service domain ontologies. We do this by describing the problem of domain ontology building in the context of the two research projects that served as case studies for developing and evaluating our framework (Section 7.2). Based on this analysis, we present an overview of the issues that constrain the development of an ontology learning solution in the Web services context (Section 7.3). Then, we present an ontology learning framework that deals with these constraints and the two concrete instantiations of this framework in Section 7.4. Implementation details of this framework in a prototype system are provided in Section 7.5.

7.2 The Problem of Building Web Service Domain Ontologies

In this section we describe the ontology building process as it took place in the context of two research projects: WonderWeb and *my*Grid. These projects offered realistic requirements, data sets and evaluation standards for our work. In both cases we detail (1) the kind of data sources used for ontology building and the (2) manually built domain ontologies. These manually built ontologies serve as Gold Standards when evaluating the automatically learned ontologies. We also (3) highlight the various difficulties that were encountered during building these domain ontologies.

The benefit of this analysis is twofold. First, these projects reveal some of the major aspects that make Web service ontology building difficult. These aspects prompt at the need of automating (at least to some extent) the acquisition of domain ontologies. The second benefit of the analysis is an overview of a set of issues that constrain the development of ontology learning methods in the context of Web services. These constraints, detailed in Section 7.3, guided us in the design of the ontology learning framework as discussed in Section 7.4.

[1]`http://wonderweb.semanticweb.org/`
[2]`http://www.mygrid.org.uk/`

7.2.1 Case Study 1: WonderWeb RDF(S) Storage Tools

Project description. The EU-funded WonderWeb research project aimed to develop an infrastructure for large-scale deployment of ontologies on the Semantic Web. The project's engineering infrastructure was provided by the KAON Application Server, a semantic middleware system which facilitates the interoperability of Semantic Web tools (Sabou et al., 2004). Ontologies that describe the functionality of Semantic Web tools and services are core to the architecture of this middleware. Since RDF(S) storage and query facilities are essential components of any Semantic Web application, they were the first ones to be integrated with KAON and thus required a domain ontology that would describe this domain. Besides WonderWeb, the ontology for describing RDF(S) storage functionality was also used in the AgentFactory project which performs configuration of semantically described Web services using agent-based design algorithms (Richards and Sabou, 2003; van Splunter et al., 2003).

Data Sources. While there are many tools offering ontology storage (a major survey (Gomez-Perez, 2002) reported on the existence of 14 such tools), only few are available as Web services (two, according to the same survey). Therefore, it is problematic to build a quality domain ontology by analyzing only the available Web services. However, since Web services are simply exposures of existing software to Web accessibility, there is a large overlap (often one-to-one correspondence) between the functionality offered by a Web service and that of the underlying implementation. Based on this observation, the domain ontology was manually built by analyzing the APIs of three RDF(S) storage tools (Sesame (Broekstra et al., 2002), Jena (McBride, 2002), KAON RDF API (Maedche et al., 2003)).

The data sources used during ontology building consisted of the *javadoc* documentation of all methods offered by these APIs. A javadoc documentation contains a general description of the method's functionality, followed by the description of its parameters, result types and exceptions to be thrown. See for example the javadoc documentation of the *add* method from the Jena API.

```
add
 Add all the statements returned by an iterator to this model.

Parameters:
 iter - An iterator which returns the statements to be added.
Returns:this model
Throws: RDFException - Generic RDF Exception
```

Manually built ontology. The manually built ontology contains 36 concepts distributed in two main hierarchies (see a snapshot of the ontology in Figure 7.1). The first hierarchy contains concepts that denote a set of functionalities offered by the analyzed APIs. These concepts are grouped under the *Method* concept which is similar in meaning to the OWL-S *Profile* concept. This hierarchy contains four main categories of methods for: adding data (*AddData*), removing data (*RemoveData*), retrieving data (*RetrieveData*) and querying (*QueryMethod*). Naturally, several specializations of these methods exist. For example, depending on the granularity of the added data, methods exist for adding a single RDF statement (*AddStatement*) or a whole ontology (*AddOntology*). Note that this hierarchy reflects a certain conceptualization and is *not* unique. Besides the *Method* hierarchy, the ontology also contains the elements of the RDF Data

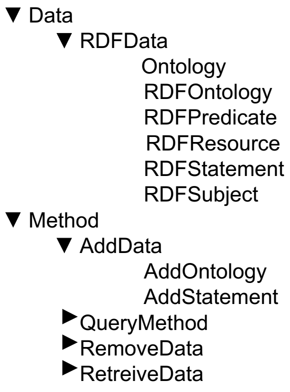


Figure 7.1: RDF(S) Storage ontology snapshot.

Model (e.g., *RDFStatement, RDFPredicate*) and their hierarchy, grouped under the *Data* concept.

The ontology is rich in knowledge useful for several reasoning tasks. For example, the definition of methods was enriched in multiple ways such as: imposing restrictions on the type and cardinality of their parameters or by describing their effects and types of special behavior (e.g., *idempotent*). The building of this manual ontology was a good indication that API documentations are rich enough to allow building Web service domain ontologies.

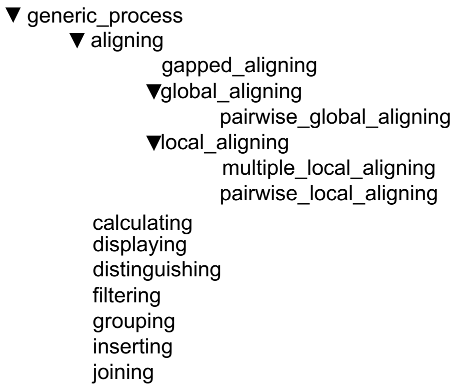
Encountered Problems. The major impediment in building a domain ontology for describing RDF(S) storage tools was the choice of data sources from which to build the domain ontology. Once the decision taken, it took three weeks (for one person) to build the ontology. This time includes the time to read and understand the API documentations as well as the time to identify overlapping functionalities offered by the APIs and to model them in an ontology.

7.2.2 Case Study 2: *my*Grid Bioinformatics Services

Project description. *my*Grid is a UK EPSRC e-Science pilot project building semantic grid middleware to support *in silico* experiments in biology. The experimental protocol is captured as a workflow, with many steps performed by Web services. Core to the infrastructure is an ontology for describing the functionality of these services and the semantics of the manipulated data. A key role of the ontology is to facilitate user driven discovery of services at the time of workflow construction. In contrast to efforts such as OWL-S and WSMO, the ontology is not currently intended to support workflow specification, agent-driven automated service discovery, automatic invocation, or monitoring.

Data Sources. The ontology was built manually initially using the documentation for 100 services as a source of relevant terms. These services are part of the EMBOSS

(European Molecular Biology Open Software Suite) service collection[3] and are further referred to as EMBOSS services. Each EMBOSS service has a detailed description containing (among others) a short description of the service, detailed information about its command line arguments, examples of the input/output file formats, its relation with other services in the collection or even references to scientific publications describing its functionality.



```
▼ generic_process
    ▼ aligning
          gapped_aligning
        ▼global_aligning
              pairwise_global_aligning
        ▼local_aligning
              multiple_local_aligning
              pairwise_local_aligning
      calculating
      displaying
      distinguishing
      filtering
      grouping
      inserting
      joining
```

Figure 7.2: ^{my}Grid ontology snapshot.

Manually built ontology. The manually built ^{my}Grid ontology is much larger and more complex than the RDF(S) related ontology. It contains over 550 concepts distributed over a set of distinct subsections covering the domains of molecular biology, bioinformatics, informatics and generic tasks, all under a common upper level structure. However, currently only a part of this ontology (accounting for 23% of its concepts) provides concepts for annotating Web service descriptions in a forms-based annotation tool. The so obtained semantic Web service descriptions are used for facilitating service discovery (Wroe et al., 2004). The ^{my}Grid ontology contains only a small number of concepts denoting functionality (23) (see a snapshot of this part of the ontology in Figure 7.2). Observe that a different modelling principle is used here compared to the RDF(S) related ontology. Namely, the functionality concepts simply denote generic actions that can be performed in bioinformatics without being linked to the involved data structures. A possible explanation for this choice is that in bioinformatics one can perform these operations on a multitude of data structures and thus, enumerating all these combinations would be impractical.

Encountered Problems. Several factors hampered the building of this ontology. First, ontology building in itself is *time consuming*. The ontology was initially built with two months of effort from an ontology expert with four years experience in building description logic based biomedical ontologies. A second impediment is the *dynamic nature* of the field. The exponential rise in the number of bioinformatics Web services over the past year required a further two months effort to maintain and extend the ontology. However, its content currently lags behind that needed to describe the 1000+ services available to the community. Thirdly, *lack of tools* hampered the process. At the time of

[3]`http://www.hgmp.mrc.ac.uk/Software/EMBOSS/Apps/`

development, tool support for handling separate ontology modules was minimal, hence the existence of a single substantial ontology.

A fourth impediment was the *lack of guidelines* on how to build the domain specific ontology, or indeed how to relate it to upper level ontologies. Since at that time DAML-S (the predecessor of OWL-S) was still under development, the ontology curator devised their own generic Web service description schema based on DAML-S but much simplified to reflect narrower requirements. Lacking guidance from the Web services field, the curator relied on design principles employed in other large open source biomedical ontologies such as openGALEN (Rector and Rogers, 1999) and the TAMBIS ontology (Baker et al., 1999).

7.2.3 Conclusions

The analysis of the ontology building process presented in the previous two subsections lead to two major conclusions. First, ontology building was experienced as a difficult process in both projects. This prompts to the need of a (semi-)automated solution for this problem. In this thesis we investigate the use of ontology learning techniques for (semi-)automating domain ontology building. Second, the Web service context presents a set of constraints that have to be taken into account when designing a (semi-)automatic solution. In this subsection we detail both conclusions.

1. Ontology building is difficult and should be automated. Both case studies agree on a set of problematic factors that hampered the ontology building activity.

High number of textual documents. The ontology curators had to analyze (read, understand and identify common concepts in) a *high number of textual documents* (over hundred in both cases) to ensure the quality of their ontologies. The number of analyzed documents is increasing as more Web services become available.

Lack of guidelines. A second impediment was the *the lack of guidelines* on what knowledge such ontologies should contain and what design principles they should follow. This resulted in different groups building different ontologies to describe Web services in the same domain, as reportedly happened in bioinformatics (Lord et al., 2004). A difference in modelling style can already be seen if we compare the ontologies produced by the two case studies. In case study 1, the names of the functionality concepts denote both the action of the functionality and the participating data structure (e.g., *BookTicket*). In case study 2, only the verb denoting the action is used in the naming the functionality concepts (e.g., *Aligning*).

These factors make ontology building a time consuming activity (several months of work) creating a demand for tools that support ontology curators to extract ontologies from *large and rapidly changing textual data collections*.

2. Several constraints have to be taken into account when building an automated ontology learning solution. We conclude that the two ontology building activities differ in several aspects. First, the application domains are different: computer science vs. biology related. Second, different kinds of data sources are used as a basis for ontology building: javadoc descriptions of several tool APIs in case study 1 and detailed service documentations in case study 2. These sources also differ in their grammatical quality, the descriptions used in case study 1 having a lower quality from this perspective. The manually built ontologies are also different. The ^{my}Grid ontology is much larger and more complex than the RDF(S) related ontology. This is not necessarily an advantage since experience has shown that only a small fraction of the ontology is currently used for Web service annotation.

Despite these differences there are several characteristics that are exhibited by both case studies. These characteristics, described in detail in the next section, acted as a set of requirements for setting up an ontology learning framework specialized for the Web services domain (see Section 7.4).

7.3 Requirements for an Ontology Learning Solution

During our analysis of the ontology building process in the two case studies, we identified a set of characteristics that have a major influence on the design of an ontology learning solution. These characteristics relate to two aspects of the learning problem.

The quality of the analyzed texts. The textual comments attached to Web services are characterized by a low grammatical quality (see Subsection 7.3.1) and several sublanguage features (see Subsection 7.3.2). These characteristics have partially caused the suboptinal functioning of existing ontology learning tools. We overcame the limitations of these characteristics by carefully choosing the NLP tools used within our ontology learning framework.

The characteristics of the learned ontology. Domain ontologies employed for Web service descriptions conceptualize both static and procedural knowledge (see Subsection 7.3.3). However, current ontology learning efforts are centered on extracting static knowledge. Therefore, we adapted existing techniques to the extraction of procedural knowledge as well.

7.3.1 Dealing with Low Grammatical Quality

Low grammatical quality is a strong characteristic of the natural language descriptions associated with Web services. Indeed, these descriptions are mostly short, informative comments written by developers. Typically, punctuation is completely ignored and several spelling mistakes are present. Naturally, services that have many users expose better documentation while less-used services barely contain snippets of abbreviated text.

The evident drawback of this low grammatical quality of the analyzed texts is that they are difficult to process with off the shelf NLP tools. Existing NLP tools were trained on high quality newspaper corpora which offer a considerable higher quality than Web service documentations. For example, some rule based part of speech taggers are sensitive to the capitalization of words considering most capitalized words as nouns. A possible remedy is to use preprocessing such as, e.g., capitalization of the first words of the sentence, adding some punctuation, etc.

There are also two advantages of working with such documentations. First, these texts usually employ simple sentences instead of using complicated phrases. This reduced ambiguity favors the use deeper linguistic analysis. For example, dependency parsers work better on short sentences than on complex phrases. The second advantage is that these texts use natural language in a specific (they belong to a sublanguage). This characteristic makes them amenable to automatic analysis as discussed in the next subsection.

7.3.2 Dealing with Sublanguage Characteristics

Software documentation in general, and Web service descriptions in particular, employ natural language in a specific way. They belong to what is defined as a sublanguage in (Grishman and Kittredge, 1986). A sublanguage is a specialized form of natural language which is used within a particular domain or subject matter and characterized by a specialized vocabulary, semantic relations and syntax (e.g., weather reports, real estate advertisements). Harris, one of the first researchers to study the use of natural language in restricted domains, introduced the notion of **sublanguage word classes** defined as sets of words that are acceptable in the same context within a sublanguage (Harris, 1968). An intuitive example from the medical domain is that in the context *___revealed a tumor* we might find words such as *X-ray, film, scan*. These words belong to the MEDICAL_TESTs sublanguage word class. There are several constraints on the co-occurrences of word classes in a sublanguage. For example, many valid sentences in the medical sublanguage have the form *MEDICAL_TEST revealed DISEASE* while sentences of the form *DISEASE revealed MEDICAL_TEST* are meaningless in this sublanguage (even if grammatically valid). These constraints are called **selectional constraints**.

Several word classes and selectional constraints can be determined in the Web service sublanguage we are analyzing. For example, by considering EXT_VB a word class of verbs that indicate an extraction process (e.g., *extract, get, retrieve*) a frequently occurring pattern which involves this word class and the preposition “from” can be used to easily determine the output and the source of the action.

```
Selectional Constraint (Pattern):
   EXT_VB OUTPUT from SOURCE.

Examples:
   Extract data from aaindex.
   Extract cds , mrna and translations from feature tables.
   Get data from cutg.
   Retrieve features from a sequence.
```

Knowledge about word classes and their selectional constraints in a certain sublanguage can greatly support several Natural Language Processing tasks, such as Information Extraction (Grishman, 2001). Using sublanguage analysis techniques has also a direct applicability in Ontology Learning since word classes often denote semantic classes. Also, selectional constraints can help to determine the members of a word class given some knowledge about the members of other word classes involved in the restrictions.

One of the major problematic aspects of sublanguage analysis is that determining the interesting word classes and their selectional constraints is a time consuming process. There has been promising research on (partly) automating this process ((Grishman et al., 1986; Riloff, 1996)). However, our intention was, for the first design of our framework, to focus on few but frequently occurring sublanguage features that do not need laborious work to be identified. Such are patterns that do not rely on lexical information but only on syntactic structures. For example, one of the straightforward observations was that, in this sublanguage, almost any verb indicates an action performed by a Web service. So, a word class *ACTION* would include any identified verbs. Also, noun phrases that appear after an action verb denote a participant in the action, forming the word class *ACTION_PARTICIPANT*. These word classes can be easily identified by relying only on a minimal linguistic analysis. A co-occurrence of these word classes identifies a Web service functionality and provides the basic material for an ontology learning algorithm.

7.3.3 Learning Ontologies of Procedural Knowledge

Ontology learning has to be adapted not only to deal with the characteristics of the input data but also to produce ontologies that are fit for the task of describing Web services. Web service domain ontologies contain both static (i.e., domain entity concepts) and procedural knowledge (i.e., functionalities offered by Web services). Existing ontology learning efforts, to our knowledge, have only focused on deriving static knowledge. One of the contributions of our work is to extend these techniques to the acquisition of procedural knowledge as well.

Another factor to consider is that, due to the lack of guidelines in modelling procedural knowledge for Web services, different modelling styles are emerging (e.g., see the two different styles adopted by the ontologies of our case studies). Our framework provides methods for generating ontologies that follow any of the two modelling styles. Even more, the framework can be extended to follow novel modelling styles as well.

7.4 A Framework for Learning Web Service Domain Ontologies

In the previous section we identified a set of particularities that condition ontology learning when performed in the context of Web services. These characteristics require the adaptation of existing ontology learning methods. Our literature study yielded that the ontology learning field offers a wide range of different approaches to ontology acquisition. However, while most work is targeted on specific domains we are not aware of any efforts that analyze software documentation style texts. Several generic ontology learning tools exist, most prominently Text-To-Onto (Maedche and Staab, 2004), OntoLearn (Navigli and Velardi., 2004) or OntoLT (Buitelaar et al., 2004b), but they are either not available for experimenting or they are workbenches of generic methods that can be fine-tuned for a certain domain.

We tried to extract a domain ontology from our corpora using Text-to-Onto, the only tool publicly available at the time of our experiments. The results were suboptimal due to the strong particularities of our corpus which hampered the efficacy of the generic methods implemented by the tool.

In this section we present an ontology learning framework which is tailored to address the particularities of the Web services domain. We first describe the learning framework as a whole (Section 7.4.1), then we detail each of its steps.

7.4.1 Overview of the Framework

The ontology learning framework consists of several steps, as depicted in Figure 7.3. We briefly describe these steps and show how the characteristics of the Web services context influenced their design.

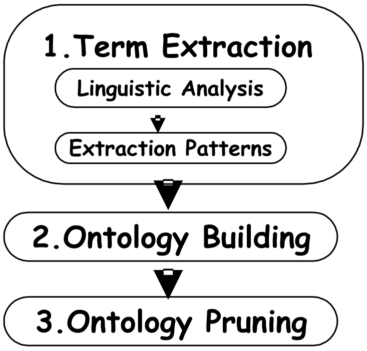


Figure 7.3: The main steps of the ontology learning framework.

1. **Term Extraction.** In the first step we identify words in the corpus that are relevant for ontology building. A word or a set of words that are identified as useful for ontology building form a "term". Term extraction is done in two steps. First, in a *linguistic analysis* phase the corpus is annotated with linguistic information. Then, a set of *extraction rules* are applied on this linguistic information to identify the potentially interesting terms.

   The characteristics of the Web services domain influenced our design choices in several ways. First, to overcome the limitations of the poor grammatical quality of the texts we have employed linguistic analysis of different complexity. As it is evident from the results of our experiments, more complex analysis led to better results. Then, the small size of the corpus and its sublanguage features facilitated the use of a rule-based solution. Namely, the sublanguage features of the corpora allowed us to easily observe a few heuristics for identifying important information and implement them in our extraction rules.

2. **Ontology Building.** In the second step of the framework, the previously identified terms are centralized, analyzed and transformed in corresponding concepts and their hierarchical relations. The ontology building phase derives both static and procedural knowledge in the form of a hierarchy of frequent domain concepts and a hierarchy of Web service functionalities. The strong sublanguage features of the analyzed corpora allow extracting terms that are highly relevant for ontology building. Therefore, it suffices to use simple ontology learning techniques and to adapt them to the requirements of the domain (e.g., extract procedural knowledge).

3. **Ontology Pruning.** The low grammatical quality of the corpus as well as its sublanguage characteristics cause a suboptimal functioning of the used linguistic tools.

Therefore, some of the derived concepts do not have any domain relevance. The pruning stage excludes these potentially uninteresting concepts from the ontology.

In the next subsections we detail all three steps of the learning framework.

7.4.2 Step1: Term Extraction

The term extraction phase identifies (sets of) words (i.e., *terms*) in the corpus that are relevant for ontology building. This phase can be realized in different ways, for example, by using *linguistic analysis* of different complexity. We report on two instantiations of the framework which use two different kinds of linguistic knowledge. The first instantiation, ***M_POS***, uses basic Part-of-Speech (POS) information while the second, ***M_DEP***, relies on deeper dependency parsing techniques to acquire linguistic knowledge.

The different linguistic information requires implementing different extraction patterns for the extraction rules: *surface patterns* in the first case and *syntactic patterns* in the second. While the form of these pattern based rules differ (as will be described in what follows), the heuristics behind them remain the same. Independently of the technical implementation, we distinguish two major categories of rules according to the type of information they derive.

Rules for *identifying domain concepts* rely on the observation that domain concepts correspond to nouns in a corpus. Given the small size of the corpora and the concise style of the Web service documentations the majority of nouns denote potentially interesting domain concepts. We extract entire noun phrases where a noun phrase consists of a head noun preceded by an arbitrary (zero or more) number of modifiers (nouns or adjectives).

Rules for *identifying functionalities* implement the previously described sublanguage characteristic, i.e., that verbs and related nouns are good indicators of Web service functionality.

In what follows we detail two concrete instantiations of the term extraction step which rely on different linguistic information.

Method 1: Part-of-Speech Based Term Extraction

The first instantiation of the framework relies on the simplest linguistic information, i.e., part of speech tags (POS). A POS tagger is used to perform the **linguistic analysis** phase. The tagger assigns each word in the sentence a corresponding POS tag. For example, in the sentence below, the tagger identified a verb (i.e., *find*), two nouns (i.e., *sites*, *proteins*), an adjective (i.e., *antigenic*) and a preposition (i.e., *in*).

```
Find(VB) antigenic(JJ) sites(NN) in(Prep) proteins(NN).
```

Following the general steps described by the framework, a set of extraction rules are applied on the derived linguistic information. The extraction patters which form the left hand side of the rules are *surface patterns* which, besides POS tag linguistic information, rely on surface knowledge such as the order of words in the sentence.

1. Identifying domain concepts. We stated above that extraction patterns are written to extract both static (domain concepts) and procedural (service functionalities) knowledge. The *surface pattern* that extracts noun phrases implements the heuristic observation described above. This rule is specified in JAPE (Cunningham et al., 2000), a rich and flexible regular expression based rule mechanism.

```
( (DET)*
  (ADJ|NOUN|POS)*
  (NOUN) ):np
-->:np.NP={}
```

The pattern in the left hand side of the rule (i.e., before "→") identifies noun phrases. Noun phrases are word sequences that start with zero or more determiners (identified by the (DET)* part of the pattern). Determiners can be followed by zero or more adjectives, nouns or possession indicators in any order (identified by the (ADJ|NOUN|POS)* part of the pattern). A noun phrase mandatorily finishes with a noun, called head noun ((NOUN)). DET, ADJ, NOUN and POS are macros and act as placeholders for other rules identifying terms that are part of these categories. These macros rely on the actual POS tag information. For example, the ADJ macro has the following definition:

```
Macro: ADJ (
  {Token.category == JJ, Token.kind == word} |
  {Token.category == JJR, Token.kind == word} |
  {Token.category == JJS, Token.kind == word}
)
```

The macro contains the disjunction of three patterns. This means that the macro will fire if a word satisfies any of these three patterns. Each of these three patterns identifies words which were assigned one of the JJ, JJR and JJS POS tags. POS tags are assigned in the *"category" feature* of a *"Token" annotation*. Note that annotations and their features are used in GATE (Cunningham et al., 2002) to encode various information about the analyzed documents.

Any word sequence identified by the left hand side of a rule can be referenced in its right hand side. The text snippet identified by a (part of) a pattern is associated to a variable which than can be reused in the right hand side. For example, *np* identifies all noun phrases. This string is then used in the right hand side of the rule which specifies that strings denoted by *np* should be annotated with the *NP* annotation. In the example sentence, this rule identifies *"antigenic sites"*[4] (ADJ NOUN) and *"proteins"* (NOUN) as noun phrases.

2. Identifying functionalities. One *surface pattern* identifies pairs of verbs and following noun phrases as potential functionality information to be added to the domain ontology. Having identified noun phrases (NP) and verbs (VB) with two previous rules, the JAPE rule for identifying and annotating functionalities is straightforward.

```
( {VB}{NP} ):funct
-->:funct.Functionality = {}
```

[4]We use this notation convention to present *"terms"* extracted from the corpus.

In the example sentence, this rule identifies *"find"_"antigenic site"* as a verb phrase denoting a possible functionality in bioinformatics.

Method 2: Dependency Relation Based Term Extraction

In a second instantiation of the framework, ***M_DEP***, we experiment with richer linguistic information than POS tags, i.e., dependency relations. Dependency parsing offers a deeper linguistic analysis than POS tagging being a commonly used method in computational linguistics. A dependency relation is an asymmetric binary relation between a word called **head** and a word called **modifier**.

We use Minipar (Lin, 1998), a state of the art dependency parser with a reported high performance (88% precision and 80% recall with respect to dependency relations). As an example, we list in Table 7.1 Minipar's analysis for our example sentence. For each word, the following information is provided : (i) its position in the sentence; (ii) its form as it appears in the sentence; (iii) its lemma; (iv) its part of speech; (v) the name of the dependency relation between this word and the head (e.g., obj) and (vi) the position of the head word modified by the current word. In the example, *antigenic* is an adjective which modifies the noun *sites*, and *sites* is the object of the verb *find*.

Pos.	**Word**	**Lemma**	**POS**	**Relation**	**Head**
1	find	find	V	-	-
2	antigenic	antigenic	A	mod	3
3	sites	site	N	obj	1
4	in	in	Prep	mod	3
5	proteins	protein	N	pcpmp-n	4

Table 7.1: An example Minipar output.

The benefit of using richer linguistic information is that the potentially interesting information can be extracted in an easier way. Naturally, while the same heuristics are used, the extraction patterns must be re-implemented. In this case the patterns are defined on the syntactic relations within the sentences and therefore called *syntactic patterns*.

1. Identifying domain concepts. The first category of patterns, those that identify domain concepts, explore the *nn* (noun modifier of a noun) and *mod* (adjective modifier of a noun) dependency relations to detect noun phrases. When such relations are identified, the head noun together with its modifiers are annotated as being a noun phrase. Regular expressions are not enough to encode these more complex patterns (they do not allow variables). We use extra java code on the right hand side of the JAPE extraction rules to accomplish this.

2. Identifying functionalities. The pattern for functionality identification relies on the *obj* relationship and identifies pairs of verbs and their objects. If the object is the head of a noun phrase then the whole noun phrase is extracted. This pattern relies on the output of the previous NP extraction pattern.

This pattern captures the desired information in the majority of cases with a few exceptions. One of the exceptions occurs when several verbs in a sentence refer to the same object. For example, the sentence *Replace or delete sequence sections* suggests that both *"replace"_"sequence section"* and *"delete"_"sequence section"* are valid functionali-

Position	Word	Lemma	POS	Relation	Head
1	replace	replace	V	-	-
2	or	or	U	lex-mod	1
3	delete	delete	V	lex-dep	1
4	sequence	sequence	N	nn	5
5	sections	section	N	obj	1

Table 7.2: Verb dependency example.

Position	Word	Lemma	POS	Relation	Head
1	pick	pick	V	-	-
2	pcr	pcr	N	nn	3
3	primers	primer	N	obj	1
5	hybridi-zation	hybridi-zation	N	nn	6
6	oligos	oligos	N	conj	3

Table 7.3: Noun dependency example.

ties in this domain that we wish to extract. However, Minipar's output does not directly encode the verb-object relation between *delete* and *section* (see Table 7.2). On the other hand, the analysis denotes that there is a dependency relation between the two verbs of the sentence. Whenever two or more verbs are related by a logical operator they should be bound to a single noun (the object of one of the verbs). One of our extraction patterns identifies cases when several verbs are related via the *lexdep* or *conj* relations. These relations denote cases when verbs are related via logical operators such as "or", "and" (e.g., *Reverse and complement a sequence*) or ",". Often there are cases when the logical dependency between more than two verbs is partially specified and we have to explicitly define all dependents based on the transitivity of this relation (e.g., if dependency(v1,v2) and dependency(v2,v3) then dependency(v1,v3)).

Another exception is when several objects are in a conjunctive relation. For example, from *Pick pcr primers and hybridization oligos* we wish to extract both *"pick"_"pcr primer"* and *"pick"_"hybridization oligos"* functionalities. However, the Minipar output specifies only the first verb-object relation (see Table 7.3). Nevertheless, knowing that there is a conjunctive relation between *primers* and *oligos* we can deduce that *oligos* also plays an object relation with respect to the verb *pick*. Just as with verbs, we have written a pattern that identifies conjunctive NPs and deduces the additional knowledge. The patterns that identify dependency of verbs and objects are performed before the pattern that identifies functionalities.

Summarizing Section 7.4.2, note that the sublanguage nature of the Web service specific corpora allowed us to extract sufficient material for ontology building by using only relatively simple, off the shelf natural language processing techniques. There are several advantages of using these simple extraction methods. First, they are fast. Second, they rely on off the shelf, thoroughly researched and high-performance techniques (POS tagging, dependency parsing). Finally, the pattern based extraction rules can be adjusted or extended by the users of the system according to the needs of their particular data sets.

7.4.3 Step2: Ontology Building

The second step of our framework, ontology building, collects the results of the pattern based extraction. Noun phrases are a basis for deriving a data structure hierarchy and the functionality information is used for building a functionality hierarchy. We employ the lemma (i.e., base form) of the extracted terms for ontology building.

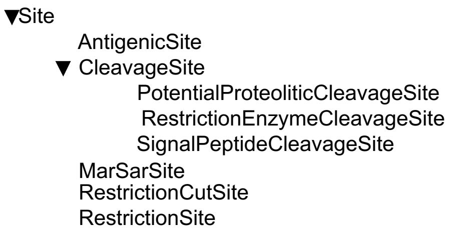

▼Site
AntigenicSite
▼ CleavageSite
PotentialProteoliticCleavageSite
RestrictionEnzymeCleavageSite
SignalPeptideCleavageSite
MarSarSite
RestrictionCutSite
RestrictionSite

Figure 7.4: The automatically extracted *Site* concept.

Building the data structure hierarchy. We observed that many of the terms mentioned in the analyzed corpora (and especially in the bioinformatics corpus) have a high level of compositionality, in the sense that they incorporate other meaningful terms as proper substrings. Our observation is confirmed by a recent study of the Gene Ontology terms which proved that 63,5% of all terms in this domain are compositional in nature (Ogren et al., 2004). Another observation, also proved by this study, is that compositionality indicates the existence of a semantic relationship between terms. If a term t_1 is obtained by adding a modifier in to another term t_2 then t_1 is more specific than t_2. This translates in the ontological subsumption relationship.

The hierarchy building algorithm reflects these observations. If a concept A's lexicalization is a proper substring *ending* another concept B's lexicalization (e.g., *Site* in *AntigenicSite*) then A is more generic than B and the corresponding subsumption relationship is added to the ontology. Also, if the lexicalization of two concepts B and C end with the same substring we speculate that this substring represents a valid domain concept (even if it does not appear as a stand alone term in the corpus) and add it as a parent concept for B and C. As an example, Figure 7.4 depicts the *Site* data structure hierarchy. Such compositionality based hierarchy building, also called *vertical relations*, has been used in (Buitelaar et al., 2004b; Velardi et al., 2001) and (Cimiano et al., 2004c).

Building the functionality hierarchy. There are no clear guidelines in the field of semantic Web services about how functionality hierarchies should look like. The OWL-S/IRS/WSMO style of modelling functionalities includes both the verb of the action and a directly involved data element in the functionality (e.g., *BookTicket*). This modelling style was followed in case study 1 (see Figure 7.1). On the other hand, in the bioinformatics domain ontology developed in case study 2, functionalities are concepts denoting action (e.g., *Aligning*) without any connection to the data structures see (Figure 7.2). We provide ontology building modules that produce functionality hierarchies fulfilling either of these modelling styles, i.e., creating verb-noun phrase (e.g., *Delete_SequenceSection*) or only verb (e.g., *Deleting*) based concepts.

Note that we do not take a strong stand on the right way to model functionality hierarchies. There are several reasons for this. First, in this initial iteration of our work we were more concerned with correctly extracting the needed information than with the way it is conceptualized (once the right information is extracted it can be conceptualized in several different ways). Second, while we acknowledge that this is an important issue that requires further investigation, we believe that modelling guidelines should be provided by those who are using the semantic Web services technology. In fact, our observation, based on the experimental work that we performed, is that different domains would benefit from different ways of conceptualizing the functionality knowledge. Therefore, we believe that as the technology will be used in more domains, common guidelines will emerge on the appropriate ways to model functionalities (probably dependent on domain characteristics). To this end we have not constrained the use of a particular conceptualization in our tool but allowed its extension with different functionality hierarchy building modules.

7.4.4 Step3: Ontology Pruning

The first two main steps of the framework, term extraction and ontology building, result in an initial ontology. These steps only rely on our initial heuristics to select the potential concepts. However, even if they capture strong sublanguage characteristics, our heuristics are not perfect and some of the derived concepts are not domain relevant. Also, several irrelevant concepts are derived due to the suboptimal functioning of NLP tools on the low quality texts. The pruning module aims to filter out irrelevant concepts from the learned ontologies.

Maedche describes two major strategies for performing ontology pruning (Maedche, 2002). First, the *baseline pruning* strategy is based on the assumption that frequent terms in a corpus are likely to denote domain concepts. Conversely, concepts that are based on low frequency terms should be eliminated from the ontology. The second pruning strategy, *relative pruning*, is based both on the frequency of the terms in the analyzed corpus and in an independent reference corpus. Only concepts that rely on terms that are frequent in both analyzed and relative corpora will be maintained in the ontology.

Lacking a reference corpus in both domains, we use a *baseline pruning* strategy in our current implementations. We consider the average frequency ($\overline{Freq}$) of the n learned concepts as a threshold value and prune all concepts that have a lower frequency than this value.

$$\overline{Freq} = \frac{\sum_{i=1}^{n} freq(concept_i)}{n}$$

Another heuristic for the pruning is based on the observation that noun phrases included within a functionality annotation by our rules are more likely to denote domain concepts. Therefore, if a low frequency data structure concept's lexicalization was identified within a functionality annotation and the corresponding *Functionality* concept was not pruned then the data structure concept will not be pruned either.

7.4.5 Possible Extensions of the Framework

The extraction framework and its two instantiations alow extracting basic information about Web services, such as functionality types and domain concepts that play the role of parameters. Ongoing experiments explore new ways to extend this basic framework in order to acquire richer information about the services. In this section we briefly overview some of our future plans for extending the framework. Note, however, that the next chapter, presents only the evaluation of the basic framework (without these extensions).

A first possible extension should not only identify parameters but should also be able to distinguish which of them are inputs or outputs. For this, more refined rules can be defined. For example, the word "given" in front of a noun phrase often indicates that it plays the role of an input. We are currently working on identifying such heuristics. Some example heuristics that would identify inputs are:

```
* given NP(,NP)*

e.g., Gets current weather given a zip code.
e.g., Given amount, interest rate, and term, this service
will calculate load payments.

* VB(Action) NP(output) between NPs(inputs)

e.g., Calculates the distance between two U.S. ZIP codes.
e.g., Translates text between a variety of languages.
```

The second source of information for determining inputs, outputs and functionalities are the WSDL files that describe Web services. In particular, the names of the methods and messages contain useful information about the service. From preliminary investigations it seams that WSDL files are often more accurate in providing this information than the textual descriptions attached to Web services. Our idea is that a combination of both sources should give the best results.

Besides operational features, such as inputs and outputs, other features can be important when choosing a service. In particular the geographic area where the service is active is an important consideration. Our ontology learning framework should be able to extract the geographical constraints indicated by Web services (these concepts can be used to describe the geographic radius of the service). Note that this issue is becoming more evident in online Web service repositories where services from different parts of the world and acting in different geographic regions are registered.

Some online Web service repositories attempt to deduce this feature from the country extension of the URL where the service description is published. However, this seldom indicates the geographic region for which the service was built. For example, a Web service that *"validates and enhances contact information for any address in India"* can be published at a .com address[5]. Conversely, a Web service whose URL contains a certain country identifier (e.g., France[6]) might perform a service that is independent of geographic constraints (e.g., in the case of the example service - cipher/decipher).

[5]`http://ws.strikeiron.com/IndianAddressVerification?WSDL`

[6]`http://www.quisque.com/fr/chasses/crypto/cesar.asmx?WSDL`

An alternative solution to determining geographic constraints for a service is to use Named Entity Recognition (NER) systems. Such systems automatically identify geographic entities, persons and organizations in free text. NER technology matured in the previous decades to reach performances of 80-90% Precision and Recall for a generic system (such as ANNIE) and 90-95% Precision and Recall for systems that are tuned to the needs of particular domains (Cunningham et al., 2002).

```
Search through all Swedish telephone subscribers.

Search UK Index.

This webservice return longitude, latitude and height
from a given city. Only for France.

Lookup ATM Locations by Zip Code (US Only).
```

For example, for the Web service descriptions above, our experiments show that the ANNIE NER system recognizes *Swedish, UK, US, France* as references to the corresponding countries. We observed that, in some cases, the restriction is strengthened by the use of “only” in constructions such as “only for/in country” or “country only”. These constructs can be easily identified using a regular expression based rule mechanism, for example in the term extraction step of our framework.

Another possible extension of the framework is at the level of used algorithms. For example, more complex algorithms should be used to derive the taxonomy hierarchy. In this direction we experimented with encoding a set of Hearst based patterns. However, the occurrences of such patterns are rare in the textual sources attached to Web services. In fact, when analyzing around 450 descriptions (extracted from the SalCentral[7] online Web service repository), only 10 contained subsumption information identifiable with Hearst patterns. We will further explore the use of WordNet for hierarchy learning in this domain.

7.5 Implementation Details

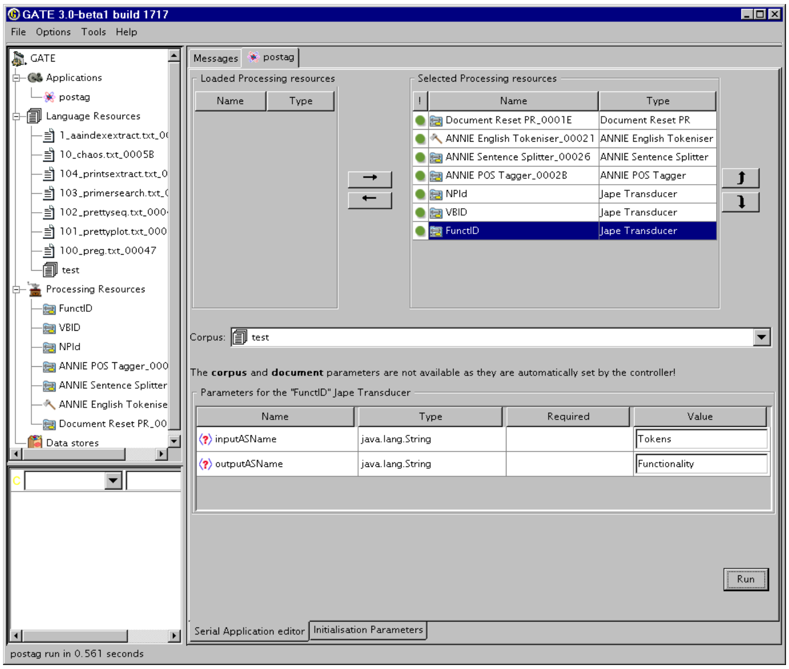
An important part of our work was the implementation of a prototype system. We aimed to build a system that is easy to use both by developers and users of ontology learning methods. Such a user friendly tool brings us closer to the ultimate goal of our work, that of facilitating the process of building Web service domain ontologies. First, we aim for a modular, easy to run and understand implementation that can be easily modified and adapted to new application domains. We achieve this by using the GATE framework (Cunningham et al., 2002) (Sections 7.5.1 and 7.5.2). Second, we use visual techniques to provide a user friendly and easy to understand presentation of several data sets derived during ontology learning (i.e, the extracted terms, the learned ontology) in order to support frequently occurring tasks (Section 7.5.3).

[7] `http://www.salcentral.com/`

7.5.1 The GATE Framework

As developers of ontology learning solutions we found that GATE offered several features important for the development of ontology learning tools, such as, modularity, traceability, portability and evaluation. Indeed, GATE is one of the few available generic NLP frameworks that has been developed to offer *"robustness, re-usability, experimental repeatability and scalability"* (Bontcheva et al., 2004). Also, to our knowledge, it is the only NLP framework to offer support for ontology based language engineering. The application that we built, being part of the framework, inherits these features.

GATE is an infrastructure and framework for developing natural language processing applications. *Processing Resources* (i.e., processing algorithms such as POS taggers, dependency parsers, etc) and *Language Resources* (i.e., data elements such as corpora and ontologies) can be combined in applications either programmatically or visually. GATE also offers a wide range of Processing Resources and the possibility to extend this set by own, customized modules.


GATE 3.0-beta1 build 1717
File
Options
Tools
Help
Messages
postag
Serial Application editor
Initialisation Parameters
Run

GATE
- Applications
  - postag
- Language Resources
  - 1_aaindexextract.txt_0(
  - 10_chaos.txt_0005B
  - 104_printsextract.txt_0
  - 103_primersearch.txt_(
  - 102_prettyseq.txt_000·
  - 101_prettyplot.txt_000
  - 100_preg.txt_00047
  - test
- Processing Resources
  - FunctID
  - VBID
  - NPId
  - ANNIE POS Tagger_000
  - ANNIE Sentence Splitter
  - ANNIE English Tokenise
  - Document Reset PR_00
- Data stores

Loaded Processing resources

Name	Type

Selected Processing resources

!	Name	Type
●	Document Reset PR_0001E	Document Reset PR
●	ANNIE English Tokeniser_00021	ANNIE English Tokeniser
●	ANNIE Sentence Splitter_00026	ANNIE Sentence Splitter
●	ANNIE POS Tagger_0002B	ANNIE POS Tagger
●	NPId	Jape Transducer
●	VBID	Jape Transducer
●	FunctID	Jape Transducer

Corpus: test

The **corpus** and **document** parameters are not available as they are automatically set by the controller!

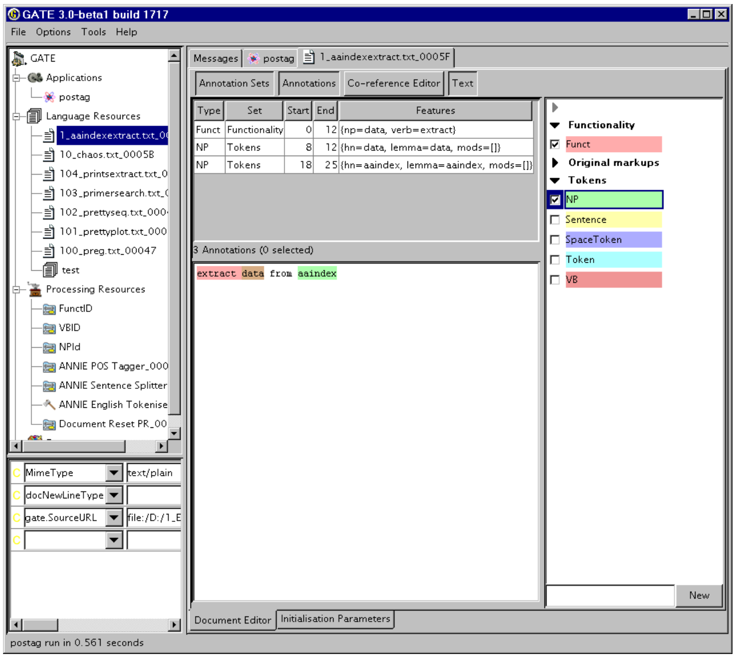
Parameters for the "FunctID" Jape Transducer

Name	Type	Required	Value
inputASName	java.lang.String		Tokens
outputASName	java.lang.String		Functionality

postag run in 0.561 seconds

Figure 7.5: Building an application with the GATE user interface.

Modularity. GATE allows for building modular applications. Applications can be built up by pipelining a set of Processing Resources either programmatically or visually. Using GATE one can visually load NLP modules (e.g., tokenizers, POS taggers) and data structures (e.g., corpora, ontologies) and combine them in different applications. For example, Figure 7.5 depicts the GATE user interface. The left pane shows the currently loaded Processing and Language Resources. The right pane shows how these resources are combined in a pipeline to build up an application (called *postag*). These resources run over all the documents provided by the *test* corpus. Note that this application performs the Linguistic Analysis and Term Extraction steps of the part of speech based ontology learning method (M_POS).



Type	Set	Start	End	Features
Funct	Functionality	0	12	{np=data, verb=extract}
NP	Tokens	8	12	{hn=data, lemma=data, mods=[]}
NP	Tokens	18	25	{hn=aaindex, lemma=aaindex, mods=[]}

3 Annotations (0 selected)

extract data from aaindex

MimeType	text/plain
docNewLineType	
gate.SourceURL	file:/D:/1_E

postag run in 0.561 seconds

Figure 7.6: Inspecting annotations with the GATE user interface.

Traceability. Besides easy combination of Processing Resources, GATE also provides a user interface for the inspection of the results produced by each of these resources. Input and output data to processing resources is represented in terms of so called "annotations". An annotation is a description which is added to a certain part of the analyzed text. Annotations can be added on individual words, word compounds, sentences or paragraphs. Annotations can be easily visualized in GATE. In Figure 7.6 we inspect one of the documents in the corpus (highlighted in the left pane, under Language Resources) and all its annotations. The text of the document can be seen in the bottom-middle pane. The right pane contains all annotations, grouped in so called annotation sets. For example, the *Tokens* annotation set contains annotations that depict different information. The *Token* annotation is added to identify all tokens in the sentence. The *NP* annotation labels all noun phrases in the text. The *Functionality* annotation set contain only the *Funct* annotation which spreads on word combinations that might denote a functionality offered by the service. When one annotation is selected all its instances are highlighted in the text. In our case both *NP* and *Funct* annotations are highlighted. Note that there can be overlaps between annotations. Besides their name, annotations can contain a set of features which are attribute - value pairs. One can view these features in the top-middle pane. For example, each *Funct* annotation has a feature that contains the label of the noun phrase (*np*) and a feature whose value is the verb that participates in the functionality (*verb*). These annotations can be easily inspected for all documents allowing an insight in the results of the Processing Resources.

Portability. By using this modular architecture of GATE it is easy to adapt an application to novel needs by simply replacing or modifying some of the modules. For example, one can use different language processing modules, or adjust some of the JAPE rules.

Evaluation. GATE also offers two mechanisms for quantitative evaluation. The first one, *AnnotationDiff*, compares two sets of annotations added to a document. This functionality is useful to compare an automatically annotated document with a manually annotated version of the same document or one could compare the annotations added by two different systems (or different versions of the same system). The other evaluation tool, the benchmarking tool, performs the evaluation at a corpus level rather than at document level. The tool compares how a given annotating is assigned to *all* documents in two different versions of a corpus. Precision, Recall and F-measure are compared at document level and then at corpus level.

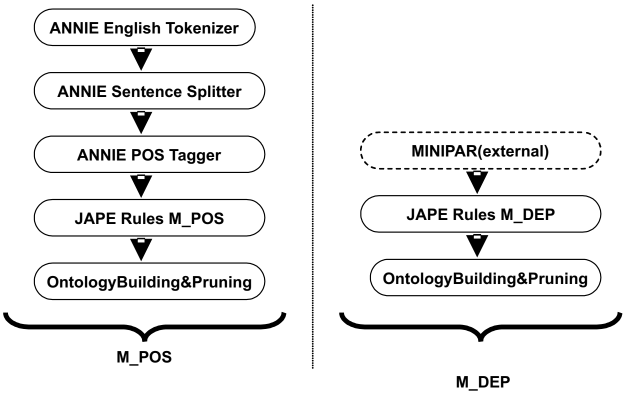


Figure 7.7: The GATE implementation of M_POS and M_DEP.

7.5.2 Two Concrete Implementations

We implemented both ontology learning methods by using GATE. Many of the off the shelf techniques on which our methods rely on were readily available in GATE. For example (as depicted in the left hand side of Figure 7.7), the *Linguistic Analysis* step of the Part of Speech based method, **M_POS**, was entirely performed using processing resources offered by GATE: a tokenizer (*ANNIE English Tokenizer*), a sentence splitter (*ANNIE Sentence Splitter*) and the Hepple POS tagger (Hepple, 2000) (available as the *ANNIE POS Tagger* processing resource). In the case of the dependency based method, **M_DEP**, (see the right hand side of the Figure 7.7), we performed the linguistic preprocessing external to GATE by using MINIPAR (Lin, 1998). For both approaches, the *Extraction patterns* were implemented using the JAPE regular expression based rule mechanism (Cunningham et al., 2000) which is part of GATE. The final two steps are jointly performed by a single module (*OntologyBuilding&Pruning*) which was implemented as a GATE Processing Resources and therefore it is usable from the GATE GUI.

The data used by our methods (such as the linguistic information or the structures identified by patterns) is represented as *annotations* on the analyzed texts. Both extraction rules and individual modules operate on these annotations and represent their output as annotations.

We greatly benefitted from the support of GATE during implementation. To briefly mention the most important benefits, first, we could *reuse* several existing libraries for document management and ontology representation. Second, by declaring parts of our method as GATE Processing Resources and using the offered annotation based system as a data transfer mechanism between these parts, we can run and manage our tool via the *GATE GUI*. It is now possible (1) to build and configure modular applications by visually selecting existing Processing Resources (our own or provided by GATE), (2) to select different corpora or (3) to inspect the annotation based output of each processing module. All these allow easy debugging and make the extraction process transparent to the end users thus increasing the usability of the tool. Finally, we have used the *data storage* and *evaluation* facilities of GATE during the development and fine-tuning stages of our prototype.

7.5.3 Visual Support for Ontology Learning

In the previous section we discussed the implementation details of our prototype and showed how by using GATE we achieved usability features important for developers. In this section we discuss how visual techniques can be used to present the learned ontologies and therefore, to make our tool easier to use both for developers and end users.

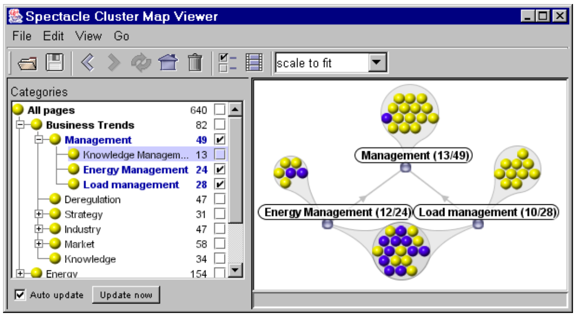
Some systems already exist that exploit visualization techniques for the Semantic Web. For example, ontology authoring tools (Gomez-Perez, 2002) offer visualization facilities that support the needs of the ontology development stage. Our analysis of these tools (detailed in (Fluit et al., 2004)) concluded that the visualizations used by authoring tools are mainly schema based. The reason for this is that the users of these tools are ontology engineers that need to get an insight in the complexity of the ontology that they build. Therefore, these tools employ schema visualization techniques that primarily focus on the structure of the ontology, i.e., its concepts and their relations. Schema visualizations often do not make any graphical distinction between schema information and instances. As a result, they do not scale to visualise many instances. Also, they fail to show instance level overlaps between classes.

Besides ontology authoring tools, we are aware of three ontology learning approaches that employ visual techniques. First, the Text-To-Onto (Maedche and Staab, 2004) tool employs a TouchGraph based schema visualization to present the extracted ontologies. Second, in conjunction with the work presented in (Reinberger et al., 2004), a visualization technique is developed to present related terms (lexons) extracted from a corpus (Pretorius, 2004). Finally, Text2Onto also relies on a graph based visualization of the extracted concepts. While useful, these visualizations only depict relations between entities of the same type (i.e., concepts, terms). However, due to the complexity of ontology learning, it is often crucial to know how entities of different types interrelate (e.g., *from which documents was a concept extracted?*). To achieve this we used the Cluster Map (Fluit et al., 2002), a visual technique developed by the Dutch company Aduna[8].

[8]`http://aduna.biz`

Note that, unlike the commonly used schema based visualization techniques, the Cluster Map technique was optimized for visualizing large, instantiated ontologies.

The Cluster Map is used to visualize light-weight ontologies that describe a domain through a set of classes (concepts) and their hierarchical relation. The technique visualizes *instances* of a set of *classes* according to their classification into these classes. Due to the specialization relationship that is encoded in the hierarchy, the set of objects in a subclass is a subset of its superclass. The set of subclasses of a class is *incomplete* when their union does not contain all the objects of the superclass. Classes that share instances are *overlapping* if no specialization relationship holds between them. These characteristics are quit common for taxonomies. However, they are difficult to show satisfactorily in textual representations or in schema like techniques that are currently used in the Semantic Web. The Cluster Map offers an alternative in this matter.



All pages	640	☐
Business Trends	82	☐
Management	49	✓
Knowledge Managem...	13	☐
Energy Management	24	✓
Load management	28	✓
Deregulation	47	☐
Strategy	31	☐
Industry	47	☐
Market	58	☐
Knowledge	34	☐
Energy	154	☐

Figure 7.8: Cluster Map example.

Figure 7.8 shows an example Cluster Map that visualizes a set of documents classified according to topics discussed in those documents. Each small sphere represents an instance. The classes are represented as rounded rectangles, stating their names and cardinalities. Directed edges connect classes and point from specific to generic (e.g., *Load Management* is a subclass of *Management*). Balloon-shaped edges connect instances to their most specific class(es). Instances with the same class membership are grouped in clusters (similar to Venn Diagrams). Our example contains four clusters; one of them represents the overlap of the *Load management* and *Energy Management* classes.

The organization of the graph is computed using a variant of the well-known spring-embedder algorithm (Eades, 1984). On the one hand the class and cluster nodes repel each other. On the other hand the edges, connecting two classes or clusters to their classes, produce an attractive force between the connected nodes; i.e., they work as "springs".

The added value of our visualisation lies in its expressivity. The classes and their relationships (the vocabulary of the domain) are easy to detect. Also, it is immediately apparent which items belong to one or multiple classes, which classes overlap (e.g., *EnergyManagement* and *LoadManagement*) and which do not. The subclasses of the root class are *incomplete* as their union does not cover the superclass: some members of *Management* were not further classified.

Another interesting aspect of the visualization is that geometric closeness in the map is related to semantic closeness. This is a consequence of the graph layout algorithm.

Classes are semantically close if they share many objects. Indeed, the more objects two classes share, the closer they are represented. *Objects* are semantically close if they belong to the same class(es). Indeed, objects with the same class membership are clustered.

In our prototype, we adapted the Cluster Map to support frequently occurring situations both during the development and the deployment of an ontology learning method (see details in (Sabou, 2005b)). These different visualizations have been embedded in the GATE framework as plugins. One of the plugins supports the evaluation of ontology extraction and as such complements the text based evaluation tools offered by GATE. The second plugin, demonstrated here, visualizes the extracted ontology and shows how the derived concepts relate based on the documents from where they were extracted. Finally, the third plugin helps identifying concepts that have been derived from several data sources in cases when ontology learning is performed on multiple different corpora.

Situation Description. Ontology learning methods are not perfect. Besides often extracting irrelevant concepts, they are limited in grouping concepts under more generic concepts (discovering abstractions) as well as discovering relations between them. Therefore, any automatically extracted ontology has to be inspected by a domain expert which judges the domain relevance of the concepts and enriches the ontology with important abstractions and relations. To understand the meaning of a concept and to evaluate its correctness, it is often necessary to inspect the contexts in which it is used in the corpus, i.e., to access the documents from which it was extracted. Further, to discover new relations between concepts (or possible abstractions) an insight has to be provided on how a selected set of concepts interrelate at document level.

Proposed Visualization. To support these analysis tasks, we use the Cluster Map in a “traditional” way: each extracted concept forms a Cluster Map class and the documents from which it was extracted become its instances. Each document instance contains the position of the terms which resulted in deriving concepts. Documents from which no concept was learned are added under the *Top* class.

Examples. This visualization allows accessing all documents from which a concept was learned. This quick access to *evidence* information is essential for understanding the intended meaning of the concept.

To detect relations between a set of extracted concepts, we visualize these concepts and analyze the resulting image. For example, we visualized the functionality concepts extracted for the domain of bioinformatics. We observed that two groups of interconnected functionalities emerged (see part A and B of Figure 7.9). Each group represents functionalities that are often offered simultaneously by Web services. At a closer look we observe that the first group (part A) contains functionalities that search or modify content, while in the second group (part B) we find functionalities concerned more with input/output operations such as *Reading* or *Writing*. The domain expert can easily access (with a simple mouse click) and inspect the documents that interrelate these concepts and decide if it is the case to set up new abstract categories (e.g., *ContentServices* and *InputOutputServices*).

In another example, part D of Figure 7.9, we visualized both Method (*Evaluating* and *Creating*) and DataStructure type concepts derived from our first case study on RDF(S) stores. On the obtained visualization two groups of terms are formed around the verbs. The visual closeness of these concepts rightly suggests that they have some shared characteristics. The first group (by *Evaluation*) contains concepts that are involved in evaluating a query (i.e., a *SesameServer*, a *Repository* on that server as well as a *Query*).

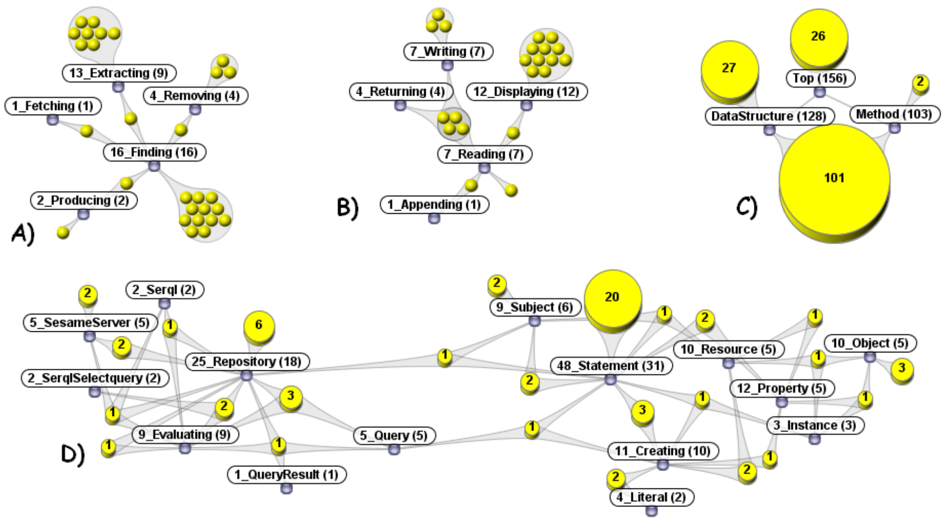


Figure 7.9: Understanding the extracted ontology. A), B) Related Web service Functionalities; C) *Method* and *DataStructure* concept type identification in the corpus; D) Detecting similar concepts based on verb's selectional restriction.

The second group contains concepts that describe elements of the RDF data model. In principle, this visualization is equivalent with the *selectional restriction* technique which is often used in ontology learning to detect semantically related concepts (e.g. (Cimiano et al., 2003)). This technique identifies terms that often occur in the same syntactic context as likely to be semantically related. For example, the objects of the verb *to drive* in *He drives the car* and *She drives a truck* share the characteristic of being drivable and therefore belong to the same semantic class (e.g., *Vehicle*).

Another task frequently performed by the method's developer is to measure how well the ontology covers the domain. For example, in our application domain we expect that each document should be described by at least one *Method* and one *DataStructure* concept. However, when visualizing all the documents in the corpus (*Top*), all documents from which a *Method* concept was extracted and all documents resulting in a *DataStructure* concept (see part C) we found out that only 101 out of 156 documents fulfil our expectation and that no concepts were extracted from 26 documents. It is easy to filter out documents that provided no (or only one type of) concepts and investigate the reasons. Solving them can improve the ontology learning.

In this section we described a set of visualizations that support frequently occurring tasks during the development and the deployment of ontology learning tools. To build these visualizations, we exploit the fact that Cluster Maps visualize the relations between two types of entities (instances and classes). In the presented visualizations different types of ontology learning related entities (documents, terms, concepts, document sources) play the role of instances and classes to support different tasks. Our conclusion is that the Cluster Map technique offers a powerful visual paradigm that can easily be adapted to support a wide range of generic (e.g., analysis, comparison, monitoring) or ontology learning specific (e.g., evaluation) tasks. The interactive GUI allows creating different visualizations just with a few mouse-clicks.

We are aware that, while flexible, the Cluster Map paradigm requires a certain amount of training to be used to its full potential. This is especially true when different entities play the role of classes and instances. For a novice user it can be confusing when changing between these different visualizations.

More generally, we believe that visual techniques can enhance the development and deployment of ontology learning methods. Naturally, the Cluster Map is *NOT* the only technique that can be used. Other techniques might be useful to support these (or other) situations. These experiments, even if not backed up by a rigorous user study, are encouraging for further pursuing the integration of ontology learning solutions with visual techniques.

7.6 Summary

In this chapter we analyzed the domain ontology building process in two case studies. The conclusion of this study was that the high number of Web services that exist per domain as well as current lack of modelling guidelines hamper the building process and motivate the development of tools that support ontology building in this domain. We also observed a set of characteristics that are specific to the Web services context and which need to be addressed when designing (semi-)automatic solution to this problem.

The design of the presented ontology learning framework is influenced by our findings. The framework aims to provide support when dealing with a high number of Web services. It also allows deriving ontologies that are built according to different modelling styles in order to compensate for the current lack of guidelines in this direction. Further, the characteristics of the Web services context (related to the quality of the analyzed data sets and the nature of the derived knowledge) influenced the main design decisions such as the choice of NLP techniques, the rationale behind the extraction rules or the output format of the conceptualization stage. Finally, it is notable that relatively simple, off the shelf natural language processing techniques sufficed for implementing the framework. These rely on well studied techniques, are fast and can be easily adjusted to particular needs of the domain. We have implemented our framework in a prototype system by using the GATE NLP workbench and extending it with visual support for presenting the results of ontology learning methods.

In the next chapter we detail issues related to the evaluation of the framework.

Chapter 8

Evaluation

In the previous chapter we presented a framework that adapts ontology learning in the context of Web services. We also described two learning methods based on this framework that employ linguistic analysis of different complexity. In this chapter we evaluate the framework. We evaluate both methods in two different domains, verifying the quality of the extracted ontologies against high quality hand-built ontologies of these domains. This evaluation indicates that (1) our extraction is indeed applicable in different domains, that (2) deeper linguistic analysis leads to better results and that (3) good results can be achieved using relatively simple off the shelf techniques.

We continuously extended and updated our view on ontology learning evaluation. Our first considerations on this issue were published in the ECAI-2004 Workshop on Ontology Learning and Population (ECAI-OLP) (Sabou, 2004a). Then the metrics were refined in a follow-up book chapter (Sabou, 2005a), in two papers published at the Third International Semantic Web Conference (Sabou, 2004b) and the 14th International World Wide Web Conference (Sabou et al., 2005a) and in an article that appeared in the Journal of Web Semantics (Sabou et al., 2005b).

8.1 Introduction

Evaluation is an integral part of the development of any tool or algorithm. The evaluation of the previously presented framework represented an important part of our work. Performing this evaluation was not straightforward because there are no standard benchmarks nor well-defined criteria for evaluating ontology learning algorithms. We performed two partial evaluations as we developed each of the methods in the framework. We verified the performance of the part of speech method (**M_POS**) on the data sets provided by case study 1 (Sabou, 2004b) and we evaluated the dependency parsing based method (**M_DEP**) in the context of case study 2 (Sabou et al., 2005a). However, these evaluation experiments, besides applying the methods on two different data sets, used slightly different evaluation metrics (as our strategy was evolving). Therefore, these partial results are not enough to answer the following questions:

Q1: Is the framework applicable across domains? The goal of our work is to build a framework that adapts ontology learning to the context of Web services but which is independent of the domain of the analyzed services. A proof of this domain independence would be that both methods would perform similarly on data sets from different domains.

Q2: Which extraction method performs best? Since our methods were evaluated on different data sets and using slightly different evaluation criteria, it is hard to compare them. To achieve a fair comparison we should (1) apply the methods on the same data sets and (2) use the same set of evaluation strategy and metrics.

Our approach to find out the answer to these questions is to take evaluation a step further by applying both methods on data drawn from both case studies[1]. Also, we use the same evaluation criteria for assessing the methods.

Since ontology learning evaluation is a non-trivial task, we start this chapter by giving an overview of some evaluation practices used by the community (Section 8.2). We use a subset of these practices for our evaluation as described in detail in Section 8.3. We describe the experimental corpora in Section 8.4 and discuss our experimental results in Section 8.5. We finally conclude in Section 8.6.

8.2 Ontology Learning Evaluation Practices

Evaluation of ontology learning is an important but largely unsolved issue, as reported by papers in a recent workshop (Buitelaar et al., 2004a). Two evaluation stages are typically performed when evaluating text based ontology learning methods:

Term level evaluation assesses the performance of extracting terms relevant for ontology learning from the corpus. Naturally, the quality of term extraction has a direct influence on the quality of the built ontology. This evaluation can easily be performed by using the well-established recall/precision metrics.

Ontology quality evaluation assesses the quality of the learned ontology. Two different ontology evaluation approaches were identified by Maedche (Maedche, 2002) depending on what is considered a “quality” ontology.

In an **application specific ontology evaluation** the quality of an ontology is directly proportional to the performance of an application that uses it. Several papers report on successfully using ontologies in various tasks such as text clustering and classification tasks (Hotho et al., 2003; Bloehdorn and Hotho, 2004) or information extraction (Faure and Poibeau, 2000). However, initial considerations on task-based ontology evaluation are only reported in (Porzel and Malaka, 2004). Two problematic issues surface for such evaluations. First, it is often difficult to asses the quality of the supported task or the performance of the application (e.g., search). Second, an experimental environment needs to be created where no other factors but the ontology influences the performance of the application. However, in the case of complex applications it is often hard to achieve such a “clean” experimental environment.

In a **Gold Standard based ontology evaluation** the quality of the ontology is expressed by its similarity to a manually built *Gold Standard ontology*. In some cases the

[1]All experimental data (corpora, extracted and gold standard ontologies) can be downloaded from `http://www.cs.vu.nl/~marta/experiments/extraction.html`.

authors use a Gold Standard ontology which was extracted from different corpora than used by the learning method ((Reinberger and Spyns, 2004)). Other authors use Gold Standard ontologies extracted strictly from the automatically analyzed corpora ((Cimiano et al., 2004b, 2003)). One of the difficulties encountered by this approach is that comparing two ontologies is rather difficult. According to (Maedche and Staab, 2002), one of the few works on measuring the similarity between ontologies, one can compare ontologies at two different levels: lexical and conceptual. Lexical comparison assesses the similarity between the lexicons (set of labels denoting concepts) of the two ontologies. At the conceptual level the taxonomic structures and the relations in the ontologies are compared.

The Gold Standard evaluation approach assumes that the Gold Standard ontology contains all the extractable concepts from a certain corpus and it contains only those. In reality though, Gold Standards omit many potential concepts in the corpus and introduce concepts from other sources (such as the domain knowledge of the expert). The evaluation results are influenced by these imperfections of the Gold Standard. To compensate for these errors, a **concept-per-concept evaluation by a domain expert** can be performed. Such an evaluation is presented in (Navigli et al., 2004). Expert evaluation can be performed also in cases when a Gold Standard is not available and its construction is too costly just for the sake of the experiment. Ideally, the evaluation should be performed by several experts.

8.3 Chosen Evaluation Criteria

We employ a combination of these evaluation strategies *to asses and compare* the quality of the implemented learning methods. We first asses the performance of the term extraction algorithm (marked 1 in Figure 8.1). To evaluate ontology quality we first rely on the domain experts' concept per concept based evaluation (2). The domain experts in both case studies are the curators of the corresponding Gold Standard ontologies. Then, we compare the extracted ontologies to the Gold Standard ontologies provided by each case study (3). In what follows, we present the methodology and metrics for performing each type of evaluation.

8.3.1 Term Extraction

This evaluation stage is only concerned with the performance of the term extraction modules. To measure the performance of these modules we manually identified all the relevant terms to be extracted from the corpus. Misspelled terms are not considered for extraction. We used the Benchmark Evaluation Tool offered by GATE to compare this set of terms with the ones that were identified through pattern based extraction. We use Term Recall ($TRecall$) to quantify the ratio of (manually classified) relevant terms that are extracted from the analyzed corpus ($correct_{extracted}$) over all terms to be extracted from the corpus (all_{corpus}). Term Precision ($TPrecision$) denotes the ratio of correctly extracted terms over all extracted terms ($all_{extracted}$). We also compute the $Fmeasure$ by assigning an equal importance to both precision and recall.

$$TRecall = \frac{correct_{extracted}}{all_{corpus}} \quad ; \quad TPrecision = \frac{correct_{extracted}}{all_{extracted}}$$

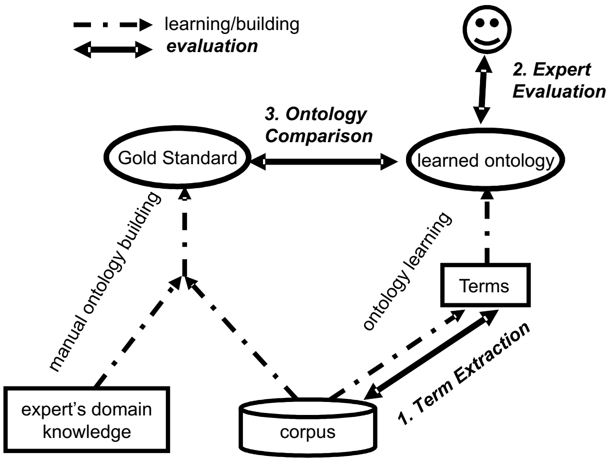


Figure 8.1: Chosen evaluation strategies.

$$Fmeasure = \frac{2 * TPrecision * TRecall}{TPrecision + TRecall}$$

8.3.2 Expert Evaluation

In this evaluation stage the domain expert performs a concept per concept evaluation of the learned ontology. We compute the ontology precision which represents the percentage of domain relevant concepts in the extracted ontology ($OPrecision$). We observed that manually built Gold Standards often omit several concepts from the corpus or introduce concepts that are not named in the corpus. Therefore, for this evaluation step, a domain expert distinguishes between two categories of relevant concepts: those that exist in the Gold Standard ($correct$) and those omitted by the Gold Standard (new). Irrelevant concepts are marked $spurious$. In terms of these three categories of concepts, $OPrecision$ is defined as follows:

$$OPrecision = \frac{correct + new}{correct + new + spurious}$$

Note that, since in this evaluation we only wish to estimate how domain representative the learned ontology is (i.e., precision), we do not need to calculate the number of Gold Standard concepts that have not been discovered by the automatic method (though this number simply represents the difference between the number of all Gold Standard concepts and the number of correctly identified concepts). We will consider the recall of the learning method in next Section 8.3.3.

We also evaluate the quality of the taxonomic relations. For this we count the number of taxonomic relations discovered between domain relevant (i.e., $correct$ and new) concepts ($allRels_{Relevant}$). Then an expert assesses how many of these taxonomic relations express indeed an *isA relation* ($allRels_{Correct}$). The $TaxoPrecision$ metric is the ratio of correctly identified isA relations over all taxonomic relations between domain relevant concepts that were automatically discovered.

$$TaxoPrecision = \frac{allRels_{Correct}}{allRels_{Relevant}}$$

Useful side-results of the expert evaluation were the opinion and suggestions of the experts (see Section 8.5.2). This qualitative evaluation provided valuable ideas for further improvements.

8.3.3 Ontology Comparison

In the final evaluation stage we compare each extracted ontology to the manually built Gold Standard ontologies in the corresponding domain. For the lexical comparison, our first metric denotes the shared concepts between the manual and extracted ontology. This metric was originally defined in (Maedche and Staab, 2002) as the *relative number of hits* (RelHit), then renamed in (Cimiano et al., 2003) to Lexical Overlap (LO). Let L_{O_1} be the set of all *domain relevant* extracted concepts ($correct$ and new) and L_{O_2} the set of all concepts of the Gold Standard. The Lexical Overlap is the ratio between the number of concepts shared by both ontologies (i.e., $correct$ - the intersection of these two sets) and the number of all Gold Standard concepts (noted *all*). Intuitively, this metric is equivalent to recall while the previously defined OPrecision represents precision. If two or more correctly extracted concepts are equivalent to a single concept in the Gold Standard (e.g., *AddModel*, *LoadOntology* are equivalent to *AddOntology*) then only one of them is counted. Therefore $L_{O_1} \cap L_{O_2}$ contains only individual concepts (noted $correct_i$).

$$LO(O_1, O_2) = \frac{|L_{O_1} \cap L_{O_2}|}{|L_{O_2}|} = \frac{correct}{all}$$

The extracted ontology can often bring important additions to the manual ontology by highlighting concepts that were ignored during its creation. We are not aware of any previously defined metric for measuring these additions. Therefore, we define Ontological Improvement (OI) as the ratio between all domain relevant extracted concepts that are not in the Gold Standard (noted *new*) and all the concepts of the Gold Standard.

$$OI(O_1, O_2) = \frac{|L_{O_1} \setminus L_{O_2}|}{|L_{O_2}|} = \frac{new}{all}$$

For comparing the taxonomic structures of the Gold Standard and the extracted ontology we employ a similar strategy as during the expert evaluation stage. We first count the number of taxonomic relations that were discovered between two Gold Standard concepts ($allRels_{RelevantGS}$). Then we count the number of relations that are qualified as *isA relations* by the Gold Standard ($allRels_{CorrectGS}$). The $TaxoPrecision_{GS}$ is the ratio of these two numbers.

$$TaxoPrecision_{GS} = \frac{allRels_{CorrectGS}}{allRels_{RelevantGS}}$$

This taxonomic comparison is simpler than the cotopy based comparison introduced by Maedche (Maedche, 2002). However, it is feasible to be performed because the compared hierarchies are not too deep and the overlap between them is quite small.

8.4 Experimental Corpora

The experimental corpora were provided by the two research projects.

Case study 1: RDF(S) tools. The first corpus, *C_RDFS*, contains 112 documents extracted from the documentation of the tools used to build the manual ontology (51 documents from Jena's API, 37 from the KAON RDF API and 24 from Sesame's API). Each document in the corpus contains the javadoc description of one method. Previous work showed that the short textual descriptions of these methods contain the most information and other javadoc elements such as the method syntax and the description of the parameters introduce a lot of noise severely diminishing the performance of the extraction (Sabou, 2004b). Therefore, we only use such short descriptions in these experiments. Also, we exclude the syntax of the method because it introduces irrelevant technical terms such as *java, com, org*. Note that earlier work on building software libraries using information retrieval methods adopted similar decisions to exclude syntax details from the analyzed corpus (Helm and Maarek, 1991). For exemplification, we show some examples of typical descriptions that we analyze:

```
List all resources which are subjects of statements. Subsequent
operations on those resource may modify this model. (Jena)

Returns a resource uri that is unique for this model. (KAON)

Uploads data from an inputstream to a repository. (Sesame)
```

Case Study 2: Bioinformatics services. The corpus for this domain, *C_BIO*, consisted of 158 individual bioinformatics service descriptions as available at the EMBOSS web site[2]. We worked only on the short method descriptions since they are significant for Web service descriptions in general being similar to descriptions found in online Web service repositories such as XMethods[3]. The detailed descriptions of the EMBOSS services present a specific layout which makes extraction much easier. However, using it would have biased our extraction methods towards this particular kind of documentation. Therefore, we analyze such short descriptions only:

```
Replace or delete sequence sections.

Find antigenic sites in proteins.

Cai codon usage statistic.
```

8.5 Results

In this section we present an evaluation and comparison of the two implementations of the framework. We ran both implementations on both corpora presented in Section 8.4 and used the evaluation criteria described in Section 8.3 to evaluate them. The ontology building algorithm was adjusted to follow the modelling principle employed by each Gold Standard ontology (i.e., producing compound functionality concepts for case study 1 and simple action verb based concepts for case study 2). In order to get an insight in

[2]`http://www.hgmp.mrc.ac.uk/Software/EMBOSS/Apps/`
[3]`http://www.xmethods.net/`

the efficiency of our pruning heuristics we evaluated both a pruned (i.e., completing all processing steps in Figure 7.3) and an un-pruned (i.e., excluding the last processing step from Figure 7.3) version of the extracted ontologies.

8.5.1 Term Extraction

The results of the first evaluation stage (see Table 8.1) indicate a better performance of *M_DEP* in both domains, the F-measure being higher for *M_DEP* than for *M_POS*. In each corpus *M_DEP* resulted in higher recall than *M_POS* even if the precision slightly decreased. However, in the context of ontology learning, in general, and for Web service domain ontology learning in particular, recall is often more important than precision: the domain experts prefer deleting a few concepts rather than missing some important concepts. Note that our results for precision are in the same range as those of the OntoLearn system (Navigli et al., 2004) - their precision is ranging from 72.9% to about 80.0%. However, we obtain a higher recall compared to the 52.7% values achieved by OntoLearn.

	C_RDFS		***C_BIO***	
	M_POS	**M_DEP**	**M_POS**	**M_DEP**
$correct_{extr}$	471	549	319	384
all_{corpus}	631	631	446	446
all_{extr}	658	774	393	480
TRecall	**75%**	**87%**	**72%**	**86%**
TPrecision	**72%**	**70%**	**81%**	**80%**
F-measure	**73**	**78**	**76**	**83**

Table 8.1: Term extraction results for both case studies.

The errors are mostly due to mistakes in the output of the linguistic analysis tools. It is well known that generic tools perform worse on specialized corpora than on newspaper style texts that were used for their initial training. For example, verbs at the beginning of the sentence are often mistaken for nouns thus causing a lower recall. It is likely that these performance values will remain in this range unless we train the linguistic analysis tools for this specific sublanguage. Note also that the performance of dependency parsing is sensitive to the length and complexity of the analyzed texts. Fortunately, the majority of sentences in our corpora are simple and allow a correct analysis. This partially explains the better performance of the dependency parsing based implementation.

A second source for errors are spelling and punctuation mistakes. The RDF(S) related corpus *C_RDFS* has, from this perspective, a lower quality than the bioinformatics corpus *C_BIO* and, indeed, this affects the $Precision$ of the extracted set. This leads to the conclusion that textual sources from Web service catalogs should be preferable to low quality code documentation.

8.5.2 Expert Evaluation

The results of the expert evaluation for the extracted ontologies are shown in Table 8.2 (for the RDF(S) domain) and Table 8.3 (for the bioinformatics domain). For both data sets the *M_DEP* method resulted in a slightly decreased ontology precision as a direct consequence of a lower term extraction precision (see *TPrecision* in Tables 8.2 and 8.3). The explanation of this correlation is that more incorrect extracted terms will lead to more incorrect concepts. Actually, we can trace down the cause of this behavior to the quality of the analyzed texts, and conclude that low quality texts will lead to less precise ontologies.

The pruning mechanism increased the ontology precision in both domains and for both methods leading to precisions in the range of 57% (*M_DEP*, case study 1) to 74% (*M_POS*, case study 2). This means that more than half of the concepts of all pruned ontologies are relevant for the analyzed domain thus offering a good start for building a domain ontology.

	Not Pruned		**Pruned**	
	M_POS	**M_DEP**	**M_POS**	**M_DEP**
$correct_{(i)}$	29(23)	35(27)	24(20))	29(21)
new	70	77	45	65
$spurious$	165	211	46	71
OPrecision	**38%**	**35%**	**60%**	**57%**
LO	**46%**	**54%**	**40%**	**42%**
OI	**140%**	**154%**	**90%**	**130%**

Table 8.2: Evaluation results for the RDF(S) domain, case study 1, in the expert evaluation and ontology comparison phases.

	Not Pruned		**Pruned**	
	M_POS	**M_DEP**	**M_POS**	**M_DEP**
$correct$	25	27	12	18
new	140	157	64	80
$spurious$	98	105	26	39
OPrecision	**63%**	**63%**	**74%**	**72%**
LO	**20%**	**22%**	**10%**	**14%**
OI	**112%**	**126%**	**51%**	**64%**

Table 8.3: Evaluation results for the Bioinformatics domain, case study 2, in the expert evaluation and ontology comparison phases.

The taxonomic evaluation results (see Tables 8.4 and 8.5) show that both methods identified a similar number of taxonomic relations per corpus (18/17 for *C_RDFS* and 78/73 for *C_BIO*). Naturally, *C_BIO* resulted in much more taxonomic relations given the high level of compositionality of bioinformatics concepts. For both corpora and both extraction methods all extracted taxonomic relations were correct ($TaxoPrecision$ = 100%). This indicates that the hierarchy building algorithm that we used, even if simple, performs well. A similar conclusion was drawn by a comparative study of several taxonomy building algorithms which proved that the vertical relation based algorithm had the highest precision (Cimiano et al., 2004c). However, this algorithm has a low recall being able to identify only compositionality based taxonomic relations. Note that besides correctly discovering the taxonomic relations existing in the Gold Standards, many new taxonomic relations were discovered as well (either between Gold Standard - Gold Standard concepts, Gold Standard - new concepts or new - new concepts).

The effect of the pruning mechanism on the taxonomic structures is different for the two corpora. Namely, in *C_BIO* more than half of the correct taxonomic relations disappear after pruning (unlike *C_RDFS* where the effect of pruning is not so radical). One of the major reasons for this behavior is that, in bioinformatics, due to the compositionality of terms, deep data structure concept hierarchies are created where the frequency of the concepts decreases with their generality. These low frequency specialized concepts were often pruned even if important, thus many taxonomic relations being deleted with them. A solution would be to decrease the pruning threshold when advancing deeper into the hierarchy. Also, since the ontology precision was already high without pruning, we might have adopted a lower value for the overall pruning threshold.

Qualitative Evaluation. Besides our quantitative results, we derived several interesting observations from the comments of the domain experts who performed the evaluation.

Recall vs. Precision. It seams that the cleanness of the ontology is not of major importance for the ontology engineer. Often even concepts that are not included in the final ontology are useful to give an insight in the domain itself and to guide further abstraction activities. We should therefore concentrate on increasing the recall of the term extraction process even at the expense of its precision. This could be done by extending our original set of extraction patterns to cover more cases, as well as by adjusting the extraction tools to this sublanguage through training.

Synonymy. During the evaluation, the expert recognized several potential synonym sets such as: {*find, search, identify, extract, locate, report, scan*}, {*fetch, retrieve, return*}, {*pick, select*}, {*produce, calculate*} or {*reverse, invert*}. Synonymy information is an important piece of knowledge for semantic Web services. Especially search and matchmaking algorithms would benefit from knowing which concepts are equivalent. The ontology engineer can decide to include synonymy in his ontology in different ways. For example, he can maintain all these concepts and describe their synonymy via an explicit mapping (e.g., owl:equivalentClass). Alternatively, he can choose to maintain one single concept per synonym set and link all lexical synonyms to this concept. Automatic acquisition of synonymy information remains important future work.

Abstractions. The experts often redistributed the extracted domain concepts according to their domain view. For example, two subclasses identified for *Protein* belong to different domains, molecular biology and bioinformatics, and have to be placed in the corresponding hierarchies accordingly. Such abstractions need to be still manually created according to the ontology engineers view on the domain. However, the abstraction step is considerably supported if the expert has an overview of relevant domain concepts.

Support. The curators considered the extracted ontologies as a useful start for deriving a domain ontology. Several complex structures could be directly included in a final ontology (e.g., the *Site* hierarchy in Figure 7.4), or provided helpful hints on how certain concepts interrelate. The most appreciated contribution was that the learned ontologies even suggested new additions for the manually built ontologies.

8.5.3 Ontology Comparison

In the final evaluation step we perform a comparison of the automatically learned ontologies with the corresponding manually built Gold Standard ontologies. In case study 1, the unpruned ontology extracted with *M_DEP* contains more than half of the concepts existing in the manually built ontology (LO = 54%) and many new potential concepts (OI = 154% in Table 8.2). These values are lower for the unpruned ontology derived with *M_POS*. Lexical overlap was computed based on the individual correctly extracted concepts ($correct_i$) shown within brackets. The behavior of the pruning mechanism was satisfactory. While pruning has almost doubled the ontology precision (from 38% to 60% for *M_POS* and from 35% to 57% for *M_DEP*) it only slightly affected the lexical overlap. Ontological improvement was more affected (e.g., reduced from 140% to 90% for *M_POS*) because many of the newly identified concepts possibly have a low domain relevance. Therefore our pruning distinguishes between important domain concepts and less important concepts.

	Not Pruned		**Pruned**	
	M_POS	**M_DEP**	**M_POS**	**M_DEP**
$allRels_{Relevant}$	18	17	12	14
$allRels_{Corect}$	18	17	12	14
$TaxoPrecision$	**100%**	**100%**	**100%**	**100%**
$allRels_{RelevantGS}$	4	3	2	2
$allRels_{CorectGS}$	4	3	2	2
$TaxoPrecision_{GS}$	**100%**	**100%**	**100%**	**100%**

Table 8.4: Taxonomy Evaluation results for the RDF(S) domain, case study 1.

	Not Pruned		**Pruned**	
	M_POS	**M_DEP**	**M_POS**	**M_DEP**
$allRels_{Relevant}$	78	73	27	30
$allRels_{Corect}$	78	73	27	30
$TaxoPrecision$	**100%**	**100%**	**100%**	**100%**
$allRels_{RelevantGS}$	10	11	5	6
$allRels_{CorectGS}$	10	11	5	6
$TaxoPrecision_{GS}$	**100%**	**100%**	**100%**	**100%**

Table 8.5: Taxonomy Evaluation results for the Bioinformatics domain, case study 2.

In the case of case study 2, the comparison with the bioinformatics Gold Standard (the ontology currently used in Web service description), shows the same trend but registers less success than the RDF(S) case study (see Table 8.3). The unpruned ontologies cover only 20 - 22% of the manual ontology for both methods, even if they suggest many new possible concepts (OI = 112%/126%). The pruning behavior is less satisfactory in this case: it reduces both the lexical overlap and the ontology improvement to half while resulting in a low ontology precision increase. This behavior is also explained by the fact that low frequency specialized terms were pruned even if important (see our explanation in 8.5.2).

Why is the lexical overlap so low? The major cause for ontological losses was that the curator also included concepts about the fields of biology, bioinformatics and informatics that are not present in the corpus. For this he relied on his expert knowledge and other ontologies in the domain (see Section 7.2). See, for example, the organism - primate hierarchy or the hierarchy of measurement units in Figure 8.2. Further, the curator relied on a set of compound concepts to define different "views" on services. For example, the left hand side of Figure 8.3 contains a snapshot of a set of concepts that define different kinds of services based on their compositionality or the types of inputs and outputs. These concepts that define views represent 18% of all concepts in the Gold Standard. Our algorithm is not able to learn such views, however, it is feasible to extend it in this direction.

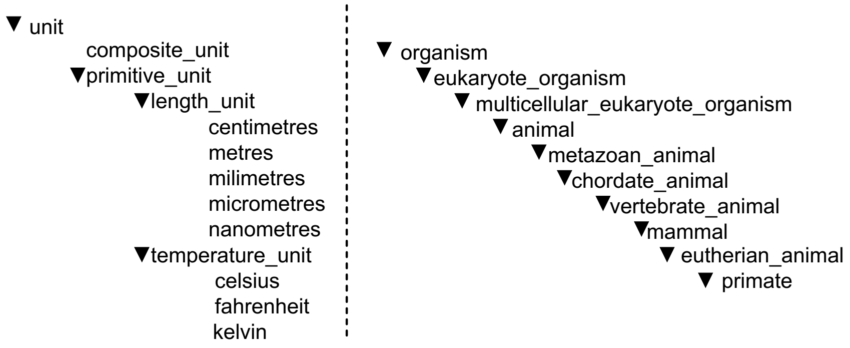


Figure 8.2: Example of parts in the ^{my}Grid ontology that are not mentioned in the corpus.

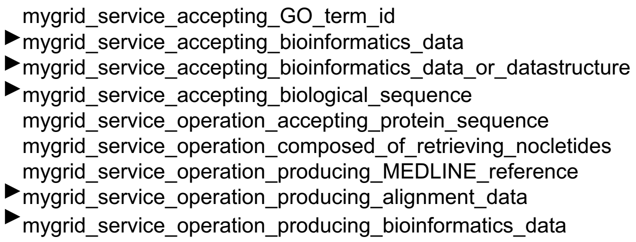

mygrid_service_accepting_GO_term_id
▶mygrid_service_accepting_bioinformatics_data
▶mygrid_service_accepting_bioinformatics_data_or_datastructure
▶mygrid_service_accepting_biological_sequence
mygrid_service_operation_accepting_protein_sequence
mygrid_service_operation_composed_of_retrieving_nocletides
mygrid_service_operation_producing_MEDLINE_reference
▶mygrid_service_operation_producing_alignment_data
▶mygrid_service_operation_producing_bioinformatics_data

Figure 8.3: A hierarchy of different views defined on services.

Why is the ontological improvement so high? Our results suggest that the ^{my}Grid ontology curator worked rather independently from the given corpus during the building of the Gold Standard as he missed many concepts named in the corpus. Post-experimental interviewing revealed that the examination of the corpus was not meticulous. He used "just in time ontology development": concepts were added if needed for describing a certain service. Note also that he worked on a subset of our analyzed corpus (100 service descriptions instead of 158 analyzed by us). Name changes could also account for the mismatch. The curator expanded abbreviations or created a preferred term for several synonyms (e.g., *Retrieving* for fetch, get, return). In fact, he acknowledged that the automatic approach leads to a more faithful reflection of the terms the community uses to describe their services.

8.5.4 Comparison with Other Ontology Learning Tools

While the primary goal of this research was that of adapting existing techniques to the Web services context rather than developing new ones, we still attempted to compare our tool with other existing tools. This task was hampered by several factors. First, few ontology learning tools are publicly available for download and experimentation. We only know about TextToOnto and OntoLT. However, these tools are essentially ontology learning workbenches that provide a set of generic techniques and allow their customization to the user's needs. For example, both tools offer a way to encode domain specific extraction patterns. Therefore, after tuning these tools to execute our patterns (which are the elements that make our tool tailored for Web services) we expect to have similar results. The differences would be caused by the performance of the underlying language processing tools (e.g., all tools use different POS taggers).

Despite these considerations we ran TextToOnto with its standard term extraction pattern on the data set provided by our first case study. The results are presented in comparison with the results of our methods in Table 8.6. As expected, the results are lower than those obtained when using domain specific extraction patterns. In particular the lexical overlap and the ontology improvement metrics rich values lower than half of those obtained with our methods. One of the straightforward explanations is that TextToOnto's standard pattern ignores all functionality type concepts. Indeed, such concepts are specific to our Web service context and are irrelevant for generic ontology learning tasks. Therefore, TextToOnto's lower performance indicates that it was not fine-tuned for the context of Web services. In any case these results do not suggest that TextToOnto is a non-performant *generic* ontology learning tool.

	Not Pruned		
	M_POS	**M_DEP**	**TextToOnto**
$correct_{(i)}$	29(23)	35(27)	9
new	70	77	39
$spurious$	165	211	154
OPrecision	**38%**	**35%**	**23%**
LO	**46%**	**54%**	**18%**
OI	**140%**	**154%**	**78%**

Table 8.6: A comparison of our methods with TextToOnto on the data provided by case study 1.

To complete our comparison, we searched for performance results that could give an insight in how our tool compares to others. This is again difficult because many ontology learning related papers do not report any evaluation results or use metrics that are incompatible with ours. We found that (Cimiano et al., 2003) (and other papers related to this work) reported on a formal concept analysis based ontology learning algorithm where the lexical overlap reached a maximum value of 27.71%. We provide this result as an indication of a typical performance value and we are aware that it is not enough to give an overview of the state of the art performance in ontology learning. In fact, the ontology learning community just reached the stage when an effort is made towards establishing evaluation benchmarks and metrics to be adopted by the whole community with the purpose of being able to compare the different efforts.

8.6 Discussion

The work presented in Chapter 7 and Chapter 8 is motivated by the observation that, while domain ontologies are of major importance for semantic Web services, their acquisition is a time-consuming task which is made more difficult with the increasing number of Web services. The ultimate goal of our research is to build tool support for (semi-) automatic ontology learning in the context of Web services. In these chapters, we presented the first stage of the research which *pioneered* ontology learning for Web services. Ontology learning in the context of Web services raises several non-trivial questions while inheriting some unsolved issues from ontology learning in general. The aim of the work was to better understand the problem at hand and to investigate what technologies might be feasible to use. We addressed a set of fundamental issues such as: data source selection, choosing the appropriate learning algorithms, deciding on evaluation methodologies and metrics, considering usability. The contribution of our work lies in identifying and tackling these issues as well as offering (partial) solutions to them. We discuss our findings in this section and point out future work that can be based on our findings so far.

Selecting Data Sources. Traditional ontology learning is predominantly focused on learning ontologies that describe a set of textual documents. In this case the data sources for ontology learning are those textual sources. However, there are several possible data sources that could be used for learning Web service ontologies. First, resources connected to the underlying implementation might provide useful knowledge about the functionality of the Web service since Web services are simply web accessible software products. Such are the source code, its textual documentation or existing UML diagrams. Second, one could use Web service specific data sources such as associated WSDL files or activity logs.

During our work, we observed that Web services are almost always accompanied by a short textual description of their functionality which helps a user to quickly understand what the service does. Such descriptions exist in on-line Web service repositories such as XMethods and also in javadoc style code documentations. Besides being the most available sources, short textual descriptions of Web services (1) are characterized by a low grammatical quality and (2) use a specific sublanguage that makes them amendable to automatic processing. In our work we only considered these textual documentations. Current experiments, not reported here, show that WSDL documents also contain valuable information about the service that they describe being more detailed than the short textual descriptions we have considered. As future work, we will combine these two sources.

Choosing Learning Techniques. The goal of our work was to adapt existing ontology learning techniques for this specific domain rather than developing novel ones. The choice of these techniques depends on the kind of data sources considered. For example, UML diagrams would require semantic mapping techniques (Falkovych et al., 2003) which are essentially different from natural language processing techniques used for textual sources. We designed a framework for learning ontologies from textual Web service descriptions and implemented two methods within this framework that use natural language processing techniques of different complexity. During the design and evaluation of this framework we derived the following observations:

Simple techniques work fine in well-defined contexts. Despite our methods are based on relatively simple, off the shelf term extraction and ontology building techniques, the learned ontologies have a good quality (as we argue in the evaluation part of this discussion). One explanation of this phenomenon is that we are considering a well-defined ontology learning task and work on specialized texts with strong sublanguage characteristics. Our context differs from efforts to design generic ontology learning methods which have to run on any kind of textual sources and build only generic ontological structures. Therefore, generic ontology learning methods are harder to build and they rely on more complex techniques. We believe that since Semantic Web technology is used in a variety of specialized domains, tools that allow easy adaptation of basic ontology learning methods will have an increased practical importance. We consider this work as a first step towards context directed ontology learning.

Deeper linguistic analysis seams to increase performance. The dependency relationship based method performs better than the POS tag based method. First, it increases the recall of the term extraction from the corpus with little impact on the term extraction precision. Second, while the extracted ontologies have a lower precision this is compensated by higher values for ontological overlap and improvement (but experts consider precision less important than domain coverage). Another argument for the use of dependency parsing is that the richer dependency information makes it much easier to write and establish new syntactic patterns than surface ones.

It should be possible to build a domain independent tool. The sublanguage features on which our methods build can be identified in Web service descriptions written for various domains. Therefore, we believe that it is feasible to build an ontology learning tool that is tailored to the context of Web service descriptions but which is applicable across different application domains. A good indication is that both our methods perform similarly in two different domains. However, corpus particularities can influence the extraction performance. For example, punctuation and spelling mistakes lead to a low term extraction precision, and consequently, to a less precise ontology.

We envision several improvements for the basic framework presented here. First, we wish to extend the method with more fine-grained extraction patterns to complement the current high coverage patterns. There are a considerable number of sublanguage patterns that were not used in this iteration but could provide rich input for the ontology building. We also wish to exploit pronoun resolution and prepositional phrases. Machine learning techniques could help in discovering some new patterns. It is interesting to investigate if these fine-grained lexical-based patterns would still make our framework applicable across different domains. Second, we want to enhance the ontology building step. Use of synonymy information during the conceptualisation step is an important development possibility. Further, we would like to concentrate on strategies that enrich the basic extracted ontology. For example, defining different views on a set of domain concepts or providing a set of patterns that act on the already extracted semantic information.

Evaluation. To achieve the goal of this first stage of research, that of understanding the applicability of ontology learning techniques in the context of Web services, our evaluation was directed towards getting an insight in the performance of the learning methods. A possible extension of our evaluation would be to test the robustness of the methods, i.e., to see how their performance is affected when applied on incrementally enlarged data sets in the same domain. One of the major future tasks is to perform a task-based evaluation of the extracted ontologies in a Web service context, e.g., by powering Web service tasks such as search, matchmaking, etc. This would complement the current evaluation and indicate the appropriateness of the learned ontologies for Web service tasks. However, we believe that the current evaluation is sufficient to encourage the continuation of this line of work.

One of our major observations is that the evaluation of ontology quality is difficult because the Gold Standards do not faithfully represent the knowledge in the corpus: the domain experts omit several concepts because it is not feasible to read and analyze all available documents in a reasonable time frame. Complementary, our methods extract ontologies that contain a high percentage of domain relevant concepts from a corpus. The amount (and domain relevance) of extracted concepts can be influenced by tuning the pruning algorithm.

The quality of the ontologies, even if extracted by using simple methods, was encouraging. We state this based on the fact that similar work on ontology learning in open domain reports on a maximum lexical overlap of 27,71% while we reach 54% procent in some cases. Further, during the qualitative evaluation, the experts indicated that the extracted ontologies represent more faithfully the knowledge in the corpus and that they provide a useful start for building a domain ontology. Indeed, providing ontology curators with ontologies that contain half of the extractable concepts is a considerable help for this time consuming task.

Chapter 9

Conclusions and Future Work

The goal of this thesis is to enhance the quality of Web service ontologies and to facilitate the process of building these ontologies. To achieve this goal we investigated the following three research questions:

- *Q1. What requirements should Web service ontologies fulfill?*
- *Q2. How to enhance the quality of generic Web service ontologies?*
- *Q3. Is semi-automatic acquisition of Web service domain ontologies feasible?*

In the first part of the thesis we investigated the first research question and established a set of requirements for Web service ontologies based on our experiences with semantic Web services technology. In the second part of the thesis we focused on the second research question by describing a set of methods to check whether a generic Web service ontology fulfills the requirements stated in the first part of the thesis. These methods also help enhancing the quality of the ontologies. In the final part of the thesis we turned our attention to Web service domain ontologies and described our efforts to build a semi-automatic solution for their acquisition (as stated by our third research question).

In this chapter we discuss our conclusions and contributions related to the three research questions that we investigated (in Section 9.1) and present ideas for future work (in Section 9.2).

9.1 Conclusions and Contributions

In this section we detail our major conclusions and describe our contributions to the state of the art. We organize our discussion around the three major topics introduced by our research questions.

The general conclusion of our work is that Web service ontologies should fulfill several requirements (as discussed in Section 9.1.1) and that several methods can be employed to facilitate achieving these requirements. One part of the presented methods relate to generic Web service ontologies and are concerned with improving their quality (see Section 9.1.2). The rest of the methods aid the semi-automatic acquisition of Web service domain ontologies as discussed in Section 9.1.3.

9.1.1 Requirements for Web Service Ontologies

Based on our work with the emerging semantic Web services technology we concluded that:

Web service ontologies should fulfill a set of requirements. A major requirement for a generic Web service ontology is high quality. In particular, the ontology should provide a high *modelling expressiveness* so that a large variety of services can be modelled. *Clear semantics and a rich formalization* of these semantics would ensure the support for complex reasoning tasks performed by artificial agents. Then, another measure of quality is whether the captured generic knowledge is *adaptable* for use in similar domains. It would also be desirable that a mapping (*harmonization*) between different evolving generic ontologies can be achieved in order to ensure a basis for their comparison. Finally, if such a generic ontology is to be used in a consistent way by a large and open group of users, it is compulsory that adequate usage examples and guidelines are provided (we call this characteristic *usability*). Web service domain ontologies should provide a broad domain coverage and they should contain the knowledge that describes large and dynamic Web service collections.

Our contribution is to identify and describe these requirements.

9.1.2 Enhancing Generic Web Service Ontologies

In the second part of the thesis we presented a set of methods that could enhance the quality of generic Web service ontologies. We used these methods to test the quality of OWL-S, to identify its limitations and to propose solutions. The outcome of our work is often generally reusable when developing other generic ontologies as well (see Section 9.1.4). We hereby briefly describe the main methods that we used, the conclusions we reached and our contributions:

The modelling expressiveness of the OWL-S generic Web service ontology has been tested and improved by using it to describe real life services. For testing the modelling expressiveness of OWL-S we used it to describe a set of services (Chapter 4). This exercise yielded the conclusion that *OWL-S had a low modelling expressiveness*. We found that the ontology relies on an unclear conceptual model as it used different metaphors when representing a service at Profile and Process level. Also, the links between these different parts were unclear. This lead to discovering several internal inconsistencies and limitations to model situations as ad-hoc polymorphism or complex internal structures.

Our contribution is to identify, exemplify and document some modelling use cases that should be covered by any generic Web service ontology. For OWL-S, we provided a set of solutions to the identified limitations. Some of these suggestions have already been incorporated in newer versions of the ontology.

The adaptability of OWL-S has been tested by reusing it in a related context. One of the positive conclusions was that *OWL-S was easily adapted to similar domains (high adaptability)*. In particular, we reused several of the design principles that underly OWL-S to build an ontology that would describe API-based software components and that would support several tasks within a semantic middleware (see Chapter 5). The result of our work, the ASSW ontology was embedded in the ASSW application server already.

Our contributions are (1) identifying the reusable parts of OWL-S and (2) the extension of OWL-S for use with middleware systems. This work was published in the *middleware community and appreciated as being useful and innovative (Oberle et al., 2004a).*

The alignment of OWL-S to a foundational ontology ensured clear semantics and a rich formalization of these semantics. An investigation of OWL-S from an ontological perspective revealed that it presented conceptual ambiguity, poor axiomatization, loose design and narrow scope thus leading us to the conclusion that *the ontology has a low clarity in semantics and these semantics are poorly formalized* (see Chapter 6). To overcome these limitations we performed an alignment to DOLCE, a rigorously formalized foundational ontology. To bridge the conceptual gap between the high level concepts of DOLCE and the concepts of OWL-S we designed an intermediate ontology, the Core Ontology of Services (COS).

One of the contributions of our work, form DOLCE's perspective, is to extend the DOLCE library with this software related module. Besides offering a set of valuable observations to the OWL-S community, the contributions of this part of our work are the development of COS and the alignment methodology that can be generically reused for aligning other generic ontologies as well. By the alignment of multiple ontologies to COS, a mapping (harmonization) can be achieved between their concepts.

9.1.3 Learning Web Service Domain Ontologies

In the third part of the thesis we investigated the feasibility of an automated solution to support the process of building Web service domain ontologies. Our work resulted in the following conclusions and contributions.

The Web service context exhibits some specific features that can be exploited for developing an automated ontology acquisition solution. From the two case studies that we performed we concluded that domain ontology building scenarios can be different but that they exhibit a few important characteristics that have to be taken into account when developing an automated solution (see Chapter 7). We observed that the generally available data sets for ontology learning are textual descriptions of the offered functionalities (e.g., API documentation, or Web service comments). These textual descriptions have a low grammatical quality and they employ natural language in a specific way (they use a sublanguage). Further, both static and procedural knowledge should be extracted by the automated tool.

Our contribution is to get a better understanding of the problem at hand and identify aspects that can be used to build automated ontology acquisition methods.

Simple ontology learning methods can be adapted to the Web service context. After understanding the particularities of the Web service context we built a framework that adapted ontology learning methods to the specificities of the domain. This framework relies on the observation that, due to the sublanguage characteristic of the analyzed texts, simple pattern based techniques can be used to extract semantic structures from the regularities of the sublanguage. This framework allows the implementation of different extraction methods. For example, we used linguistic analysis of different complexity to perform ontology learning.

Our contribution is to draw up the general lines of the framework and to provide two different implementations (see Chapter 7).

Evaluation shows that the extracted ontologies have a good quality. Evaluation of ontology learning methods is not a straightforward task. We analyzed several different approaches and used a combination of these to define our evaluation metrics. The results of our evaluation show that (1) *our framework, even if designed for the Web service context, is applicable across application domains* (our methods perform comparatively in two different domains). The results also imply that (2) *deeper linguistic analysis leads to better results* being less sensitive to the imperfections of the corpus. Finally, the experts involved in the evaluation indicate that (3) *the extracted ontologies represent faithfully the knowledge in the corpus and that they provide a useful start for building a domain ontology*. In fact, the extracted ontologies contain, on average, more than 50% of domain relevant concepts. Comparatively to other ontology learning efforts in open domains we were able to obtain higher values for our ontology overlap metric (almost double) . The strength of our method is in complementing human experts and supporting them in a meticulous inspection of the corpus thus contributing to a broad coverage of the domain's terminology.

Our contribution at this point, besides giving an insight in the performance of the learning methods, is to establish a baseline for evaluating ontology learning in the Web services context. We collected two data sets and their associated manually built Gold Standard ontologies that can be used as a benchmark for evaluating new solutions (see Chapter 7). We also established an evaluation scheme that can be used to evaluate ontology learning in this domain.

We provide an easy to extend prototype by implementing our methods in a generic NLP framework. To achieve our goal of facilitating the acquisition of Web service ontologies, besides building high performance ontology learning methods, we need to implement them in tools that are easy to adapt and use by experts. Aware of this requirement, we implemented our prototype so that it is easy to debug, extend or adapt by developers of ontology learning methods in the Web service context (see Chapter 7). Our implementation relied on the use of a generic NLP workbench, GATE. In our implementation we experimented with visual means to present the extracted knowledge.

Our contribution is a prototype system that can be easily adapted to further needs of the Web services context.

9.1.4 A Note on the Generality of our Results

The work presented in this thesis was performed in the context of OWL-S because at that time it was the only generic Web service ontology. During our work other semantic Web service frameworks have merged, most notably WSMO. In this section we argue that several of our results are general enough in order to be reused in the context of other semantic Web service efforts as well.

Our claim is based on the comparison of these Web service efforts as reported in two recent research papers, (Cabral et al., 2004; Lara et al., 2005). The main conclusion of both papers is that these Web service frameworks cover complementary aspects of Web services. For example, OWL-S relies on an agent oriented approach while WSMO focuses on business requirements such as trust and security (Cabral et al., 2004). This fact makes the comparison of the ontologies non trivial. These Web service frameworks also differ in their maturity level. Indeed, as concluded by (Lara et al., 2005), OWL-S is more mature in some aspects having a well defined process model and grounding component. On the other hand, WSMO has a clearer conceptual model and makes a clear difference between requestors and providers. A common characteristic is that all approaches rely on the use of ontologies to describe Web services. The OWL-S ontology has to be complemented by a domain ontology, then, in the case of WSMO domain ontologies and task ontologies are used in combination.

The authors of the comparison in (Lara et al., 2005) conclude that further future work is needed to better compare WSMO and OWL-S. First, a set of real use cases should be modelled with both approaches to determine their applicability in real-world settings. Second, a formal mapping should be established between these two ontologies after WSMO will be completed in all aspects.

We hereby provide a brief overview of our results that are reusable in the context of WSMO as well:

Modelling use cases. The modelling use cases identified when describing our services are generally valid and should be covered by any generic Web service ontology. Indeed, they could be reused in the context of the first line of future work identified in (Lara et al., 2005) - that of modelling several real use cases using both ontologies to assess their applicability in real life settings.

The alignment methodology. Our alignment methodology can serve as an example when aligning other generic Web service ontologies to DOLCE or other foundational ontologies. Indeed, alignment could have several advantages at this stage of development in the semantic Web services field. First, it could help to identify potential conceptual ambiguities in the currently developed WSMO ontology. Second, by aligning both WSMO and OWL-S to DOLCE it would be easier to construct a formal mapping and to get a better understanding of the differences between these ontologies.

The Core Ontology of Services. The COS ontology provides an conceptual bridge between the high level concepts of DOLCE and the domain of software descriptions. We claim that this ontology will facilitate the alignment of other generic Web service ontologies to DOLCE because these ontologies will have to be aligned to the concepts of COS rather than directly to the generic concepts provided by DOLCE.

The methods for learning Web service domain ontologies. These methods are independent of the generic Web service ontology that is used. Nevertheless, different semantic Web services frameworks might rely on different ways of organizing the domain knowledge. Our methods should provide a good basis for adaptation to the particular needs of a certain framework.

9.2 Future Work

In this thesis we used a variety of methods to enhance the quality of Web service ontologies and to facilitate the process of building them with the ultimate goal of supporting the semantic Web services research as a whole. However, even if we covered a variety of issues on this topic we envision that important research still has to be done in the context of enhancing the semantic Web services field (see Section 9.2.1). Our work has also a significance for the fields of software engineering and ontology learning. We detail possible future work on these fields in Sections 9.2.2 and 9.2.3.

9.2.1 Semantic Web Services

There are several possibilities of extending our research in the area of semantic Web services.

1. Enhancing and harmonizing generic Web service ontologies. As we discussed in previous sections, several of the results of our work with respect to generic Web service ontologies can be reused when building any other generic ontology. Therefore, we encourage applying the results of this work in the context of novel initiatives such as WSMO. In particular, an important task would be the harmonization of several generic Web service ontologies through their alignment to the Core Ontology of Services. Such a harmonization could benefit the field in several ways. First, it would be beneficial to determine flaws and problematic aspects of these ontologies. Second, this harmonization would provide a basis for comparing the different ontologies thus allowing their complementary development.

There is an awareness in the semantic Web services community that some level of standardization is becoming necessary, especially given the increasing number of different approaches that became recently available. This need became evident at a recent W3C workshop that aimed at identifying potential standardization work in the area of semantic Web services (Battle and Martin, 2005). Participants to this forum have identified several approaches to realizing semantic Web services. These are: Web-Service Modelling Ontology (WSMO), the OWL Service ontology (OWL-S), Web-Service Semantics (WSDL-S[1]) (Miller et al., 2005), the First-Order Ontology for Semantic Web-Services (FLOWS) (Berardi et al., 2004), the Siemens Service Description Reference Model (SDRM), IRS III and Meteor-S[2]. The general opinion was that *"agreement on a common core set of specifications would be feasible"* and desirable, however, the different maturity level of the approaches might not permit any standardization efforts yet.

2. Ontology learning in the Web services context - continued. While our work on ontology learning addresses several important issues thus pioneering this research direction, in its current status, it can be mostly considered as a *proof of concept* that ontology learning can be performed in the context of Web services. We are aware of several limitations and we envision various ways to further develop this work. In fact, this line of work will be continued in a recently funded European research project, TAO. In particular, the following topics can be addressed:

Extracting more complex information. The implemented methods only extract simple information about services (their parameters and functionality). However, much more has to and can be learned form these descriptions. For example, the role of each parameter (input, output, precondition), constraints on the services activity (e.g., geographic radius) or quality ratings.

Using and combining more extraction sources. One way to obtain more complex information is to consider other extraction sources as well. For example, recent experiments have shown that WSDL descriptions also contain valuable and often more precise and detailed information than the textual documentations. An obvious challenge when using multiple sources is combining the knowledge extracted

[1] `http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.html`
[2] `http://lsdis.cs.uga.edu/projects/meteor-s/`

from them. Note that this issue is one of the developing topics in ontology learning (Cimiano and Voelker, 2005).

Scaling up the learning methods. We achieved encouraging results when only using relatively simple ontology learning methods. An obvious future direction is to scale up these methods in several ways. First, the use of more complex taxonomy building algorithms and methods that consider synonymy (e.g., WordNet based) are a straightforward improvement. Second, we have to consider that current methods were developed on repositories of Web services from a single domain. This simplifying assumption fails on open repositories, such as those on the Web that contain services from several different domains. A good way to drop all the implicit assumptions that we made during the development of the framework is to apply our methods in the context of open Web based service repositories.

Automatically detecting extraction rules. One of the possible critiques of our framework is that is relies on manually identified extraction rules. While the identification of the heuristics which underly these few rules was almost straightforward, we are aware that an automation of this process could be beneficial. There has been promising research on (partly) automating this process in the context of sublanguages ((Riloff, 1996; Grishman et al., 1986)) which we hope to be able to reuse and extend in our context.

Task based evaluation. Our current evaluation, even if complex, was only useful to get an insight in the performance of our methods. An obvious next step in our work is to evaluate the appropriateness of the automatically learned ontologies to support Web service related tasks.

Completing and extending the usability studies. While an important first step, our usability considerations are not backed up by usability tests. We wish to identify more scenarios where visualization is useful and to experiment with other visual paradigms than the Cluster Map. Naturally, all these new experiments should undergo usability tests.

3. Tools for Semantic Web services. An obvious way to continue the general direction of offering support for developing the Web services technology is building supportive tools. In fact, this need for semantic Web service tools is also clearly expressed in a recent article on this topic which acknowledges that *"the lack of standardization and tool support has so far been a limiting factor for adoption of semantic Web services technology"* (Elenius, 2005).

Several tools have been developed lately to support different tasks involved in creating semantic Web services such as automatic markup generation from WSDL (Mindswaps OWL-S API[3], OWL-S IDE[4]), finding a suitable domain ontology (METEOR-S (Patil et al., 2004)), (semi-)automatic annotation of Web services (ASSAM[5] (Hess et al., 2004)), editing and visual composition of semantic Web services (OWL-S Editor for Protege[6] (Ele-

[3]`http://www.mindswap.org/2004/owl-s/api/`
[4]`http://projects.semwebcentral.org/projects/owl-s-ide/`
[5]`http://moguntia.ucd.ie/projects/annotator/`
[6]`http://owlseditor.semwebcentral.org/`

nius et al., 2005), the OWL-S Editor developed at University of Malta[7] (Scicluna et al., 2004), ODE SWS[8] (Gomez-Perez et al., 2004)). Note, however, that all of these tools are research prototypes and that there are no *"commercial efforts to provide tools supporting SWS technologies per se"* (Battle and Martin, 2005). An obvious next step is the development of workbenches that support the entire process of semantic Web service development.

Another observation is that little automated support is provided by current tools. We argue that automatic methods should be better integrated with existing or next generation semantic Web services tools. In particular, our current efforts try to combine the automatic ontology learning and the automatic semantic annotation as done in the ASSAM tool. As such, the whole task of providing the semantic annotation (i.e., acquire the domain ontology and annotate Web services with its concepts) would be automated.

Finally, we think that it would also be beneficial to leverage on the advantages of using ontologically founded methodologies. For example, ontological design principles (e.g., such as the distinction between a situation and its description) could be formulated as reusable ontology design patterns and integrated in semantic Web service tools in a way that makes them easy to reuse.

4. Methodologies for building ontology based standards. Web service description standards built to support the semantic Web services technology are naturally based on ontological modelling. This is the case both for OWL-S and WSMO and possibly for other recommendations built in the future. Therefore we believe that it is important to analyze and formalize the process of creating these ontology based standards. This will help to better understand the development life cycle of the standard, the processes that are involved, the personnel that is needed and the resources that need to be allocated.

9.2.2 Semantic Web Enabled Software Engineering

Another promising research direction is the integration of semantic Web technologies in software engineering. In fact, during this thesis we have already initiated such an integration through the use of semantic technologies to support several tasks in the context of an application server middleware (see Chapter 5). The creation of the ASSW ontology and its follow up integration in the ASSW server proved useful and was well received by the middleware community (Oberle et al., 2004a).

Another way to contribute to the software engineering field is to adapt our work on (semi-)automatic ontology learning to improve online Web service repositories. Our analysis of the current techniques used to organize these libraries yielded that they rely on no or almost no semantic information. We believe that one of the reasons is the cost and difficulty to acquire semantic descriptions of these large and dynamic Web service collections. In fact, the situation of Web service repositories is similar to the situation of software libraries which, as concluded by a major survey (Mili et al., 1998), predominantly employ simplistic techniques that do not rely on knowledge acquisition. We detailed this analysis and our ideas about how the current situation could be improved by semantic Web technology in (Sabou and Pan, 2005).

[7] `http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html`

[8] `http://kw.dia.fi.upm.es/odesws/content.htm`

Our efforts in the direction of using semantic Web techniques in a software engineering context are not unique but they belong to an increasing trend that supports this integration. For example, the use of semantic Web techniques in software engineering is the topic of the Semantic Web Enabled Software Engineering (SWESE) workshop[9] collocated with the International Semantic Web Conference 2005.

9.2.3 Ontology Learning

A final research field where we envision possible future work is the field of ontology learning.

1. Adaptive ontology learning. With the continuous uptake of semantic Web technology in a variety of domains the need for adapting ontology learning solutions to these domains is also increasing. This trend has been identified and supported by the newest ontology learning tools such as OntoLT and Text2Onto. In particular, OntoLT allows the adaptation of pattern based ontology building approaches to different domains. Text2Onto, on the other hand, offers a comprehensive library of generic ontology learning modules that can be combined in a particular setting (however, they cannot be modified to fit certain domain constraints). Through their goal of bringing ontology learning closer to non-experts, these tools differ from the plethora of previous work which was mostly concerned with discovering new techniques.

Based on our experience in the context of Web services, we believe that in concrete domains, where a set of constraints provide a well defined ontology learning problem, simpler learning methods will provide good results. It is essential to provide easy to use tools and comprehensive guidelines about the use of these tools to allow the adaptation of ontology learning by diverse communities.

2. Evaluation of ontology learning tools. The issue of evaluation is probably the most important one for the development of the ontology learning field. In this stage, where several approaches exist, their comparative evaluation is a prerequisite for the further development of the field. However, as concluded at a workshop with the same topic (Buitelaar et al., 2004a), different techniques use different steps and have different results. For example, clustering techniques have an essentially different outcome than term extraction based methods. These factors hamper the development of a standard set of evaluation metrics. Nevertheless, initial efforts towards an ontology learning challenge are currently performed.

3. Usability in ontology learning tools. The wide use of ontology learning techniques does not only require that they are easily adaptable to new domains but also that they are incorporated in tools that are easy to use. In fact, the more general topic of ensuring a smooth user interaction with semantic based tools and technologies is the topic of several workshops (e.g., the End User Semantic Web Interaction Workshop[10] at ISWC 2005 and the End User Aspects of the Semantic Web Workshop[11] at ESWC2005).

Our experience is that the newest ontology learning tools, OntoLT and Text2Onto, support usability by providing provenance information or even using visual techniques to present their results. However, there is no concrete effort to identify a set of usability

[9]`http://www.mel.nist.gov/msid/conferences/SWESE/`

[10]`http://www.ifi.unizh.ch/ddis/iswc2005ws.html`

[11]`http://kmi.open.ac.uk/events/usersweb/`

features and to experiment with techniques that would best implement them. In this thesis we provided our view on some usability features and presented ways in which we achieved these features. We believe though that this effort needs to be detailed and taken to a community level.

Appendix A

DOLCE+’s Synchro-task in OWL Abstract Syntax

```
Class(<dolce+:synchro-task> partial
  <dolce+:elementary-task>
  restriction(<dolce+:predecessor>
   someValuesFrom (unionOf(
    <dolce+:concurrent-task>
    <dolce+:partly-concurrent-task>)))
  restriction(<dolce+:direct-predecessor>
   minCardinality(2))
  restriction(<dolce+:sequences>
   allValuesFrom
   (dolce+:planning-activity))
  restriction(<dolce+:represented-by>
   allValuesFrom (<dolce+:join-node>))
  annotation(<rdfs:label> "synchro-task")
  annotation(<rdfs:comment>
   "A task that joins a set of
    tasks after a branching"))
```

Samenvatting

Twee nieuwe ontwikkelingen veranderen op het moment het gezicht van het World Wide Web. Aan de ene kant maakt Web service technologie via web standaarden uniforme toegang mogelijk tot software componenten die aanwezig zijn op verschillende platformen en geschreven in verschillende programmeertalen. Dit heeft geresulteerd in de beschikbaarheid van software componenten die een verscheidenheid aan functionaliteit aanbieden (van monetaire conversie tot het boeken van vluchten) via het Web. Aan de andere kant reikt de Semantisch Web technologie technieken aan voor het verrijken van bestaande Web data met op logica gebaseerde formele beschrijvingen van hun betekenis. Deze semantische "markup" is machinaal te verwerken en faciliteert toegang en integratie van de enorme hoeveelheid Web data. De kern van Semantisch Web technologie bestaat onder andere uit *ontologieën,* formele representaties van algemene wereld kennis dat de elementen biedt om semantische beschrijvingen te construeren.

Een beperking van Web service technologie is dat het vinden en bijeenbrengen van services nog steeds menselijke handelingen vereist, terwijl het aantal Web services stijgt. Semantische Web Service technologie groeide uit als een antwoord op dit probleem uitgaande van het idee Web services te verrijken met een semantische beschrijving van hun functionaliteit om het vinden en integreren van services te faciliteren. Semantische Web service beschrijvingen steunen op twee soorten ontologieën. Ten eerste specificeert een *generieke Web service ontologie* de centrale aspecten van Web services onafhankelijk van het domain waarin zij opereren (b.v. inputs, outputs). Ten tweede biedt een *domein ontologie* concepten uit het domein van de Web service aan om de beschrijvende template die gebouwd is met concepten uit de generieke ontologie in te vullen (b.v. *Voedsel, Hotel, VoedselBestelling, HotelBoeking*). Wij noemen de generieke en domein ontologieën samen *Web service ontologieën*.

Een belangrijk probleem van semantische Web service technologie, dat wij in dit proefschrift ter hand nemen, is dat het bouwen van Web service ontologieën een tijdrovende en complexe taak is. Dit vormt een beperking in de ontwikkeling van het veld. Het proefschrift onderzoekt en formuleert oplossingen voor de problemen bij het construeren van zowel generieke als domein ontologieën voor Web service beschrijvingen.

In Hoofdstuk 2 beschrijven we in meer detail Web services, het Semantisch Web en semantische Web service technologie en leiden een set van vereisten af die Web service ontologieën dienen te vervullen. Een belangrijke vereiste voor een generieke Web service ontologie is een hoge kwaliteit. De ontologie moet vooral een hoge modelleer expressiviteit bieden zodat een grote varieteit van situaties kan worden gemodelleerd. Duidelijke semantiek en rijke formalizatie van deze semantiek verzekert ondersteuning voor complexe redeneertaken uitgevoerd door kunstmatige agenten. Een andere kwaliteitsmaat is of de beschreven generieke kennis ook aan te passen is voor gebruik in op elkaar gelijkende domeinen. Het is tevens wenselijk dat een afbeelding (harmonizatie) tussen verschillende evoluerende generieke ontologieën mogelijk is om een basis voor onderlinge vergelijking te verzekeren. Ten laatste, als een generieke ontologie door een grote en open groep gebruikers op een consistente wijze moet worden kunnen gebruikt is het verplicht dat adequate gebruiksvoorbeelden en richtlijnen worden verstrekt. Voor domein ontologieën is het belangrijk dat ze een breed domein bestrijken en dat ze opschalen naar een grote verscheidenheid aan services.

In het tweede deel van het proefschrift onderzoeken we methoden die de kwaliteit van generieke Web service ontologieën kunnen verbeteren. Hiervoor testen wij de kwaliteit van de *de facto* standaard voor Web service beschrijvingen, OWL-S. We bestuderen elke eis die we aan web-service ontologieen stellen, trekken conclusies over de beperkingen van de OWL-S ontology, en stellen een aantal oplossingen voor (die ook vaak van toepassing zullen zijn voor andere generieke ontologieen).

Om de modellerings expressiviteit en bruikbaarheid van OWL-S te testen hebben we het gebruikt om een set van services te beschrijven (Hoofdstuk 4). Dit werk leverde als resultaat op dat OWL-S een lage modellerings expressiviteit heeft. We concluderen dat de ontologie op een onduidelijk conceptueel model steunt omdat het verschillende metaforen gebruikte om een service te representeren op Profiel en Proces niveaus. Ook waren de verbindingen tussen de verschillende delen onduidelijk. Dit leidde tot de ontdekking van verschillende interne inconsistenties en beperkingen in modelleringssituaties zoals ad-hoc polymorphisme of complexe interne structuren. Wij bieden een set van oplossingen aan voor de geidentificeerde beperkingen, en sommigen zijn inmiddels verwerkt in nieuwere versies van de ontologie. Een andere bijdrage was het identificeren en documenteren van enige modelleer eisen die in rekenschap moeten worden genomen door iedere Web service ontologie.

Ons gebruik van OWL-S doet ons ook conclusies trekken over haar bruikbaarheid. We concluderen dat OWL-S moeilijk te leren viel omdat voorbeeld beschrijvingen van verschillende services ontbraken. Evenzo ontbraken richtlijnen voor het modelleren van bepaalde situaties. Onze contributie betrof het aanreiken van een set voorbeeld Web service beschrijvingen dat de bestaande didactische voorbeelden van het OWL-S comité kunnen aanvullen. Ook leidde de discussie rond onze modelleerbeslissingen (en twijfels) zoals gedocumenteerd in verschillende artikelen, ook al zijn deze niet vergelijkbaar met modelleer richtlijnen, ertoe dat het besef bij de gebruiksgemeenschap ontstond dat er problematische aspecten waren in hun ontologie.

Een van de positieve conclusies betrof dat OWL-S eenvoudig aanpasbaar is voor op elkaar gelijkende domeinen (hoge aanpasbaarheid). Meer specifiek hebben we verschillende ontwerp principes die het fundament van OWL-S vormen hergebruikt om een ontologie te bouwen die API gebaseerde software componenten kan beschrijven en dat verschillende taken in semantische middleware kan ondersteunen (zie Hoofdstuk 5). Het resultaat van ons werk, de ASSW ontologie, werd ingebed in en verbeterde de werking van de ASSW applicatie server.

Een analyse van OWL-S van het ontologische perspectief toonde toonde conceptuele ambiguiteit aan, matige axiomatisatie en een smal perspectief, en leidde ons tot de conclusie dat de ontologie een lage semantische helderheid bezit en dat deze semantiek tevens op zwakke wijze geformaliseerd is (zie Hoofdstuk 6). Om deze beperkingen te verwijderen gebruiken we een verbinding (*“alignment”*) met DOLCE, een diepgaand bestudeerde en rigoreus geformaliseerde basis ontologie. Om het epistemologische gat tusssen de hoog niveau concepten van DOLCE en de concepten van OWL-S te overbruggen hebben wij een tussenliggende ontologie ontworpen: de Kern Ontologie van Services (KOS). Een van de bijdragen van ons werk vanuit het perspectief van DOLCE is het uitbreiden van de DOLCE bibliotheek met deze software gerelateerde module. Behalve het aanbieden van een set waardevolle observaties aan de OWL-S gebruikersgemeenschap zijn de bijdragen van dit deel van ons werk het ontwikkelen van KOS en de verbindingsmethodologie die op algemene wijze kan worden herbruikt om ook andere basis ontologieën te verbinden. Door het verbinden van verschillende ontologieën met KOS kan een harmonizatie van hun concepten worden bereikt.

In het laatste deel van het proefschrift onderzochten we de haalbaarheid van het bouwen van een geautomatiseerde oplossing om het bouwproces van Web service domein ontologieën te ondersteunen. Uit de twee case studies die we uitvoerden concluderen we dat scenario's voor het bouwen van domein ontologieën zeer verschillend kunnen zijn maar dat zij een aantal belangrijke karakteristieken hebben waarmee rekening dient te worden gehouden wanneer men een geautomatiseerde oplossing ontwikkelt. We observeerden dat de over het algemeen beschikbare data sets voor het automatisch leren van ontologieën textuele beschrijvingen van aangeboden functionaliteit zijn (b.v. API documentatie, of Web service commentaren). Deze textuele beschrijvingen hebben een lage grammaticale kwaliteit en gebruiken natuurlijke taal op een zeer specifieke manier (ze gebruiken een subtaal). Verder dient zowel statische als dynamische kennis afgeleid te worden door geautomatiseerde programma's. Onze bijdrage is het identificeren van deze eigenschappen.

Onze tweede conclusie was dat eenvoudige ontologie leermethoden kunnen worden aangepast voor de Web service context. Nadat we de eigenaardigheden van de Web service context begrepen hebben we een raamwerk gebouwd dat ontologie leermethoden aanpast aan de karakteristieken van het domein. Dit raamwerk steunt op de observatie dat, vanwege de karakteristieken van de gebruikte subtaal in de geanalyseerde teksten, simpele patroon gebaseerde technieken kunnen worden gebruikt om semantische structuren te extraheren uit de regelmatigheden van de subtaal. Dit raamwerk staat de implementatie van verschillende extractie methoden toe. We hebben linguistieke analyses van verschillende complexiteit gebruikt om ontologie lering uit te voeren. Onze bijdrage is het opstellen van algemene contouren van het raamwerk en het demonstreren van twee aparte implementaties (zie Hoofdstuk 7)

Evaluatie van ontologie leer methoden is geen eenvoudige taak. We analyseerden een aantal verschillende aanpakken en gebruikten een combinatie van methoden om onze evaluatie metriek te definiëren. De resultaten van onze evaluatie tonen aan dat (1) ons raamwerk, al is het ontworpen voor de Web service context, toepasbaar is op verschillende domeinen (onze methoden presteren vergelijkbaar in twee verschillende domeinen). De resultaten tonen ook aan dat (2) diepere linguistische analyse leidt tot betere resultaten omdat zij minder gevoelig is voor de onvolkomenheden van het corpus. Ten laatste gaven de experts die betrokken waren bij de evaluatie aan dat (3) de geëxtraheerde ontologieën een betrouwbaarder representatie van de kennis in het corpus zijn dan de hand gebouwde Gouden Standaard ontologieën en dan ze een nuttig startpunt voor het bouwen van een domein ontologie vormen. De geëxtraheerde ontologieën bevatten zelfs meer dan 50% van de relevante domein concepten. Onze bijdrage is, behalve het verschaffen van inzicht in de prestaties van van de leer methoden, het vestigen van een basismeting voor evaluatie van ontologie leer methoden in de Web services context. We verzamelden twee data sets en hun bijbehorende met de hand gebouwde Gouden Standaard ontologieën die kunnen worden gebruikt als een toets voor het evalueren van nieuwe oplossingen (zie Hoofdstuk 8).

Om het doel van het ondersteunen van het verkrijgen van Web service domein ontologieën volledig te bereiken dienen we, naast het bouwen van ontologie lerings methoden met een hoge prestatie, ze te implementeren in programma's die eenvoudig te gebruiken zijn door experts. Ons bewust van deze eis, hebben we een prototype geïmplementeerd dat (1) eenvoudig te debuggen, uit te breiden en aan te passen is door ontwerpers van ontologie lerings methoden in deze context en dat (2) een intuïtieve gebruikersinterface voor domein experts heeft die ontologieën willen extraheren met het programma (zie Hoofdstuk 7). We bereikten deze gebruikseigenschappen door het gebruik van de GATE software en het gebruik van visuele metaforen om de geëxtraheerde kennis te presenteren.

Ook al hebben we een verscheidenheid aan kwesties betreffende het onderwerp van het bouwen van Web service ontologieën behandeld, stellen wij ons voor dat verder belangwekkend onderzoek gedaan kan worden in de context van het verbeteren van het semantische Web service veld. In het bijzonder, als de voorstellen voor generieke Web service ontologieën een bepaald niveau van rijpheid bereiken, zal het belangrijk worden ze te harmonizeren. Verder is de aanpak voor het leren van Web service domein ontologieën zoals hier gepresenteerd slechts een [aantoonbaarheidsstudie] en dient verder uitgebreid en ontwikkeld te worden. Toekomstig werk kan zich ook concentreren op het dichter bij elkaar brengen van software engineering en Semantisch Web technologie in zijn algemeenheid. Ten laatste zou het ontologie lerings veld profijt hebben van verder onderzoek van adaptieve ontologie lering, evaluatie van ontologie lerings methoden en het ontwikkelen van bruikbaarder ontologie lerings programma's.

Bibliography

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., and Sycara, K. (2002a). DAML-S: Web Service Description for the Semantic Web. In (Horrocks and Hendler, 2002), pages 348 – 364.

Ankolekar, A., Huch, F., and Sycara, K. (2002b). Concurrent Execution Semantics for DAML-S with Subtypes. In (Horrocks and Hendler, 2002), pages 318 – 333.

Ankolenkar, A., Hutch, F., and Sycara, K. (2002). Concurrent Semantics for the Web Services Specification Language DAML-S. In *Proceedings of The Fifth International Conference on Coordination Models and Languages*.

Aussenac-Gilles, N. (2005). Supervised Text Analyses for Ontology and Terminology Engineering. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*.

Baida, Z., Gordijn, J., Omelayenko, B., and Akkermans, H. (2004). A Shared Service Terminology for Online Service Provisioning. In *Proceedings of the Sixth International Conference on Electronic Commerce (ICEC04)*, Delft, The Netherlands. ACM.

Baker, P., Goble, C., Bechhofer, S., Paton, N., Stevens, R., and Brass, A. (1999). An Ontology for Bioinformatics Applications. *Bioinformatics*, 15(6):510 – 520.

Barbera-Medina, W., Padget, J., and Aird, M. (2003). Brokerage for Mathematical Services in MONET. In (Cavedon and Maamar, 2003).

Battle, S. and Martin, D. (2005). W3C Workshop on Frameworks for Semantics in Web Services Summary Report. Available online at: `http://www.w3.org/2005/04/FSWS/workshop-report.html`.

Bechhofer, S., Horrocks, I., Goble, C., and Stevens, R. (2001). OilEd: a reason-able ontology editor for the Semantic Web. In *Proceedings of the Joint German Austrian Conference on Artificial Intelligence*, volume 2174 of *LNAI*, pages 396 – 408. Springer-Verlag.

Berardi, D., Gruninger, M., Hull, R., and McIlraith., S. (2004). Towards a First-Order Ontology for Web Services. In *Proceedings of the W3C Workshop on Constraints and Capabilities for Web Services*.

Berland, M. and Charniak, E. (1999). Finding Parts in Very Large Corpora. In *Proceedings of the 37th Annual Meeting of the ACL*.

Berners-Lee, T. (1989). Information Management: A Proposal. Available online at: `http://www.w3.org/History/1989/proposal.html`.

Berners-Lee, T. (1999). *Weaving the Web*. Harpur, San Francisco.

Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):34 – 43.

Biggerstaff, T., Mitbander, B., and Webster, D. (1993). The Concept Assignment Problem in Program Understanding. In *Proceedings of the 15th International Conference on Software Engineering*, pages 482 – 498, Los Alamitos, CA. IEEE Computing Society Press.

Bisson, G., Nedellec, C., and Caamero, L. (2000). Designing clustering methods for ontology building: The Mo'K workbench. In (Staab et al., 2000).

Bloehdorn, S. and Hotho, A. (2004). Text Classification by Boosting Weak Learners based on Terms and Concepts. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 331 – 334. IEEE Computer Society Press.

Bontcheva, K., Tablan, V., Maynard, D., and Cunningham, H. (2004). Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349 – 373.

Borgida, A. and Devanbu, P. (1999). Adding more DL to IDL: towards more knowledgeable component inter-operability. In *Proceedings of the 21st international conference on Software engineering*, pages 378 – 387. IEEE Computer Society Press.

Borgo, S., Gangemi, A., Guarino, N., Masolo, C., and Oltramari, A. (2002). Ontology RoadMap. WonderWeb Deliverable D15.

Borgo, S. and Leitao, P. (2004). The Role of Foundational Ontologies in Manufacturing Domain Applications. In (Meersman and Tari, 2004), pages 670 – 688.

Borst, W. (1997). *Construction of Engineering Ontologies*. PhD thesis, University of Tweente, Enschede, NL–Centre for Telematica and Information Technology.

Brazier, F. and Wijngaards, N. (2001). Automated servicing of agents. *Artificial Intelligence and Simulation of Behaviour, Special Issue on Agent Technology*, 1(1):5 – 20.

Brison, J., Martin, D., McIlraith, S., and Stein, L. (2002). Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web. In Zhong, N., Liu, J., and Yao, Y., editors, *Web Intelligence*. Springer-Verlag, Berlin.

Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In (Horrocks and Hendler, 2002), pages 348 – 363.

Buitelaar, P., Cimiano, P., and Magnini, B., editors (2004a). *Proceedings of the ECAI Workshop on Ontology Learning and Population: Towards Evaluation of Text-based Methods in the Semantic Web and Knowledge Discovery Life Cycle*. Valencia, Spain.

Buitelaar, P., Olejnik, D., and Sintek, M. (2004b). A Protege Plug-In for Ontology Extraction from Text Based on Linguistic Analysis. In (Bussler et al., 2004).

Bussler, C., Davies, J., Fensel, D., and Studer, R., editors (2004). *Proceedings of the First European Semantic Web Symposium (ESWS2004)*, volume 3053 of *LNCS*. Springer-Verlag, Heraklion, Crete, Greece.

Cabral, L., Domingue, J., Motta, E., Payne, T., and Hakimpour, F. (2004). Approaches to Semantic Web Services: An Overview and Comparisons. In (Bussler et al., 2004), pages 225 – 239.

Cardoso, J. and Sheth, A. (2002). Semantic e-Workflow Composition. Technical report, LSDIS Lab, Computer Science, University of Georgia.

Cavedon, L. and Maamar, Z., editors (2003). *Proceedings of the AAMAS Workshop on Web Services and Agent-Based Engineering*. Melbourne, Australia.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2003). Web Services Description Language (WSDL). Working Draft. Available online at: `http://www.w3.org/TR/wsdl`.

Cimiano, P., Handschuh, S., and Staab, S. (2004a). Towards the Self-Annotating Web. In (Feldman et al., 2004), pages 462 – 471.

Cimiano, P., Hotho, A., and Staab, S. (2004b). Clustering concept hierarchies from text. In *Proceedings of LREC*.

Cimiano, P., Pivk, A., Schmidt-Thieme, L., and Staab, S. (2004c). Learning Taxonomic Relations from Heterogeneous Evidence. In (Buitelaar et al., 2004a).

Cimiano, P., Staab, S., and Tane, J. (2003). Automatic Acquisition of Taxonomies from Text: FCA meets NLP. In *Proceedings of the ECML/PKDD Workshop on Adaptive Text Extraction and Mining, Cavtat–Dubrovnik, Croatia*.

Cimiano, P. and Voelker, J. (2005). Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. In *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'2005)*.

Corcho, O., Fernandez-Lopez, M., and Gomez-Perez, A. (2003). Methodologies, tools and languages for building ontologies. Where is the meeting point? *Data and Knowledge Engineering*, 46(1):41 – 46.

Cunningham, H., Maynard, D., Bontcheva, K., and Tablan, V. (2002). GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics*.

Cunningham, H., Maynard, D., and Tablan, V. (2000). JAPE: a Java Annotation Patterns Engine (Second Edition). Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield.

DAML Services Coalition (2002). DAML-S: Semantic Markup for Web Services. DAML-S v. 0.7 White Paper. Available online at: `http://www.daml.org/services/daml-s/0.7/daml-s.pdf`.

de Aalst, W. V. (2003). Dont Go with the Flow: Web Services Compostition Standards Exposed. *IEEE Intelligent Systems*, 18(1):72 – 76.

Delteil, A., Faron-Zucker, C., and Dieng, R. (2001). Learning Ontologies from RDF Annotations. In *Proceedings of the IJCAI Workshop on Ontology Learning*, Seattle.

Diaz, R. P. (1991). Implementing Faceted Classification for Software Reuse. *Communications of the ACM*, 34(5):88 – 97.

Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium*, 42:149 – 160.

Elenius, D. (2005). Tools for Semantic Web Services. *AgentLink News*, 18:8 – 11.

Elenius, D., Denker, G., Martin, D., Gilham, F., Khouri, J., Sadaati, S., and Senanayake, R. (2005). The OWL-S Editor A Development Tool for Semantic Web Services. In (Gomez-Perez and Euzenat, 2005), pages 78 – 92.

Elrad, T., Filman, R., and Bader, A. (2001). Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10):29 – 32.

Falkovych, K., Sabou, M., and Stuckenschmidt, H. (2003). UML for the Semantic Web: Transformation-Based Approaches. In Omelayenko, B. and Klein, M., editors, *Knowledge Transformation for the Semantic Web*, volume 95 of *Frontiers in Artificial Intelligence and Applications*, pages 92 – 107. IOS Press, Amsterdam.

Faure, D. and Nedellec, C. (1998). ASIUM: learning subcategorization frames and restrictions of selection. In Kodratoff, Y., editor, *Proceedings of Workshop on Text Mining, 10th European Conference on Machine Learning (ECML 98)*.

Faure, D. and Poibeau, T. (2000). First experiments of using semantic knowledge learned by ASIUM for information extraction task using INTEX. In (Staab et al., 2000).

Feldman, S., Uretsky, M., Najork, M., and Wills, C., editors (2004). *Proceedings of the 13th International World Wide Web Conference (WWW'04)*. ACM Press, New York, NY, USA.

Fensel, D., Benjamins, R., Motta, E., and Wielinga, B. (1999). UPML: A framework for knowledge system reuse. In Dean, T., editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99*, pages 16 – 23, Stockholm, Sweden. Morgan Kaufmann.

Fensel, D., Decker, S., Erdmann, M., and Studer, R. (1998). Ontobroker: Or How to Enable Intelligent Access to the WWW. In *Proceedings of the 11th Knowledge Acquisition for Knowledge-Based System Workshop (KAW98)*. Banff, Kanada.

Fensel, D., Sycara, K., and Mylopoulos, J., editors (2003). *The SemanticWeb - ISWC 2003, Second International Semantic Web Conference, Proceedings*, volume 2870 of *LNCS*. Springer-Verlag, Sanibel Island, FL, USA.

Fluit, C., Sabou, M., and van Harmelen, F. (2002). Ontology-based Information Visualisation. In Geroimenko, V., editor, *Visualising the Semantic Web*. Springer-Verlag.

Fluit, C., Sabou, M., and van Harmelen, F. (2004). Supporting User Tasks through Visualisation of Light-weight Ontologies. In (Staab and Studer, 2004), pages 415 – 434.

Fluit, C., Sabou, M., and van Harmelen, F. (2005). Ontology-based Information Visualization: Towards Semantic Web Applications. In Geroimenko, V., editor, *Visualising the Semantic Web, Second Edition*. Springer-Verlag.

Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., and Schneider, L. (2002). Sweetening Ontologies with DOLCE. In Gomez-Perez, A. and Benjamins, V., editors, *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2002). Ontologies and the Semantic Web.*, volume 2473 of *LNCS*. Springer-Verlag, Siguenza, Spain.

Gangemi, A., Mika, P., Sabou, M., and Oberle, D. (2003). An Ontology of Services and Service Descriptions. Technical report, Laboratory for Applied Ontology, National Research Council, I-00137 Rome, Italy.

Gibbins, N., Harris, S., and Shadbolt, N. (2003). Agent-based Semantic Web Services. In (Hencsey and White, 2003), pages 710 – 717.

Gomez-Perez, A. (2002). A survey on ontology tools. OntoWeb Delieverable 1.3.

Gomez-Perez, A. and Euzenat, J., editors (2005). *The Semantic Web: Research and Applications: Second European Semantic Web Conference, ESWC, Proceedings*, volume 3532 of *LNCS*. Springer-Verlag, Heraklion, Crete, Greece.

Gomez-Perez, A., Fernandez-Lopez, M., and Corcho, O. (2003). *Ontological Engineering*. Advanced Information and Knowledge Processing. Springer-Verlag.

Gomez-Perez, A., Gonzalez-Cabero, R., and Lama, M. (2004). Development of Semantic Web Services at the Knowledge Level. In *Proceedings of the European Conference on Web Services (ECOWS)*, Erfurt, Germany.

Gomez-Perez, A. and Manzano-Mancho, D. (2003). A Survey of Ontology Learning Methods and Techniques. OntoWeb Delieverable 1.5.

Grishman, R. (2001). Adaptive Information Extraction and Sublanguage Analysis. In *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining*.

Grishman, R., Hirschman, L., and Nhan, N. T. (1986). Discovery Procedures for Sublanguage Selectional Patterns: Initial Experiments. *Computational Linguistics*, B12(3):205 – 216.

Grishman, R. and Kittredge, R., editors (1986). *Analyzing Language in Restricted Domains: Sublanguage Description and Processing*. Lawrence Erlbaum Assoc., Hillsdale, NJ.

Grosof, B., Gruninger, M., Kifer, M., Martin, D., McGuinness, D., Parsia, B., Payne, T., and Tate, A. (2004). Semantic Web Services Language Requirements. `http://www.daml.org/services/swsl/requirements/swsl-requirements.shtml`.

Gruber, T. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199 – 220.

Guarino, N. (1997). Semantic Matching: Formal Ontological Distinctions for Information Organization, Extraction, and Integration. In Pazienza, M., editor, *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, volume 1299 of *LNCS*, pages 139 – 170. Springer-Verlag.

Guarino, N. (1998). Formal Ontology and Information Systems. In Guarino, N., editor, *Formal Ontology in Information Systems. Proceedings of FOIS'98*, pages 3 – 15. IOS Press, Amsterdam, Trento, Italy.

Guarino, N. and Welty, C. (2004). An Overview of OntoClean. In (Staab and Studer, 2004), pages 151 – 171.

Guizzardi, G. (2005). *Ontological Foundations for Structural Conceptual Models*. PhD thesis, Telematica Instituut.

Harris, Z. (1968). *Mathematical Structures of Language*. Wiley Interscience, New York.

Hearst, M. (1992). Automatic Acquisition of Hyponyms in Large Text Corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*.

Helm, R. and Maarek, Y. (1991). Integrating Information Retrieval and Domain Specific Approaches for Browsing and Retrieval in Object-Oriented Class Libraries. In *Proceedings of Object-oriented Programming Systems, Languages, and Applications*, pages 47 – 61, New York, NY, USA. ACM Press.

Hencsey, G. and White, B., editors (2003). *Proceedings of the 12th International World Wide Web Conference*. ACM Press, Budapest, Hungary.

Hendler, J. (2001). Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30 – 37.

Hepple, M. (2000). Independence and commitment: Assumptions for rapid training and execution of rule-based pos taggers. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-2000)*, Hong Kong.

Hess, A., Johnston, E., and Kushmerick, N. (2004). ASSAM: A Tool for Semi-Automatically Annotating Semantic Web Services. In (McIlraith et al., 2004), pages 320 – 335.

Hess, A. and Kushmerick, N. (2003). Learning to Attach Semantic Metadata to Web Services. In (Fensel et al., 2003), pages 358 – 274.

Hess, A. and Kushmerick, N. (2004). Machine Learning for Annotating Semantic Web Services. In *AAAI Spring Symposium on Semantic Web Services*.

Horrocks, I. and Hendler, J. A., editors (2002). *The Semantic Web - ISWC 2002, First International Semantic Web Conference*, volume 2342 of *LNCS*. Springer-Verlag, Sardinia, Italy.

Horrocks, I., Patel-Schneider, P., and van Harmelen, F. (2003). From $\mathcal{SHIQ}$ and RDF to OWL: The Making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7 – 26.

Hotho, A., Staab, S., and Stumme., G. (2003). WordNet improves Text Document Clustering. In *Proceedings of the Semantic Web Workshop at SIGIR-2003, 26th Annual International ACM SIGIR Conference*, Toronto, Canada.

Hyvonen, E., editor (2002). *The Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications*. HIIT Publications, University of Helsinki.

Jannink, J. and Wiederhold, G. (1999). Thesaurus Entry Extraction from an On-line Dictionary. In *Proceedings of Fusion*, Sunnyvale, CA.

Kivela, A. and Hyvonen, E. (2002). Ontological Theories for the Semantic Web. In (Hyvonen, 2002).

Lara, R., Lausen, H., Arroyo, S., de Bruijn, J., and Fensel, D. (2003). Semantic Web Services: description requirements and current technologies. In *Proceedings of the International Workshop on Electronic Commerce, Agents, and Semantic Web Services held in conjunction with the Fifth International Conference on Electronic Commerce (ICEC 2003)*, Pittsburgh.

Lara, R., Polleres, A., Lausen, H., Roman, D., de Bruijn, J., and Fensel, D. (2005). A Conceptual Comparison between WSMO and OWL-S. WSMO Deliverable, D.4.1, v.0.1.

Lassila, O. (2002). Serendipitous Interoperability. In (Hyvonen, 2002).

Laukkanen, M. and Helin, H. (2003). Composing Workflows of Semantic Web Services. In (Cavedon and Maamar, 2003).

Lei, L. and Horrocks, I. (2003). A Software Framework For Matchmaking Based on Semantic Web Technology. In (Hencsey and White, 2003), pages 331 – 339.

Lin, D. (1998). Dependency-based Evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems, First International Conference on Language Resources and Evaluation*, Granada, Spain.

Lopes, A., Gaio, S., and Botelho, L. (2002). From DAML-S to Executable Code. In *Proceedings of the AAMAS Workshop on Challenges in Open Agent Systems*.

Lord, P., Alper, P., Wroe, C., and Goble, C. (2005). Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery. In (Gomez-Perez and Euzenat, 2005), pages 17 – 31.

Lord, P., Bechhofer, S., Wilkinson, M., Schiltz, G., Gessler, D., Hull, D., Goble, C., and Stein, L. (2004). Applying Semantic Web Services to bioinformatics: Experiences gained, lessons learnt. In (McIlraith et al., 2004), pages 350 – 365.

Luke, S., Spector, L., and Rager, D. (1996). Ontology-Based Knowledge Discovery on the World-Wide Web. In *AAAI96 Workshop on Internet-based Information Systems*.

Maarek, Y., Berry, D., and Kaiser., G. (1991). An Information Retrieval Approach for Automatically Constructing Software Libraries. *IEEE Transactions on Software Engineering*, 17(8):800 – 813.

MacGregor, R. (1991). Using a Description Classifier to Enhance Deductive Inference. In *Proceedings Seventh IEEE Conference on AI Applications*, pages 141 – 147.

Maedche, A. (2002). *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers.

Maedche, A., Motik, B., and Stojanovic, L. (2003). Managing Multiple and Distributed Ontologies in the Semantic Web. *VLDB Journal*, 12(4):286 – 302.

Maedche, A. and Staab., S. (2000). Discovering Conceptual Relations from Text. In Horn, W., editor, *Proceedings of the Fourteenth European Conference on Artificial Intelligence (ECAI 2000)*, pages 321 – 325, Berlin, Germany. IOS Press.

Maedche, A. and Staab, S. (2001). Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72 – 79.

Maedche, A. and Staab, S. (2002). Measuring Similarity between Ontologies. In *Proceedings of European Knoeledge Ackquisition Workshop (EKAW)*.

Maedche, A. and Staab, S. (2004). Ontology Learning. In (Staab and Studer, 2004), pages 173 – 190.

Martin, D., Burstein, M., Denker, G., Hobbs, J., Kagal, L., Lassila, O., McDermott, D., McIlraith, S., Paolucci, M., Parsia, B., Payne, T., Sabou, M., Sirin, E., Solanki, M., Srinivasan, N., and Sycara, K. (2003). OWL-S 1.0 white paper. `http://www.daml.org/services/owl-s/1.0/`.

Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., and Sycara, K. (2004). Bringing Semantics to Web Services: The OWL-S Approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, USA.

Masolo, C., Borgo, S., Gangemi, A., Guarino, N., and Oltramari, A. (2003). Ontology Library (final). WonderWeb Deliverable D18.

Massonet, P. and van Lamsweerde, A. (1997). Analogical Reuse of Requirements Frameworks. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pages 26 – 39, Annapolis, MD, USA. IEEE Computer Society.

McBride, B. (2002). Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55 – 59.

McBride, B. (2004). The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS. In (Staab and Studer, 2004), pages 51 – 66.

McGuinness, D. (2002). Ontologies Come of Age. In Fensel, D., Hendler, J., Lieberman, H., and Wahlster, W., editors, *Spinning the Semantic Web: bringing the World Wide Web to Its Full Potential*. MIT Press.

McGuinness, D. and van Harmelen, F. (2004). Web ontology language (OWL) overview. Available online at: `http://www.w3.org/TR/owl-features/`. W3C Recommendation.

McIlraith, S., Plexousakis, D., and van Harmelen, F., editors (2004). *The Semantic Web - ISWC 2004, Third International Semantic Web Conference, Proceedings*, volume 3298 of *LNCS*. Springer-Verlag, Hiroshima, Japan.

McIlraith, S., Son, T., and Zeng, H. (2001). Semantic Web Services. *IEEE Intelligent Systems. Special Issue on the Semantic Web*, 16(2):46 – 53.

Meersman, R. and Tari, Z., editors (2004). *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences, Proceedings*, volume 3290 of *LNCS*. Springer-Verlag, Agia Napa, Cyprus.

Mika, P., Oberle, D., Gangemi, A., and Sabou, M. (2004a). Foundations for Service Ontologies: Aligning OWL-S to DOLCE. In (Feldman et al., 2004), pages 563 – 572.

Mika, P., Sabou, M., Gangemi, A., and Oberle, D. (2004b). Foundations for DAML-S: Aligning DAML-S to DOLCE. In *First International Semantic Web Services Symposium (SWS2004), AAAI Spring Symposium Series*.

Mili, A., Mili, R., and Mittermeir, R. (1998). A survey of software reuse libraries. *Annals of Software Engineering*, 5:349 – 414.

Mili, H., Ah-Ki, E., Godin, R., and Mcheick, H. (1997). Another nail to the coffin of faceted controlled-vocabulary component classification and retrieval. *SIGSOFT Software Engineering Notes*, 22(3):89 – 98.

Miller, J., Verma, K., Rajasekaran, P., Sheth, A., Aggarwal, R., and Sivashanmugam, K. (2005). WSDL-S: A Proposal to W3C WSDL 2.0 Committee. White paper.

Motta, E., Domingue, J., Cabral, L., and Gaspari, M. (2003). IRSII: A Framework and Infrastructure for Semantic Web Services. In (Fensel et al., 2003), pages 306 – 318.

Narayanan, S. and McIlraith, S. (2003). Analysis and simulation of Web Services. *Computer Networks*, 42(5):675 – 693.

Navigli, R. and Velardi., P. (2004). Learning Domain Ontologies from Document Warehouses and Dedicated Websites. *Computational Linguistics*, 30(2).

Navigli, R., Velardi, P., Cucchiarelli, A., and Neri, F. (2004). Quantitative and Qualitative Evaluation of the OntoLearn Ontology Learning System. In (Buitelaar et al., 2004a).

Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W., and Musen, M. A. (2001). Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems*, 16(2):60 – 71.

Oberle, D., Eberhart, A., Staab, S., and Volz, R. (2004a). Developing and Managing Software Components in an Ontology-based Application Server. In *Proceedings of the Fifth International Middleware Conference*.

Oberle, D., Staab, S., Studer, R., and Volz, R. (2004b). KAON SERVER Demonstrator. WonderWeb Deliverable 7.

Oberle, D., Staab, S., Studer, R., and Volz, R. (2004c). Supporting Application Development in the Semantic Web. *ACM Transactions on Internet Technology (TOIT)*, 4(4).

Object Modelling Group (2002). IDL / Language Mapping Specification - Java to IDL. 1.2.

Ogren, P., Cohen, K., Acquaah-Mensah, G., Eberlein, J., and Hunter, L. (2004). The Compositional Structure of Gene Ontology Terms. In *Proceedings of the Pacific Symposium on Biocomputing*.

Paolucci, M., Sycara, K., and Kawamura, T. (2002). Delivering Semantic Web Services. Technical Report CMU-RI-TR-02-28, Robotics Institute, Carnegie Mellon University.

Patil, A., Oundhakar, S., Sheth, A., and Verma, K. (2004). METEOR-S Web service Annotation Framework. In (Feldman et al., 2004), pages 553 – 562.

Pease, A., Niles, I., and Li, J. (2002). The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications. In *Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*. Edmonton, Canada.

Pepper, S. and Schwab, S. (2003). Curing the Web's Identity Crisis. Technical report, Ontopia (`http://www.ontopia.net`).

Porzel, R. and Malaka, R. (2004). A Task-based Approach for Ontology Evaluation. In (Buitelaar et al., 2004a).

Pretorius, A. (2004). Lexon Visualization: Visualizing Binary Fact Types in Ontology Bases. In *Proceedings of the 8th International Conference on Information Visusalisation*, London.

Rector, A. and Rogers, J. (1999). Ontological issues in using a description logic to represent medical concepts: Experience from GALEN. *IMIA Working Group 6 Workshop*.

Reinberger, M. and Spyns, P. (2004). Discovering Knowledge in Texts for the Learning of DOGMA-Inspired Ontologies. In (Buitelaar et al., 2004a).

Reinberger, M.-L., Spyns, P., Pretorius, A., and Daelemans, W. (2004). Automatic initiation of an ontology. In (Meersman and Tari, 2004), pages 600 – 617.

Richards, D. and Sabou, M. (2003). Semantic Markup for Semantic Web Tools: A DAML-S description of an RDF-Store. In (Fensel et al., 2003), pages 274 – 289.

Richards, D., van Splunter, S., Brazier, F., and Sabou, M. (2003). Composing Web Services using an Agent Factory. In (Cavedon and Maamar, 2003).

Rigau, G. (1998). *Automatic Acquisition of Lexical Knowledge from MRDs*. PhD thesis, Departament de Llenguatges i Sistemes Informatics – Universitat Politecnica da Catalunya.

Riloff, E. (1996). Automatically Generating Extraction Patterns from Untagged Text. In *Proceedings of the 13th National Conference On Artificial Intelligence (AAAI)*, pages 1044–1049.

Rubin, D., Hewett, M., Oliver, D., Klein, T., and Altman, R. (2002). Automatic data acquisition into ontologies from pharmacogenetics relational data sources using declarative object definitions and XML. In *Proceedings of the Pacific Symposium on Biology*.

Sabou, M. (2004a). Extracting Ontologies from Software Documentation: a Semi-Automatic Method and its Evaluation. In (Buitelaar et al., 2004a).

Sabou, M. (2004b). From Software APIs to Web Service Ontologies: a Semi-Automatic Extraction Method. In (McIlraith et al., 2004), pages 410 – 425.

Sabou, M. (2005a). Learning Web Service Ontologies: an Automatic Extraction Method and its Evaluation. In Buitelaar, P., Cimmiano, P., and Magnini, B., editors, *Ontology Learning from Text: Methods, Evaluation and Applications*, volume 123 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, July.

Sabou, M. (2005b). Visual Support for Ontology Learning: an Experience Report. In *Proceedings of the 9th International Conference on Information Visualisation (IV05)*, London, UK.

Sabou, M., Oberle, D., and Richards, D. (2004). Enhancing Application Servers with Semantics. In *Proceedings of the First Australian Workshop on Engineering Service-Oriented Systems (AWESOS)*, Melbourne, Australia.

Sabou, M. and Pan, J. (2005). Towards Improving Web Service Repositories through Semantic Web Techniques. In *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE) collocated with the 4th International Semantic Web Conference (ISWC 2005)*.

Sabou, M., Richards, D., and van Splunter, S. (2003). An experience report on using DAML-S. In *Proceedings of the Workshop on E-Services and the Semantic Web, The 12th WWW Conference*, Budapest, Hungary.

Sabou, M., Wroe, C., Goble, C., and Mishne, G. (2005a). Learning Domain Ontologies for Web Service Descriptions: an Experiment in Bioinformatics. In *Proceedings of the 14th International World Wide Web Conference*, Chiba, Japan.

Sabou, M., Wroe, C., Goble, C., and Stuckenschmidt, H. (2005b). Learning Domain Ontologies for Semantic Web Service Descriptions. *Journal of Web Semantics*, 3(4).

Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., van de Velde, W., and Wielinga, B. (1999). *Knowledge Engineering and Management The CommonKADS Methodology*. The MIT Press, Cambridge, Massachusetts.

Scicluna, J., Abela, C., and Montebello, M. (2004). Visual Modelling of OWL-S Services. In *Proceedings of the IADIS International Conference WWW/Internet*, Madrid, Spain.

Seguela, P. and Aussenac-Gilles, N. (1999). Extraction de relations semantiques entre termes et enrichissement de modeles du domaine. In *Actes de la Conference IC'99 - Plate-forme AFIA*.

Sheshagiri, M., desJardins, M., and Finin, T. (2003). A Planner for Composing Services Described in DAML-S. In (Cavedon and Maamar, 2003).

Smith, B. (2003). Basic Formal Ontology. Technical report, Institute for Formal Ontology and Medical Information Science, University of Leipzig.

Smith, B. and Welty, C. (2001). FOIS introduction: Ontology—towards a new synthesis. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, pages 3 – 9, Ogunquit, Maine, USA. ACM Press.

Somacher, M., Tomaiuolo, M., and Turci, P. (2002). Goal Delegation in Multiagent System. In *Tecniche di Intelligenza Artificiale per la ricerca di informazione sul Web*.

Staab, S., Maedche, A., and Nedellec, C., editors (2000). *Proceedings of the ECAI-2000 Ontology Learning Workshop*.

Staab, S. and Studer, R., editors (2004). *Handbook on Ontologies*. International Handbooks on Information Systems. Springer-Verlag.

Stuckenschmidt, H., Sabou, M., and Klein, M. (2004). Semantic Web Technology - Bringing Meaning to Distributed Systems. *IEEE Distributed Systems Online*.

Studer, R., Benjamins, V., and Fensel, D. (1998). Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering*, 25(1-2):161 – 197.

Suryanto, H. and Compton, P. (2001). Discovery of Ontologies from Knowledge Bases. In *Proceedings of the International Conference on Knowledge Capture*, pages 171 – 178, New York, NY, USA. ACM Press.

ten Teije, A., van Harmelen, F., and Wielinga, B. (2004). Configuration of Web Services as Parametric Design. In Motta, E., Shadbolt, N., Stutt, A., and Gibbins, N., editors, *Proceedings of the 14th International Conference on Knowledge Engineering and Knowledge Management, (EKAW-2004)*, number 3257 in LNAI, pages 321 – 336, Whittleburry Hall, UK. Springer-Verlag.

Uschold, M. (2001). Where is the Semantics in the Semantic Web? In *Proceedings of the Workshop on Ontologies in Agent Systems (OAS) at the 5th International Conference on Autonomous Agents*.

Uschold, M. and Jasper, R. (1999). A Framework for Understanding and Classifying Ontology Applications. In *Proceedings of the IJCAI99 Workshop on Ontologies and Problem-Solving Methods*. Stockholm.

van Harmelen, F. (2002). How the Semantic Web will change KR: challenges and opportunities for a new research agenda. *The Knowledge Engineering Review*, 17(1):93 – 96.

van Heijst, G., Schreiber, A. T., and Wielinga, B. J. (1997). Using Explicit Ontologies in KBS Development. *International Journal of Human-Computer Studies*, 46(2/3):183 – 292.

van Splunter, S., Sabou, M., Brazier, F., and Richards, D. (2003). Configuring Web Service, using Structurings and Techniques from Agent Configuration. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence*, Halifax, Canada,.

Velardi, P., Missikoff, M., and Fabriani, P. (2001). Using Text Processing Techniques to Automatically enrich a Domain Ontology. In *Proceedings of the International Conference on Formal Ontology in Information Systems*, pages 270 – 284. ACM Press.

Vrandecic, D., Pinto, H., Sure, Y., and Tempich, C. (2005). The DILIGENT Knowledge Processes. *Journal of Knowledge Management*.

W3C (2002). Web services architecture requirements. W3C Web Services Architecture Working Draft, Available online at: `http://www.w3.org/TR/2002/WD-wsa-reqs-20021114`.

Wroe, C., Goble, C., Greenwood, M., Lord, P., Miles, S., Papay, J., Payne, T., and Moreau, L. (2004). Automating Experiments Using Semantic Data on a Bioinformatics Grid. *IEEE Intelligent Systems*, 19(1):48 – 55.

Wroe, C., Stevens, R., Goble, C., Roberts, A., and Greenwood, M. (2003). A Suite of DAML+OIL Ontologies to Describe Bioinformatics Web Services and Data. *Journal of Cooperative Information Science*, 12(2):197 – 224.

Zaremski, A. and Wing, J. (1997). Specification Matching of Software Components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333 – 369.

SIKS Dissertation Series

1998

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB) *Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems*

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation

2000

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE) *Prototyping of CMS Storage Management*

2000-3 Carolien M.T. Metselaar (UvA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectie

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coupé (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VU)
Knowledge Management: The Role of Mental Models in Business Systems Design

2002

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel (KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (UvA)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UvA)
Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UvA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

2004

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UvA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek (UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UvA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

2005

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM)
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems