An Evaluation Platform for Semantic Web Technology

by

Cécile Åberg

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2007
Abstract

The vision of the Semantic Web aims at enhancing today’s Web in order to provide a more efficient and reliable environment for both providers and consumers of Web resources (i.e. information and services). To deploy the Semantic Web, various technologies have been developed, such as machine understandable description languages, language parsers, goal matchers, and resource composition algorithms. Since the Semantic Web is just emerging, each technology tends to make assumptions about different aspects of the Semantic Web’s architecture and use, such as the kind of applications that will be deployed, the resource descriptions, the consumers’ and providers’ requirements, and the existence and capabilities of other technologies.

In order to ensure the deployment of a robust and useful Semantic Web and the applications that will rely on it, several aspects of the technologies must be investigated, such as whether the assumptions made are reasonable, whether the existing technologies allow construction of a usable Semantic Web, and the systematic identification of which technology to use when designing new applications.

In this thesis we provide a means of investigating these aspects for service discovery, which is a critical task in the context of the Semantic Web. We propose a simulation and evaluation platform for evaluating current and future Semantic Web technology with different resource sets and consumer and provider requirements. For this purpose we provide a model to represent the Semantic Web, a model of the evaluation platform, an implementation of the evaluation platform as a multi-agent system, and an illustrative use of the platform to evaluate some service discovery technology in a travel scenario.

The implementation of the platform shows the feasibility of our evaluation approach. We show how the platform provides a controlled setting to support the systematic identification of bottlenecks and other challenges for new Semantic Web applications. Finally, the evaluation shows that the platform can be used to assess technology with respect to both hardware issues such as the kind and number of computers involved in a discovery scenario, and other issues such as the evaluation of the quality of the service discovery result.
I was not alone on the path to the PhD. To all of you who encouraged me, questioned me, taught me: thank you.

I would especially like to thank and express all my gratitude to my supervisor, Professor Nahid Shahmehri. She has always been there to encourage me and has continually found ways to help me grow as a researcher.

I am also grateful to the members of my advisory committee: Dr. Patrick Lambrix and Dr. Lena Strömbäck. Patrick Lambrix has helped me greatly in formulating ideas in a more rigorous manner. He has been a constant source of inspiration. Lena Strömbäck was always available to provide good advice and engage in useful discussions.

Many thanks to Dr. Johan Åberg for his comments on my writing and research methodology. Thanks go also to Dr. Juha Takkinen, project manager of the SwebButler project, to Mikael Svensson from Medius for useful discussions on the development and use of Workflow systems, and to Jesper Holmqvist for help with the implementation of the sButler prototype. I also want to thank Brittany Shahmehri for her prompt, thorough, and effective language reviews on both the content of this thesis and several other research publications.

Thanks to all the members of IISLAB, past and present, for making the work environment so enjoyable, inspiring.... and tasty!

The constant support of my family and friends has also been extremely important. I want to especially express my gratitude to Johan, maman, papa, Anne, Eric, Flo, and Claudine. Thank you for listening so much and so well. Thank you Emma, wild angel, sweet devil, for your smile, your laughter, your rightful anger, and your never failing knowledge of what is truly important.

The cover of this thesis has been designed by Anders Petson. Many thanks.

The financial support from Vinnova (the Swedish Agency for Innovation Systems, project SwebButler 22485-1) and the EU Network of Excellence REW-ERSE (Sixth Framework Programme project 506779) is gratefully acknowledged.

Cécile Åberg
Linköping, June 2006
Faites que le rêve dévore votre vie,
afin que la vie ne dévore pas votre rêve.

Let dreams consume your life
so that life does not consume your dreams.

Antoine de St Exupéry
I let the PhD dream consume my life. It has not been a peaceful dream. More than once, I felt like Cyrano de Bergerac\(^1\): in love with something too beautiful for me. Like Cyrano’s love, my dream was often misunderstood and sometimes mocked. Like Cyrano, I could not help but serve anyway, with all my energy, wit, and panache. And since there is no Cyrano without a tirade of the nose, a PhD student channeling Cyrano must compose her tirade of the PhD dream. The tirade provides a list of PhD dream descriptions corresponding to different moods of the PhD student, varying from aggressive to poetic. The tirade is in French. Comme il se doit.

Le thésard: Rêver? J’en ai une expérience variée vraiment. Veuillez que je vous l’explique à l’Edmond Rostand: En variant le ton, comme Cyrano, tenez: Agressif: ” Moi, voyez vous, si j’osais rêver, Il faudrait sur-le-champ que je me réveillasse ! ” Amical: ” C’est doux de rêver. Mais le temps passe. Il faut être alerte pour de la thèse voir le bout. ” Descriptif: ” La paupière tombe, l’œil divague, c’est mou! Que dis-je, mou ?. .. je suis complètement abruti! ” Curieux: ” De quoi sert cet air naïf, ébahi ? Mimique d’air inspiré, ou fatigue sincère? ” Gracieux: ” Aimai-je à ce point les chimères Que longuement, yeux mi-clos, je m’imaginais en penseur et, tranquille, doucement m’endormai. ” Truculent: ” Quand je décris mon rêve éveillé, mon discours est il si joliment arrangé, que l’audience ne crie au fou, à l’illuminé ? ” Prévenant: ” Garde toi, tête rêveuse, du poids des songes. Au risque de tomber lourdement sur le sol. ” Tendre: ” Il faudrait faire un petit parasol de peur que mon rêve au soleil ne se fane ! ” Pédant: ” Autour du monde, je distribue le mane de mes travaux. Debout, de Hong-Kong à Boston, je me rêve expert, scientifique. J’en fait des tonnes. ” Cavalier: ” Quoi, l’ami, rêver est à la mode ? Pour perdre tout sens pratique, c’est vraiment très commode! ”, Emphatique: ” Aucun vent ne peut, rêve idéal, T’emporter tout entier, excepté le mistral ! ” Dramatique: ” Abandonner son rêve, c’est mourir. ” Admiratif: ” Quel port de paupière, à pâlir! ” Lyrique: ” Me prendrais-je pour un grandiose morphée? ” Naïf: ” Réveur? Moi qui me croyais être benêt. ” Respectueux: ” Souffrez, Mon rêve, qu’on vous salue, C’est là ce qui s’appelle avoir de suprêmes vues! ”

---
\(^1\)Cyrano is the guy with the long nose.
Campagnard: "Hé, ardé ! C’est-y un rêve ? Nanain !
C’est que qu’e théorie planante ou que qu’baratin !"
Militaire: "Rêvons! Rêvons! D’une thèse finie!" (air: Marseillaise)
Pratique: "Tant qu’à rêver maintenant et ici
autant rêver que le jury me dise ‘Bravo!’"
Enfin, parodiant Pyrame en un sanglot:
"Le voilà donc ce rêve qui du chef de son maître
a détruit l’harmonie! Il en blâmit, le traître!"
Contents

1 Introduction .............................................. 13
   1.1 Research problem .................................. 15
   1.2 Contributions .................................... 15
   1.3 Outline of the thesis ............................... 16
   1.4 Relation to previous published work of the author ... 17

2 The Semantic Web ........................................ 19
   2.1 Machine-understandable languages ................. 21
   2.2 Semantic annotation description languages ......... 24
   2.3 Semantic-aware tools .............................. 25
   2.4 Semantic Web operations ............................ 25
   2.5 Difficulties to overcome for deployment of the Semantic Web ... 26

3 Illustrative Scenario .................................... 27

4 Model for the Simulation and Evaluation Platform ........ 31
   4.1 Requirements for a simulation and evaluation platform ... 32
   4.2 Platform model ..................................... 33
       4.2.1 Modeling assumptions about the Semantic Web ....... 33
       4.2.2 Modeling the Semantic Web ...................... 34
       4.2.3 The platform .................................. 35

5 Implementation of the Simulation and Evaluation Platform .... 41
   5.1 Support for the operation component ............... 42
   5.2 Evaluation support ................................ 45
   5.3 Settings .......................................... 45
   5.4 A multi-agent system .............................. 46
   5.5 Related work ..................................... 49

6 sButler: a Requester Agent ............................... 51
   6.1 Organizational workflows ......................... 52
   6.2 A Model for the integration of organizational workflows and the Semantic Web ...................... 54
   6.3 sButler architecture ............................... 57
### 7 OWL-DTP

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 The problem of query generation for service retrieval</td>
<td>61</td>
</tr>
<tr>
<td>7.2 The DTP logical view</td>
<td>64</td>
</tr>
<tr>
<td>7.3 Definition of the DTP language extension</td>
<td>66</td>
</tr>
<tr>
<td>7.4 Ontologies for the DTP language extension</td>
<td>67</td>
</tr>
<tr>
<td>7.4.1 The MIT process handbook as a source of knowledge</td>
<td>67</td>
</tr>
<tr>
<td>7.4.2 A conceptual structure for the MIT process handbook</td>
<td>70</td>
</tr>
<tr>
<td>7.4.3 Specifying constraints on Activity concepts</td>
<td>75</td>
</tr>
<tr>
<td>7.4.4 Using the MIT process handbook as a knowledge resource on business processes</td>
<td>76</td>
</tr>
<tr>
<td>7.4.5 Using the DTP language extension to describe queries and Web services</td>
<td>79</td>
</tr>
<tr>
<td>7.5 Matchmaking with the DTP language extension</td>
<td>83</td>
</tr>
<tr>
<td>7.5.1 Matching categories</td>
<td>83</td>
</tr>
<tr>
<td>7.5.2 Different matchmaking approaches</td>
<td>84</td>
</tr>
<tr>
<td>7.5.3 The current matchmaking approaches are not satisfactory</td>
<td>91</td>
</tr>
<tr>
<td>7.5.4 Using the existing matchmaking algorithm</td>
<td>91</td>
</tr>
<tr>
<td>7.6 OWL-DTP</td>
<td>92</td>
</tr>
<tr>
<td>7.7 Comparison of OWL-DTP, OWL-S and WSMO</td>
<td>93</td>
</tr>
<tr>
<td>7.7.1 OWL-S</td>
<td>94</td>
</tr>
<tr>
<td>7.7.2 WSMO</td>
<td>94</td>
</tr>
<tr>
<td>7.7.3 Comparison method</td>
<td>95</td>
</tr>
<tr>
<td>7.7.4 Test suite</td>
<td>95</td>
</tr>
<tr>
<td>7.7.5 Expressing queries with OWL-S, WSMO, and OWL-DTP</td>
<td>96</td>
</tr>
<tr>
<td>7.7.6 Discussion</td>
<td>107</td>
</tr>
</tbody>
</table>

### 8 Prototype Implementation of sButler making use of OWL-DTP

### 9 Platform: Illustration of Use, Evaluation, and Lessons Learned

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Illustration</td>
<td>117</td>
</tr>
<tr>
<td>9.1.1 Assumptions</td>
<td>117</td>
</tr>
<tr>
<td>9.1.2 Integrating the assumptions in the evaluation platform</td>
<td>118</td>
</tr>
<tr>
<td>9.1.3 Evaluation of the service discovery approach</td>
<td>120</td>
</tr>
<tr>
<td>9.1.4 Platform evaluation</td>
<td>121</td>
</tr>
<tr>
<td>9.2 Lessons learned</td>
<td>121</td>
</tr>
</tbody>
</table>

### 10 Conclusion and Future Work

Bibliography

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A OWL-DTP - version of 20051124</td>
<td>135</td>
</tr>
<tr>
<td>B MIT Process Handbook Structure - version of 20051124</td>
<td>137</td>
</tr>
<tr>
<td>C Resource Ontology - version of 20051124</td>
<td>141</td>
</tr>
<tr>
<td>D Extract from the Transaction Ontology - version of 20051124</td>
<td>145</td>
</tr>
</tbody>
</table>
CONTENTS

E  Extract from the Process Ontology - version of 20051124  151

F  Test Suite  153
    F.1  Disclaimer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  153
    F.2  Header of the Xth OWL-DTP query . . . . . . . . . . . . . . . . . . . . . . . . .  153
    F.3  The domain knowledge used in the OWL-S queries . . . . . . . . . . . . . . . .  154
    F.4  The domain knowledge used in the WSMO queries . . . . . . . . . . . . . . . . .  155
    F.5  The domain knowledge used in the OWL-DTP queries . . . . . . . . . . . . . . .  156

G  Acronyms  159
Chapter 1

Introduction

The Internet and the Web provide an environment for business-to-business and business-to-consumer exchanges in a virtual world where distance means very little, providers can advertise their products globally and consumers from all over the world can obtain access to these products. The travel industry is one leading example of the positive business outcomes of this new on-line and electronic business. Today, most travel providers, from airplane and train companies to travel agencies, allow consumers to book and buy travel tickets over the Internet, providing for the most mature Web industry (Davis, 2006) and the biggest on-line consumer investment with $3.2 billion in travel spending in 2005 (Lipsman, 2006). However, the problem of finding suitable providers that can satisfy a consumer’s need is made more complex by the fact that the Web is a heterogeneous and continually changing environment. Continual changes in data are especially present in the travel domain where, as pointed out by Wöber (2006), “there are only few other economic activities where the generation, gathering, processing, application, and communication of information is so important for day-to-day operations.” In a continually changing environment, a provider that fits a consumer’s need at query time may no longer fit the need at execution time. Additionally, heterogeneity of data format is also present since there is no guarantee that different travel providers will adopt compatible data structures, similar information presentation, or even the same language. For example, in this context, a potential travel consumer that wants to identify the best possible solution when planning to travel for a conference from Linköping in Sweden to Chiba in Japan will have to consider multiple Web resources providing information expressed mostly in English and Japanese. Such a planning process is time consuming and requires language skills and travel planning experience, as well as trust that the information provided (e.g. bus timetables, tickets availability) is up to date. Thanks to both its popularity and complexity, the domain of travel offers good examples of Web applications that need support to be more efficient and reliable.

The vision of the Semantic Web (Berners-Lee, Hendler, & Lassila, 2001) aims at enhancing today’s Web in order to provide a more efficient and reliable
environment for both providers and consumers of Web resources (i.e. information and services). On the Semantic Web each Web resource is attached to a machine-understandable description. This description can be used by software agents to reason about resource relevance with respect to predefined user goals. The agents can also compose descriptions to match complex goals. Planning for a conference trip is most often such a goal, in that it requires several tickets be booked, corresponding to the different legs of the travel, booking hotel room(s), and orienting oneself in unfamiliar geographical settings to find locations such as restaurants or meeting places. Some technology has been developed to deploy the vision of the Semantic Web. Examples of such technologies are machine-understandable description languages, language parsers, goal matchers, and resource composition algorithms. This technology has already allowed the emergence of the Semantic Web as a fast growing collection of machine-understandable resources. The speed of this growth is illustrated by Swoogle\textsuperscript{1}'s indexing statistics as follows: in November 2005, Swoogle had indexed 45 million triples (i.e. machine understandable semantic statements) on the Web, and this number was up to more than 300 million in August 2006. Moreover, applications are appearing that are able to understand and use semantic annotations, such as the Semantic Web search engine Swoogle (Ding et al., 2004) and Piggy Bank (Huynh, Mazzocchi, & Karger, 2005), which enhances Web browsing.

Since the Semantic Web is just emerging, each technology tends to make certain assumptions about different aspects of the Semantic Web's architecture and use, such as the kind of applications that will be deployed, the number and level of semantics of the published resource descriptions, the consumer and provider's requirements, and the existence and capabilities of other technology. For example, for approaches to resource discovery, such assumptions specify implicit requirements about the scalability (e.g. in terms of response time, bandwidth use, or cost) and the quality of the result (e.g. measured as precision and recall). Furthermore, common examples of consumer travel requirements include the maximal acceptable time to get a travel plan, accessibility constraints to transports and hotel rooms, and preferences in terms of dates of departure and arrival. Similarly, common examples of travel provider requirements relate to the description of the services that they are willing to provide, such as the type of payment that they allow, and the kind of customer profile that they wish to deal with.

In order to ensure the deployment of a robust and useful Semantic Web and the applications that will rely on it, several aspects of the technology must be investigated, such as whether the assumptions made are reasonable, whether the existing technology allows a usable Semantic Web to be built, and the systematic identification of which technology to use when designing new applications.

In this thesis, we propose a simulation and evaluation platform to perform such investigations.

\textsuperscript{1}Swoogle (Ding et al., 2004) is a Semantic Web search engine that crawls the Web to index the concepts described in the documents written in Semantic Web languages (RDF, RDFS, DAML+OIL, OWL).
1.1 Research problem

In its current state, the Semantic Web initiative is very active and provides a lot of new technology. Most often this is complex technology (e.g., reasoners and planners build on advanced concepts of AI) that must dwell in a broadly distributed environment (this is the World Wide Web!) and that manages a vast set of resources. As a direct consequence of the complexity of the Semantic Web environment, evaluation of the technology is currently performed in a rather uncontrolled and incomplete manner. This is because it is difficult for every development team to be aware of all the possible scenarios of use of the Semantic Web. In this situation, there is a high risk of overlooking some important use scenarios and adopting the first technology that seems to work well enough rather than the best one. As a result, there is currently a strong need for better means of evaluation of Semantic Web technology. In this thesis we strive to provide that by answering the following question:

How to evaluate existing Semantic Web technology?

The main problem addressed in this thesis is how to provide the means to evaluate the current state of the Semantic Web technology in terms of its capability to support the development of Semantic Web applications.

In order to solve the problem we must design a tool to support the evaluation of Semantic Web technology. We must also demonstrate the feasibility and usability of the tool.

1.2 Contributions

The thesis provides two main contributions as follows:

- A platform to evaluate Semantic Web technology
  
  In order to solve the main problem above, we define the model of a support tool as an evaluation and simulation platform that aims at providing realistic insights into the different aspects of the integration and use of Semantic Web technology. We further provide an implementation of the platform that focuses on the simulation of service discovery scenarios. Service discovery is one of the main tasks to be performed on the Semantic Web; it consists of matching requests and Web resource descriptions. The implementation of the platform required the identification of the different actors that tend to take part in the service discovery process, as well as the definition of a protocol of communication between the different actors. The implementation is an API that allows the construction of specific multi-agent systems corresponding to simulations of the Semantic Web, integrating different sets of service discovery technology, resources, and consumer and provider requirements. We also illustrate the use of the API and demonstrate its usability by implementing and evaluating a specific simulation in a travel scenario.
• **New Semantic Web technology to evaluate the platform**

In order to demonstrate the feasibility and usability of the platform, we need some Semantic Web technology that can be integrated and evaluated in the evaluation platform. This technology must be advanced enough to be representative. However, the more advanced current technology is often not yet freely available. Moreover, in order to show how the platform can be used by developers to further develop and evaluate their own technology, we need to have some technology that can be easily updated. Also, for evaluating the platform, it would be an advantage to use technology that we know well and can change when needed. We thus design our own Semantic Web application that allows mediation between organizations and the Semantic Web in order to allow on-the-fly delegation of the execution of organizational tasks. This application requires both the development of new Semantic Web technology and its integration with existing Semantic Web technology. Moreover, the implementation of this application allowed us to acquire realistic insights into the processes of developing and using Semantic Web technology.

The new Semantic Web technology consists of:

**sButler:** a requester agent. A requester agent is a service discovery actor that supports the seamless integration of the use of the Semantic Web in a user environment. We provide a model and prototype implementation for sButler, a requester agent that supports the use of the Semantic Web in organizational business processes. The feasibility of sButler is shown with the evaluation platform in a travel scenario.

**OWL-DTP:** a Semantic Web service description language that is able to integrate and provide business process vocabulary. The language description includes three ontologies to describe different aspects of business processes. The comparison of OWL-DTP with two other approaches (OWL-S and WSMO) further demonstrates the need for languages that actively support the generalization of more expressive business process languages. OWL-DTP’s feasibility is also shown with the evaluation platform.

### 1.3 Outline of the thesis

**Chapter 2:** Background information on the Semantic Web vision and the existing Semantic Web technology.

**Chapter 3:** An overview of the travel scenario that is used to illustrate the problem and the solution proposed in the thesis.

**Chapter 4:** The model of the simulation and evaluation platform.
1.4. Relation to previous published work of the author

Chapter 5: The implementation of the simulation and evaluation platform as a multi-agent system that allows the evaluation of service discovery approaches.

Chapter 6: The design of the new service discovery approach that we use to illustrate and evaluate the implementation of the platform. The design includes the architecture of the sButler, a requester agent able to support the integration of organizational business processes and the Semantic Web.

Chapter 7: Our solution to the problem of query generation that must be solved to provide an implementation of the sButler. The solution includes the definition of OWL-DTP, a Semantic Web service description language.

Chapter 8: A prototype implementation of sButler that makes use of OWL-DTP.

Chapter 9: Illustrative use of the platform to evaluate the feasibility of sButler and OWL-DTP in a travel scenario.

Chapter 10: We conclude and provide directions for future work.

1.4 Relation to previous published work of the author

The content of this thesis relates to previous work by the author and colleagues at the laboratory for intelligent information systems (IISLAB).

- Chapters 4, 5, and 9 are based on Aberg, Aberg, Lambrix, & Shahmehri, 2006 and (Aberg, Lambrix, & Shahmehri, 2005).

- The problem of integrating organizational business processes and the Semantic Web, and the outline of the solution that lead to the design of sButler described in Chapter 6 is based on Shahmehri, Takkinen, & Aberg, 2003.

- The sButler’s model, architecture and prototype implementation of Chapters 6 and 8 are presented in Aberg, Lambrix, Takkinen, & Shahmehri, 2005.

- The description of OWL-DTP in Chapter 7 builds on some of the work presented in Aberg, 2005.
Chapter 2

The Semantic Web

The vision of the Semantic Web is to provide a Web where all published material is understandable by software agents. This allows for the automatic retrieval of information and the establishment of business cooperation. Examples of activities that can be automated with the Semantic Web include an information retrieval task where a requester wants to know the address of the nearest car dealer, or the establishment of a business contract where a requester wants to plan a conference trip. Besides the ability to consult all the existing published resources, such activities require an understanding of what the user wants. The two example activities above require the understanding of what a car dealer is (i.e. a car dealer is not the same thing as a car owner nor a car builder) and how to derive the distance between the car dealer and the user address, as well as knowing that, for the user, planning a conference trip includes booking and buying travel tickets, reserving a hotel room, and registering for the conference.

To perform such activities on the current Web, requesters enter keywords in a search engine or go directly to Web sites of services that they know provide the service that they need (e.g. www.amazon.com for buying a book). These are rather efficient methods, given that the requesters are somewhat Web-literate, do not mind juggling the choice of words to express a query, nor mind browsing through Web pages that do not answer their needs directly. An overview of the methods is provided by Baeza-Yates and Ribeiro-Neto (1999a). These approaches typically adopt heuristics to generate machine-understandable representations of the Web resources, which are in turn used as indexes for the retrieval. However, most Web resources’ content (e.g. text, image) is written by humans in languages tailored to human understanding, and not machine understanding. In this context, current methods for index generation cannot capture the semantics of the resources. The methods can only define heuristics that use the syntax of the resource to attach meaning to the resource. As complex and refined as the index generation method may be, the resulting resource representations are thus always approximations of the original semantic of the resource. As a result, it is impossible for these methods to ensure that the retrieval will always be satisfactory with respect to the requester’s needs. The Semantic Web
The Semantic Web vision aims at providing ways to reliably process the semantics of the resource in order to significantly improve the speed and reliability of Web resource retrieval. This is achieved by systematically attaching machine-understandable representations to all published material.

As a result, the Semantic Web can be seen as a set of semantically annotated Web resources. A Web resource may be a text, a picture, a piece of software, or a representation of an element of the real world such as a person, etc. Semantic annotations describe the semantics of the resources so that software agents can reason about them in order to reach a predefined goal. The goals of the agents vary from application to application, but they all rely on the operation of finding and using the resources necessary to perform the goal. To enable the deployment of the Semantic Web, technology is being developed for representing semantic annotations, for finding them, for reasoning about them and for using the resources that they annotate. The technology provides:

**Machine-understandable languages** to describe the semantical content of Web resources. RDF and OWL are such languages. They allow the description (often context dependent) concepts and relations between concepts. For example, let us consider a resource that is the Web page of a person that is both a university teacher and a PhD student who takes doctoral courses. Machine-understandable languages may provide the vocabulary to express the relation between this person and the list of courses that he teaches, as well as another relation between this person and the list of courses that he takes, and yet another relation between this person and the hours when he is available to answer his students’ questions.

**Semantic annotation description languages** that provide the set of language constructs for describing the properties, capabilities, and use rules of the Web resources in an unambiguous, machine-understandable manner. OWL-S and OWL-DTP are such languages. Semantic annotation description languages are the machine-understandable languages that provide a uniform format to describe the specific concept of Web resource.

**Semantic-aware tools** that use and manage the semantic annotations, as well as the ontologies\(^1\) that the annotations may refer to.

**Semantic Web operations** that include resource retrieval, resource use, and Semantic Web management operations such as handling the changes in the resources’ content. Semantic Web operations use semantic-aware tools.

Given this technology, so-called Semantic Web applications can be developed. Semantic Web applications are the applications that make use of the semantic annotation of resources. As a result, the set of Semantic Web applications is quite a large one, including the semantic-aware tools and Semantic Web operations above, as well as the applications that exploit these technologies.

\(^1\)From (Neches et al., 1991): “An ontology defines the basic terms and relations comprising the vocabulary of a topic area as well as the rules for combining terms and relations to define extensions to the vocabulary.”
The following sections provide more detailed information for each of the four categories of technologies that we have just introduced.

2.1 Machine-understandable languages

One important first step in building the Semantic Web vision is to specify the machine-understandable language(s) in which the description of the published material is to be written. Figure 2.1 shows the layers of the Semantic Web as proposed by Tim Berners-Lee\(^2\) in a talk at the XML-2000 Conference. The figure shows his vision of the stack of languages to be developed\(^3\). The principle of the stack is that the higher level languages use the syntax and semantics defined by the lower level languages. As a result, the higher the language, the more expressive the language and the more complex the use and management of Web resources can be. To further illustrate the use of the language stack, Figure 2.1 also provides a tentative description of the categories of concepts to be described by these languages (see the Self description language, data and rules icons.) Concretely, the layers are meant to be used as follows, starting from the bottom:

1. Definition of basic types:
   - Uri (Unique Resource Identifier): All Internet resources available on the World Wide Web are referenced by a Uri. As stated in the URI Activity Statement (2005):
     
     The Uris are [...] simple text strings that refer to Internet resources. Uris may refer to documents, resources, to people, and indirectly to anything. [...] Document formats [e.g. HTML] and protocols [e.g. HTTP] may come and go, but Uris will remain as the glue that binds the Web together.
     
     Notice that URLs (Unique Resource Locators) are specific Uris which are given to resources that can be retrieved.
   - Unicode (1991-2006) is an encoding system that specifies a machine-understandable code for letters, digits, punctuation and some control characters. The code defines the alphabet that can be used to write machine-understandable words and sentences.

2. Providing some basic syntax and naming mechanisms:

\(^2\)As designer and writer of the first WWW server in 1991, Tim Berners-Lee is often considered to be the “father” of the World Wide Web. He is the director of the World Wide Web Consortium (W3C) which coordinates Web development worldwide.

\(^3\)We are aware that the stack has been critiqued and changes have been proposed since 2000. However, the original stack is still providing a good intuitive view of the different categories of languages required for developing the Semantic Web.
• XML\(^4\) (1996-2003) is a markup language and a W3C\(^5\) standard. Typically, markup languages allow for the annotation of free text with tags which provide meta information about the text. For example, the statement \textit{the car is red} can be annotated as follows: \texttt{<vehicle>the\ car</vehicle>\ is\ <color>red</color>}. XML provides a very basic conceptualization system where the tags are concepts.

• NS (or XML namespaces (2004)) is a W3C standard naming mechanism for the tags defined with XML. This way, one may refer to tags which are defined in different name spaces but have the same names.

• XML Schema (2000-2003) is a W3C standard that provides a typing definition mechanism that allows for the definition of tag definitions that can then be used by different XML files. A schema allows for the definition of structured concepts. For example, a schema can specify the concept \texttt{<person>} to be composed of two other concepts \texttt{<firstname>} and \texttt{<lastname>}. In turn, an XML definition can refer to the schemas so that when a \texttt{<person>} concept is defined, its definition must follow the structure specified in the schema. Typically, by defining an XML schema, one defines a new domain specific XML language. OFX (1997-2006) stands for “Open Financial eXchange” and is an example of an XML language for the electronic exchange of financial data between financial institutions, businesses and consumers via the Internet.

3. Allowing for richer data modeling by providing the notion of relationships:

• RDF\(^6\) (1997-2004) is a W3C standardized XML language that allows for writing statements composed of three elements: subject, object and predicate where the predicate expresses the relationship between the subject and the object. Each subject, object and predicate has a unique Uri.

• RDF Schema (2004b) builds on XML schemas and specifies a typing mechanism which is similar to the class mechanism of the object-oriented paradigm.

4. Providing knowledge and languages to describe that knowledge:

The knowledge is represented in ontologies. A definition of ontology that is much referenced in the literature is as follows: “An ontology is an explicit specification of a conceptualization.” (Gruber, 1993). More specifically, and as Lambrix (2004) puts it:

\begin{quote}
Intuitively, ontologies can be seen as defining the basic terms and relations of a domain of interest, as well as the rules for
\end{quote}

\footnote{XML stands for eXtensible Markup Language.}
\footnote{W3C stands for World Wide Web Consortium. It is the organization that coordinates Web development worldwide.}
\footnote{RDF stands for Resource Description Framework.}
2.1. Machine-understandable languages

combining these terms and relations. Ontologies are being used nowadays in many areas [...] for communication between people and organizations, as the basis for interoperability between systems, and as query models and indexes to repositories of information.

The ontology vocabulary describes knowledge about domains. An ontology is described with a language. Such languages may share expressivity power while differing in their potential domain coverage. For example, OWL (2004) may be used to describe any domain (with respect to its expressivity limits) while OWL-S (2005) can only be used to describe Web services.

5. Reasoning about the existing knowledge:

The logic layer provides for language constructors and tools that allow for reasoning about ontologies to establish the consistency and correctness of specific concepts or to infer new concepts that are not explicitly stated. Queries, assertions and rules are examples of language constructs defined in this layer.

6. Ensuring consistency and correctness of the knowledge:

The proof layer provides tools for generating the logical path of reasoning to establish consistency and correctness of concepts. They typically require the ability to describe rules.

7. Trusting the Semantic Web:

The Semantic Web does not assert that all statements found on the Web are “true”. Furthermore, all statements occur in some context. As a result, each application needs this context to evaluate the trustworthiness of the statements. The trust layer provides the mechanisms that are able to demonstrate the truthfulness or the quality of a resource with respect to a specific context of use. This is typically done through systems of authentication which require that logical reasons for trusting the data are provided.

Concrete work is currently underway to provide languages for each layer. Starting from the bottom, for the three first layers W3C standards exist, the fourth layer is well on its way (with the OWL language standardized and applications being developed) and the other ones are being tackled. The layering work is not straightforward, as illustrated by Patel-Schneider and Fensel (2002), who show that the fact that “the RDF Schema theory of classes and properties is very weak” makes it difficult to directly build OWL on top of RDF Schema. As a result, alternative layering mechanisms must be proposed such as considering an RDF layer that provides several versions of RDF including the original RDF and a new RDF(DL) on which OWL DL can be built in a more straightforward manner. Besides these discussions, which show that some adjustment may be required in the layer cake, further work has been conducted to provide query
languages and retrieval engines for each of the four first layers, as described in (Furche et al., 2004).

2.2 Semantic annotation description languages

As mentioned in the introduction of this chapter, semantic annotation languages correspond to the category of machine-understandable languages that describe Web resources. Traditionally, Web resources are either information nuggets or executable services.

Semantic Web services is the common term used to describe semantic annotations for executable services. The notion of Semantic Web service builds on the notion of Web service (Web Service, 2002a). Web services are programmable, machine-understandable interfaces that can be attached to an executable service in order to provide the information necessary for software agents to decide if they need to use the specific resource or not. The current consensus is that these descriptions specify the data format required for calling the actual service (i.e. a port and a set of inputs and outputs described using WSDL (2004c)). With the rise of the Semantic Web, it has become common to argue for the need for Web services that also provide semantical descriptions of the services. Semantic Web services are such Web services, in that they provide descriptions based on the semantics of the services. There are several propositions for Semantic Web service description languages, such as OWL-S (OWL-S, 2005), WSMO (WSMO, 2004), and OWL-DTP (OWL-DTP, 2005-2006). It is becoming more and more common to consider that all the resources on the Semantic Web are services.
For example, information nuggets can be seen as services whose purpose is to provide a specific nugget of information. In the following, and if not expressed otherwise, we adopt the assumption that all the annotations of Semantic Web resources are Semantic Web services. In the remainder of the thesis, we commonly use the term Web service for Semantic Web service.

### 2.3 Semantic-aware tools

As mentioned above, semantic-aware tools are the tools that use and manage the semantic annotations, as well as the ontologies that the annotations may refer to. Examples of tools that use semantic annotations are Semantically enhanced web browsers like Piggy Bank (Huynh et al., 2005), Semantic Web search engines like Swoogle (Ding et al., 2004), and Semantic Web query languages (Furche et al., 2004). Examples of ontology management tools are ontology editors such as Protégé (1995-2006) and ontology alignment systems such as SAMBO (SAMBO, 2005-2006). Examples of management tools for the semantic annotations are automatic generators of semantic annotations such as the one-click publishing process of IRS III illustrated in (Domingue, Cabral, Hakimpour, Sell, & Motta, 2004). Examples of tools that use semantic annotations and the ontologies that the annotations refer to are logic reasoners. Currently the most successful reasoners are using description logics, such as Racer (Racer, 1999-2006). Reasoners relying on other logics (e.g. F-logic (Bruijn, Polleres, Lara, & Fensel, 2005)) are also being proposed.

### 2.4 Semantic Web operations

As mentioned above, Semantic Web operations use semantic-aware tools. The operations are complex, and solutions are just emerging for applications where semantic annotations are specifically formulated as Semantic Web services. As pointed out by Lara, Lausen, Arroyo, de Bruijn, and Fensel (2003), and further detailed in the description of the Web service modeling framework (Fensel & Bussler, 2002), Semantic Web services are designed to support the operation of service discovery\(^7\), composition and interoperability. WSMX (WSMX, 2004-2006) and IRS III (Domingue et al., 2004) apply the recommendation of the framework to describe service discovery and service execution. The integration of heterogeneous services to form new composite services on a need-to-use basis is a very active current domain of research with initiatives by the W3C (e.g. Web Services Choreography (2002b)), the agent community (e.g. Racing, 2004 and SWORD, 2002) and the industry (e.g. Peltz, 2003).

There are some attempts to describe operations that handle changes in the state of the resources published on the Semantic Web. However, the semantic-aware tools required for such technology are still under development. Work done

\(^7\)When the semantic annotations are Semantic Web services, the operation of resource retrieval is called service discovery.
in the REWERSE network (Rewerse, 2004) aims at providing such technology.

2.5 Difficulties to overcome for deployment of the Semantic Web

Because the Semantic Web should allow machines to reason about Web resources instead of humans (Berners-Lee et al., 2001), its development relies largely on advances in artificial intelligence (e.g. knowledge representation, logic reasoners, machine learning, multi-agent systems), but also on other domains such as distributed systems and trust management. Furthermore, the Semantic Web is such a complex environment that the development of each new technology seems to reveal new challenges and the need for more technology. As a result, more than 5 years from the official beginning of this adventure, the Semantic Web technology is only starting to be mature enough to provide for added value, and the first real Semantic Web applications are appearing, e.g. (WSMX, 2004-2006), (Huynh et al., 2005), (Ding et al., 2004).

Similarly to Web technology, the Semantic Web technology and applications must overcome the difficulties associated with the huge size and versatility of the Web. Specifically, the Web technology must allow for:

- **Scalability**: the Semantic Web allows access to a very large amount of knowledge in terms of concepts, resources, and annotations (i.e. as of August 2006, Swoogle indexed more than 300,000,000 statements). To be successful, the agents navigating on the Semantic Web will have to be able to process this information in a reasonable time and using a reasonable amount of CPU.

- **Heterogeneity**: the Semantic Web is an heterogeneous environment with respect to resource domain, technology, and consumer and provider’s requirements. For example, in the travel industry, different providers adopt different data formats, data storage tools and presentation languages to describe travel information. The software agent that will crawl the Semantic Web will have to be able to handle this heterogeneity to support consumers and providers in reaching their goals (e.g. buying train tickets.)

- **Change**: as on the Web, resources on the Semantic Web continually appear, are updated and disappear, and technology evolves constantly. In the travel domain, flight providers may update flight timetables continually making it impossible for consumers to be sure that their flight will actually be departing at the predefined time. Supporting consumers and providers in the handling of the continuous changes in their environment is another challenge that software agents must overcome.
Chapter 3

Illustrative Scenario

Travel scenarios have been used by others to illustrate the need for and use of Semantic Web technology, e.g. (McIlraith & Son, 2001), (Dalal, Temel, Little, Potts, & Webber, 2003), (Benatallah, Sheng, & Dumas, 2003). Travel scenarios are interesting for several reasons. As mentioned in the introduction, the travel industry is a successful example of a Web-based business from both the consumers’ and providers’ points of view (Davis, 2006) (Lipsman, 2006). Today, most travel providers, from airplane and train companies to travel agencies allow consumers to book and buy travel tickets over the Internet, providing for a large set of real consumer queries, provider services and formatted travel knowledge. Furthermore, the travel domain is a complex domain where the state of the market is both continually changing (Lipsman, 2006), and heterogeneous (e.g. in terms of users’ preferences: even for the same user, the circumstances and constraints may differ from one travel planning to the next.) Moreover, the task of planning travel usually requires the use of several services (e.g. to buy tickets, to book hotel rooms) and illustrates the need for the management of multiple services. It is also relatively easy to modify a travel scenario to test different capabilities of service discovery algorithms. Finally, travel planning is an activity and a domain which is easily understood by most people. For all these reasons, we adopt a travel scenario to illustrate our work.

The travel scenario is as follows. The travel consumers are members of the research staff at a university. To facilitate the management of the university, the staff is provided with a common workflow representation of basic work activities such as conference travel. The part of the workflow that describes the conference travel activity specifies that when researchers of this university go to a conference, they perform a number of tasks as illustrated by the left workflow in figure 3.1. First, the researchers apply for funding for travel. This is done by filling in administrative forms and handing them over to the accounting department several weeks before the trip. The second step is to plan the actual trip. This may include tasks such as booking transportation to the conference location. Moreover, the travel itinerary produced provides the process instance for the next task as illustrated by the right workflow in figure 3.1. The third
step is to actually travel to the conference and participate. This includes such things as finding the right gate at the airport or rescheduling the trip if a plane is late. Finally, the researchers must report on the scientific content of the conference to their colleagues and handle administration regarding travel costs for the accounting department. These last two tasks may be performed in parallel.

Performing this conference travel workflow is quite time-consuming for the researchers:

- They must identify and fill in several forms (see **Apply for Funding** and **Report to Accounting**). Such forms are available on the internal Web pages of the university as pdf files.

- They must identify the different suitable travel routes and travel providers, as well as hotels. Such information is typically available on the current Web available on Web sites, either as text information or as interactive services able to query domain-specific databases.

- They must collect information about each service provided to decide which ones better match their preferences (e.g. time/dates, comfort, economy, allergies, etc.) Some preferences may be derived from the dates of the conference (time/dates), the policies of the university available on the internal Web pages (comfort, economy), as well as other sources (e.g. the calendar of the researcher, the previous preferences of the researcher, etc.)
They must coordinate the legs of the travel. The legs descriptions are described on travel Web pages or in emails sent by the travel Web sites.

They must take into account a set of university and conference rules (e.g. deadlines, payment means, etc.) Such information is available on the internal Web pages of the university and on the Web site of the conference.

They may need to change the travel plans on the fly (e.g. storm that closes an airport, bus missed, etc.), which requires them to start a new search for suitable providers.

They must repeat/retype the same information several times (e.g. they must provide the destination of their travel in each administrative document and in each query formulation that they use to identify suitable providers.)

This process is very repetitive and refers to many sources of information, both local (university rules, user preferences) and distant (transport means, hotels), most of which are commonly available on the Web. As a result, the Semantic Web could be used as a medium to automate, extend the coverage, and speed up the process of identifying and using the services of distant travel contractors.
Chapter 4

Model for the Simulation and Evaluation Platform

In this chapter we introduce the model for our simulation and evaluation platform, while Chapter 5 describes an implementation of this model.

As mentioned in the introduction, we aim at providing the means to investigate the current state of the Semantic Web technology in terms of its capability to support the generation of Semantic Web applications.

Semantic Web applications tend to imply many fast-changing parameters such as the amount and kind of data resources considered, the speed of the network, the expressivity of the semantic languages used to expressed the annotations and concepts, the difficulty of the reasoning to be performed, etc. Identifying and taking into account all the parameters in order to provide a complete theoretical analysis is a very difficult task. Moreover, it is not always necessary to analyze all the possible cases of use for each specific Semantic Web application or technology. Instead, we consider the already complex problem of identifying the specific application scenarios that can be successfully implemented using the existing Semantic Web technology.

When considering the new applications that refer to the Semantic Web vision and/or rely heavily on the technology being developed for the Semantic Web, we find a large array of possible application scenarios. Inter-organizational workflows (Patil, Oundhakar, Sheth, & Verma, 2004; Dan et al., 2004), information retrieval (Huynh et al., 2005) (Ding et al., 2004), e-commerce (Trastour, Bartolini, & Preist, 2002) and bioinformatics (Lambrix, 2005) are examples of the research domains that propose such new application scenarios. Moreover, each new proposed scenario tends to make its own assumptions regarding three aspects of the Semantic Web:

1. the use case, i.e. who are the users and resource providers, what motivates them to use and provide resources, and what are the social and business rules that govern their interaction,
2. the available resources together with their semantic annotations, and
3. the available technologies (e.g. description logic reasoners, rule engines, ontology managers, etc) and how they are used.

In turn, such assumptions specify implicit requirements about the technology. For example, in travel use cases, travel consumers typically specify a set of preferences in terms of maximum price, dates, time length of the travel, etc. Taking into account different categories of preferences implies different acceptable technical solutions. Moreover, the growing set of semantically annotated Web resources made available also put requirements on the technology that is going to be using them. For example, the language in which they are described (today: mostly RDF, and, less often, some flavor of OWL) specifies constraints on the reasoning that can be performed or not.

The Semantic Web is in the process of being deployed. The process is partly controlled by the establishment of standards (e.g. RDF, OWL), and is partly the result of local experiments. The incomplete control over the deployment process makes it difficult to perform a thorough experimentation of Semantic Web technology on the real Semantic Web.

For this reason we propose a common simulation and evaluation platform for Semantic Web technology where current and future Semantic Web technology can be integrated and evaluated with suitable use cases and resource sets.

This chapter describes the requirements to build such a simulation and evaluation platform, and proposes a model for the platform that fulfills these requirements.

4.1 Requirements for a simulation and evaluation platform

As mentioned above we need a platform that allows the simulation of different Semantic Web environments in a controlled manner. This leads to the following set of requirements that the platform must fulfill. More specifically:

R1 The platform allows simulation of the Semantic Web
   We want to evaluate technology in a realistic and controlled Semantic Web environment. As a result, we need to be able to simulate the Semantic Web.

R2 The platform allows evaluation of Semantic Web technology
   In order to evaluate technology we must be able to observe its behavior and measure its performance. There are also different aspects of performance to consider, such as speed, correctness, network load, security issues, etc.

R3 The simulations are controlled
   As mentioned above, different assumptions are made about the Semantic Web’s capability and use. In order to know which technology can handle
which assumptions we must be able to specify different simulations corresponding to different specifications of the use case, the available resources and the technology available.

**R4 The evaluations are controlled**

Since Web technology is continually being developed, it is not possible to identify once and for all the characteristics of the technology that must be measured, nor the measures that must be taken. As a result, the platform must allow the clear specification of the application and domain-specific parameters that define the extent to which each technology or combination of technologies is evaluated. The parameters include application- and domain-specific measures, as well as specific scenario settings such as the data set and ontologies available.

**R5 The programming effort is minimal**

The generation of the simulations requires the integration of different technologies. The users of the platform can not be expected to be expert in each of these technologies, nor to have access to the source code of the technology. As a result, the integration must be modular, and require as little programming effort as possible.

**R6 The platform is Semantic Web technology independent**

In order to allow for the integration of the large range of heterogeneous technology that is developed, the platform must be as technology independent as possible.

### 4.2 Platform model

As a first action to satisfy requirement R1 (i.e. allows simulation of the Semantic Web) and R5 (i.e. minimizing the programming effort), we propose a platform which consists of a set of tools that are able to simplify and systematize the generation of Semantic Web simulations corresponding to different sets of assumptions. The role of the platform is illustrated in Figure 4.1: the platform uses the set of assumptions to generate a monitored Semantic Web. To be able to specify the concrete operations that the platform must perform, concrete data structures for assumptions and simulations must be described. A first step towards these descriptions is to model the assumptions and the Semantic Web.

#### 4.2.1 Modeling assumptions about the Semantic Web

As mentioned previously, the Semantic Web assumptions are made about three aspects of the Semantic Web: the technology, the resources and the use case. We thus propose a model that is illustrated in Figure 4.2 and is composed of three elements:
The technology available on the Semantic Web. As described in Chapter 2, the technology consists of machine-understandable and semantic-aware description languages, semantic-aware tools, and Semantic Web operations.

The resources consist of the Web resources (i.e. services) and their attached semantic annotations (i.e. Semantic Web services).

The use case provides a set of constraints on the technology and the resources. Examples of constraints are the specific list of Uris corresponding to the available resources, the identification of the provider for each resource, the set of requesters’ queries, and the requirements the providers and requesters put on the use of the technology.

4.2.2 Modeling the Semantic Web

Berners-Lee et al. (2001) provided a generic description of the Semantic Web as a set of annotated Web resources and a set of software agents able to understand these resources. Hendler (2001) completed the description stating that
the agents may have different capabilities and are able to understand semantic annotations and collaborate with other software agents to achieve specific goals. The agents’ capabilities typically correspond to the Semantic Web technology that they embed. Additionally, the agents’ goals often correspond to combinations of queries for resources, and the term query is often used instead of goal.

Keeping in mind the different categories of Semantic Web technologies introduced in Chapter 2, we propose a Semantic Web model that highlights the distinction between the different categories of data available on the Semantic Web. The model illustrated in Figure 4.3 consists of the four following components:

The **Web resources** provide some semantic content presented in a format that may not be machine-understandable. It is illustrated as the **Web resource** knowledge base at the top left of Figure 4.3.

The **language specifications** include the machine-understandable languages and the Semantic annotation description languages. They are represented with the set of **LANGUAGE SPECIFICATION** documents in Figure 4.3.

The **machine-understandable data** includes the semantic annotations of the Web resources, the possible queries for resources, and the ontologies available to describe and reason about the annotations and the queries. The data is represented with the **DATA** box at the top right of Figure 4.3. The data is expressed with the machine-understandable languages defined by the language specifications introduced above. The semantic annotations (i.e. **SW service** documents in Figure 4.3) are attached to one or several of the Web resources described above.

The **operations** are the different Semantic Web applications and are represented by the set of **OPERATION** boxes in Figure 4.3. Such operations typically correspond to a set of **agents** that integrate one or several **semantic-aware tool(s)** (i.e. the **S. A. TOOL** boxes in Figure 4.3), and may collaborate with each other by exchanging messages whose content is written in one of the available machine-understandable or semantic-aware description **languages** (i.e. the **LANGUAGE SPECIFICATION** documents in Figure 4.3.) By enforcing a strongly decoupled architecture, the multi-agent system architecture supports the fulfillment of requirements R5 (minimal programming effort) and R6 (i.e. technology independence).

### 4.2.3 The platform

Given the two models above, we can describe the simulation generation role of the platform as a mapping service conditioned by the use case, as illustrated in Figure 4.4. By not adding any assumptions about the capabilities of the technology of the use cases, the mapping service supports the fulfillment of requirement R6 (i.e. technology independence). Specifically, the assumptions’
Web resources are mapped to the Semantic Web model’s Web resources, the assumptions’ semantic annotations are mapped to the Semantic Web model’s sets of ontologies, and Web service and query documents which are part of the machine-understandable data component, the assumptions’ semantic-aware description languages and machine-understandable languages are mapped to the set of language specification documents, the assumptions’ Semantic Web operations are mapped to the set of Semantic Web model’s Operations, and the assumptions semantic aware tools are mapped to the Semantic Web model’s set of semantic aware tools which is part of the operation component.

To perform this controlled mapping, the platform provides some specific support for generating each of the components of the Semantic Web model. The result is a more detailed version of the platform principle of Figure 4.1 as illustrated in Figure 4.5. The principle still consists of providing a platform that uses a set of assumptions to generate a monitored Semantic Web. The models introduced above for the assumptions and the Semantic Web are providing more detailed descriptions of the original Assumptions and Model Simulation boxes. In addition, the following set of support tools is provided to fill in the central Platform box.

Support to instantiate the different components of the Semantic Web model (see the Support to instantiate box in the PLATFORM box of Figure 4.5). This support is especially designed to facilitate the integration of the
4.2. Platform model

Figure 4.4: Mapping service provided by the platform
Figure 4.5: Platform model
4.2. Platform model

different Semantic Web technology and thus supporting the fulfillment of requirement R5 (i.e. minimal programming effort.) This support is itself composed of four support components as follows:

- Support to instantiate the Web resource component.
  This support consists of using the assumptions about the resources to:
  1. generate the Web resource component by gathering the resource Uris in a single database,
  2. gather the semantic annotation in another database that refers to the database of resource Uris, and
  3. generate a set of service provider agents in charge of advertising and managing the resources.

- Support to instantiate the language specification component.
  This support consists of identifying the set of languages used in the use case, the technology, and the data.

- Support to instantiate the machine-understandable data component.
  This support consists of gathering the semantic annotations, the queries defined by the use case, and the ontologies to which the annotations and the queries refer.

- Support to instantiate the operation components.
  This support provides the means to generate the Semantic Web applications specified in the use case as operations as introduced in the Semantic Web model above (i.e. as multi-agent systems where each agent packages specific uses of semantic-aware tools.)

**Setting specification support** fulfills requirement R3 (i.e. the simulations are controlled) by allowing the definition of different settings of the same simulation in terms of, for example, the number of resources used or the number of agents available with a specific behavior. This is handled by a specific set of support tools represented by the Settings box in the PLATFORM box in Figure 4.5.

**Evaluation support** fulfills requirement R2 (i.e. allows evaluation) and R4 (controlled evaluation) by providing a monitoring mechanism. We propose to adopt an event listening approach where the different components of the simulation can generate events. To complete the fulfillment of requirement R5 (i.e. minimal programming effort), and as illustrated by the Evaluation support box in the PLATFORM box in Figure 4.5, the platform provides the API for implementing specific monitoring behaviors that listen to specific events and compute specific evaluation reports, and a monitoring agent in charge of running parallel threads for each of these behaviors.

We have provided a model for the platform that fulfills the six requirements introduced in Section 4.1. In order to show its feasibility and usability, this
model is implemented, and its use is illustrated, evaluated, and discussed in the rest of the thesis.

In the next chapter we provide an implementation of the platform for the simulation of service discovery scenarios.
Chapter 5

Implementation of the Simulation and Evaluation Platform

In order to demonstrate the feasibility of the platform model introduced in Chapter 4, this chapter describes an implementation of the model. The implementation focuses on the operation of service discovery. We will later describe some service discovery technology in Chapters 6 through 8 and illustrate the use of the platform to evaluate some service discovery technology in Chapter 9.

The implementation focuses on the operation of service discovery for two reasons: 1) the operation component of the platform is the most complex component to implement, and 2) service discovery is one of the basic operations that the Semantic Web must provide in order to speed up and generalize its use.

The operation component is the most complex component to implement because each operation requires identification of the categories of agents that will participate, identification of the generic abilities that each agent will implement, and assurance that the agents establish coherent collaborations with respect to the operation’s function.

Moreover, service discovery is the operation that performs resource retrieval on the Semantic Web (see Chapter 2). This operation is the subject of a great deal of interest right now, both from organizations that want to use it and the community of developers of Semantic Web technology, which includes industry, e.g. (Metatomix, 2000), and academics. Organizations see service discovery as one of the first concrete Semantic Web applications that would truly help them to manage their business processes more efficiently, both internally and externally, e.g. when they need to establish new business contracts with other organizations (Zhang & Hung, 2006). Academics see the possibility of finally applying complex and advanced AI technology, and helping to solve important problems relying on the analysis of large semi-structured data structures, such as the design of new genetic therapies (Lambrix, 2005). There is no standard for
service discovery yet, and the approaches for design and implementation are a current, very active topic of discussion in the Semantic Web community (Workshop WF08 at WWW2005, 2005). Service discovery is thus a good candidate for simulation and evaluation.

As illustrated in Figure 5.1, the current state of the platform implementation provides a solid implementation for the operation component and evaluation support, as well as more basic implementations of the other kinds of support. The following sections describe in detail the implementation of the different kinds of support.

5.1 Support for the operation component

We implemented the following set of support tools for supporting the simulation of service discovery operations:

- The description of the different categories of agents that typically take part in the operation of service discovery. Concretely, the description consists of:
  - A set of agent categories (see Table 5.1 and Figure 5.2).
  - The generic protocol of communication that takes place between these agent categories to perform service discovery (see Figure 5.3).
  - An API description of each agent category in terms of the minimal set of functions that they must provide, and the minimal set of messages that each agent is expected to be able to interpret.
  - An illustrative implementation of one Semantic Web simulation corresponding to the travel scenario introduced in Chapter 3.

With these tools, the potential users of the platform do not have to identify their own agent categories, but can focus on specifying the agent categories that are taking part in the operation that they want to implement, and the mode of collaboration they must adopt.
### Table 5.1: Categories of agents participating in service discovery

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requester</td>
<td>A Requester is able to formulate a query for a specific service, and to send it to the agent(s) able to start up the process of service discovery, i.e. the Web service discovery agents described next. A Requester may also be able to enact a service once it is discovered.</td>
</tr>
<tr>
<td>Web service discovery agent</td>
<td>A Web service discovery agent is able to find the services that match a given query for services. Web service discovery agents may also be able to discover compositions of services that match the query.</td>
</tr>
<tr>
<td>Web service manager</td>
<td>A Web service manager is a directory of Semantic Web services. Web service managers are associated to one or several Semantic Web service description languages such as OWL-S, WSMO or OWL-DTP. A Web service manager is able to answer queries for specific Web services. A Web service manager does not perform composition of services.</td>
</tr>
<tr>
<td>Service provider</td>
<td>A Service provider sends service advertisements to Web service managers. The service advertisements are formulated as Semantic Web services.</td>
</tr>
<tr>
<td>Ontology Agent (OA)</td>
<td>An Ontology Agent (OA) is able to reason about a specific domain (e.g. Travel, Car.) Any agent can delegate part of their reasoning to ontology agents. OAs can answer several types of queries such as “is A a subconcept of B?”, “what are all the subconcepts of C?” or “what are all the instances of concept C?”</td>
</tr>
</tbody>
</table>
Implementation of the Simulation and Evaluation Platform

Figure 5.2: Service discovery’s agents

Figure 5.3: Protocol of communication to perform service discovery
• One default implementation for each agent category. This is useful for users who do not wish to specify all the agent behaviors, but only the specific ones corresponding to the specific technology that they want to test.

Specifically, the set of agent categories consists of: requesters, Web service discovery agents, Web service managers, service providers and ontology agents.

In order to provide service discovery, the agents tend to collaborate as follows. Requester agents are able to formulate queries for specific services and send them to Web service discovery agents (see arrow 1 in Figure 5.2 and arrow 1 in Figure 5.3), which are themselves able to retrieve the needed services and possibly combine them to match the query. In order to find the services, the Web service discovery agents must access the Web service registry that is maintained by the Web service managers (see arrow 2 in Figure 5.2 and arrow 2 in Figure 5.3). The Web service manager answers the Web service discovery agent (see arrow 3 in Figure 5.2 and arrow 3 in Figure 5.3), which is then able to compute an answer to the query and return it to the requester (see arrow 4 in Figure 5.2 and arrow 4 in Figure 5.3). In order to allow the agents to be domain- and application-independent, domain-specific reasoning (that is allowed and encouraged by the vision of the Semantic Web) can be delegated to specific ontology agents where each agent is able to reason about one or a set of specific domains (see the arrows pointing to the ontology agents in Figure 5.2 and arrow 2', 2", 3', and 3" in Figure 5.3). For example, a Travel ontology agent is able to reason about travel legs. Finally, service providers are the agents that submit the descriptions of their services to the Web service manager (see the tightly dashed arrows in Figure 5.2). Service providers may be contacted by requesters who want to enact one or several of their services (see the wider dashed arrows in Figure 5.2).

5.2 Evaluation support

The implemented evaluation support consists of one monitoring agent and one illustrative agent behavior able to compute the time to get an answer to a specific request message.

5.3 Settings

The rest of the support corresponds to the specification of the following parameters:

• The list of resources (as Uris) available on the simulated Semantic Web (cf. Web resources component in Figure 5.1.)

• The Semantic Web languages that each agent understands (cf. Language specification and machine-understandable data components in Figure 5.1.)
• The number of agents of each category that participate in the scenario (cf. Settings box in Figure 5.1.)

• The mapping between the service providers and the resources represented as Uris (cf. Settings box in Figure 5.1.)

• The mapping between the agents and the machines on which they run (cf. Settings box in Figure 5.1.)

• The order in which the agents appear in the scenario. It is also possible to time the appearances (cf. Settings box in Figure 5.1.)

5.4 A multi-agent system

When it comes to the actual implementation of these tools, we considered two facts. First, the implementation of the operations requires integration of different technologies written in different programming languages, possibly running on different machines. Second, to allow for the comparison of different technologies and different settings, the evaluation platform should provide the means to minimize the effort required by changing one or several technologies used by the operations. By providing the possibility to describe applications whose architecture is strongly decoupled, multi-agent systems as defined by FIPA\(^1\), provide an environment that supports these needs. We thus implemented the support above in Jade (Jade, 2000-2006), a Java framework that provides the means to describe FIPA multi-agent systems. To do that we needed to take into account the specifics of FIPA multi-agent systems. According to FIPA,

an agent is a computational process that implements the autonomous, communicating functionality of an application. Agents communicate using the Agent Communication Language (ACL). An agent is the fundamental actor on an Agent Platform which combines one or more capabilities, as published in a service description, into a unified and integrated execution model. An agent must have at least one owner, for example, based on organizational affiliation or human user ownership, and an agent must support at least one notion of identity. This notion of identity is the Agent Identifier (AID) that labels an agent so that it may be distinguished unambiguously within the Agent universe. An agent may be registered at a number of transport addresses at which it can be contacted (FIPA AMS, 2004).

In FIPA multi-agent systems, agents advertise their capabilities as services. These services are described according to a specific format and are registered in a Directory Facilitator (DF). The DF is not a semantic-aware tool as defined in Chapter 2. The DF’s reasoning about service description is limited to text

\(^1\)The Foundation for Intelligent Physical Agents (FIPA) is a non-profit organization aimed at producing standards for the interoperation of heterogeneous software agents.
5.4. A multi-agent system

As a result, the DF cannot be used as a Web service manager as defined in Table 5.1. However, the DF is a central element in the FIPA agent platform. In order to implement our support platform as a FIPA multi-agent system, we have to integrate the use of the DF in the process of service discovery. The result is an extension of the service discovery model introduced in the previous section with a DF agent and a set of request messages sent to this agent (see the new DF entity and the arrows pointing to it in 5.4 and 5.5 which are themselves extensions of Figures 5.2 and 5.3.) With the integration of the DF, it is now possible for each agent that needs to send a message to another agent with a specific capability (e.g. Web service manager), to query the DF for the address(es) of all the agents with this capability. Figure 5.5 illustrates the new protocol of communication that extends the protocol introduced in Figure 5.3 with requests to the DF.

Given this new protocol of communication, our implementation in Jade provides the following support to build specific service discovery operations on the Semantic Web:

- An API as a set of Java interfaces. For each agent category introduced in Table 5.1 above, the API specifies the minimum set of behaviors that they must provide as well as the minimum set of messages that they must understand.

Each of the agents’ behavior can embed one or several Semantic Web tech-
Figure 5.5: Service discovery communication protocol in a FIPA multi-agent system
5.5 Related work

To our knowledge, there is no other simulation and evaluation platform for Semantic Web technology available today. Other attempts to evaluate Semantic Web technology tend to focus on one specific category of semantic-aware tools. This is the case for (Guo, Qasem, Pan, & Heflin, 2006), which focuses on providing test sets and evaluations of different Semantic Web reasoners. These focused approaches define the specific metrics and support tools that can be integrated into our platform in order to provide the holistic evaluation of a set of technologies working together.

The similarity of the paradigms of the Semantic Web and multi-agent systems has been acknowledged by others. However, this other work concentrates on providing an interface between multi-agent systems and the Semantic Web. Jade does go in the direction of supporting the integration of the Web paradigm in multi-agent systems by providing the ability to use the HTTP protocol as the communication protocol between agents. However, more advanced features such as the management of Semantic Web resources are not taken into account by any other agent approach that we know of. IRS III (Domingue et al., 2004) and WSMX do provide platforms to manage the life cycle of Semantic Web services in terms of service discovery. However, they force the use of one Semantic Web service representation (i.e. WSMO), which may not fit all Semantic Web use cases. Furthermore, they do not provide any means to evaluate and compare different approaches.
In the next three chapters we describe the Semantic Web technology that we have developed, which we are going to integrate together with other Semantic Web technologies in the illustration and evaluation of our platform in Chapter 9.
Chapter 6

sButler: a Requester Agent

Now that we have described the evaluation platform, we are going to introduce and describe the new service discovery technology that we have developed, which we are going to evaluate with the platform in Chapter 9. In the current chapter we focus on describing the design of a new service discovery approach while different aspects of the implementation will be described in Chapters 7 and 8.

By allowing software agents to communicate and understand the information published on the Web, the Semantic Web enables new ways of doing business and consuming services. Semantic Web technology will provide an environment where the comparison of different business contracts can be conducted in a matter of minutes, new contractors can be discovered continuously, and the organizations’ routines can be automatically updated to reflect new forms of cooperation. All this provides an agility in the implementation of business processes that will allow for new business visions such as the e-business models described by Timmer (1998) or the Real-Time Infrastructure (RTI) envisioned by Gartner’s business analysts (Gartner, 2004a).

The Semantic Web provides an infrastructure for software-agent-to-software-agent communication. However, organizations do not necessarily use this infrastructure to communicate internally. The organization does not “talk Semantic Web”. As mentioned by van der Aalst, ter Hofstede, and Weske (2003) organizations that use computer-assisted business management support (e.g. ERP\(^1\) or BPM\(^2\) tools) adopt the workflow paradigm to describe their work routines. As pointed out by Gartner (2004b), the organizations should not need to change their routines to gain new technological (and business) power, but it should be possible to integrate the new technology into existing routines. As a result, the integration of the usage of the Semantic Web requires the definition of a model of interaction between the internal world of the organization and the external world of the Semantic Web. In this chapter, we provide such a model, and the architecture of the specific requester agent that will implement it. We provide more information on the notion of organizational workflows in the next section.

---

\(^1\)ERP stands for Enterprise Resource Planning

\(^2\)BPM stands for Business Process Management
We then introduce our model for the integration of organizational workflows and the Semantic Web in Section 6.2 and an architecture of sButler, the central element of the model, in Section 6.3.

### 6.1 Organizational workflows

Businesses are influenced by many constantly-evolving parameters, which provide complex business representations. This complexity makes it difficult for human beings to fully understand the business consequences of their decisions.

Since the early sixties, IT has been seen as a potential source of tools to gather, represent and manage business knowledge and provide support for decision making. Today, acronyms such as ERP (enterprise resource planning), or BPM (business process management) are used to describe different categories of such tools. They are heavily used, especially in large corporations in the banking and insurance industry, but also in smaller companies and in other domains of work. Concretely, as revealed by van der Aalst et al. (2003), the business process management life-cycle has the following sequence of phases:

- The design phase: the processes are (re)designed.
- The configuration phase: a design is implemented by configuring a process aware information system (such as a workflow management system).
- The enactment phase: the operational business process is executed using the configured system.
- The diagnosis phase: the operational processes are analyzed to identify problems and to find things that can be improved. (This may lead to redesign of the process.)

In this context, Reijers and Heusinkveld (2004) and van der Aalst et al. (2003) point out that the current process management tools heavily rely on workflow technology, especially in the configuration and enactment phases.

The workflow paradigm provides a model to represent business processes. The terminology for this paradigm has been standardized by the Workflow Management Council (WFMC) (WFMC, 1993). The empirical study described by Reijers (2004) also indicates that workflow management systems tend to help significantly decrease the time required to perform business activities. The study examines 17 processes from 6 different organizations (one governmental agency, one health insurer, one regional public works department, one local municipality, one insurance intermediary, and one domiciliary care agency). The main reason for a decrease in time for these processes is that workflow management systems typically decrease the need for coordination. As mentioned by van der Aalst et al. (2003), one strength of workflows is the existence of some formal models such as Petri nets and workflow nets (van der Aalst, 1998) for workflows. They allow for the thorough, theoretically grounded and unambiguous analysis
that is greatly needed for decision making in the complex business domain. Finally, based on the experience of its members (developers and users of workflow management systems) and the analysis of many case studies (a sample of ca 30 of which are published in Workflow Case Studies, 2001-2004), the WfMC has identified the following key benefits of using workflows:

- Improved efficiency - the automation of many business processes results in the elimination of many unnecessary steps.
- Better process control - improved management of business processes achieved through standardizing working methods and the availability of audit trails.
- Improved customer service - consistency in the processes leads to greater predictability on levels of response to customers.
- Flexibility - software control over processes enables their redesign in line with changing business needs.
- Business process improvement - a focus on business processes leads to their streamlining and simplification.

When it comes to the representation of workflows, Hollingsworth (1995) specifies in the Workflow Reference Model that a Workflow (or Business Process) is composed of tasks (or activities), that are performed by Workflow participants that may be grouped in organizational roles. An organizational role defines the set of capabilities that each of its participants must have.

As specified by WfMC (1999), a workflow is defined and created, and its execution managed by a Workflow Management System (WfMS) through the use of software run on one or more workflow engines. Each workflow engine is able to interpret the process definition, interact with workflow participants and, where required, invoke the use of IT tools and applications. Once a workflow is defined, it may be executed. This is done by the workflow enactment service, which generates a process instance and controls its enactment. A process instance is the representation of a single enactment of a process. As such, a process instance uses its own input and output data. Operations performed during the process instance enactment may be executed by external applications, i.e. applications that are not part of the WfMS. As a result, communication between the WfMS and these applications takes place. The WfMC defines an invoked application interface as a standard format for this communication. The interface specifies which information may be passed on to the invoked applications, such as the attributes of the tasks that are to be executed (e.g. the preconditions of the tasks).

Figure 6.1 illustrates the representation of a simple workflow as a graph of five tasks. This workflow representation focuses on showing the order in which the 5 tasks must happen. Task2 and task3 can be executed in parallel. It is possible to represent many kinds of process patterns with workflow languages, such as loops and conditional subprocesses. Some workflow languages also allow the specification of the information that is passed from one task to the next.
6.2 A Model for the integration of organizational workflows and the Semantic Web

In this section, we define a global model for integrating the Semantic Web into the work routines of an organization. To do this, we must define a model for the organization’s work routines, a model for the Semantic Web, and the actual integration. We adopt workflows as a model for organizations’ work routines, and Semantic Web services as a model for the Semantic Web. Integration is achieved using an sButler agent that mediates the interactions between the workflows and the Semantic Web.

We choose workflows because this is a well-established method for describing organizations’ routines. Concretely, we adopt the WfMC definition of workflows (see Section 6.1). Such workflows are composed of tasks, and the process instance generation and enactment may be delegated to an external application. In this case, communications, such as the request to generate a process instance for a specific task, take place through the well-defined invoked application interface.

To adopt a representation of the Semantic Web, we consider the needs of the organizations. As mentioned in the introduction of this chapter, the organizations need to use the Semantic Web to find and access distant contractors in a dynamic manner. The Semantic Web service approach provides for such a need by striving to describe the Semantic Web as a source of distant contractors modeled as Semantic Web services. We further assume in our model that communication in the Semantic Web is achieved through the use of messages whose content is written in a Semantic Web language, typically a flavor of OWL (2004).
6.2. A Model for the integration of organizational workflows and the Semantic Web

In our model, the integration between an organization’s routines and the Semantic Web is performed by a dedicated sButler agent. The sButler agent mediates between the organization’s WFMS and the Semantic Web service architecture. This need for mediation when dealing with service retrieval has been highlighted by both the Semantic Web and agent communities. Fensel and Bussler (2002) describe a framework for the development and management of Web services that highlights the need for different kinds of mediations including data format and process semantics mediations. By defining the notions of Contract net (2002b) and Iterated contract net (2002c) the FIPA defines its own negotiation protocol between agents.

In this context, the sButler has to handle several workflow management-related activities. One of them is the process instance generation of the parts of the workflow whose enactment is delegated to external contractors. Another one is the enactment of these process instances. A third is the management of the experience accumulated from using the Semantic Web (e.g. association of trust values to service providers).

Concretely, the sButler is an invoked application that is called by the WFMS to generate and manage process instances composed of Web services for specific tasks. These tasks are known internally to the organization as tasks performed by external contractors. As a result, in the workflow representation, their organizational role is set to external contractor. The sButler communicates further with the WFMS through the invoked application interface. Communication with the Semantic Web is achieved by submitting queries for Web services to the network. Depending on the concrete architecture of the network (i.e. peer-to-peer or client-server), the query is sent to a specific registry node or broadcast on the network to discover the nodes that provide Web services that match the query.

In its role of mediator, the sButler supports the retrieval of Web services that match the organization’s need. Therefore, the sButler needs to ensure that the discovery algorithms receive optimal input information. For this purpose, the sButler must define an internal representation of the tasks that strives to capture all the aspects of the task that are important for matching Web service representations. These aspects may include such things as the kind of transaction the task requires and the kind of result the task is expected to produce. As a result, the sButler performs a two-step translation from the organization’s representation of a task to its own internal representation and from there to the Web service representations. As Web services may be represented using different languages, different translation schemes may be needed.

Additionally, the management needs of WFMS define a number of situations where the sButler has to mediate between the WFMS and the Semantic Web. As a result, an sButler agent must provide the following capabilities:

**Process instance generation.** Given a task description, generate a process instance composed of services available on the Web. In many cases, this

---

3As we focus on the integration of the Semantic Web into an organization’s workflow, we assume in this thesis that external contractors are always related to Web services.
corresponds to retrieval of a composition of services. However, some organizations require the specification of formal business contracts with all their service partners. In these cases, the sButler also has to support possible negotiations between the organization and the distant service provider in order to further specify business contracts.

**Process instance enactment.** Support the enactment of the process instances composed of Web services. In the cases where a business contract has been established, this includes ensuring that the terms of the contract are followed and supporting the tasks that should be performed when they are not (e.g. unexpected delivery delays occur).

Besides providing capabilities that satisfy the needs of the WfMS, the sButler must take into account the characteristics of the Semantic Web and the new possibilities that it provides. One such characteristic is its dynamism. The resources available on the Web may appear, disappear and be updated in an asynchronous manner with respect to other resources and to any possible users. As a result, the sButler must be able to manage the changes that may occur asynchronously with process instance generation or enactment. These changes may occur either in the organization’s internal world (providing for more dynamic business processes) or in the Web service functionalities or settings.

The sButler must also exploit the repetitive character of an organization’s routines to learn from the experience of previous process instance generation and enactment, in order to speed up and adapt its capabilities to provide better performance. For example, in some cases the same composition of services that has proved to have high performance can be used to direct process instance generation by limiting the search domain.

In summary, the sButler may be seen as the Semantic Web interface to the organization. In practice, the sButler may be implemented either as an application that is installed and runs on the organization’s platform or as a Web service that is called by the organization when in need of a mediator with the Semantic Web.

Figure 6.2 illustrates our model. In the figure, an organization’s routine is represented by a workflow of five tasks managed by a WfMS. One of these tasks (task3), must be delegated to external contractors. As a consequence, when a process instance for this task must be generated, the generation is handed over to the sButler (step 1 in the figure). The sButler then translates the task description into a query for Web services, submits it to the Semantic Web (step 2), and uses the reply to generate a process instance for the task. The process instance is then returned to the WfMS, which uses it according to its management scheme (step 3). For example, the WfMS may expand the workflow by associating the process instance to the original task as sub-workflow. Later, the enactment and management of the process instance will also require delegation to the sButler. The sButler will again interact with the Semantic Web and return the results to the WfMS.

In the next section, we focus on providing a solution to the first capability of the sButler introduced above: process instance generation. From a Semantic
6.3 sButler architecture

Web point of view, process instance generation requires that service discovery be performed. Referring to the service discovery model provided by the service discovery operation support of the platform (see Table 5.1 in Chapter 5), the sButler is a specific category of Requester Agent.

6.3 sButler architecture

In this section, we describe an architecture for the sButler agent. We focus on the first capability of the sButler introduced in the previous section, namely the process instance generation\(^4\) for a given workflow task. This includes the generation of queries to the Semantic Web based on the given workflow task, the actual querying to find relevant (compositions of) Web services, the generation of process instances that satisfy the requirements of the workflow task, the ranking of the process instances with respect to user and organization preferences, and the integration of the process instances into the organization’s workflow.

To generate a process instance for a given task, the sButler requires a task description as input. The sButler also uses different kinds of knowledge. Some of this knowledge is represented in ontologies shared by the organization and distant contractors. Other knowledge may be organization specific and stored locally in knowledge bases. We have identified the following kinds of useful knowledge:

\(^4\)We use the term process instance instead of task instance because of the nature of the matching mechanism: Typically the task will require a composition of Web services, i.e. a process composed of several activities.
• **Domain knowledge** provides the vocabulary used by the organization to describe its work and routines. Typically, this domain knowledge describes the different input and output types of the workflow tasks. In practice, this knowledge is seldom built from scratch — rather, existing ontologies are used or extended. Examples are ontologies about products, documents and activities.

• **Transaction knowledge** provides the vocabulary used to describe the different Web transactions that may be established between organizations and Web service providers. This knowledge is represented in shared ontologies. For instance, the MIT process handbook (2003) defines transactions such as the Acquire activity and its specializations.

• **Process knowledge** provides the vocabulary for describing constraints on Web service processes. The MIT process handbook (2003) is a source of information for such knowledge. Also, the quality of service definitions provided by the WfMC in (WF-XML, 2000) and the METEOR project in (Cardoso & Sheth, 2003) are specific cases of process knowledge.

• **Decomposition knowledge** defines the concepts that may be used to decompose queries and provides explicit rules for decomposition and re-assembly. Some of this knowledge may already be represented more or less explicitly in Domain knowledge. Additional decomposition rules (e.g. based on decomposition heuristics) may also be included.

• **User and organization preference knowledge** specifies preferences of the users and organizations. This knowledge is typically expressed in the terms of Domain, Transaction and Process knowledge.

• **Query knowledge** provides experience data from previous generation and management of queries for Web services. This knowledge typically refers to specific workflow tasks.

The sButler aims to represent all the aspects of the task that are important for matching Web service representations. Our internal task model requires the following aspects: First, the required result needs to be represented. This may be, for instance, a concrete object, an action, or information about an object or an action. We use Domain Knowledge to represent this information. Additionally, we represent the kind of process that is performed to create the desired result. This is described using Process knowledge. Finally, another important aspect is the type of transaction the task requires. This may include such things as the delivery mode and the kinds of ownerships that are transferred (e.g. renting vs. buying). This is described using Transaction knowledge. These aspects focus on what the organization requires from the task. There may be other aspects that are relevant and focus on other areas such as the location of the Web services or technical requirements for communication with the Semantic Web. These are beyond the scope of this thesis.
Query Generation is the process that supports the generation of a query for Web services and is responsible for the two-step translation performed by the sButler. During the first translation step, the sButler uses the task description provided by the organization to create an internal representation of the task. It uses the organization’s Domain knowledge as well as Transaction, Process knowledge, and Query knowledge. The sButler may also interact with the organization to obtain as much relevant information as possible. The second step translates the internal representation into Web service representations as required by the existing discovery algorithms. The result is a query for Web services.

Task Decomposition is the process of locally identifying possible decompositions of the task. One possible approach consists of decomposing the desired object and building a new query for each part of the object. In this case, some Decomposition Knowledge is required. This module also uses Query knowledge.

Service Discovery is the actual querying for Web services. Queries for services that perform part of the task (in the case of decomposable tasks) or the whole task are submitted to service discovery agents available on
the Semantic Web. The result is a set of candidate Web services for each query submitted.

**Instance Generation** is the process of putting the parts together to build process instances that produce the whole desired object. This is done using *Decomposition Knowledge*. The result is a set of candidate process instances.

**Instance Ranking** consists of ranking the process instances according to some criteria. These criteria are typically represented in a *User and organization preference* knowledge base.

**Instance Selection** may be done in two modes. In the automatic mode, one of the highest ranked instances is automatically selected as the process instance for the task. In the manual mode, the user chooses a process instance based on an ordered list of process instance descriptions presented by the sButler. In both cases, the sButler returns the process instance to the WfMS which uses it according to its management scheme. Typically, when the time comes to execute the process instance, the WfMS will delegate the enactment to the sButler.

**Evaluation** aims at improving the performance of process instance generation by using previous experience. For instance, many process instances may be generated for the same task and information about the selection can be stored. This experience may be used the next time the user requires similar information. Also, during service discovery, knowledge may be gained, e.g. some Web service registries may systematically take too long to answer. All of this accumulated knowledge is stored in *Query knowledge*.

The next chapter discusses the problem of query generation for service retrieval and introduces the OWL-DTP language in order to fulfill the sButler’s architecture requirement of including *Domain, Transaction, and Process knowledge* to describe queries and services. Further on, in Chapter 8, we provide an implementation of sButler using OWL-DTP. This implementation is then evaluated with the simulation and evaluation platform in Chapter 9.
Chapter 7

OWL-DTP

In order to provide an implementation of the sButler’s architecture introduced in the previous chapter, the problem of query generation must be examined. This is the purpose of this chapter. We describe the problem and define OWL-DTP, an alternative Semantic Web service description language. We then go on to compare OWL-DTP with OWL-S and WSMO, which are the two other most common approaches to Semantic Web service description.

7.1 The problem of query generation for service retrieval

Baeza-Yates and Ribeiro-Neto (1999a) describe the Web information retrieval process as including both the operation of query generation (i.e. specification of a user need + execution of query operations) and the query processing that provides the information relevant to the query. We consider service retrieval to be the retrieval process for services on the Semantic Web. As such, the process of service retrieval includes both the operation of query generation and query processing.

Current research related to service retrieval mainly focuses on service discovery. There are three aspects of service discovery as follows:

1. Design of description formats for Web services such as WSDL (2004c), OWL-S (2005) and WSMO (2004).


3. Design of Web service matchmaking algorithms, e.g. OWL-S Matchmaker (2002). These algorithms tend to adopt the same format for representing available Web services and the queries for such Web services.

Typical operations that may be performed during query processing are query validation, optimization, submission to query engines, and results ranking.
These are all aspects of query processing, but they do not address the issue of query generation.

The current Web service description formats on which all the three aspects are based do not support query generation. These formats are designed to describe Web services, not queries. Also, they are designed with respect to the data format requirements of the composition algorithms. For example, both OWL-S and the Web service ontology of WSMO describe services with the notions of inputs, outputs, preconditions and effects (IOPEs). IOPEs are typical data formats used by planning technology (i.e. the main technology used by current Web service composition algorithms.) The capability of the Web service description formats to represent the real needs of the requester has been overlooked. As a result, the queries for services written according to existing formats may not be able to describe the complete and real need of the requester, but describe instead an approximation of this need. This is an especially important issue for the Semantic Web, which, as mentioned in Chapter 2, is meant to reduce the level of approximation traditionally associated with Web retrieval.

In this section, we analyze the problem of representing the real need of the requester. This work is required to design a good query generation module for our sButler architecture, introduced in the previous chapter.

As mentioned by Baeza-Yates and Ribeiro-Neto (1999a), before the retrieval process can even be initiated, it is necessary to define the set of objects to be retrieved. For unstructured objects like text documents and images, this implies the description of the object’s logical view (i.e. the representation of the object to the retrieval system). Typically, such a logical view is modeled with a set of aspects (or features). This set of aspects provides all the information about the object that the retrieval system is able to use. This means that if a requester writes a query that refers to aspects of an object that are not described by the logical view, then this part of the query cannot be taken into account during the retrieval process. In other words, we can say that the model of the logical view for services provides the vocabulary available to the requester for expressing its needs in terms of services. The capability of a retrieval engine to provide requesters with objects that match their need is typically measured in terms of precision and recall\(^2\) (Baeza-Yates & Ribeiro-Neto, 1999a). As a result, to provide service retrieval with good precision and recall, a definition of the model that will represent services must take into account the requester’s view of the service.

On the Semantic Web, and according to the W3C (1994-2006), the logical view of a service is called a Web service. It is a file written in the RDF syntax, which is the current mandatory Semantic Web syntax. There are currently two popular Web service description languages: WSDL (2004c) and OWL-S (2005). WSDL is a W3C standard and OWL-S is being evaluated as a potential W3C recommendation. As mentioned earlier in Chapter 2, WSDL specifies the format to call the services. The semantical description is very limited. OWL-S extends

\(^2\)Precision is the fraction of the retrieved objects that are actually relevant to the user’s need, while Recall is the fraction of the relevant objects that have been retrieved from all the relevant objects that actually exist.
WSDL to provide the information necessary for the composition of services (e.g. preconditions and effects) and the management of inter-organizational service compositions (i.e. the description of the service in terms of a composition of other atomic and composite services).

Current approaches to interfacing workflow systems and the Semantic Web using OWL-S (e.g. Cardoso and Sheth (2003)) reduce the description of workflow tasks to sets of inputs, outputs, preconditions and effects (commonly referred to as IOPEs). Such task descriptions do not consider the definition of the process that the task is meant to perform, i.e. what the different steps of the process are actually doing, and in which order. This leads to unsatisfying retrieval for several reasons. One reason is that the process definition contains some important semantics that the IOPEs do not capture. For example, the processes of both a car dealer and a car manufacturer may require a car description and a payment mean as input while providing a set of cars as output. In this context, a requester who wants to find car dealers will not be able to distinguish between them and car manufacturers. Another reason for unsatisfying retrieval is that the organization likely has incomplete or biased knowledge of what is necessary for a process instance to be enacted by another organization. As a result, the organization’s description of preconditions and effects is approximate with respect to the view of the service providers. For example, when looking for a car, requesters may not directly provide information on the means of payment they plan to use, even if it is a required input from the car dealer’s point of view. A third reason is that a delegated task is a specific category of task whose usage must be integrated into an organization’s routine through the establishment of a business transaction. The transaction may require the establishment of a contract, or for both the requester and provider to follow a specific protocol of communications, or a transfer of ownership on information or an object. The description of these aspects is meaningful both for the provider and the requester and, as such, should be taken into account in the queries and service descriptions.

To sum up, the commonly adopted IOPE view is not always able to provide a complete description of requesters’ needs and providers’ services. Moreover, both requesters and providers need support to formulate their needs and services in an unambiguous manner. More generally, a Semantic Web service description language must be capable of representing Semantic Web services and queries in an expressive and unambiguous way. We have identified the following requirements as being necessary in order to achieve this:

**R1** The language must be capable of expressing all the relevant aspects of a (provided or required) service.

**R2** There must be a semantic definition of each aspect that is well-defined to avoid ambiguous use of the language.

**R3** There must be practical means to enforce that the language constructs are used in a manner that is uniform and according to their intended use.
7.2 The DTP logical view

To address the problem of query and service representation described above, we propose a holistic logical view of a service composed of the following three aspects.

**Desired result** is the resource that the requesters need to acquire through the use of a distant service. The resource may be a physical object (e.g. a car, a book), a service (e.g. pay bills), or information (e.g. the phone number of the nearest doctor).

**Transaction** describes a business transaction with a distant contractor. The description of the transaction may include the description of the process of the transaction, and constraints on this process such as quality of service constraints.

**Process constraints** describe the process performed by the provider to produce the desired result. While the specific description of the process may be known only to the service provider, the requester may still want to set some constraints with respect to ethics, law, quality of service, etc.

Constraint specifications on both the transaction and on the process typically include the quality of service requirement as specified by the WfMC in (WF-XML, 2000) and the developers of the METEOR workflow management system (Anyanwu, Sheth, Cardoso, Miller, & Kochut, 2003).

In the rest of this thesis, we will refer to this logical view as the DTP logical view where D stands for Desired result, T for Transaction, and P for Process constraints.

The DTP logical view is not only describing queries for services, but also the different aspects of a service on which a query for services can be formulated. However, to ensure optimal performance, it is not sufficient to provide tools for describing all necessary aspects of the Web service. Some user support that ensures the proper use of these tools is also necessary. To be more specific about what proper use is, we refer to the notion of descriptive power of a model or language. This notion should not be mixed up with the notion of expressive power. Descriptive power corresponds to the different aspects available to describe a meaning. As a result, two languages with the same expressive power may have different descriptive powers. For the user of the language, a greater descriptive power provides for well-formed semantics for specific domains. An example of such a domain is the domain of services. We illustrate the importance of using the full descriptive power of the DTP logical view with the following example. A requester specifies that she wants to rent a car, while Web service descriptions provide only information about the objects produced (e.g. cars) and not about the category of the service (e.g. renting or selling). As a result, part of the requester’s query is left unused and the precision of the retrieval

---

*The expressive power of a language is the power to express relations and derivations of a language with the operators of the language.*
will suffer. In this example, the requester uses a language with more descriptive power than the language of the Web service provider. This results in a situation where the full descriptive power of the requester’s query cannot be exploited. This is a typical retrieval problem (Baeza-Yates & Ribeiro-Neto, 1999b) that is solved by providing the means to enforce the use of the full descriptive power of the vocabulary. We propose a set of guidelines for describing queries and Web services in a uniform manner as follows:

- **Domain knowledge** must specify objects by referring to their standard definition. Projects like CYC (1994-2006) show how difficult and time-consuming it is to build up one unique consistent body of knowledge for describing all domains. A more promising approach that already shows signs of success is based on the assumption that each requester or service provider can build or reuse a number of knowledge sources that describe the things they use everyday. In some cases, these knowledge sources can even be based on some standard knowledge such as the definition of DAML-Time (Hobbs, 2003), which strives to express temporal concepts and properties common to any formalization of time. In such a case, the use of ontology-merging algorithms to check for the consistency of newly created knowledge with respect to existing knowledge would allow for consistently building up one large and diverse body of knowledge. This knowledge would not be static, but relationships between concepts could appear, disappear, and be updated according to the changing needs of its users.

- **Transaction knowledge** must specify a transaction definition that is available and used by at least a part of the Semantic Web community. Ideally, there is one unique source of **Transaction knowledge**.

- **Process constraints** express rules about **Process knowledge**. Ideally, there is one unique source of **Process knowledge**.

In Section 7.4.4 we show how to use the MIT process handbook (2003) as a source of knowledge for defining each of these three knowledge sources.

In order to further enforce the use of the guidelines on the correct use of the DTP logic view, we also suggest the development of an editing tool for service descriptions and queries. However, the design and construction of such a tool is outside the scope of this thesis.

In the next section we provide a detailed description of the DTP language extension. The DTP language extension allows extension of a category of knowledge representation (KR) languages to represent queries and services according to the three aspects of **Desired result**, **Transaction**, and **Process constraints** introduced above. The category of KR languages considered are the languages that allow for the representation of concepts, relations and is-a hierarchies.
7.3 Definition of the DTP language extension

The DTP language extension describes services with the three aspects introduced above: Desired result, Transaction, and Process constraints. The definition of the language extension relies on three parts:

1. **The conceptualization** of the notions of query and service

   The DTP language extension defines one unique concept to describe both queries and services. This concept is called `WebService` and has three predefined properties:

   - `hasDesiredResult` whose range is the concept `Resource`. A `WebService` has one or several fillers for this property.
   - `hasTransaction` whose range is the concept `Transaction`. A `WebService` has exactly one filler for this property.
   - `hasProcessConstraint` whose range is the concept `Process`. A `WebService` has exactly one filler for this property.

2. **The ontologies** that make the above definitions concrete and provide some standard vocabulary to the user

   There are three ontologies: the resource, transaction, and process ontologies which define the notions of `Resource`, `Transaction` and `Process`. Section 7.4 further discusses the design of the `Resource`, `Transaction`, and `Process` ontologies and proposes some candidate ontologies that include some of the knowledge gathered in the MIT process handbook (2003).

3. **The definition of the algorithm to match a query and a service description**

   A DTP query \( Q \) and a DTP service \( S \) match iff

   - (a) for all desired results \( D_q \) of the query \( Q \), there exists a desired result \( D_s \) of the service \( S \) such that \( D_q \ matches_d D_s \), and
   - (b) the transaction \( T_q \) of the query \( Q \) and the transaction \( T_s \) of the service \( S \) are such that \( T_q \ matches_t T_s \), and
   - (c) the process constraint \( P_q \) of the query \( Q \) and the process constraint \( P_s \) of the service \( S \) are such that \( P_q \ matches_p P_s \).

   Section 7.5 further discusses the different approaches to matchmaking and proposes a matchmaking algorithm for languages adopting the DTP language extension.

   The next section discusses the design of the `Resource`, `Transaction`, and `Process` ontologies.
7.4 Ontologies for the DTP language extension

As mentioned in the previous section, the DTP language extension requires the existence of three ontologies to describe Resource, Transaction and Process concepts. In this section we show how the MIT process handbook (2003) can be used as a source of such knowledge. We proceed as follows: In Section 7.4.1, we describe the MIT process handbook. In Section 7.4.2 we define a conceptual structure for representing the knowledge provided by the MIT process handbook with an ontology that is suited to the needs of the DTP language extension. In Section 7.4.3, we explain how to represent the knowledge constraints that are needed to describe queries and Web services. In Section 7.4.4 we describe how to use the structured MIT process handbook knowledge to represent Resource, Transaction and Process ontologies. Finally, in Section 7.4.5 we show how this knowledge enables the description of queries and Web services.

7.4.1 The MIT process handbook as a source of knowledge

The Semantic Web implies the communication from software agent to software agent through machine-understandable messages. The current approach is to develop languages with RDF syntax, with good enough descriptive power to allow for automatic resource discovery. For the retrieval of services, we have argued that there is a need for a language with the capability to describe requirements for the aspects associated with using a service over a network: resources, transaction and process. We propose a language that provides three ontologies to define the concepts of Resource, Transaction and Process.

The MIT process handbook provides a very exhaustive, very thorough and structured source of knowledge about these three concepts. The result of the analysis of a large sample of real business processes, the MIT process handbook provides a large and organized collection of vocabulary for describing processes (or activities). In its academic version, the handbook also provides a basic categorization of resources. This process handbook is not a static source of knowledge. It is continually updated and growing to capture new business trends, including electronic ones. The MIT process handbook currently provides eight generic categories of activities: Create, Combine, Modify, Manage, Separate, Preserve, Decide, and Destroy. These categories describe the possible ways of providing a desired result. Therefore, they provide some vocabulary to describe service processes, including transactions over a network. Moreover, the academic version of the MIT process handbook also provides a hierarchical description of the resources that the activities may use and produce. The most generic resource categories are Location, Actor, Informational entity, and Physical entity. Such resource description can be used to categorize desired results.

The MIT process handbook provides information about business processes, where each business process is defined as an activity and the activities are organized into two hierarchies.

The first hierarchy is an is-a hierarchy of concepts. Each of these con-
cepts represents an activity. Examples of activities are Buy, Buy information resource, Buy over Internet, or Buy in electronic store using posted prices. Each child of an Activity concept is a specialization of this Activity. For example, the activities Buy information resource, Buy over Internet and Buy in electronic store using posted prices are child activities of the Buy activity. The handbook acknowledges a specific case of specialization corresponding to the association of a value to a specific criterion. Such a criterion is used to specialize the Buy activity with respect to the category of the thing bought. Examples of specializations are Buy information resource and Buy physical resource. The group of specializations that correspond to different values of the same criterion is called a bundle. Each bundle is labeled with a name. For example, the bundle of specializations made with respect to the category of the thing bought is called the Buy what bundle. The name of the bundle is called a bundle criterion. Buy using what pricing mechanism is another example of bundle criterion that labels the bundle composed of the following activities:

- Buy using bulk purchasing network
- Buy in electronic store using lottery
- Buy in electronic store using auction
- Buy in electronic store using posted prices

Figure 7.1 illustrates the is-a hierarchy for three activities of the MIT process handbook defining two is-a relationships as follows:

- Buy in electronic store is-a Buy over internet
- Buy in electronic store using posted prices is-a Buy in electronic store

The second specialization is based on the specification of a value for a bundle criterion while the first one is not. The bundle criterion involved in the second specialization is the Buy using what pricing mechanism introduced above.

The second hierarchy provided by the MIT process handbook is a part-of hierarchy. This hierarchy associates an activity with the sub-activities of which it is composed. In this context, an activity $a_i$ is said to be part-of an activity $a_j$ if $a_i$ participates in the composition of activities that make up the process of $a_j$. In the current state of the MIT process handbook, the structure of this composition is always a sequence. However, we learn from the workflow community’s work that process modeling often requires a more complex organization of the activities (van der Aalst & ter Hofstede, 2002). Thus, we foresee that the capability to describe more complex compositions will be required. Figure 7.2 illustrates the part-of hierarchy for the Buy activity, which may be decomposed into a sequence of seven other activities. Notice that these activities may themselves be specialized and/or be decomposed into a composition of other activities.
7.4. Ontologies for the DTP language extension

Figure 7.1: Example of an *is-a* hierarchy between activities of the MIT process handbook

Figure 7.2: Example of a *part-of* hierarchy between activities of the MIT process handbook
In the next section we propose a structure for the MIT process handbook as a means to use the process handbook as a source of knowledge to describe queries and Web services.

### 7.4.2 A conceptual structure for the MIT process handbook

As stated in Section 7.4.1, we want to use the process handbook as a source of knowledge for the resource, transactions and process ontology required by the DTP language extension. By providing a conceptualization of the notions of resource and business process, the process handbook is an ontology. However, some notions, like the bundle criterion, are not fully structured. In this section we propose a complementary structure for the business process knowledge of the MIT process handbook.

In order to represent the notion of bundle criterion, we introduce the notion of domain for a bundle criterion. An example of a domain for the Buy using what pricing mechanism bundle criterion introduced in the previous section, is the disjunction of the following values:

- bulk purchasing network
- lottery
- auction
- posted prices

Each of these values is used in the MIT process handbook to define each specialization of the Buy activity that is conducted according to the Buy using what pricing mechanism criterion introduced in the previous section.

We define the concept of **Bundle criterion** with the following properties:

- **Activity** The name of the activity that may be specialized with this bundle criterion.
- **Name** The name of a bundle criterion for this activity. It is a string. (Note: Activity + Name = unique identifier of the bundle criterion concept.)
- **Domain** The domain of the criterion. It may be either a concept name or a disjunction of concept names.
- **Description** The natural language description of the criterion. It is a string.

Figure 7.3 shows how the Buy using what pricing mechanism bundle criterion is represented by our proposed structure. Using the UML formalism, the figure provides a **Bundle criterion** class and its **Buy using what pricing mechanism** instance. The instance has a domain attribute that refers to the **Pricing mechanism** concept. The **Pricing mechanism** concept is a disjunction of four concepts: bulk purchasing network, lottery, auction, posted prices.
As stated in Section 7.4.1, some activities are specialized according to a bundle criterion. Such a specialization corresponds to selecting one value from the domain of the bundle criterion. We define the concept of Bundle criterion specialization with the following properties:

**Criterion** The criterion that is specialized. It is a Bundle criterion concept as defined above.

**Domain** The specialized domain of the criterion. This is a subdomain of the criterion’s domain. It may be either a concept name or a disjunction of concept names.

Figure 7.4 shows how our structure represents the bundle criterion specialization that restricts the domain of the Buy using what pricing mechanism bundle criterion to the unique concept posted price. Using the UML formalism, the figure provides a Bundle criterion specialization class and its #BCS0 instance. The instance has a criterion attribute that refers to the Buy using what pricing mechanism bundle criterion, and a domain attribute that is set to the unique concept posted price.

Finally, we structure the process handbook as an ontology of activity concepts such that, given the set $A$ of all activity names in the MIT process handbook, the ontology describes Activity concepts with the following properties:

**Name** The name of the activity such that the name belongs to $A$ and no other concept has the same name.

**Description** The natural language description of the activity.
Participants The set of activities which compose the current activity. This is a set of activity concepts.

Plan The plan that tells how these activities are organized. Depending on the application, one may want to use YAWL (van der Aalst & ter Hofstede, 2002) or the process part of OWL-S (2005), or BPEL4WS (2002)

Table 7.1: Properties of the Plan concept

Bundle criterion specializations The set of bundle criterion specializations that the activity implements locally. This is a set of Bundle criterion specialization concepts. The set may be empty.

Parents The set of parent activities. This is a set of activity concepts. This property allows for the representation of the is-a hierarchy of the MIT process handbook.

Direct-decomposition The activities that compose the current activity and the organization of these activities. This is a Plan concept whose properties are described in Table 7.1. This property allows for the representation of the part-of hierarchy of the MIT process handbook.

Figure 7.5 shows how our structure represents Buy in electronic store using posted prices activity. This activity is described with:

---

4If some parent activities are also specialized according to one or several criteria, the bundle criterion specializations are stored by these parents.
Ontologies for the DTP language extension

Figure 7.5: Example UML model of an Activity
A parent: the **Buy over internet** activity. We assume that this concept, as well as its own possible parents, bundle criterion specializations, and direct decomposition are already defined.

- The bundle criterion specialization **#BCS0** as introduced in Figure 7.4 above.

- A Direct-Decomposition that provides a set of participant activities and a plan as illustrated in Figure 7.6.

This proposed conceptual structure for the MIT process handbook provides an ontology that is thus composed of a hierarchy of **Activity** concepts, a set of **Bundle criterion** concepts, a set of **Bundle criterion specializations** concepts and a set of **Plan** concepts. Further, **Activity** concepts may refer to **Bundle criterion specialization** concepts which, in turn, refer to **Bundle criterion** concepts.

Moreover, the domains of the **Bundle criterion** and **Bundle criterion specialization** concepts may be described either with a set of values, or an existing concept, or a disjunction of concepts. The concepts used to describe the domains either correspond to notions defined by the MIT process handbook,
or to concepts existing in some other domain knowledge resource. Pricing mechanism is an example of concepts defined by the MIT process. This is a notion derived from the Buy using what pricing mechanism criterion.

Notice that the proposed ontology structure takes into account only the structured knowledge available in the MIT process handbook. Klein and Bernstein (2001) point out that the process handbook should allow processes to be associated with properties that capture information such as “typical performance values (e.g. how long a process takes to execute), as well as pre-, post- and during conditions”. The academic version of the process handbook provides the slots to fill in this information. However, neither the public nor academic versions of the handbook provide values for these slots. Some of this information is available in an unstructured manner (i.e. natural language formulation) in some of the activities’ description property. As a result, we can foresee that our proposed structure for representing activity concepts may need to be augmented with a larger set of properties that would be able to capture this information.

Now that we have a conceptual structure for the MIT process handbook, we can build a new ontology of activities, bundle criteria, and bundle criterion specializations, that describe the vocabulary provided by the MIT process handbook.

In next section, we provide some means of expressing restrictions on the Activity concepts. This is necessary to be able to use the ontology as the vocabulary for describing queries and Web services.

### 7.4.3 Specifying constraints on Activity concepts

We want to describe aspects of queries and Web services. A query may be expressed on different levels of granularity. For example, the requester may want to Acquire something, and does not care about the details of the transaction, while another requester may want to acquire by auction. As a result, a means of specifying restrictions on activity properties, such as intervals of acceptable values, is required. Such restrictions can be specified on the Activity concepts of our representation of the MIT process handbook by specifying constraints on their properties. We identify three categories of constraints that need to be expressed as follows:

**Hierarchical constraints** specify some ancestors and descendants of a concept. E.g. “the activity must be among the descendants of the Get activity.”

**Domain constraints** constrain a domain. Example of domains that may be constrained are the domains of bundle criteria or bundle criterion specializations (see the previous section). This is done by specifying the new domain in terms of a concept C which is subsumed by the original domain. E.g. the criterion is a bundle criterion specialization #BCS0 (defined in Figure 7.4) with the Domain constraint, Domain = {posted prices} while
the original domain is \{bulk purchasing network, lottery, auction, posted prices\}.

**Plan constraints** are placed on the characteristics of a plan. They may set the following restrictions:

- that one specific activity must belong to the plan, or not.
- that the structure of the plan must follow some specific organizational rule (e.g. activity \(a_i\) comes after activity \(a_j\)).

The first two categories of constraints are expressed by defining a sub-concept of the original concept with restrictions on the domains of some of its properties. As a result, a concept that is more strongly constrained will be a specialization of a concept that is less strongly constrained.

### 7.4.4 Using the MIT process handbook as a knowledge resource on business processes

We have claimed that the MIT process handbook provides a good base for generating the **Transaction** and **Process** ontologies required by the DTP language extension. The MIT process handbook also provides some important grounds for describing the **Desired result** of the DTP language extension. The following summarizes our proposed use of the MIT process handbook as a knowledge resource for building the ontologies used by the DTP language extension.

**Resource ontology**

As mentioned in Section 7.3, the **Desired result** describes the result that the requester expects from using the service. This may be a concrete object, a change of state (like a bank deposit) or a set of these.

The academic version of the MIT process handbook provides a basic categorization of resources. Examples of categories of resources are **Location**, **Actor**, **informational entity** (such as **service information** or **contract**), or **physical entity** (such as **money** or **mechanical device**). Typically, the desired results are specializations of these resources. For example, a car is a **physical entity** and the contact information of a car dealer is an **informational entity**. Such resource category information is often overlooked by domain ontology writers and can result in ambiguous concept definitions. For example, a car ontology may describe the concept **Car** with different properties such as the **brand**, the **year** of construction and the **color**. However, when it comes to interpreting the query **get {red Clio from 1998}**, there is an ambiguity about the kind of resource required. Should the result be a picture of the Clio, an instance in the ontology, or an actual drivable car? The resource categorization of the MIT process handbook provides the complementary knowledge that allows a domain ontology to be put in the right context.
7.4. Ontologies for the DTP language extension

**Transaction ontology**

A transaction describes the category of business processes to be conducted between two actors: the service provider and the requester.

The Transaction ontology provides requesters (resp. service providers) with the vocabulary to describe the kind of transaction(s) they agree to participate in (resp. are able to support). From the requester’s point of view, the transaction vocabulary that is put at their disposal is also a way of knowing what possible transactions actually exist, as well as the alternative existing solutions that they are perhaps not aware of.

There have been several attempts to model transaction knowledge. Both CEL (2004) and (Neal et al., 2003) provide languages for modeling contracts that mix our notions of transaction and process. However, these approaches do not provide exhaustive descriptions of valid domains for the different components of the contract like the MIT process handbook does. Concretely, among all the business processes described in the handbook we have identified the following hierarchies of activities that describe transactions:

- Acquire corresponds to the following sequence of activities: Receive physical resource, Determine timing, Identify needs or requirements. Specializations of Acquire include Buy and Acquire not for money.

- Exchange corresponds to the following sequence of activities: Communicate information, Identify person or organization, Transfer payment, Identify options or goals, Identify needs or requirements. Transfer something. Specializations of Exchange include Buy and Barter.

We also identified some Rent and Lend activities. However, they describe the business process of rental companies and not the collaboration of requesters and providers in performing a rental. While the MIT process handbook provides wide coverage of business processes, the handbook is an ongoing process that is not completed yet. We expect the description of the rental transaction to be described in the future.

For the moment, we propose a transaction ontology composed of the set of specializations of the Acquire and Exchange activities. This includes both their parts and the bundle criteria that they refer to. Notice that the specializations of the different activities include the specializations of their parts. For example, Pay is one subactivity of Buy which may be specialized as a Pay using credit card activity. Moreover, in some cases, some bundle criteria constrain the Desired result. This is the case for the specializations of Acquire that are done according to the Acquire what bundle criterion. Buy physical entity is an example of such specialization, which puts a constraint on the type of the Desired result: it must be a physical entity. As a result, the consistency of the constraints established with the Transaction ontology must be checked against the definition of the Desired result. The Resource ontology introduced above provides a common-ground language that allows for enforcing the consistency of the description. The Transaction ontology is built according to the structure introduced in Section 7.4.1.
Process ontology

When using a distant process, the performance of the requester’s own working process may be influenced by some characteristics of the execution requirement of the distant process. There are different issues, such as technical compatibility (communication protocols used, data format adopted, etc), planning issues (e.g. the time needed for the distant process to complete the task may have a drastic influence on the user’s work process), data availability (during its execution, the distant process may require data that the user cannot provide, either because it does not exist or because it is confidential), ethics and policy issues (e.g. the resources used by the process is not to be wood from Amazonia). Some of these issues are officially identified and described as quality of services (see Wi-XML (2000) and Anyanwu et al. (2003)). Notice, however, that some of these issues may concern a specification of parts of the Transaction. Typically, the Determine timing activity that is a part of the Acquire activity, and the Identify own need to change when?, which is a specialization of a part of the Acquire activity, are two examples of planning constraint specifications that are described using the Transaction ontology.

The Process ontology supports bridging the gap between the knowledge of the requester and the knowledge of the service provider. Typically, the requester knows nothing about the internal routines of the service. As a result, the requester can only specify requirements on constraints blindly. For example, when buying furniture, the requester may require that the furniture is hand-made. Another example concerns organizations that want to do business only with companies that implement some ISO 9000 quality standard.

Summary

The MIT process handbook provides the vocabulary to describe:

- The type of some resources, e.g. human, information, physical object, raw material.
- Activities composing business processes.
- The preconditions, effect and during conditions for some process. This is currently available in a non-structured manner (i.e. natural language) in the current versions of the handbook.
- A name for the underlying business model of the provider, e.g. a provider that provides cars may be a car constructor or a car dealer. In the MIT process handbook, the car constructor’s business model is described with the Create physical asset (Manufacturer) activity while the car dealer’s business model is a Distribute physical asset (Wholesaler/retailer) activity.

To allow for the description of the different categories of process constraints introduced above, some additional mechanisms are needed:
7.4. Ontologies for the DTP language extension

- Performance evaluation techniques: Static information about performance is not always a reliable manner for evaluating a service. Some dynamic performance evaluation must be provided. Moreover, performance data provided by providers themselves is not to be trusted. The intervention of a third party is desirable in this case.

- The existing information expressed in natural language in the activity description of the MIT process handbook must be formalized, either as a static value or as a function that dynamically generates this value.

- For some activities, the MIT process handbook already provides some knowledge in a formal manner. However, requesters cannot always know which operating process providers are executing. As a result, requesters cannot always identify the relevant activities and their attributes.

In this thesis, we do not provide solutions for the representation of knowledge that is not explicitly expressed in the MIT process handbook. We focus on the integration of the MIT process handbook, as is, since this is already an important task. Further work may, then, explore these issues, perhaps even proposing an extension to our current conceptual structure for the MIT process handbook activities, which would allow for the representation of process constraints in a more systematic manner.

Moreover, because different applications may require definition of specific subconcepts of Transaction and Process, the problem of the extension of the process and transaction ontologies must be addressed. At the moment we envision several possible scenarios for extensions. One scenario is that users specify the extensions that they need. This approach implies the need for translations between user-defined ontologies, which in turn makes it necessary to deal with the problem of alignment of ontologies. A second scenario is that users gather in communities regarding specific domains of activities and interests, and the communities establish the extensions, and then use the extensions. A third scenario is that a board of experts in the domain of business and process representation continually observe the business world and generate new extensions when necessary. In the third scenario, it is also possible for non-expert users to submit extension demands to the experts. Currently we have not chosen which scenarios are to be adopted and which should be rejected.

7.4.5 Using the DTP language extension to describe queries and Web services

Both a query and a Web service are templates: They provide the description of a set of services. For example, a Web service may advertise a service by saying that it sells cars, without specifying the brand. This service will, thus, be selected in the result set for a query requiring a Renault Scenic. As mentioned in Section 7.3, we consider three important aspects of both queries and Web services: the desired result, the category of transaction and a set of constraints on the process. This is done as follows:
Figure 7.7: UML model for a query

- Specification of a **Desired result**. This is one concept described using **Domain knowledge**. Each concept is a specialization of a resource category of the MIT process handbook as specified in Section 7.4.4.

- Specification of a **Transaction**. This is done by selecting the desired transaction in the hierarchy of activities provided by the **Transaction** ontology. As an **Activity**, a transaction may be a composition of other **Activity** concepts. As a result, it is possible to further specify a set of constraints on each of these **Activity** concepts as specified in Section 7.4.3. In some cases, the transaction refers to a resource (e.g. **Acquire physical object**). In these cases, the consistency of the resource with respect to the description of the **Desired result** must be checked. As a result, the generation process of queries and Web service descriptions must ensure that such cross-references are done correctly (i.e. that they reference the same concept). For example, an inconsistent selection can be forbidden by the query or Web Service editor by hiding the parts of the **Transaction** ontology that are not valid with respect to the description of the **Desired result**.

- Specification of **Process constraints**. The user is presented with a set of process attributes and may specify a constraint for each attribute. One example of such an attribute is the **Business model** (e.g. **Entrepreneur**, **Manufacturer**). The **Process** ontologies provides the ranges of values for these attributes.

Figure 7.7 provides the UML model for describing a query or Web service. Figures 7.8 and 7.9 illustrate the use of the DTP language extension by providing a query for buying a **Renault Scenic** in **Linköping** and a Web service selling **Cars** in the region of **Östergötland** (which includes Linköping). The examples illustrate the case where the query and Web service description include a cross reference between the description of the **Desired result** and the **Transaction**. In Figure 7.8, both the **Transaction** and the **Desired result** refer to the domain knowledge concept **D0** (a car whose model is Renault Scenic). Similarly, in Figure 7.9, both the **Transaction** and the **Desired result** refer to the domain knowledge concept **Car**.

In this section, we have shown how to use the MIT process handbook knowledge, and designed a structure that allows its practical and systematic use by
Figure 7.8: UML model for using a query
Figure 7.9: UML model for using a service
the DTP language extension.

The next section completes the definition of our language extension by discussing the design of a matchmaking algorithm for languages adopting the DTP language extension.

7.5 Matchmaking with the DTP language extension

In this section we discuss the different approaches to matchmaking and propose a matchmaking algorithm for languages adopting the DTP language extension.

A user writes a query for Web services to perform the Web service retrieval that may lead to the use of a specific service. The query is submitted to a retrieval engine. The manner in which this engine interprets and takes into account the requester’s and provider’s service representations influences the quality of the retrieval in terms of precision and recall.

Concretely, given a set of Web services, the process of Web service retrieval consists of checking each service description against the query to see if they match. One popular approach to this process is called matchmaking. Service matchmaking is one step in the process of Web service discovery\(^5\). When doing matchmaking, different categories of matching may be performed and different technologies may be used. In the following, we describe matching categories and discuss the different possible matching approaches.

7.5.1 Matching categories

Let us consider a query and a Web service, both of which are described by the set of aspects \(\{a_1...a_n\}\) such that each \(a_i\) is associated with a domain of possible values corresponding to the concept \(Q_i\) for the query and \(W_i\) for the Web service. The query and the Web service do not match iff: \(\forall a_i \in \{a_1...a_n\} : Q_i \cap W_i = \emptyset\). Otherwise, the service and the query agree, at least on some aspects, and we say that there is partial matching. We further identify the following special cases of partial matching:

- **Exact matching** \(\forall a_i \in \{a_1...a_n\} : Q_i = W_i\)

  This matching ensures that the Web service is exactly what the user wants.

- **Query included** \(\forall a_i \in \{a_1...a_n\} : Q_i \subseteq W_i\)

  This matching tells that the Web service provides at least the Web service desired by the user. For example, the Web service allows for the rental of Renault and Volvo cars while the user queries for a service that rents Volvo cars.

---

\(^5\)Service discovery consists of physically finding Web services, matching the Web services against the query, and returning to the requester the Web services that match.
- **Web service included**  \( \forall a_i \in \{a_1...a_n\} : W_i \subseteq Q_i \)

This matching tells that the Web service is a specialization of the desired Web service. For example, the Web service allows for the rental of Volvo cars while the user queries for a service that rents cars with no preference for the car brand.

### 7.5.2 Different matchmaking approaches

We identify two categories of tools to represent and discover Web services: the tools developed by the W3C and FIPA communities for, respectively, the Web and the Semantic Web, and the agent technology. Both communities provide architectural, resource retrieval and message exchange models. The two following sections describe the approaches developed by each community for service matchmaking.

**Web-based matchmaking**

Currently the W3C provides two popular Web service description languages: WSDL and OWL-S. In practice OWL-S is an extension of WSDL. The main differences are documented by Pilioura, Tsalgatidou, and Batsakis (2003). The WSDL approach describes services with a WSDL (2004c) document that may be packaged in a SOAP (2003) envelope for message exchange purposes. UDDI for WSDL (UDDI, 2000-2006) is the standard method adopted for publishing and discovering WSDL documents. It defines the format and use of registries of WSDL Web services. It allows for submitting either new Web services for publication or requests for Web services. The UDDI matchmaking process performs pattern matching on some specific properties (i.e. name, identifier, taxonomy). With respect to the matching categories introduced above, the UDDI matchmaking process provides **exact matching**. As pointed out by Pilioura et al. (2003), it is now often recognized that the descriptive power provided by these properties does not cover the querying needs of the requesters. For example, these properties do not allow for the description of the aspects that we introduced in the DTP language extension (i.e. desired result, transaction, and process constraints).

In contrast, OWL-S (2005) allows for completing the WSDL description with some additional information about the semantics of the service. OWL-S’ syntax and semantics are based on those defined by OWL (2004), a language for ontology writing. Specifically, there are several versions of OWL, and the OWL-S 1.0 release (OWL-S, 2005) provides for a version of OWL-S that relies on some OWL full elements, as well as an updated version, which is OWL-DL (2004a) compliant. OWL ontologies are hierarchies of concepts. Each new concept may either be a child of the root (the **Thing** concept) or of one or several other existing concepts. A concept is represented as a class that specifies a number of properties and a position in the class hierarchy (through the subclass mechanism). Each property has a name (a string) and a domain (a concept or a datatype such as **String** or **Integer**). OWL-S defines a specific OWL concept
called Service which is to be used for defining all Web services. A Service describes three aspects of the service:

- The Profile describes the services with a service name, a textual description, contact information of the service provider, a service category, service parameters that may specify inputs, outputs, preconditions and effects, as well as other properties of the service.

The mechanism of categorization of profiles allows for the creation of a hierarchy of profiles according to specific existing hierarchies such as hierarchies of products (NAICS, UNSPC), problem solving capabilities or commercial services.

The preconditions and effects are meant to provide factual descriptions of the initial and the final state of the process performed by the service. However, there is no clear definition of how to distinguish between a precondition and an input (e.g. should the need for a credit card be expressed as an input or as a precondition.)

- The Model describes the composite structure of the service (i.e. its decomposition into other Services).

- Grounding provides practical information for actually contacting and executing the service. Typically, Grounding corresponds to a WSDL definition of a service.

Since there are some subsets of OWL that are equivalent to decidable description logics (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003), the current most popular matchmaking tools for OWL are description logic reasoning engines, and the language used to describe concepts is OWL DL. OWL DL provides for a subset of OWL operators that describe ontologies to which description logic reasoning is applicable. Examples of OWL DL based matchmaking tools are Protégé (1995-2006), Racer (1999-2006), and OilEd (Bechhofer, Horrocks, Goble, & Stevens, 2001). Being a description logic reasoning engine, the principal operations an OWL DL matchmaker performs are classification and satisfiability, subsumption and instance checking. Subsumption represents the is-a relation in the concept hierarchy. Classification is the computation of a concept hierarchy based on subsumption. Some Semantic Web based matchmaking approaches extend this basic matchmaking algorithm to address specific issues. For example, the OWL-S Matchmaker (2002) adapts the OWL matchmaking algorithm to service matchmaking by defining an algorithm which performs OWL matchmaking of the OWL-S Profile’s inputs and outputs, combined with some RuleML (2001-2006) rule reasoning. OWL-QL (Fikes, Hayes, & Horrocks, 2003) is another example. This approach provides

6Protégé really uses Racer’s DL reasoning engine.

7As of Jan. 11, 2005, OilEd 3.5 really generates DAML+OIL code and reasons about a subset of OWL constructors corresponding to the FaCT description logic. (DAML+OIL is an ancestor language for OWL). There are ongoing efforts to allow OilEd to integrate OWL DL code and to use other DL reasoning engines such as Racer.
a query language and an engine that allows setting of a query in a context (e.g. which color of wine goes with seafood?)

**FIPA agent-based matchmaking**

The Foundation for Intelligent Physical Agents (FIPA) is a non-profit organization aimed at producing standards for the interoperation of heterogeneous software agents (FIPA, 1996-2006). According to (FIPA AMS, 2004):

> An agent is a computational process that implements the autonomous, communicating functionality of an application. Agents communicate using the Agent Communication Language [(ACL)](8). An agent is the fundamental actor on an Agent Platform which combines one or more capabilities, as published in a service description, into a unified and integrated execution model. An agent must have at least one owner, for example, based on organizational affiliation or human user ownership, and an agent must support at least one notion of identity. This notion of identity is the Agent Identifier (AID) that labels an agent so that it may be distinguished unambiguously within the Agent universe. An agent may be registered at a number of transport addresses at which it can be contacted.

Thanks to FIPA’s initiative, new standards are constantly under development to extend the capabilities of the architecture for multi-agent systems.

At first glance, the agent technology may seem less suited to the RDF based communication required by the Web service definition. However, the agent community has provided some very important algorithms for addressing the Semantic Web issue of Web service management, especially for Web service matchmaking (OWL-S Matchmaker (2002)](9) and Web service composition (Racing (2004), SWORD (2002)). Moreover, an initiative like agentcities (2001-2006) may provide a good platform for Semantic Web simulators. There is also an ongoing effort to interface agents with Web services (FIPA-ServiceTC, 2003). One reason for the easy integration of agent technology into the management of Web services is the fact that both service providers and agents advertise their capabilities as services.

In multi-agent systems, these services are described according to a specific format and are registered in a Directory Facilitator (DF). Several DFs may share all the information about services, agents and other DFs in multi-agent systems. When querying a DF for a service, it is possible to set the number of DFs that may be visited before the query is answered. By allowing the distribution and duplication of information among several DFs, the FIPA multi-agent systems allow for a decentralized architecture such as the pure and hybrid forms of peer-to-peer.

---

8 ACL (2002a) is a language based on speech-act theory which describes the different actions that an agent either informs other agents that it is doing, or demands other agents to perform.

9 The OWL-S Matchmaker builds on LARKS (Sycara, Widoff, Klusch, & Lu, 2002), a matchmaker for multi-agent systems.
FIPA provides a standard approach to service matchmaking. The approach relies on ontologies such that:

- A concept may be either a *class name* or a *parameter*.

- A class name may be associated with a set of parameters in a manner similar to the way an ontology concept is associated with a set of properties. A parameter is a pair (name, domain) where the name is a *String* and the domain may be either a class name or a *String*.

When it comes to matchmaking, a query is called a *template object* while the logical view of the desired object is called a *registered object*, and the matching criterion used is described in Table 7.2. With respect to the categories of matching introduced above, this approach provides a *Query included* matching. When it comes to service retrieval, the FIPA service ontology defines the format of a service as in Table 7.3 and the service descriptions are stored in a DF. Both the registration of new service descriptions and the queries for specific services correspond to specific ACL messages. Table 7.4 specifies the format of a query for a specific service (FIPA DSS, 2003) and Table 7.5 provides the resulting syntax for a request for service. Table 7.6 illustrates the use of the query format by providing an example of request for services that sell Renault cars. The DF is also in charge of performing the matchmaking as specified in Table 7.2 above. However, this standard ontology formalism does not provide mechanisms for describing hierarchies of objects. This is a real problem since the current popular ontology languages typically specify hierarchies of concepts.

As a result, FIPA’s ontology service specification (FIPA OSS, 2001), an alternative matchmaking mechanism, which is still experimental, has been proposed. The principle is as follows: Ontologies can be written in any ontology language and each ontology is managed by an ontology agent (i.e. OA). An OA must provide an API to answer a specific set of queries (see (FIPA OSS, 2001) for the exhaustive set of predicates). The API is described through the formalism provided by the Open Knowledge Base Connectivity (OKBC) (Chaudhri, Farquhar, Fikes, Karp, & Rice, 1998). The OKBC provides both a model to represent knowledge and a set of operations for manipulating this knowledge. The model supports an object-oriented representation of knowledge and object types include constants, frames, slots, facets, classes, individuals and knowledge bases. The matchmaking algorithm is managed by the OA and may correspond to any technology, including description logic engines. Since OA specification is not a standard yet, the DF’s message interpreter is not programmed to make use of such agents. As a result, to be able to use the OA when querying for services, one must write a specific agent that extends the capability of the DF to the use of OAs, and use this agent to perform the service retrieval.
The first thing to note about the matching operation is that the search action receives, as its first argument, an object description that evaluates to a structured object that will be used as an object template during the execution of the search action. In the following explanation, the expression parameter template and value template are used to denote a parameter of the object template, and the value of the parameter of the object template, respectively.

A registered object matches an object template if:

1. The class name of the object (that is, the object type) is the same as the class name of the object description template, and,

2. Each parameter of the object template is matched by a parameter of the object description.

A parameter matches a parameter template if the parameter name is the same as the template parameter name, and its value matches the value template. Since the value of a parameter is a term, the rules for a term to match another term template must be given. Before, it must be acknowledged that the values of the parameters of descriptions kept by the AMS or by the DF can only be either a constant, set, sequence (see FIPA SL (2002d)) or other object descriptions (for example, a service-description).

The search action evaluates functional expressions before the object template is matched against the descriptions kept by the AMS or by the DF. This means that if the value of a parameter of an object description is a functional term (for example, (plus 2 3)), then what is seen by the matching process is the result of evaluating the functional term within the context of the receiving agent. A constant matches a constant template if they are equal. Informally, a sequence matches a sequence template if the elements of the sequence template are matched by elements of the sequence appearing in the same order. Formally, the following recursive rules apply:

1. An empty sequence matches an empty sequence, and,

2. The sequence (cons x sequence1) matches the sequence template (cons y sequence2) if:
   - x matches y and sequence1 matches sequence2, or,
   - sequence1 matches (cons y sequence2).

Table 7.2: The matching criterion used in the search for an object registered with an agent, as defined in (FIPA AMS, 2004)
### 7.5. Matchmaking with the DTP language extension

#### Table 7.3: Service Description: this type of object represents the description of each service registered with the DF (FIPA AMS, 2004)

<table>
<thead>
<tr>
<th>Frame</th>
<th>service-description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology</td>
<td>FIPA-Agent-Management</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Presence</th>
<th>Type</th>
<th>Reserved Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The name of the service</td>
<td>Optional</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td>The type of the service</td>
<td>Optional</td>
<td>String</td>
<td>fipa-df, fipa-ams</td>
</tr>
<tr>
<td>protocol</td>
<td>A list of interaction protocols supported by the service</td>
<td>Optional</td>
<td>Set of String</td>
<td></td>
</tr>
<tr>
<td>ontology</td>
<td>A list of ontologies supported by the service</td>
<td>Optional</td>
<td>Set of String</td>
<td>FIPA-Agent-Management</td>
</tr>
<tr>
<td>language</td>
<td>A list of content languages supported by the service</td>
<td>Optional</td>
<td>Set of String</td>
<td></td>
</tr>
<tr>
<td>ownership</td>
<td>The owner of the service</td>
<td>Optional</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>properties</td>
<td>A list of properties that discriminate the service</td>
<td>Optional</td>
<td>Set of property</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 7.4: Search for df-agent-description registrations within FIPA’s agent discovery service (FIPA DSS, 2003)

<table>
<thead>
<tr>
<th>Function</th>
<th>search</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology</td>
<td>fipa-agent-discovery</td>
</tr>
<tr>
<td>Supported by</td>
<td>ADS (Agent Discovery Service)</td>
</tr>
</tbody>
</table>
| Description | An agent may search for certain df-agent-descriptions by passing a df-agent-description template to the ADS. A successful search can return one or more df-agent-descriptions that satisfy the search criteria and are returned within a fixed amount of time. A null set is returned when no df-agent-description entries satisfy the criteria. A null set is also returned when the defined search duration is exceeded, even if some results would have been received later on. To prevent a search on all available DMs [i.e. Discovery Middlewares], the DM-IDs [*] of the desired DMs can be passed. Further, the maximum number of returned results per agent platform can be defined.  

[* A DM ID is a string reserved for a single technology] |
| Domain | df-agent-description * search-constraints |
| Range  | Set of df-agent-descriptions |
| Arity  | 2 |


(search
  (df-agent-description
    :services
    (set (service-description
      :name nameX
      :type typeX
      :protocol (set protocolX)
      :ontology (set ontologyX)
      :language (set FIPA-SL FIPA-SL1)
      :ownership (set ownerX)
      :properties (set
        (property :name nameP1 :value valueP1)
        (property :name nameP2 :value valueP2))))
    (search-constraints
     :max-depth 2
     :max-results 10)
    :ontology fipa-agent-discovery)

Table 7.5: Syntax of a query for service in ACL

(request
  ...
  :content
  (search
    (df-agent-description
      :services
      (set (service-description
        :type buy
        :ontology (set Transaction Car)
        :language (set FIPA-SL FIPA-SL1)
        :properties (set (property :name article-to-buy
          :value (any (is-car ?c)
            (and
              (hasBrand Renault ?c)
              (hasType Scenic ?c)
            )))
          )))
      :ontology fipa-agent-discovery))

Table 7.6: Example of query for a service that sells Renault Scenic cars
7.5.3 The current matchmaking approaches are not satisfactory

Neither the Semantic Web nor the FIPA multi-agent service matchmaking approaches provide any guidelines for the uniform use of formats and ontologies for query and Web service description. To illustrate our claim, let us look again at the FIPA example for a query given in Table 7.6. This example assumes the existence and correct use of a set of knowledge sources: Transaction knowledge, and a Car ontology that describes the notion of car, car brand and car type.

As discussed in Section 7.1, in the absence of guidelines for the uniform use of formats and ontologies for query and Web service descriptions, the current matchmaking approaches cannot ensure service retrieval with good precision and recall. Still, the technology available is very popular and widely accepted. As a result, to be truly viable, a better retrieval solution must allow for service retrieval of Web services written in either the current Semantic Web or FIPA service description languages.

7.5.4 Using the existing matchmaking algorithm

By both providing a theoretical definition of the language components, and the ontologies that provide the range of valid values for these components, the DTP language extension aims at supporting the uniform description of queries and services. The next step in providing a working service retrieval approach for queries and services described with the DTP language extension is to define the accompanying matching algorithm. Concretely, the matching must be performed on three components: the desired result, the transaction and the process constraints. Let us consider $D_a$, $T_a$, $P_a$ the desired result, transaction and process constraints described in the DTP language extension for a query or a Web service $a$. In this context, a Web service $s$ matches a query $q$ iff:

- $D_q \subseteq D_s$ (this is the query included matching of the desired result), and
- $T_q \subseteq T_s$ (this is the query included matching of the transaction), and
- $P_q \subseteq P_s$ (this is the query included matching of the business model).

Notice that if we reduce the notion of process constraints to the description of the Business model, as suggested in Section 7.4.4, the classical subsumption is a good candidate for matchmaking queries and services.

The structure of the Transaction and Process ontologies introduced in Section 7.4.2 allow for Bundle criterion domains that are disjunctions of concepts, such as the domain of the Buy what criterion, i.e. human or information or physical entity or raw material. This requires a representation language that is able to express the disjunction (i.e. OR).

OWL is a language that can express the disjunction and for which subsumption algorithms exist. We examine the following popular algorithms:
• The description logic reasoner OWLJessKB (2003-2006). The semantics of the language are implemented using the Horn Logic reasoner Jess (1995-2006). OWLJessKB can integrate some OWL code.

• The description logic reasoner Racer (1999-2006). Racer provides a SHIQ description logic reasoner. An additional package, such as Jena + DIG must be used for Racer to integrate OWL ontologies.

A comparison of the two algorithms is provided in Table 7.7. Illustrations of the use of both algorithms for sample queries and Web services are provided in Chapter 8 for OWLJessKB, and in Chapter 9 for Racer.

The next section defines OWL-DTP, the extension of the knowledge representation language OWL DL with the DTP language extension.

7.6 OWL-DTP

In order to use the DTP language extension on the Semantic Web, we use the extension to extend the knowledge representation language OWL DL. The resulting extended language is called OWL-DTP and it implements the three parts of the DTP language extensions as follows:

1. The conceptualization of the notions of query and service

   Similarly to the way OWL-S has been defined, the extended part of OWL-DTP is defined as a set of markup language constructs for describing the properties and capabilities of Semantic Web services. The markup language constructs follow the DTP language extension definition as follows:

   • All queries and services are WebService concepts.
   • WebService concepts have three properties:
     - hasDesiredResult takes a Resource as value. Resource is the root class of the resource ontology built from the knowledge available in the MIT process handbook. Examples of subconcepts of Resource are Actor, Location and InformationEntity. All domain specific concepts such as Car or InsuranceContract must be added to the hierarchy of resources in order to be recognized as resources. This way, Car can be a subconcept of PhysicalEntity, InsuranceContract may be a subconcept of both PhysicalEntity and InformationEntity.
7.7 Comparison of OWL-DTP, OWL-S and WSMO

- **hasTransaction** takes a **Transaction** as value. **Transaction** and its subconcepts represent the processes that describe business transactions in the MIT process handbook such as **Buy** and **Acquire for free**. **Transaction** refers to a subconcept of **Process** in the MIT process handbook.

- **hasProcessConstraint** takes a **Process** as value. All the processes described in the MIT process handbook are **Process** concepts.

Appendix A provides the current OWL DL definition of the markup language constructs for OWL-DTP.

2. **The ontologies** are heavily based on the knowledge available in the MIT process handbook and adopt the structure that we designed and described in Section 7.4. The corresponding OWL DL definition of the markup language constructs required to describe the main concept of **Activity** is provided in Appendix B. Moreover, Appendix C provides the OWL DL definition of the **Resource** ontology, and examples of the **Transaction** and **Process** ontologies are available in Appendices D and E.

3. **The matching algorithm**: as stated in Section 7.5, all three **matches** operations perform **query included** matching. Moreover, given the transaction and process ontologies above, the three operations are further defined as follows:

- **matches_d**’s definition is domain dependent. For example, a trip whose arrival and departures are both located in Europe matches a trip whose departure is located in Paris (France) and arrival is located in Stockholm (Sweden). In this example, the matching depends on the semantic of the relation **locatedIn** that is specific to the travel domain. In other domains, **matches_d** may correspond to the classical subsumption operation.

- as mentioned in Section 7.5, the current hierarchies provided by the transaction and process ontologies allow the definition of the **matches_t** and **matches_p** relations as the classical subsumption operation.

The next section provides a comparison of OWL-DTP with two other approaches to Semantic Web service description language.

### 7.7 Comparison of OWL-DTP, OWL-S and WSMO

While OWL-S and WSMO are based on a process view of Web services, OWL-DTP strives to provide a more holistic view for both the requesters and the
providers. In order to clarify the concrete consequences of the different approaches, we propose comparing the three approaches on their capability to express different queries for services. We start by providing more detailed information on OWL-S and WSMO.

### 7.7.1 OWL-S

OWL-S is an *OWL-based Web Service Ontology* developed by the DAML program. OWL-S, under its earlier name of DAML-S, was the first Semantic Web service language proposition\(^\text{10}\), with DAML-S 0.5 being released in 2001. As such, OWL-S (and DAML-S) has, for the last few years, been the one format of reference used by developers of technology for the Semantic Web. With respect to service discovery, OWL-S represents services and queries as processes. The service aspects considered by OWL-S are a set of inputs, outputs, preconditions and effects (IOPEs), and a set of (label, value) pairs that describe quality of service concerns. OWL-S also allows the association of the service to a general capability (cf. profileHierarchy) and to service categorizations (e.g. NAICS). There is no standard algorithm for matching queries and services expressed in OWL-S. The OWL-S Matchmaker (2002) and OWLSM (2004-2006) both assume that queries and services are expressed in OWL-S but implement two different OWL-S matching algorithms.

### 7.7.2 WSMO

WSMO (Web Service Modeling Ontology) defines a class of ontology languages corresponding to the requirements expressed by WSMF (Fensel & Bussler, 2002), the Web service modeling framework. WSMO languages describe queries for services with so-called *goals* and the services themselves with so-called *Web services*. With respect to service discovery, a goal expresses the following aspects of a query for services: core non-functional properties (e.g. name and author of the query), requested capability of the service, and requested interface (i.e. the interaction that the requester would like to have with the provider). Similarly, Web services are described with non-functional properties (i.e. core properties as well as other service-specific properties such as reliability, performance, trust, etc.), capability and interfaces (choreography and orchestration). The *capability* aspect of a goal and Web service is described with preconditions, assumptions, postconditions and effects. Preconditions describe the information space that is required before enactment of the service, and postconditions describe the state the space is guaranteed to achieve after enactment of the service. Assumptions and effects describe the state of the world before and after the execution of the service. Service discovery consists of finding the Web services that have a capability and an interface that match a given goal’s capability and interface. WSML (Web Service Modeling Language) provides the formal syntax for WSMO. Further, several flavors of WSML exist that correspond to the sets

\(^{10}\)In this thesis we do not consider languages such as WSDL or BPEL4WS since they do not provide for any actual semantics as pointed out in (Lara et al., 2003), among others.
of operators whose semantics can be expressed with specific logics. For exam-
ple, WSML-DL’s semantics can be expressed with the SHIQ description logic,
allowing use of SHIQ reasoners in the matching process.

### 7.7.3 Comparison method

The OWL-S, WSMO, and OWL-DTP approaches have their own characteristics
and corresponding strengths and weaknesses to support service discovery. We
have designed a test suite composed of queries that highlight characteristics
and differences of the approaches, while also providing good coverage of the
different aspects of services that a query may want to set restrictions on. To
ensure that the queries are realistic we looked at the services currently available
on the Web and composed queries that one might want to express to find these
services. The evaluation consists of attempting to use each approach to express
each query and analyze whether or not it is possible and whether there are
difficulties with respect to matching the expected services. Our proposed test
suite is by no means complete, and we hope that other researchers will take
up this and extend the test suite with additional queries and scenarios, so that
these approaches as well as future ones can be further evaluated.

Our initial test suite consists of a set of service discovery queries, categorized
into different types of challenges, as described in the next section.

### 7.7.4 Test suite

The test suite is made up of five challenges and five accompanying queries.
The challenges are selected to target different aspects of service descriptions.
The aspect that is the most commonly described in existing benchmarks is the
“needed resource.” However, there are other very important aspects to consider
as exemplified by challenges 2 to 5 below.

**Challenge01** Describing a needed resource.

**Query01.a** Book a flight from Monterey airport to JFK airport (New
York).

**Challenge02** Specifying the conditions on the use of the service.

**Query02.a** Get a bed delivered within a week.

**Challenge03** Specifying the kind of business transaction provided by the ser-
vice.

**Query03.a** Get a research article for free.

**Challenge04** Specifying conditions on the means used by the provider to pro-
vide the service.

**Query04.a** Get a t-shirt with the assurance that it has not been produced
through child labor.
### Challenge05

Specifying constraints on the service provider.

*Query05.a* Get a football match ticket from a trusted provider.

### 7.7.5 Expressing queries with OWL-S, WSMO, and OWL-DTP

When trying to express the queries introduced above, we found that the three approaches had two main difficulties:

**aspect ambiguity** The approaches provide several ways to use language constructors to represent the same need, while the matching algorithms are not treating them as equivalent. This situation makes it possible for requesters and providers to describe the same service using different formulations that will never match each other. This difficulty is typically due to an incomplete implementation of requirement R2.

**undefined concept** It is possible to express the query (and candidate services). However, this requires the definition of a specific concept on which both the requester and the candidate providers have to agree to avoid misunderstandings. This difficulty is typically due to an incomplete implementation of the requirement R1 and/or R3 introduced in Section 7.1.

Table 7.8 provides a summary of the difficulties and successes of each approach with respect to each query.
Table 7.9: Query01 in OWL-S: Extract of the BravoAir service profile

In the following, the code examples for OWL-S and WSMO have been written based on the authors’ understanding of the following language specifications:

**OWL-S 1.1:** OWL-S 1.1 Release (OWL-S, 2005).

The specification of OWL-S does not provide specific guidelines for writing Web service queries. As a result our proposed code may not correspond to the intended use of the language by the authors of the specification. However, the language allows these formulations.

**WSMO:** (Roman et al., 2005).

Moreover, for the OWL-DTP code we consider the language definition provided in Section 7.6. All of the OWL-DTP source code, ontologies and queries have been successfully validated with the wOWLidator (2005-2006) from BBN technologies. All the OWL-DTP queries have the same header as pictured in Appendix F.2. We will not repeat this header in the following OWL-DTP queries. Further, for each approach, the definition of the domain knowledge referred by the queries is provided in Appendices F.3 to F.5.

**Query01.a** Book a flight from Monterey airport to JFK airport (New York).

**OWL-S:** The semantics of the markup constructors for the OWL-S profile allows for different ways to write the profile. To illustrate, let us consider the profile of the service OWL-S 1.1 BravoAir profile (2004) (see an extract in Table 7.9), and the two possible ways of expressing the query in OWL-S pictured in Table 7.10.

In the first example, the query describes the desired result as a **Result**. In the second example the query describes the desired result as an **Output**. The service profile describes a **Result** corresponding to **HaveSeatResult** and an **Output** whose range is **FlightsFound**. Whether the service will be returned as an answer to the queries depends on the matching algorithms. For example, neither the algorithms of the OWL-S Matchmaker nor OWLSM take into account
Version 1 of the query profile: the desired result is represented as a Result.

<profile:Profile rdf:ID="Query01a1profile">
  <profile:hasInput rdf:resource="&domainknowledge;#MontereyAirport"/>
  <profile:hasInput rdf:resource="&domainknowledge;#JFKAirport"/>
  <profile:hasResult rdf:resource="&ba_process;#FlightsFound"/>
</profile:Profile>

Version 2 of the query profile: the desired result is represented as an Output.

<profile:Profile rdf:ID="Query01a2profile">
  <profile:hasInput rdf:resource="&domainknowledge;#MontereyAirport"/>
  <profile:hasInput rdf:resource="&domainknowledge;#JFKAirport"/>
  <profile:hasOutput rdf:resource="&ba_process;#FlightsFound"/>
</profile:Profile>

Table 7.10: Query01 - two versions for the OWL-S approach

the information described as a Result. Thus the first query formulation will not match the service. However, both matching algorithms qualify the match initiated by the second query as successful. OWL-S suffers from ambiguous aspect.

WSMO provides more fully defined aspects to describe the notion of capability. WSMO avoids the problem of ambiguity. As illustrated in Table 7.11, the query is expressed with a precondition axiom.

OWL-DTP also provides more fully defined aspects to describe its notion of WebService. Moreover OWL-DTP provides some matching guidelines by requiring that corresponding aspects are matched. As illustrated in Table 7.12, the OWL-DTP also avoids this ambiguity problem.

Query02.a Get a bed delivered within a week.

OWL-S allows definition of a serviceParameter maxDeliveryTime, which can then be set to 1 week in the profile. The resulting query is provided in Table 7.13. However, in the case where this service parameter does not exist as a common concept for the community with which the requester expects to interact, we are confronted with the problem of undefined concept.

WSMO does not define a specific non-functional property of the service for describing the notion of delivery delay. As a result, and as illustrated in Table 7.14, the only way to define this query is to define a postcondition that checks that the difference between the date of the query and the date of the delivery is less than or equal to a week. This formulation implies the existence of the notion of query and delivery date which are not requester/provider domain-dependent knowledge. As a result we identify another case of undefined concept.
7.7. Comparison of OWL-DTP, OWL-S and WSMO

instance Query01a subConceptOf webService
hasCapability Query01aCapability

instance Query01aCapability memberOf Capability
hasPrecondition preConditionOfWSCapabilityForQuery01a

axiom preconditionOfWSCapabilityForQuery01a
definedBy
exists ?departureAirport, ?arrivalAirport
  (?trip[
    departureAirport hasValue ?departureAirport
    arrivalAirport hasValue ?arrivalAirport
  ]memberOf trip and
  (?departureAirport hasCity Monterey) and
  (?arrivalAirport hasName JFK)
)

Table 7.11: Query01 - the WSMO approach

[header]
<ws:WebService rdf:ID="Query01a">
  <ws:hasDesiredResult rdf:resource="#DRQuery01a"/>
  <ws:hasTransaction rdf:resource="#TQuery01a"/>
  <ws:hasProcessConstraint rdf:resource="#PCQuery01a"/>
</ws:WebService>

<d:Flight rdf:ID="DRQuery01a">
  <d:hasDeparture rdf:resource="&d;#MontereyAirport"/>
  <d:hasArrival rdf:resource="&d;#JFKAirport"/>
</d:Flight>

<t:Buy rdf:ID="TQuery01a"/>
<p:MITphProcess rdf:ID="PCQuery01a"/>

</rdf:RDF>

Table 7.12: Query01 - the OWL-DTP approach
Table 7.13: Query02 - the OWL-S approach

Table 7.14: Query02 - the WSMO approach
OWL-DTP provides transaction-specific means of formulating such a query. For example, and as illustrated in Table 7.15, the transaction Acquire has a subprocess Determine timing that allows definition of all timing constraints. However, all categories of timing constraints (such as a specific delivery period) are not currently provided by the MIT process handbook. In this context OWL-DTP provides a partial solution to the undefined concept problem.

Query03.a Get a research article for free.

OWL-S allows several ways to specify such a query, as shown in Table 7.16. One approach is to specify an Output with a range set to FreeResearchArticle. A second approach is to specify the profile Hierarchy as an AcquireForFree profile. Each approach adopts a different modeling solution. In the first approach, domain knowledge and business knowledge are merged. In the second approach, the two types of knowledge are kept separated. Neither approach provides a means of determining, for a given community of requesters and providers, who is responsible for generating a uniformly acceptable definition of the shared business knowledge. Also, because there are several ways to specify the query, there is a high risk that the matching will fail in the case where the query is expressed in one way while the service description is expressed in another. In this case OWL-S suffers from both undefined concept and ambiguous aspect.

WSMO provides one way to express such a query: specifying a post-condition that states the existence of a FreeResearchArticle. As shown in Table 7.17, this implies that both services and providers know what a FreeThing is. This is not a notion that is defined by WSMO nor one that is requester or provider domain dependent. This is a notion that corresponds to the domain of business descriptions. WSMO does not provide for a way to disconnect the business description from the desired result. As a result we identify another case of undefined concept.

OWL-DTP provides a predefined collection of transactions that specify the notion of free thing. One example is the AcquireForFree transaction. As a result, and as shown in Table 7.18, OWL-DTP allows definition of the query by setting the transaction to AcquireForFree (or one of its descendants) and the desired result to Article.

Query04.a Get a t-shirt with the assurance that it has not been produced through child labor.

OWL-S allows definition of a serviceParameter noChildLabor, which can then be set to true in the profile. The resulting query is provided in Table 7.19. Similarly to the case of Query02.a this formulation implies the problem of undefined concept.
<ws:WebService rdf:ID="Query02a">
<ws:hasDesiredResult rdf:resource="#DRQuery02a"/>
<ws:hasProcessConstraint rdf:resource="#PCQuery02a"/>
<ws:hasTransaction rdf:resource="#TQuery02a"/>
</ws:WebService>

<d:Bed rdf:ID="DRQuery02a"/>
<AcquireWithDeliveryInAWeek rdf:ID="TQuery02a"/>
<p:MITphProcess rdf:ID="PCQuery02a"/>

<!-- extension to the transaction and process ontologies -->
<owl:Class rdf:ID="DDAcquireWithDeliveryInAWeek">
<rdfs:subClassOf rdf:resource="&s;#DirectDecomposition"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&s;#hasParticipant"/>
<owl:allValuesFrom rdf:resource="&p;#IdentifyNeedsOrRequirements"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&s;#hasParticipant"/>
<owl:allValuesFrom rdf:resource="&p;#ReceivePhysicalResource"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="AcquireWithDeliveryInAWeek">
<rdfs:subClassOf rdf:resource="&t;#Acquire"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="&s;#hasDirectDecomposition"/>
<owl:allValuesFrom rdf:resource="#DDAcquireWithDeliveryInAWeek"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="DetermineTimingOfDeliveryWithinAWeek">
<rdfs:subClassOf rdf:resource="&p;#DetermineTiming"/>
</owl:Class>

</owl:RDF>

Table 7.15: Query02 - the OWL-DTP approach
7.7. Comparison of OWL-DTP, OWL-S and WSMO

Version 1:
<profile:hasOutput rdf:resource="&domainknowledge;#FreeResearchArticle"/>

Version 2:
<profileHierarchy:AcquireForFree rdf:ID="Profile_Query03'">
<profile:hasOutput rdf:resource="&domainknowledge;#ResearchArticle"/>
</profileHierarchy:AcquireForFree>

Table 7.16: Query03 - the OWL-S approach

instance Query03a subConceptOf webService
hasCapability Query03aCapability

instance Query03aCapability memberOf Capability
hasPostcondition postconditionOfWSCapabilityForQuery03a

axiom postconditionOfWSCapabilityForQuery03a
definedBy
exists ?providedThing
(?providedThing memberOf ProvidedThing and
?providedThing.type memberOf FreeThing and
?providedThing.type memberOf ResearchArticle)

Table 7.17: Query03 - the WSMO approach

[header]

<ws:WebService rdf:ID="Query03a">
<ws:hasDesiredResult rdf:resource="DRQuery03a'"/>
<ws:hasTransaction rdf:resource="TQuery03a''"/>
<ws:hasProcessConstraint rdf:resource="PCQuery03a''"/>
</ws:WebService>

d:ResearchPaper rdf:ID="DRQuery03a''/>
<t:AcquireNotForMoney rdf:ID="TQuery03a''"/>
<p:MITphProcess rdf:ID="PCQuery03a''"/>

</rdf:RDF>

Table 7.18: Query03 - the OWL-DTP approach
<profile:Profile rdf:ID="Query04aprofile">
    <profile:hasOutput rdf:resource="&domainknolwedge;TShirt"/>
    <profile:serviceParameter>
        <profile:ServiceParameter>
            <profile:serviceParameterName rdf:datatype="&xsd;#string">
                noChildLabor
            </profile:serviceParameterName>
            <profile:sParameter rdf:resource="&concepts;#True"/>
        </profile:ServiceParameter>
    </profile:serviceParameter>
    </profile:Profile>
</profile:Profile>

Table 7.19: Query04 - the OWL-S approach

**WSMO** does not define a specific non-functional property of the service that describes ethical issues. As a result, and as illustrated in Table 7.20, the only way to define this query is to define an effect that states that “no child labor was performed.” This formulation implies the problem of **undefined concept**.

**OWL-DTP** does not provide for the notion of ethical conditions of work. However, the process constraint aspect is typically made to specify such constraints. As illustrated in Table 7.21, this is done by specifying a new process concept `ProcessWithNoChildLabor`. As for OWL-S and WSMO above, this formulation implies the problem of **undefined concept**. Note however that the notion of ethical policy is not completely overlooked by OWL-DTP since the MIT process handbook provides the description of a process to `ManageLegalAndEthicalIssues`. However, in its current implementation the process is not useful for the purpose of this query.

**Query05.a** Get a football match ticket from a trusted provider.

**OWL-S** allows definition of a `serviceParameter trusted`, which can then be set to true in the profile. The resulting query is provided in Table 7.22. Similarly to the cases of Query02.a and Query04.a, this formulation implies the problem of **undefined concept**.

**WSMO** does define a specific non-functional property of the service that describes trust. As a result, it is possible to express a query as provided in Table 7.23. However, neither the syntactical details, nor the coverage, nor the ontologies necessary to specify such a concept are specified yet.

**OWL-DTP’s** current versions of the process and transaction ontologies do not provide for the notion of trust. However, the process constraint aspect is made to specify such constraints. As illustrated in Table 7.24, this is done by specifying a new process concept `TrustedProcess`, triggering again the problem of **undefined concept**.
instance Query04a subConceptOf webService
   hasCapability Query04aCapability

instance Query04aCapability memberOf Capability
   hasPostcondition postconditionOfWSCapabilityForQuery04a1
   hasPostcondition postconditionOfWSCapabilityForQuery04a2
   hasEffect effectOfWSCapabilityForQuery04a

axiom postconditionOfWSCapabilityForQuery04a1
definedBy
   exists ?providedThing
   (?providedThing memberOf ProvidedThing and
    ?providedThing.type memberOf TShirt)

axiom postconditionOfWSCapabilityForQuery04a2
definedBy
   exists ?ethicalrule
   (?ethicalrule memberOf EthicalRule and
    exists ?ethicalrule.nochildLabor)

axiom effectOfWSCapabilityForQuery04a
definedBy
   ?ethicalrule.nochildLabor = true

Table 7.20: Query04 - the WSMO approach

[header]

<ws:WebService rdf:ID="Query04OWLDTP">
  <ws:hasDesiredResult rdf:resource="#DRQuery04a"/>
  <ws:hasTransaction rdf:resource="#TQuery04a"/>
  <ws:hasProcessConstraint rdf:resource="#PCQuery04a"/>
</ws:WebService>

<d:TShirt rdf:ID="DRQuery04a"/>
<t:Acquire rdf:ID="TQuery04a"/>
<MakeProductWithNoChildLabor rdf:ID="PCQuery04a"/>

<!-- extension to the process ontology -->

<owl:Class rdf:ID="MakeProductWithNoChildLabor">
  <rdfs:subClassOf rdf:resource="&p;#MakeProduct"/>
</owl:Class>

</rdf:RDF>

Table 7.21: Query04 - the OWL-DTP approach
<profile:Profile rdf:ID="Query05aprofile">
  <profile:hasOutput rdf:resource="&domainknowledgewls;FootballMatchTicket"/>
  <profile:serviceParameter>
    <profile:ServiceParameter>
      <profile:serviceParameterName rdf:datatype="&xsd;#string">Trusted</profile:serviceParameterName>
      <profile:sParameter rdf:resource="&concepts;#True"/>
    </profile:ServiceParameter>
  </profile:serviceParameter>
</profile:Profile>

Table 7.22: Query05 - the OWL-S approach

instance Query05a subConceptOf webService
hasCapability Query05aCapability

instance Query05aCapability memberOf Capability
hasnonFunctionalProperties trust hasValue [trustedProvider]*
hasPostcondition postconditionOfWSCapabilityForQuery05a

axiom postconditionOfWSCapabilityForQuery05a
  definedBy
  exists ?providedThing
  (?providedThing memberOf ProvidedThing and
   ?providedThing.type memberOf FootballMatchTicket)

* disclaimer: the syntax of the notion of trust is not well defined yet.

Table 7.23: Query05 - the WSMO approach
7.7. Comparison of OWL-DTP, OWL-S and WSMO

Table 7.24: Query05 - the OWL-DTP approach

Note though, that processes dealing with trust, such as InformationUsingTrustBasedAdvisor, are described in the MIT process handbook, opening the path to future systematic definitions of the notion of trusted processes.

To sum up, the three approaches perform differently on the set of queries, with OWL-S being the approach that suffers the most from ambiguity-related problems.

7.7.6 Discussion

Based on the evaluation above we assess the three approaches with respect to each of the requirements introduced in Section 7.1.

R1 The language must be capable of expressing all the relevant aspects of a service.

By adopting a service representation as a process, OWL-S and WSMO overlook the need to represent capabilities in a systematic and concrete manner (as illustrated by their difficulties with respect to Query03.a and Query04.a). As a result, it is possible for different users (i.e. requesters and providers) to specify their own way of defining capabilities, which will likely generate problems of understanding between different communities of users. OWL-DTP takes another approach, focusing on providing abstract definitions of capabilities, and connecting them to process definitions. The current process definitions of OWL-DTP are extensible to include the classical IOPE view of a process, making it a language which has the potential to provide the different views of the notions
of query and service descriptions that are required by different applications and users.

**R2** There must be a semantic definition of each aspect that is well-defined to avoid ambiguous use of the language.

OWL-S does not fully define each element of its profile. For example, the difference between the notions of input and precondition is not clearly defined, leading to the problem of ambiguous aspect in Query01.a. Similarly, the use of the profileHierarchy is not fully defined, allowing for many different interpretations of this aspect of the Web service. A concrete consequence of these weak definitions is the diversity of interpretations when designing OWL-S matching algorithms.

WSMO provides a definition of each aspect by giving a clear natural language definition, and/or by associating a terminology (i.e. non-functional properties) to some aspects. However, the difference between what belongs to the information space (i.e. expressed with preconditions and postconditions) and the world (i.e. expressed with assumptions and effects) may be domain and application dependent. As a result, in some interdisciplinary cases, the language can be ambiguous. Furthermore, no standard matching algorithm has been described yet. This situation reinforces the risk of implementing different interpretations of the language at service discovery time, leading to different matching results.

The DTP extension clearly specifies the range of each language constructor, both logically and in natural language (see section 7.3). This avoids ambiguous use of the DTP aspects for all queries in the test suite.

**R3** There must be practical means to enforce that the language constructs are used in a manner that is uniform and according to their intended use.

One of the major difficulties when implementing queries and services for the test suite is ensuring that we have used the language constructors as they were meant to be used, and in a manner that would make the matching unambiguous. This is due to the fact that there are no semantical compilers of queries and service descriptions. However, each approach provides some support.

OWL-S provides some illustration of the use of the profileHierarchy and the serviceCategory. OWL-S 1.1 also provides a candidate Author ontology to describe the contact information of the service/query designer. Although these illustrations provide a better understanding of the semantic definition of the constructor, they do not systematically enforce the intended use of the language.

WSMO provides for a bit more specific guidance by associating a terminology to the notion of a non-functional property. This terminology can itself be further developed into ontologies written in different languages.

By requiring that standard ontologies be associated with each DTP aspect, the DTP language extension goes one step further than WSMO. It is true that in practice the standard ontologies must often be extended to integrate domain and application specific notions of resources, transactions and process constraints. The extension may be domain dependent, but the fact that all the extensions share a common standard root ontology always allows for provision of a common
7.7. Comparison of OWL-DTP, OWL-S and WSMO

ground of understanding that decreases the risk of ambiguity when interpreting domain dependent concepts. Moreover, OWL-DTP is the only approach that puts requirements on the implementation of the matching algorithm.

Summary

WSMO and OWL-S both describe services as processes. OWL-S is the pioneer of the Semantic Web service languages, and its design makes a certain amount of ambiguity inevitable. WSMO strives to eliminate ambiguity while staying general enough to be domain independent. OWL-DTP strives to provide a more abstract view of the notion of services, as well as an ontology-based mechanism to decrease the risk of ambiguous interpretation of services and queries. OWL-DTP also allows description of process constraints by referring to the specific steps of the process, thus providing some specific context to the constraint. In the current version of OWL-DTP, the IOPE view of the processes is not provided.

The next chapter provides a prototype implementation of sButler using OWL-DTP as a language to describe requesters’ queries.
Chapter 8

Prototype Implementation of sButler making use of OWL-DTP

Our sButler prototype follows the model and architecture described in Chapter 6. It is implemented in Java, and the knowledge bases are OWL lite files integrated into a Jess (1995-2006) knowledge base through the facilities provided by the OWLJessKB (2003-2006) reasoning engine. Additional rules (e.g. to formulate user and organization preferences) are directly written as Jess triplets. Each Java implemented sButler module may query the Jess knowledge base using the OWLJessKB Java API. Figure 8.1 illustrates the use of the different knowledge sources by the sButler prototype’s process instance generation modules.

We simulate the organization’s WfMS with JGraph (2003-2006), a Java graph visualization library that provides a graphical visualization of the workflow as a task flow. It also provides an annotation mechanism for the tasks delegated to the sButler that is used to assign task descriptions and attributes. The Semantic Web is represented by a set of Web services. The Web services and sButler’s internal representation of the task share the same model composed of two of the DTP aspects introduced in Section 7.1: the desired result and the kind of transaction the organization wants to enter into. The Web services and Web service queries are all represented by OWL lite files integrated into a Jess knowledge base. The service discovery is simulated using the OWLJessKB matchmaking algorithm.

Considering our DTP model for Web service descriptions, we observe that some local knowledge, such as user and organizational preferences, may provide information for decomposing the desired result. Therefore, we designed a simple algorithm for using local knowledge to compose services. First, we identify the local knowledge (i.e. Domain knowledge or User and organization preference knowledge) related to the required result that can be used for decomposition. Second, we use the Decomposition knowledge to perform the
decomposition. Then, we generate a set of queries for each of the subparts of the required result and run the queries. Finally, we use the Decomposition knowledge to combine the answers and produce solutions for the task. The first three steps are performed during the Task Decomposition, the fourth step is performed during the Service Discovery and the last step is performed during the Instance Generation.

We use the conference travel scenario introduced in Chapter 3 to test our prototype. We set up the three knowledge bases so that they contain the following concept descriptions:

- The Domain knowledge models the OfficeSupply and Travel concepts. The Travel concept may be specialized into RoundTripTravel or OneWayTravel. Travel is composed of ItineraryLegs which have the properties hasDeparture, hasArrival and hasTransportMode. The domains of the hasDeparture and hasArrival properties are locations, while the domain of hasTransportMode is TransportMode. Examples of Transport Mode are train, boat, plane, and taxi.

- The Transaction knowledge models the Acquire activity specialization hierarchy from the MIT process handbook. This includes the activity Buy that may be decomposed into the sequence of activities Identify potential sources, Identify own needs, Select supplier, Place order, Receive, Pay, and Manage suppliers.
• The **Decomposition knowledge** includes information related to the decomposition and assembly of itineraries. An example of a decomposition rule for an *ItineraryLeg* is that an itinerary leg from Linköping University to X using unspecified transportation can be decomposed into two itinerary legs where the first one goes from Linköping University by taxi to Linköping airport and the second leg goes from Linköping airport to X using unspecified transportation. An example of a composition rule is a rule that combines two *Travel* concepts where the final destination of the first *Travel* concept is the place of departure of the second *Travel* concept.

• The **User and organization preference knowledge** provides preferences about *Travels*. For instance, we have the preference that if the place of departure is Linköping University and the place of arrival is not in Scandinavia then the travel’s first leg should be a taxi ride from Linköping University to Linköping airport.

• The **Query knowledge** contains the definition of the task according to the sButler’s internal representation model. In this case, we assume that the workflow has already been used before and that the Evaluation module has stored the information that the transaction performs a *Buy* activity and the desired result is a *OneWayTravel* concept.

In our prototype, the **Semantic Web** is represented as a set of six travel agencies providing nine different *ItineraryLegs*.

To start the application one executes the corresponding Java class (Window A in Figure 8.2). The OWL lite files are loaded into the Jess knowledge bases and the graphical representation of the organization’s workflow appears on the screen (Window B in Figure 8.2). The tasks that the sButler supports are highlighted in the workflow interface. When the user right-clicks on the Plan Travel task, she is presented with a menu and she may choose the Visualize option. This provides information on aspects that are associated with the sButler’s internal representation of the task. The ontology hierarchies of the Transaction and Domain knowledge are displayed and the concept that corresponds to aspects of this task is highlighted. Besides visualization, the user’s other option in the menu to start the process instance generation. This is done by selecting Provide input in the menu (Window C in Figure 8.2). The sButler is invoked and starts performing the process instance generation for this task as follows.

**Query Generation** Although the aspects of the task are already represented internally (see above), the query generation process still needs to acquire the data that is specific to the process instance. This data is added to the sButler’s task description. The type of data required is computed from the specified aspects. The data required for a *OneWayTravel* are the places of departure and arrival. A window is displayed for the user to specify this information (Window D in Figure 8.2). The user may enter Linköping University for departure and New York University for arrival.
Figure 8.2: Prototype screenshots
As the prototype assumes that the sButler task representations and the Web service representations are OWL-DTP descriptions, the sButler's task descriptions can be used as queries for Web services. Once the query is formulated, the user may proceed by selecting Execute on the menu (Window C in Figure 8.2).

**Task Decomposition** tries to decompose the desired result. The sButler identifies the relevant decomposition rules in the Decomposition knowledge and applies them (Window E in Figure 8.2). The result of the decomposition of the OneWayTravel from Linköping University to New York University provides the following Travel parts:

1. A taxi leg from Linköping University to Linköping airport
2. A set of legs from Linköping airport to New York University.

**Service Discovery** submits the queries for the different travel parts to the Jess knowledge base. The answer provides a set of candidate services for the travel parts.

**Instance Generation** puts the answers to the queries for travel parts together in order to provide complete solutions. The solutions to Queries 1 and 2 are combined to provide itineraries for the whole Travel. Each itinerary corresponds to a different composition of services.

**Instance Selection** The user is presented with the different itineraries and selects one. Each itinerary is represented by a graph where the cell is a location and the arrow represents a travel leg (Window F in Figure 8.2). Clicking on the leg provides specific information about the leg such as the name of the travel agency and the means of transportation. After the instance selection, the chosen itinerary is used as the sub-workflow specification for the Go on Travel task.

The knowledge bases are important components of our architecture. The fact that we used Jess knowledge bases allowed us to do consistency checking as well as verification of the knowledge. We based the domain knowledge on existing ontologies about travel, locations and time. Therefore, the generation of knowledge was not very time-consuming for this application. For larger applications, however, there may be a need to use information from different ontologies, and tools for merging and aligning ontologies (e.g. Lambrix and Tan (2006)) may have to be used. Another advantage of using ontologies is that the information can be more easily shared between applications.

The decomposition knowledge was manually created based on our knowledge about the domain. The most useful information in the domain knowledge for this purpose was (often implicit) information about part-of relations together with their ranges and domains. It may be interesting to investigate whether parts of the decomposition knowledge may be semi-automatically generated from the domain knowledge. The user and organization preference knowledge contains static rules. These rules may be generated by learning approaches,
as in adaptive systems. Transaction knowledge is based on the MIT process handbook and can be considered to be a shared ontology. We downloaded and translated the information in the handbook into our internal structure. This needs to be done only once and is reused for all tasks.

For our application, the system loaded the knowledge bases at start-up. This took ca 30 seconds on a SUN Ultra 10 workstation with a 440-MHz UltraSPARC-IIi architecture. From then on all interaction and processing was instantaneous.
Chapter 9

Platform: Illustration of Use, Evaluation, and Lessons Learned

To illustrate the use of the platform we show how service discovery technology was integrated and evaluated for a specific use case and with a specific set of Web resources. Lessons learned from the case study with respect to the platform’s ease of use are further discussed in Section 9.2.

9.1 Illustration

9.1.1 Assumptions

Our initial assumptions with respect to the use case, the service discovery technology, and the available Web resources are as follows. For the use case, we assume the Semantic Web to be an open world where requesters and service providers can specify the kind of transaction that they agree to participate in (e.g., buying, lending, acquiring for free). The service providers provide travel itineraries and requesters query for specific travel itineraries, and expect to get answers at least as quickly as when they consult a database of travel itineraries.

As for the service discovery technology, we adopt the underlying architecture provided by the operation component of the platform implementation introduced in Section 5.1. We specialize it as follows:

- The Semantic Web service language is OWL-DTP.

- The requester agent is a modified version of the sButler prototype introduced in the previous section such that it does not perform local decomposition and allows for the specification of the third DTP aspect: the Process constraints.
• The discovery agent implements the dummy default discovery agent behavior provided by the operation support of the platform: it receives queries and passes them on to all the available Web service manager agents.

• The Web service manager agent integrates the OWL-DTP matchmaking algorithm introduced in Section 7.5.4. This algorithm requires the use of description logic reasoning for which the Racer system is used. The Jena-DIG interface is used as a Java interface to Racer. The matchmaking algorithm is implemented in a straightforward way that requires each query to be matched against each service description. Each operation of matching a query and a service requires a set of reasoning operations including some subsumption operations. The reasoning about the concept of Travel is delegated to the specific Travel Ontology Agent.

• A Travel Ontology Agent is provided that is able to reason about travel specific notions such as travel departures and arrivals. The agent is also using Racer and the Jena-DIG interface as a description logic reasoning engine.

• The service provider agents also implement the default service provider behaviors provided by the operation support of the platform. Each service provider is assigned a list of services by the Setting component (see Section 5.3). The first action of the service providers is thus to send this list of services to all running Web service manager agents. They then enter a waiting loop where they wait for queries for service enactment. On reception of such queries, their dummy behavior consists of answering the requester with a text message that says “Received your query for my service with ID = id” (where id is the actual id of the queried service).

With respect to the Web resources available we consider a set of 183 services providing travel itineraries. These services correspond to those provided by the different Web sites of the travel providers that the employees of our department are authorized to use when planning work-related trips. We also consider a set of 14 queries corresponding to real travel queries expressed by some employees when planning their trips. There are two categories of queries: queries that will require a composition of services (e.g. “Give me an itinerary from Stanford University, Palo Alto, CA, USA, to the Conference center in Chiba city, Japan”), and queries for which service composition is not always necessary (e.g. “Give me all the flights from Heathrow Airport, London, UK, to Kastrup Airport, Copenhagen, Denmark”).

9.1.2 Integrating the assumptions in the evaluation platform

We integrated the above assumptions in the platform as follows:
package SWtestplatform.interfaces;

import SWtestplatform.types.Query;

public interface RequesterInterface{
    public void RequesterSendsQuery(Query q);
    public void RequesterGetsAnAnswerForQuery(Query q);
}

Table 9.1: Java interface for the Requester agent

- We created a new package Platform01 which contains a sub-package for each agent (i.e RequesterAgent, ServiceProvider, TravelOA, WSdiscovery, WSmanager), a settings file (Settings.java) and a loading file (Platform01.java).

- The settings file contains a list of Uris of services and queries. They describe the set of semantically annotated Web resources available.

- Each sub-package contains a Java file that describes the specific agent behavior. Each agent behavior is a new Java class that extends the predefined GameAgent class (which defines the basic FIPA agent behavior) and implements the specific interface corresponding to the API described in Chapter 5. For example, Table 9.1 shows the Java code of the interface (or API) for Requester agents provided in the predefined SWtestplatform.interfaces Java package. As a result, the Java code describing the agent’s behavior corresponds to a set of methods which embed different modules of predefined technology. The set of methods corresponds to a super set of the methods specified in the interface. For example, the Java code to describe the sButler agent provides three methods: RequesterSendsQuery and RequesterGetsAnAnswerForQuery, as required by the Requester agent interface, and RequesterFormulatesQuery, which is an extra Requester agent capability provided by the sButler model. Each method runs one of the existing sButler modules (see implementation description in Chapter 6). For example, the sButler’s RequesterFormulatesQuery runs the Query Generation module pictured in Figure 8.1, while the RequesterSendsQuery behavior runs the Service Discovery module. There are two possible ways to define the body of the methods: either with a call to some predefined code, or by writing the code directly in the method.

- We use the predefined monitoring agent with the MonitorAnswerTime monitoring behavior in order to measure the time to answer for the different queries.

- The Platform01 file allows to start up the FIPA multi-agent platform and schedule the order in which the agents initially appear. When starting up
an agent, it is possible to specify a set of predefined arguments, such as the Semantic Web service language that they understand. In order to fit the current scenario, we set up all agents’ Semantic Web service language to OWL-DTP.

9.1.3 Evaluation of the service discovery approach

Scalability

We measure the scalability of the service discovery approach with respect to the number of services and the technical capabilities of the machines running the agents, by measuring the average response time to the queries. To do that we use the MonitorAnswerTime behavior provided by the platform. Additionally, all the agents trigger an event when they send or receive a message. We run the simulation in different settings where the agents run on different machines.

This first set of evaluation runs teaches us that the triplet Jena-DIG-Racer cannot run the required knowledge base on a machine with too little CPU and RAM. Concretely, the reasoner freezes if it uses the Travel ontology agent knowledge base on a PC with an x96 Family 6 Model 1 stepping 9 processor (ca. 199 MHz) and 64 MB of RAM. Further, a reasoner that uses both the knowledge base for the Web service manager and the Travel ontology agent and runs on a PC with an Athlon processor (1.33 GHz) and 512 MB of RAM, freezes after treating a random number of queries (ten, eleven or even forty). We identified one machine setting that works well for our application: the reasoner that uses the Web service manager knowledge base runs on a PC with an Athlon processor (1.33 GHz) and 512 MB of RAM, and another reasoner that uses the Travel ontology agent knowledge base runs on a PC with an Intel Pentium M processor (ca. 1400 MHz) and 512 MB of RAM. Additionally, in the machine settings providing for the best average time for the set of 183 services, we obtain an average response time to the queries of approximately 14 minutes. This is clearly not an acceptable level of performance with respect to the use case. Upon more detailed inspection we find that the reason for this great delay in response time is that the current matchmaking approach performs approximately 300 subsumption operations per query. Most of these operations are required to match the travel itineraries.

Given these observations we have designed a new matchmaking algorithm such that the Web Service manager decomposes the OWL-DTP representation into three components and indexes them at service advertisement time. The indexing of the components referring to travel itineraries is performed by the Travel ontology agent, which stores the generated indexes in a database. The indexes are then used at query time. We change the behavior of the Web Service manager and Travel ontology agent to integrate the new algorithm. The new algorithm requires two subsumption operations and one SQL query to match a query with all the available services. Running the simulation now provides an

---

1 Jena-DIG related note: both knowledge bases are defined in their own model.
answer in 10 seconds on average. This is a result that better fits the use case requirements with respect to time, even if there is still room for improvement.

The monitoring also provides the time to advertise the services. With the straightforward algorithm, it takes ca. 28 seconds to advertise 183 services in one Web service manager. With the second version of the algorithm, it takes ca. 183 seconds to advertise 183 services in one Web service manager. The preprocessing done at advertisement time takes its toll. However, it is still a reasonable amount of processing time for advertisements since they need to be done only once per service in this use case.

**Result quality**

In order to measure the quality of the result we measure precision and recall for each query. This is done by implementing a monitoring behavior that compares the set of services returned for each query with a precompiled ideal set of services. The results show that we obtain 100% precision and recall for the 3 queries that request one specific travel leg (i.e. they correspond to one or several existing services), showing that the service description language is suitable for the corresponding information needs. For the other 11 queries that requested travel itineraries composed of several legs, and thus requiring service composition, we got 0% precision and recall. This result provides us with a clear next step for the development of a complete service discovery operation, namely to package a service composition algorithm as a Web Service discovery agent behavior and evaluate how that would influence the precision and recall of the corresponding queries.

9.1.4 Platform evaluation

We have illustrated how the platform was used in a case study. We showed how service discovery technology was evaluated and analyzed, in terms of scalability and result quality, and refined based on assumptions in the use case. This analysis helped us narrow down the main performance bottleneck of the technology. After fine-tuning the matchmaking algorithm the platform also facilitated the comparison with the previous version, while indicating the unwanted side effect of increased advertisement time that the new algorithm implied. All in all, the platform helped us maintain a high-level view of the service discovery problem, while facilitating our work on the details.

9.2 Lessons learned

When implementing the service discovery approaches described above, we noticed three clear advantages of using the platform. The first advantage concerned time gain at design time. When pondering how to implement the assumptions of our case study in a service discovery operation, the platform provided us with a clear model of the operation. We immediately identified the need for a
requester, a set of service providers and a Web service manager agent. The platform also made us consider the decomposition of the matchmaking algorithm so that the travel-related part of the reasoning would be delegated to a specific ontology agent. This is a good design choice if we consider that we will later want to extend the scope of services.

The second advantage concerned both debugging and the integration of the second version of the matchmaking algorithm. In both cases, because of the strongly decoupled architecture of the implementation, including the different behaviors implemented by each agent, the code rewriting could be done locally, requiring very little, if any, rewriting of the code of other behaviors.

The third advantage is also connected to ease and rapidity of implementation: the predefined package of messages allowed us to very quickly set up the communication between agents.

All this allowed us to concentrate on the one task that was really important to us as Semantic Web technology developers: integrating the matchmaking algorithm and evaluating its performance.
Chapter 10

Conclusion and Future Work

This thesis aims at supporting the efficient deployment of the Semantic Web. As mentioned in the introduction of Chapter 4, the incomplete control of the process of deployment together with the hugeness of the Web — in terms of available resources, users and worldwide size of the network — make it difficult to measure the capability and limits of the existing Semantic Web technology. This situation makes for a definite risk of adopting technology that is not the most suitable, and of overlooking some important technological problems. In order to allow the efficient and reliable development of Semantic Web technology, there is a strong need for monitoring and evaluation tools.

In this thesis we propose such a tool: an evaluation and simulation platform for the Semantic Web. We also provide some new Semantic Web technology to support the deployment of a service discovery application.

In the following we provide our concluding remarks on the evaluation and simulation platform, and the new Semantic Web technology that we have provided. We also discuss future work.

The evaluation and simulation platform

By providing a common evaluation platform for Semantic Web technology, we aim to support a better understanding of the capability of the available technology and create a place to publish and advertise technology and use cases (including ontologies), making them easily available in order to facilitate, encourage, and speed up the establishment of collaboration between different research and development teams. We believe that the evaluation feature of the platform provides an incentive to publish and share technology, as well as complementary means to document valid technology use.

This thesis provides the model to build the platform, a description of an implementation of the platform that focuses on supporting the integration and evaluation of service discovery technology, and an illustration of the use of this platform with a scenario of service discovery in the travel domain.
The platform model is based on our observation that developers and users of the different emerging Semantic Web applications tend to make different assumptions about three aspects of the Semantic Web: the use case, the technology, and the resources available on the Semantic Web. As a result, we propose a platform model that includes a model of the assumptions, a model of the Semantic Web and a model to automatically generate a Semantic Web simulation corresponding to a set of assumptions. Notably, the Semantic Web model includes an operation component that supports the representation of Semantic Web applications as multi-agent systems. The implementation provides an API and a communication protocol for developing service discovery applications as FIPA multi-agent systems where the agents communicate using machine-understandable messages. We used the platform to integrate service discovery technology that we implemented together with other available technology. By further allowing us to identify technological weaknesses and limiting the effort to change parts of the implementation, the platform demonstrated its capability to support the deployment of Semantic Web technology.

We envision the simulation and evaluation platform becoming a piece of Semantic Web technology that can be integrated into the Semantic Web. Today, the platform is meant to be a place for exchanges and common investigation of Semantic Web technology. In time, when more Semantic Web technology has been deployed and the Semantic Web has become the prime means of access to online data and services, the platform will lose its simulation function and evolve to be integrated into the Semantic Web as a Semantic Web agent that allows monitoring of specific scenarios and technology in the real Semantic Web. And, as is meant to be the case with Semantic Web agents, there will likely be several such agents monitoring different aspects of the Semantic Web, possibly generating data to support the Trust layer, the highest layer in the Semantic Web’s layer cake (see Figure 2.1 in Chapter 2).

The new Semantic Web technology for service discovery

The service discovery technology that we developed and tested with the platform strives to solve the problem of automatic integration of Semantic Web technology in organizational workflow environments. The technology consists of an integration model whose central element is a specific requester agent called sButler. The sButler performs process instance generation (which includes service discovery) and process instance enactment. Our work includes a description of the set of modules that compose process instance generation, as well as an implementation of the modules, a machine-understandable language to describe business processes called OWL-DTP and a theoretical and practical analysis of both sButler and OWL-DTP. The practical analysis shows that it is possible to integrate our technology with other existing technology (e.g. Racer.) The use of the platform allowed us to try different integration approaches and identify their possible weaknesses. The evaluation showed the feasibility of our OWL-DTP and sButler approach, as well as the current limits of the service discovery settings that we tested (i.e. no service decomposition is a strong limiting factor.)

Our work on service discovery strives to question current approaches and
show that there are ways to formulate and handle the problem of integration of Semantic Web technology in business process management, other than the mainstream (and not yet standardized) approaches such as WSMX and OWL-S. This is illustrated by our reflection on the design of Semantic Web service languages in Chapter 7. Moreover, our work on service discovery technology gave us a better grasp of the complexity of the task of developing Semantic Web technology. The work also triggered discussions with different organizations that showed the organizations’ acute need for service discovery on one hand, and for independence from technology shifts on the other hand. This underscores the need for an extensive library of technologies such as the one described in our model of the evaluation platform.

**Future Work**

There are many opportunities for future work. As mentioned at the end of the platform paragraph above, we envision that the platform will drastically evolve from a place to share and evaluate Semantic Web technology and scenarios into a monitoring agent that will dwell on the Semantic Web. To reach this stage, the platform must first become a popular place for technology exchange. This can be done by establishing collaborations with other research teams who develop different technologies. Moreover, a more complete library of available tools and more concrete evaluation results would also make the platform more attractive. One way to do that is to extend the implementation of sButler and OWL-DTP to make them useful in more use cases. For example, the capabilities of the current sButler implementation could be extended to correspond to the full sButler model introduced in Chapter 6. This model includes the capability of process instance enactment, as well as decomposition and evaluation modules. Further, the use of OWL-DTP could be facilitated by the design and implementation of an editing tool for services and queries, as well as an extension of the current integration of knowledge from the MIT process handbook.

Service composition is an essential and difficult operation on the Semantic Web. It also provides a natural and good practical context to acquire more insight and develop the technology related to the platform, sButler and OWL-DTP. This would allow the extension of the sButler implementation and the platform support, as well as the test suite of queries introduced in Section 7.7.5. Specifically, with respect to the platform support for service discovery, new non-dummy behavior(s) for Web service discovery agents could be defined and it would be possible to perform more evaluation of different composition approaches. With respect to the sButler, we are particularly interested in further developing the notion of local decomposition of queries as introduced in Chapter 6. This is because such decomposition allows consideration of specific contexts (i.e. each organization has its own context that includes user preferences, local routines, etc.) which may define specific composition issues that a more generic approach would overlook. Local decomposition is also a way to look at the possibility of distributing the decomposition effort in order to provide for efficient service discovery algorithms.
We want to conclude by acknowledging that this work has taught us how exciting, fast-changing and challenging the domain of Semantic Web technology truly is. We are more aware than ever that the technical choices that are being made today by the World Wide Web community are going to last and deeply influence the future capability of the Semantic Web. These choices must be made very carefully. As with all emerging technology there is a definite risk of going with the first ‘good enough’ solution instead of the best one. To support the sensitive process of standardization, holistic unbiased evaluation tools are needed. Our work participates in this effort.
REFERENCES

References


REFERENCES


Appendix A

OWL-DTP - version of 20051124

This appendix provides the OWL DL definition of the mark up language constructs for OWL-DTP. The code is also available online at the following http address: http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/index.shtml

```xml
<?xml version="1.0" encoding="ISO-8859-1"?
<!DOCTYPE rdf:RDF [ 
  <!ENTITY p "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-process.owl"> 
  <!ENTITY t "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-transaction.owl"> 
  <!ENTITY r "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-resource.owl"> 
]> 

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:p="&p;#
  xmlns:r="&r;#
  xmlns:t="&t;#
  xmlns=http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/OWL-DTP.owl#" 
  xml:base="http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/OWL-DTP.owl"> 

  <owl:Ontology rdf:about=""/>

</rdf:RDF>
```

135
<owl:Ontology>

<owl:Class rdf:ID="WebService">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTransaction"/>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasProcessConstraint"/>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>

  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasDesiredResult"/>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasTransaction">
  <rdfs:domain rdf:resource="#WebService"/>
  <rdfs:range rdf:resource="&t;#Transaction"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasDesiredResult">
  <rdfs:domain rdf:resource="#WebService"/>
  <rdfs:range rdf:resource="&r;#Resource"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasProcessConstraint">
  <rdfs:domain rdf:resource="#WebService"/>
  <rdfs:range rdf:resource="&p;#MITphProcess"/>
</owl:ObjectProperty>
</rdf:RDF>
Appendix B

MIT Process Handbook Structure - version of 20051124

This appendix provides the OWL DL definition of the markup language constructs to describe the activity structure of the MIT process handbook. The code is also available online at the following http address: http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/index.shtml

<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-structure.owl#"
>
    <owl:Ontology rdf:about=""/>

    <!-- Activity -->

    <owl:Class rdf:ID="Activity"/>

    <owl:ObjectProperty rdf:ID="hasDirectDecomposition">
        <rdfs:domain rdf:resource="#Activity"/>
        <rdfs:range rdf:resource="#DirectDecomposition"/>
    </owl:ObjectProperty>

    <owl:ObjectProperty rdf:ID="hasBundleCriterionSpecialization">
        <rdfs:domain rdf:resource="#Activity"/>
    </owl:ObjectProperty>

</rdf:RDF>
<rdfs:range rdf:resource="#BundleCriterion"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasDomainSpecialization">
  <rdfs:domain rdf:resource="#BundleCriterionSpecialization"/>
  <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:ObjectProperty>

</rdf:RDF>
Appendix C

Resource Ontology - version of 20051124

This appendix provides the Resource ontology for OWL-DTP. The code is also available online at the following http address: http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/index.shtml

<?xml version="1.0" encoding="ISO-8859-1"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   xmlns="http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-resource.owl#"
   xml:base="http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-resource.owl"
>

<owl:Ontology rdf:about=""/>

<owl:Class rdf:ID="Resource"/>

<owl:Class rdf:ID="Actor"
   rdfs:subClassOf rdf:resource="#Resource"/>
</owl:Class>

<owl:Class rdf:ID="AThing"
   rdfs:subClassOf rdf:resource="#Resource"/>
</owl:Class>

<owl:Class rdf:ID="Location"/>

<owl:Class rdf:ID="NonPhysicalLocation"
   rdfs:subClassOf rdf:resource="#Location"/>
<owl:Class rdf:ID="PhysicalLocation">
  <rdfs:subClassOf rdf:resource="#Location"/>
</owl:Class>

<owl:Class rdf:ID="InformationalEntity">
  <rdfs:subClassOf rdf:resource="#AThing"/>
</owl:Class>

<owl:Class rdf:ID="Contract">
  <rdfs:subClassOf rdf:resource="#InformationalEntity"/>
</owl:Class>

<owl:Class rdf:ID="Constraint">
  <rdfs:subClassOf rdf:resource="#InformationalEntity"/>
</owl:Class>

<owl:Class rdf:ID="Schedule">
  <rdfs:subClassOf rdf:resource="#InformationalEntity"/>
</owl:Class>

<owl:Class rdf:ID="Design">
  <rdfs:subClassOf rdf:resource="#InformationalEntity"/>
</owl:Class>

<owl:Class rdf:ID="Evaluation">
  <rdfs:subClassOf rdf:resource="#InformationalEntity"/>
</owl:Class>

<owl:Class rdf:ID="EvalParts">
  <rdfs:subClassOf rdf:resource="#Evaluation"/>
</owl:Class>

<owl:Class rdf:ID="EvalQuery">
  <rdfs:subClassOf rdf:resource="#Evaluation"/>
</owl:Class>

<owl:Class rdf:ID="EvalService">
  <rdfs:subClassOf rdf:resource="#Evaluation"/>
</owl:Class>

<owl:Class rdf:ID="Software">
  <rdfs:subClassOf rdf:resource="#InformationalEntity"/>
</owl:Class>

<owl:Class rdf:ID="Message">
  <rdfs:subClassOf rdf:resource="#InformationalEntity"/>
</owl:Class>
<?xml version="1.0" encoding="UTF-8"?>
<owl:RDF>
  <owl:Class rdf:ID="PhysicalEntity">
    <rdfs:subClassOf rdf:resource="#AThing"/>
  </owl:Class>

  <owl:Class rdf:ID="Hardware">
    <rdfs:subClassOf rdf:resource="#PhysicalEntity"/>
  </owl:Class>

  <owl:Class rdf:ID="MechanicalDevice">
    <rdfs:subClassOf rdf:resource="#Hardware"/>
  </owl:Class>

  <owl:Class rdf:ID="Electro-mechanicalDevice">
    <rdfs:subClassOf rdf:resource="#Hardware"/>
  </owl:Class>

  <owl:Class rdf:ID="Computer">
    <rdfs:subClassOf rdf:resource="#Electro-mechanicalDevice"/>
  </owl:Class>

  <owl:Class rdf:ID="Appliance">
    <rdfs:subClassOf rdf:resource="#Electro-mechanicalDevice"/>
  </owl:Class>

  <owl:Class rdf:ID="Robot">
    <rdfs:subClassOf rdf:resource="#Electro-mechanicalDevice"/>
  </owl:Class>

  <owl:Class rdf:ID="Money">
    <rdfs:subClassOf rdf:resource="#PhysicalEntity"/>
  </owl:Class>

  <owl:Class rdf:about="#Location">
    <rdfs:subClassOf rdf:resource="#Resource"/>
  </owl:Class>
</rdf:RDF>
Appendix D

Extract from the Transaction Ontology - version of 20051124

This appendix provides the Resource ontology for OWL-DTP. The code is also available online at the following http address: http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/index.shtml

<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE rdf:RDF [ 
  <!ENTITY p "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-process.owl">
  <!ENTITY s "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-structure.owl"> 
]> 

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:p="&p;#"
  xmlns:s="&s;#"
  xmlns="http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-transaction.owl#"
  xml:base="http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-transaction.owl"> 

<owl:Ontology rdf:about="" >
</owl:Ontology>
<owl:Class rdf:ID="AcquireNotForMoney">
  <rdfs:subClassOf rdf:resource="#Acquire"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&s;#hasBundleCriterionSpecialization"/>
      <owl:allValuesFrom rdf:resource="#BCSAcquireNotForMoney"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="PaymentByCreditCard">
  <rdfs:subClassOf rdf:resource="&p;#Payment"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&s;#hasBundleCriterionSpecialization"/>
      <owl:allValuesFrom rdf:resource="#BCSPaymentByCreditCard"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<!-- Bundle Criterion -->

<owl:Class rdf:ID="PayHow">
  <rdfs:subClassOf rdf:resource="&s;#BundleCriterion"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&s;#hasActivity"/>
      <owl:allValuesFrom rdf:resource="&p;#Payment"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="AcquireForWhat">
  <rdfs:subClassOf rdf:resource="&s;#BundleCriterion"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&s;#hasActivity"/>
      <owl:allValuesFrom rdf:resource="#Acquire"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Restriction>
  <owl:onProperty rdf:resource="&s;#hasDomain"/>
  <owl:allValuesFrom rdf:resource="#BCValuesAcquireForWhat"/>
</owl:Restriction>
</owl:Class>

<!-- Criterion Values -->

<owl:Class rdf:ID="BCValues"/>

<owl:Class rdf:ID="BCValuesPayHow"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="&s;#hasDomain"/>
  <owl:allValuesFrom rdf:resource="#BCValues"/>
</owl:Restriction>
</owl:Class>

<owl:Class rdf:ID="BCVPayWithCreditCard"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="&s;#hasDomain"/>
  <owl:allValuesFrom rdf:resource="#BCValuesPayHow"/>
</owl:Restriction>
</owl:Class>

<owl:Class rdf:ID="BCValuesAcquireForWhat"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="&s;#hasDomain"/>
  <owl:allValuesFrom rdf:resource="#BCValuesAcquireForWhat"/>
</owl:Restriction>
</owl:Class>

<owl:Class rdf:ID="BCValuesAcquireNotForMoney"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="&s;#hasDomain"/>
  <owl:allValuesFrom rdf:resource="#BCValuesAcquireNotForMoney"/>
</owl:Restriction>
</owl:Class>

<owl:Class rdf:ID="BCVBuy"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="&s;#hasDomain"/>
  <owl:allValuesFrom rdf:resource="#BCVBuy"/>
</owl:Restriction>
</owl:Class>

<!-- Criterion Specializations -->

<owl:Class rdf:ID="BCSBuy"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="&s;#hasBundleCriterionSpecialization"/>
  <owl:allValuesFrom rdf:resource="#BCSBuy"/>
</owl:Restriction>
</owl:Class>

<owl:Class rdf:ID="BCSBuy"/>
<owl:Restriction>
  <owl:onProperty rdf:resource="&s;#hasDomainSpecialization"/>
  <owl:someValuesFrom rdf:resource="#BCVBuy"/>
</owl:Restriction>
</owl:Class>
<owl:Class rdf:ID="BCSPaymentByCreditCard">
  <rdfs:subClassOf rdf:resource="&s;#BundleCriterionSpecialization"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&s;#hasBundleCriterion"/>
      <owl:allValuesFrom rdf:resource="#PayHow"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="BCSAcquireNotForMoney">
  <rdfs:subClassOf rdf:resource="&s;#BundleCriterionSpecialization"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="&s;#hasBundleCriterion"/>
      <owl:allValuesFrom rdf:resource="#AcquireForWhat"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
</rdf:RDF>
Appendix E

Extract from the Process Ontology - version of 20051124

This appendix provides the Resource ontology for OWL-DTP. The code is also available online at the following http address: http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/index.shtml

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rdf:RDF ["ENTITY s "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-structure.owl"> ]>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:s="&s;#
xmlns="http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-process.owl"
xml:base="http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-process.owl"
>
  <owl:Ontology rdf:about=""
</owl:Ontology>

  <owl:Class rdf:ID="MITphProcess">
    <rdfs:subClassOf rdf:resource="&s;#Activity"/>
  </owl:Class>
<owl:Class rdf:ID="Create">
   <rdfs:subClassOf rdf:resource="#MITphProcess"/>
</owl:Class>

<!-- to complete with the bundle criterion CreateWhat -->

<owl:Class rdf:ID="Make">
   <rdfs:subClassOf rdf:resource="#Create"/>
</owl:Class>

<!-- to complete with the bundle criterion MakeWhat -->

<owl:Class rdf:ID="MakeProduct">
   <rdfs:subClassOf rdf:resource="#Make"/>
</owl:Class>

<owl:Class rdf:ID="Payment">
   <rdfs:subClassOf rdf:resource="#MITphProcess"/>
</owl:Class>

<owl:Class rdf:ID="IdentifyNeedsOrRequirements">
   <rdfs:subClassOf rdf:resource="#MITphProcess"/>
</owl:Class>

<owl:Class rdf:ID="DetermineTiming">
   <rdfs:subClassOf rdf:resource="#MITphProcess"/>
</owl:Class>

<owl:Class rdf:ID="ReceivePhysicalResource">
   <rdfs:subClassOf rdf:resource="#MITphProcess"/>
</owl:Class>

<owl:Class rdf:ID="IdentifyOwnNeeds">
   <rdfs:subClassOf rdf:resource="#MITphProcess"/>
</owl:Class>

<owl:Class rdf:ID="Receive">
   <rdfs:subClassOf rdf:resource="#MITphProcess"/>
</owl:Class>

<owl:Class rdf:ID="Acquire">
   <rdfs:subClassOf rdf:resource="#MITphProcess"/>
</owl:Class>

</rdf:RDF>
Appendix F

Test Suite

This appendix provides some complement to the test suite code discussed in Section 7.7.5.

F.1 Disclaimer

In the following, the code examples for OWL-S and WSMO have been written based on the authors' understanding of the following language specifications:

**OWL-S 1.1**: OWL-S 1.1 Release (OWL-S, 2005).

The specification of OWL-S does not provide specific guidelines for writing Web service queries. As a result our proposed code may not correspond to the intended use of the language by the authors of the specification. However, the language allows these formulations.

**WSMO**: (Roman et al., 2005).

Moreover, for the OWL-DTP code we consider the language definition provided in Section 7.6. All of the OWL-DTP source code, ontologies and queries have been successfully validated with the wOWLidator (2005-2006) from BBN technologies.

F.2 Header of the Xth OWL-DTP query

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rdf:RDF [ 
  <!ENTITY ws "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/OWL-DTP.owl"> 
  <!ENTITY t "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-transaction.owl"> 
  <!ENTITY p "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-process.owl"> 
]>
```
F.3 The domain knowledge used in the OWL-S queries

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY c "http://www.daml.org/services/owl-s/1.1/Concepts.owl">
]>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns:ws="&ws;#"
  xmlns:d="&d;#"
  xmlns:t="&t;#"
  xmlns:p="&p;#
  xmlns="http://www.ida.liu.se/~iislab/projects/SWSlanguages/TestSuite/
domain-knowledge-owl.owl#"
  xml:base="http://www.ida.liu.se/~iislab/projects/SWSlanguages/TestSuite/
domain-knowledge-owl.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.daml.org/services/owl-s/1.1/Concepts.owl"/>
  </owl:Ontology>

  <!-- for Query01a -->
  <owl:Class rdf:ID="MontereyAirport">
```
F.4. The domain knowledge used in the WSMO queries

F.4 The domain knowledge used in the WSMO queries

Concept trip
departureAirport ofType airport
arrivalAirport ofType airport

Concept airport
hasCity ofType city
hasName ofType string

Concept city
hasName ofType string
locatedIn ofType country

Concept country
name ofType string

instance Monterey memberOf city
locatedIn hasValue USA

instance JFKAirport memberOf airport
hasName hasValue JFK
hasCity hasValue New York

instance USA memberOf country
    hasName hasValue USA

Concept Process

Concept Delivery subConceptOf Process
    deliveryTime hasType Time

Concept time

Concept oneWeek SubConceptOf time

Concept providedThing
    hasType ofType Thing
    hasQuantity ofType Number

Concept Bed

Concept ResearchArticle

Concept FreeThing

Concept TShirt

Concept EthicalRule
    nochildLabor hasValue boolean

Concept FootballMatchTicket

F.5 The domain knowledge used in the OWL-DTP queries

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE rdf:RDF [  
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema">  
<!ENTITY r "http://www.ida.liu.se/~iislab/projects/SWSlanguages/OWL-DTP/20051124/mitph-resource.owl"> ]>
<rdf:RDF  
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  
xmlns:owl="http://www.w3.org/2002/07/owl#"  
xmlns:r="#"  
xmlns="http://www.ida.liu.se/~iislab/projects/SWSlanguages/TestSuite/domain-knowledge.owl#"  
xml:base="http://www.ida.liu.se/~iislab/projects/SWSlanguages/TestSuite/" >
F.5. The domain knowledge used in the OWL-DTP queries

domain-knowledge.owl

<owl:Ontology rdf:about="">
</owl:Ontology>

<owl:Class rdf:ID="Flight">
  <rdfs:subClassOf rdf:resource="&r;#Resource"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasDeparture">
  <rdfs:domain rdf:resource="&#Flight"/>
  <rdfs:range rdf:resource="&r;#PhysicalLocation"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasArrival">
  <rdfs:domain rdf:resource="&#Flight"/>
  <rdfs:range rdf:resource="&r;#PhysicalLocation"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Airport">
  <rdfs:subClassOf rdf:resource="&r;#PhysicalLocation"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasCity">
  <rdfs:domain rdf:resource="&#Airport"/>
  <rdfs:range rdf:resource="&r;#City"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasName">
  <rdfs:domain rdf:resource="&#Airport"/>
  <rdfs:range rdf:resource="&r;#AirportName"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="AirportName"/>

<Airport rdf:ID="MontereyAirport">
  <hasCity rdf:resource="#Monterey"/>
</Airport>

<Airport rdf:ID="JFKAirport">
  <hasName rdf:resource="#JFK"/>
</Airport>

<AirportName rdf:ID="JFK"/>

<owl:Class rdf:ID="Country">
  <rdfs:subClassOf rdf:resource="&r;#PhysicalLocation"/>
</owl:Class>
<owl:Class rdf:ID="City">
  <rdfs:subClassOf rdf:resource="&r;#PhysicalLocation"/>
</owl:Class>

<City rdf:ID="Monterey"/>

<owl:Class rdf:ID="TShirt">
  <rdfs:subClassOf rdf:resource="&r;#PhysicalEntity"/>
</owl:Class>

<owl:Class rdf:ID="Bed">
  <rdfs:subClassOf rdf:resource="&r;#PhysicalEntity"/>
</owl:Class>

<owl:Class rdf:ID="ResearchPaper">
  <rdfs:subClassOf rdf:resource="&r;#PhysicalEntity"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasTitle">
  <rdfs:domain rdf:resource="#ResearchPaper"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="FootballMatchTicket">
  <rdfs:subClassOf rdf:resource="&r;#PhysicalEntity"/>
</owl:Class>

</rdf:RDF>
Appendix G

Acronyms

ACL  Agent Communication Language (FIPA terminology)
AID  Agent IDentifier (FIPA terminology)
API  Application Program(ming) Interface
BPEL / BPEL4WS Business Process Execution Language for Web Services
BPM  Business Process Management
CEL  Contract Expression Language
CORBA Common Object Request Broker Architecture
DAML  DARPA Agent Markup Language
DAML-S DAML based Web service ontology
DF  Directory Facilitator (FIPA terminology)
DL  Description Logic
DTP  Desired result, Transaction, Process
ERP  Enterprise Resource Planning
EU  European Union
FaCT  Fast Classification of Terminology
FIPA  Foundation for Intelligent Physical Agents
HTML HyperText Markup Language
HTTP HyperText Transport Protocol (W3C terminology)
IISLAB Intelligent Information Systems LABoratory
IRS  Internet Reasoning Service
IOPE Inputs, Outputs, Preconditions, Effects
MIT  Massachusetts Institute of Technology
NAICS North American Industry Classification System
NS  XML NameSpaces
OA  Ontology Agent
OFX  Open Financial eXchange
Oil / OIL Ontology Interchange Language
OilEd Oil Editor
OKBC  Open Knowledge Base Connectivity
OMG  Object Management Group
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>OWL DL</td>
<td>A sublanguage of OWL corresponding to a description logic</td>
</tr>
<tr>
<td>OWL full</td>
<td>The complete OWL language</td>
</tr>
<tr>
<td>OWL lite</td>
<td>A sublanguage of OWL that provides for a classification hierarchy and simple constraints</td>
</tr>
<tr>
<td>OWL-QL</td>
<td>OWL Query Language</td>
</tr>
<tr>
<td>OWL-S</td>
<td>OWL-based Web service ontology</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework (W3C terminology)</td>
</tr>
<tr>
<td>REWERSE</td>
<td>REasoning on the WEb with Rules and SEmantics</td>
</tr>
<tr>
<td>RTI</td>
<td>Real-Time Infrastructure</td>
</tr>
<tr>
<td>RuleML</td>
<td>Rule Markup Language</td>
</tr>
<tr>
<td>SHIQ</td>
<td>A description logic</td>
</tr>
<tr>
<td>SHOIQ</td>
<td>A description logic</td>
</tr>
<tr>
<td>SHOQ</td>
<td>A description logic</td>
</tr>
<tr>
<td>SL</td>
<td>Semantic Language (FIPA terminology)</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language (OMG terminology)</td>
</tr>
<tr>
<td>Unicode</td>
<td>Unique, Universal, and Uniform character enCoding</td>
</tr>
<tr>
<td>UNSPC</td>
<td>United Nations Standard Product and Services Classification</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier (W3C terminology)</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Location (W3C terminology)</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>WfMC</td>
<td>Workflow Management Coalition</td>
</tr>
<tr>
<td>WfMS</td>
<td>Workflow Management System</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Service Description Language</td>
</tr>
<tr>
<td>WSMF</td>
<td>Web Service Modeling Framework</td>
</tr>
<tr>
<td>WSMO</td>
<td>Web Service Modeling Ontology</td>
</tr>
<tr>
<td>WSMX</td>
<td>Web Service Modeling eXecution environment</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>


No 292 
No 297 
No 302 
No 312 
No 338 
No 371 
No 375 
No 383 
No 396 
No 413 
No 414 
No 416 
No 429 
No 431 
No 437 
No 439 
No 448 
No 452 
No 459 
No 461 
No 462 
No 475 
No 480 
No 485 
No 494 
No 495 
No 498 
No 502 
No 503 
No 512 
No 520 
No 522 
No 526 
No 530 
No 555 
No 561 
No 563 
No 567 
No 582 
Vanja Josifovski: Design, Implementation and
<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Title</th>
<th>ISBN</th>
</tr>
</thead>
<tbody>
<tr>
<td>589</td>
<td>Rita Kovordányi</td>
<td>Modeling and Simulating Inhibitory Mechanisms in Mental Image Reinterpretation - Towards Cooperative Human-Computer Creativity, 1999</td>
<td>91-7219-506-1</td>
</tr>
<tr>
<td>592</td>
<td>Mikael Ericsson</td>
<td>Supporting the Use of Design Knowledge - An Assessment of Commenting Agents, 1999</td>
<td>91-7219-532-0</td>
</tr>
<tr>
<td>593</td>
<td>Lars Karlsson</td>
<td>Actions, Interactions and Narratives, 1999</td>
<td>91-7219-534-7</td>
</tr>
<tr>
<td>595</td>
<td>Jörgen Hansson</td>
<td>Value-Driven Multi-Class Overload Management in Real-Time Database Systems, 1999</td>
<td>91-7219-542-8</td>
</tr>
<tr>
<td>597</td>
<td>Vivian Vimarlund</td>
<td>An Economic Perspective on the Analysis of Impacts of Information Technology: From Case Studies in Health-Care towards General Models and Theories, 1999</td>
<td>91-7219-544-4</td>
</tr>
<tr>
<td>598</td>
<td>Johan Jenvald</td>
<td>Methods and Tools in Computer-Supported Taskforce Training, 1999</td>
<td>91-7219-547-9</td>
</tr>
<tr>
<td>607</td>
<td>Magnus Merkel</td>
<td>Understanding and enhancing translation by parallel text processing, 1999</td>
<td>91-7219-614-9</td>
</tr>
<tr>
<td>611</td>
<td>Silvia Coradeschi</td>
<td>Anchoring symbols to sensory data, 1999</td>
<td>91-7219-623-8</td>
</tr>
<tr>
<td>618</td>
<td>Jimmy Tjäder</td>
<td>Systemimplementering i praktiken - En studie av logiker i fyra projekt, 1999</td>
<td>91-7219-657-2</td>
</tr>
<tr>
<td>637</td>
<td>Esa Falkenroth</td>
<td>Database Technology for Control and Simulation, 2000</td>
<td>91-7219-766-8</td>
</tr>
<tr>
<td>660</td>
<td>Erik Larsson</td>
<td>An Integrated System-Level Design for Testability Methodology, 2000</td>
<td>91-7219-890-7</td>
</tr>
<tr>
<td>689</td>
<td>Joakim Gustafsson</td>
<td>Extending Temporal Action Logic, 2001</td>
<td>91-7373-017-3</td>
</tr>
</tbody>
</table>


No 808 Klas Gäre: Tre perspektiv på förändringar och förändringar i samband med införande av informa-


Linköping Studies in Information Science


