A Flexible Model for the Semi-automatic Location of Services

A dissertation presented
by

Rubén Lara Hernández

to
The Department of Computer Science
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of

Computer Science

Universidad Autónoma de Madrid
Madrid, Spain
July 2007

Thesis advisor
Pablo Castells Azpilicueta

Author
Rubén Lara Hernández

A Flexible Model for the Semi-automatic Location of Services

Abstract

SOA is attracting increasing attention as an architectural paradigm which can enable a higher reuse of IT assets, their principled and eased integration into more complex services and, therefore, a more agile adaptation and evolution of IT systems to respond to business needs. The exposition of functionalities provided by heterogeneous and possibly distributed systems as reusable, platform-independent, interoperable, and meaningful (from a business point of view) services is the pillar of SOA; existing and new pieces of functionality are exposed so that they can be seamlessly accessed by other parties. However, for services to be used, they must be first located. As SOA adoption increases and more services are available, the difficulty of locating an appropriate service for solving some particular need can become a bottleneck for the effective exploitation of services unless appropriate location mechanisms are in place.

Current service technologies rely on purely syntactic descriptions, which limits the precision of service location mechanisms; as a result, a strong manual intervention from users is demanded, which hampers an agile usage of services and, especially, the dynamic location of services at run-time. Therefore, an enhancement of current service location practices in order to enable the effective and efficient location of available services for their usage, either with the intervention of a human user or in a fully automatic manner, either inside an organization or specially if organizational boundaries are crossed to extend the possibilities of cooperation, is required.

In this work, we introduce an explicit conceptual model of services and goals, analyze the variety of usage scenarios in which the need for enhanced service location mechanisms beyond current practices arises, and propose an abstract model for the semi-automatic location of services. This model is designed with flexibility and usability as key principles so that it can be used to cover diverse usage scenarios in which different requirements might arise and in which users with different profiles might participate.

While the general service location model proposed is kept abstract, an instantiation which concretizes aspects left open in the abstract model, such as the particular types of descriptions of goals and services considered and the matching mechanisms applied to them, is proposed, serving as a proof of concept of our approach and demonstrating the feasibility of enhancing current service location mechanisms. This instantiation of the abstract model has the following salient features: a) It admits both formal and non-formal descriptions of services and goals, and of different types, granting flexibility to users in choosing what types of descriptions and matchmaking mechanisms are used,

b) The descriptions used are integrated into an existing framework for the description of services (WSMO) and formal descriptions are expressed using existing languages with formal semantics (the WSML family of languages); still, portability of descriptions to other frameworks is possible, c) It admits formal descriptions with different semantics; in particular, it enables the combined use of descriptions with first-order and logic programming semantics for matching services and goals, d) The reasoning support required over formal descriptions is mostly provided by existing reasoning infrastructure, e) The alternative types of descriptions proposed and their associated matchmaking methods keep a balance between simplicity and coverage of application needs for different application types, f) Support to users for describing their services and goals is provided, and g) A prototype implementation, as well as an evaluation of the model based on this implementation, has been accomplished.

Contents

Acknowledgments

I would like to thank all those who have directly or indirectly made this work possible: Borja Foncillas and Pablo Castells for their trust, support, and endless patience; Sinuhe Arroyo, Jos de Bruijn, Alice Carpentier, Leonarda Haid-Garcia, Uwe Keller, Holger Lausen, Francisco Martin-Recuerda, and Axel Polleres for all their support while working at DERI Innsbruck and for contributing to parts of this work; Miguel Corella for his help and ideas for building up the SETA prototype; Michael Kifer for his major contribution to the model for service discovery based on transaction logic; the DERI Galway crew for the warm and motivating environment created while working at DERI Galway; the TIF and AFI crew, especially Daniel Manzano, for their support and understanding; and, finally, all my friends who gave me the energy and motivation I needed for ending this long trip.

Dedicated to Elvira
and to my parents.

Chapter 1

Introduction and summary

1.1 Context

Over the last decades, IT has gained an increasingly prominent role in many aspects of economy, business and society, leading to a massive creation of systems and applications which offer a vast variety of functionalities targeting at different needs, and which range from in-house solutions to widely accepted products.

This plethora of different systems and applications have not only being conceived to offer a huge diversity of functionalities, but they have also been created using different architectural styles and technologies. The different technology waves we have witnessed in the last decades, from mainframe centralized solutions to distributed solutions using technologies such as CORBA [COR, 1998], have left behind in many organizations a number of heterogeneous IT systems that today have to cooperate in different ways. However, realizing cooperations between such disparate systems is a challenging and costly endeavor. Even when cooperating systems have been created using the same technologies, they have often been conceived in isolation without considering their possible use in conjunction with other systems and applications. This results on e.g. the lack of adequate documentation or clear public interfaces to access the functionality of these systems, which in turn makes their integration challenging. Furthermore, in a networked economy, cooperation needs have crossed organizational boundaries and now expand different organizations. As a consequence, systems residing in different organizations have to interact in order to realize some business cooperation.

In general, the integration of different functionalities (or business services) provided by different IT systems, possibly located at distributed locations and expanding multiple business domains, has become an extended need. These systems have often been designed in isolation or using technologies hardly amenable to interaction with other systems and technologies, but still they have

to be reused and integrated, as existing IT assets and their associated investments cannot be simply replaced. This is leading to an increasing interest in the *service-oriented architectural paradigm* [Erl, 2005; Newcomer and Lomow, 2004], which promises an increased reuse of IT systems, a reduction of the integration burden, and an agile adaptation and evolution of IT assets to respond to business needs.

Still, some limitations, especially the rigidness of systems and processes built based on the service-oriented paradigm, are motivating a body of research on the application of formal semantics to Service-Oriented Architectures (SOA), the so-called *semantic SOA* view or *Semantically Enabled Service-Oriented Architectures (SESA)* [Brodie et al., 2005], as a semantic SOA might overcome some of the limitations of the current SOA view and potentially increase its benefits.

In the following, we briefly introduce the concept of SOA and its importance, the research field of Semantic Web and semantic technologies, and how and why these two worlds are starting to converge.

1.1.1 The SOA road

The exposition of functionalities provided by heterogeneous and possibly distributed systems as reusable, platform-independent, interoperable, and meaningful (from a business point of view) services is the pillar of Service-Oriented Architectures, which are receiving increasing attention as an architectural paradigm that can enable the reuse of IT assets and their principled integration into more complex services, both within and across organizational boundaries. In this way, previously isolated and heterogeneous pieces of functionality are given visibility and exposed so that they can be seamlessly located and accessed by other parties.

Common examples used to illustrate the benefits of exposing functionalities provided by different systems and actors are the following:

Example 1.1 A Virtual Travel Agency (VTA) offers on-line services for searching and booking travel packages. For that purpose, it integrates services from different providers such as airlines, car rental companies, hotels, etc, which might be provided by different systems [Stollber et al., 2004; He et al., 2004; Lara et al., 2004b]. The integration of the offers of these providers could be considerably eased if each provider would publish interoperable and meaningful services, usable by cooperating parties without requiring any specific integration effort. For example, an airline could publish a service for booking its flights using standard, interoperable technologies, thereby easing its integration into the VTA catalogue.

□

Example 1.2 An on-line book store has to interact not only with customers, but also with shipping companies and payment systems, among others, in order to provide its service to customers. This naturally requires integrating different systems used by different providers to e.g. request the shipping or the payment of a good. The publication of the functionalities offered by shipping companies and payment agencies as interoperable services can considerable reduce the integration effort necessary to establish the cooperation between the on-line book store and business partners. A similar example is presented in [Kopecky, 2005].

□

The availability of interoperable and meaningful services enables their composition into business processes with a dramatically reduced integration effort. By combining Business Process Management (BPM) [Smith and Fingar, 2003] and SOA, organizations can place their focus on explicitly defining processes at the business level and automating them based on existing, known services. This has a two-fold benefit: a) organizations can concentrate their efforts and resources on defining, managing and improving existing and new processes, as the system integration task is considerably eased, and b) the alignment of business units and IT departments is improved, so the communication between business and IT staff is, as requirements are communicated via explicitly defined (at a business level) processes, which can be automated by IT departments using existing services.

For these reasons, organizations are increasingly interested in the SOA paradigm as a promising path towards leveraging their IT systems, gaining flexibility, responding faster to market changes, and better aligning IT and business (see [Heffner et al., 2006]). This is naturally attracting the attention of software vendors and business consultants, who are devoting important efforts to the emerging SOA market [Cearley et al., 2005]. An evidence of this activity is the appearance of a new brand of products: the Enterprise Service Bus (ESB) [Chappell, 2004], as well as the generalized support for Web services [Alonso et al., 2003] and other technologies such as JMS [Monson-Haefel and Chappell, 2000] that can be used for the implementation of services in an SOA.

Following the SOA road is expected to bring an evolution of how IT systems are conceived today; this evolution will be deeper, and bigger its benefits, if a number of research challenges, such as the dynamic (re)configuration of architectures, dynamic connectivity, end-to-end security solutions, dynamic service discovery, automatic replacement of services, automatic service composition, and other challenges described in [Papazoglou et al., 2006], are properly addressed.

1.1.2 The semantic road

In parallel to the development of the SOA paradigm, the semantic Web vision has attracted considerable research efforts. The semantic Web has been defined in [Berners-Lee et al., 2001] by

Tim Berners-Lee, director of the World-Wide Web Consortium (W3C)[1], as:

"an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation. It is based on the idea of having data on the Web defined and linked such that it can be used for more effective discovery, automation, integration, and reuse across various applications"

But, why is an extension of the current Web necessary? In the following, we will try to provide a brief answer to this question.

The World Wide Web (WWW or Web for short) has been an impressive success, in terms both of available information and of the growth rate of users [Fensel and Musen, 2001]. This success has been based to a great extent on its simplicity, making easy for information providers and consumers to put new information on-line and to access it, respectively. However, the creation of new information, based solely on simple HTML tags and textual content, has some drawbacks [Ding et al., 2002]: searches are imprecise, users face the task of reading the documents retrieved in order to extract the information desired, and the task of maintaining consistency among information sources is overwhelming, resulting in sites containing inconsistent and/or contradictory information.

Example 1.3 Let us consider a user who wants to search double hotel rooms in Madrid, next to Sol square, in a 4-stars hotel. On the current Web, an average user would use his preferred search engine and type some of his requirements, browse the results given, and manually collect information about hotels which meet these requirements. Another user could also choose to visit a particular Web site offering search facilities for hotels, and this Web site would perform a more focused search for him. However, in the latter case the Web site would most likely not be searching the Web, i.e., web pages, but directly communicating with some pre-defined hotel information providers.

□

The idea of providing information on the Web with a well-defined, formal meaning, has arose as a potential improvement to the current Web, leading to the so-called semantic Web. The vision of the semantic Web is therefore to go beyond textual Web pages, describing the information published on the Web based on explicit, shared, and formal vocabularies: the so-called *ontologies* [Gruber, 1993].

The use of ontologies gives information a shared, formal meaning, which can enable computers to automatically process information on the Web, thus overcoming the limitations of the current Web and transforming it from a huge, distributed source of textual information in which computers only retrieve content from certain locations and visualize it according to some HTML tags, to a source of well-defined information with a meaning computers can "understand", thereby

[1] http://www.w3.org/

providing users with advanced services such as precise information gathering or information consistency checking.

Example 1.4 Let us reconsider the previous example, and let us imagine a (very simplified) tourism ontology which defines a concept *hotel*, with properties *category* and *location*, and a concept *hotel room*, with a property *number of occupants*. Hotel A can publish on his Web site not only a textual description of the hotel and its rooms, but also put on-line instances of concepts in the shared tourism ontology. For example, hotel A can be described as an instance of concept *hotel*, with its *category* property having a *5-stars* value, and its *location* property having value *Carmen street, Madrid*. Furthermore, the hotel can provide the detail of the rooms it offers by describing instances of the *hotel room* concept and specifying their number of occupants. Similarly, users can describe their needs using the same terminology provided by the tourism ontology. In this way, information available on the Web is given a well-defined meaning, overcoming the difficulties of processing textual information and enabling for better search capabilities, among other features.

□

1.1.3 Towards a semantic SOA

The application of formal ontologies has transcended the limits of the Web, and it has also been investigated in other fields such as information integration and management [Alexiev et al., 2005; Castells et al., 2004; Lara et al., 2006a], development of community portals [Ding et al., 2004; Lausen et al., 2005], and service-oriented computing. In particular, the application of formal semantics to SOA has been envisioned as a potential path for addressing some of the challenges mentioned in the research roadmap given by [Papazoglou et al., 2006]. The reason is that the SOA paradigm relies on the explicit, interoperable description of the services exposed, but the languages currently used for this purpose, such as WSDL [Christensen et al., 2001], are purely syntactic. As a consequence, for example searching services in a service-oriented environment yields imprecise results, resembling the problems searching on the Web has. Furthermore, the current descriptions of interaction models of services and of data exchanged between them is also syntactic, which keeps human users in the loop as they have to interpret these descriptions and to properly wire services together. The result is that services must be statically assembled, resulting on a highly rigid composition of services into complex functionalities or processes.

In this setting, the SOA road and the semantic road have started to converge into the so-called semantic SOA, where formal semantics are exploited to overcome the limitations of current technologies. The purpose is to increase the flexibility of IT systems, reducing their rigidness to make them dynamically adapt to changes on the environment (errors on services, required quality of service, etc.) or on business requirements (price constraints, constraints on cooperating partners,

etc.) and, in general, to address the research challenges summarized in [Papazoglou et al., 2006].

The vision of a semantic SOA is attracting the attention of both researchers and industry. This increasing interest in applying formal semantics to service-oriented architectures has resulted in e.g. the creation of the OASIS Semantic Execution Environment (SEE) technical committee[2], the creation of the semantic annotations for web service description language working group[3] at the W3C, and the submission of several proposals for semantically describing Web services to the W3C [Martin et al., 2004; de Bruijn et al., 2005a; de Bruijn et al., 2005c; Battle et al., 2005c; Battle et al., 2005b; Akkiraju et al., 2005].

1.2 Motivation and main objectives

The exposition of functionalities provided by heterogeneous and possibly distributed systems as reusable, platform-independent, interoperable, and meaningful (from a business point of view) services is the pillar of SOA; existing and new pieces of functionality are given visibility and exposed so that they can be seamlessly accessed by other parties.

SOA can enable a higher reuse of IT assets and their better integration into more complex services. However, for services to be used, they must be first located. As SOA adoption increases and more services are available, the difficulty of locating an appropriate service for solving some particular need can become a bottleneck for the effective exploitation of services unless appropriate location mechanisms are in place. The need for such mechanisms arises in different types of usage scenarios, namely: a) location of services at design-time for their composition or integration into more complex systems or processes, b) run-time location and usage of services, and c) location and usage of services by end human users.

For example, a telecommunications provider might define a new business process for informing their customers about special offers and promotions. This business process might involve activities realizable by using available services such as notifying customers by different means (SMS, e-mail, etc.) or registering what offers and promotions each customer was informed of. The professionals in charge of the definition and implementation of this process will want to locate, at design-time, appropriate services that can be statically incorporated into the process for performing these activities -case a) above-. Furthermore, if any of the services statically bound to the process fails at run-time, it is desirable to dynamically locate a new service which can replace the failed service -case b)-. Finally, end human users will want to locate services which can be used to fulfill their objectives -case c)- e.g. if a user wants to contract some of the promotions offered by the telecommunications provider, he will want to locate a service which can be used to perform the

[2] http://www.oasis-open.org/
[3] http://www.w3.org/2002/ws/sawsdl/

desired contracting.

Current service technologies rely on purely syntactic descriptions, which makes service location mechanisms operating over such descriptions imprecise; as a result, they demand a strong manual intervention from users, which hampers an agile usage of services and, especially, the dynamic location of services at run-time. Therefore, an enhancement of current service location practices in order to enable the effective location of available services for their usage, either with the intervention of a human user or in a fully automatic manner, either inside an organization or specially if organizational boundaries are crossed to extend the possibilities of cooperation, is required.

Works exist trying to address the increasing need for efficiently and effectively locating services based on the semantic (formal) description of services and goals. However, most existing proposals are based on the input-output signature of services and not on the description of their value and, furthermore, they are limited to the usage of a single type of formal description of services and goals and they lack a comprehensive model for the location of services which grants flexibility to service providers and consumers in the type of descriptions used and in the mechanisms employed for locating services. Furthermore, existing works lack an analysis of the usage scenarios that must be covered by the proposed service location model, of the practical considerations which must be considered in such usage scenarios, of usability issues that might arise, and, in general, they lack a global view on the service location problem.

In this setting, we perceive the need for a model which enables the semi-automatic location of services based on the value they provide and on the value required by consumers, which is flexible enough to cover different usage scenarios, and which pays attention to the usability of the model by users with different profiles and needs. The elaboration of such a model, contributing to the enhancement of current service location practices, is one of the main objectives of the work presented in this document.

The model which will be elaborated has the purpose of providing general guidelines and design decisions for the location of services, but it will be kept abstract and particular instantiations must concretize aspects which are left open, such as the particular types of descriptions of goals and services considered and the matching mechanisms applied to them. It is also a major objective of our work to provide such an instantiation, as well as a prototype implementation, of the abstract model proposed. This instantiation will serve as a proof of concept of our proposal, will enable an evaluation of the abstract model based on this particular instantiation, and will demonstrate the feasibility of enhancing current service location practices.

1.3 Reader's guide

The necessary background for situating the work presented in this document and making a proper understanding of it possible is given in Chapter 2. Section 2.1 focuses on providing an overview of the SOA paradigm, of its main principles, of web service technologies, and of the research challenges in the SOA field which remain unsolved. In Section 2.2, the logical foundations necessary for understanding the work which will be presented in the document are provided. Furthermore, the basics of ontologies and the semantic web, as well as of the languages currently proposed for the formal description of domain knowledge, are given in this Section.

In Chapter 3 we provide a conceptual view of the core elements of an SOA that are particularly relevant to our work, such as services and goals (Section 3.2). This Chapter, besides introducing a (partial) reference model for SOA, investigates further the nature of the elements introduced in the conceptual model. In particular, a deeper insight into the type of artifacts we aim at describing for their semi-automatic location is provided by means of a formal characterization of them based on Transaction Logic (Section 3.3), as it is crucial to provide a clear and explicit model of these artifacts so that we can properly understand what types of descriptions can be provided, what aspects they capture, and what level of confidence can be expected from the results of the location process based on the different types of descriptions. Finally, in Section 3.4, we briefly introduce the frameworks proposed so far for the description of services and goals, and discuss how our conceptual model relates to the model of services and goals used by these frameworks.

In Chapter 4 we summarize the work we carried out to automate the location of services based on Transaction Logic, the results obtained, and the limitations found. This work was a first attempt we made for enhancing service discovery; limitations were found which motivated the initiation of our efforts to find a more comprehensive model which can properly work in different situations and which grants users more flexibility, but still the work has some interesting features which worth looking at.

Chapter 5 introduces our proposed abstract model for the location of services, driven by practical requirements and considerations. In Section 5.2, we characterize the main groups of applications we envision for the partially or fully automated location of services and discuss its key characteristics. Our proposal for an abstract model for the location of services will be described in Section 5.3, placing special emphasis on the principles and practical observations which have motivated the design of the model. The first part of the model proposed, which covers the description of services and the publication of these descriptions, will be discussed in Section 5.3.1; the second part of the model concentrates on the description of goals, and it will be presented in Section 5.3.2; the last part of the model, which covers the actual location and selection of services which can achieve an explicitly described goal, will be presented in Section 5.3.3.

In Chapter 6, the proposed instantiation of the first part of the abstract model presented in the previous Chapter, dealing with the description and publication of services, is detailed, as well as an associated prototype implementation (the SETA service location platform). Section 6.2 introduces the types of descriptions of services admitted by the model instantiation, while Section 6.3 will be devoted to the presentation of how service providers are supported in describing their services and how such descriptions are published.

Chapter 7 describes how, and following the guidelines provided by the abstract model in Chapter 5, we articulate the discovery of services based on their value, i.e., how we instantiate the second part of such abstract model in the SETA platform. In Section 7.2 we discuss how consumers can describe their goals in our model instantiation, how consumer knowledge is expected to be described, as it might play a role in deciding what services can be used to achieve a consumer's goal, and what type of support is offered to consumers for describing their goals. Once the consumer goal has been described, we will initiate the service discovery process, which is split into two phases presented in Sections 7.3 and 7.4.

Chapter 8 provides in Section 8.1 an evaluation of the model instantiation and the prototype implementation proposed, and an overview of related work in Section 8.2.

Finally, Chapter 9 concludes our work with a summary of major contributions and future work.

Chapter 2

Background

2.1 The SOA paradigm

Software reuse has been for years a research topic in computer science, as the need for exploiting existing pieces of functionality and capitalizing existing IT investments has soon arose in the brief history of this science.

First, the concept of modular development and structured programming emerged [w. Dijkstra, 1972], enabling the reuse of functions and routines previously developed with languages such as Pascal, Ada or C. However, and while improvements in the development of new systems and functionalities were achieved, maintainability remained an issue. The next paradigm which emerged to try to address the problems and limitations of structured programming was object-orientation [Dahl, 1987]. With languages such as C++, Java or C#, this paradigm brought a much more effective way of composing applications. Later, the challenge was to go beyond reusing code, and the reuse of components [Brad J. Cox, 1991] was sought. Proposals like DCOM, CORBA, J2EE or the .NET Framework contributed to this goal, but the use of components still left some issues unsolve like platform heterogeneity, protocol heterogeneity, and device heterogeneity.

In this context, the paradigm of service-orientation (or Service-Oriented Architectures -SOA-) [Erl, 2005; Newcomer and Lomow, 2004; Wikipedia, 2005] is gaining momentum as an architectural paradigm which can enable the engineering of new applications in terms of loosely-coupled, reusable and interoperable services, thereby overcoming the limitations of existing paradigms and practices. In the following, we will summarize the principles of SOA design, what web services are and how they relate to SOA, and the main open challenges in SOA development.

2.1.1 Principles

A number of similar and sometimes complementary definitions of SOA can be found; in our opinion, some of the most relevant definitions of SOA are:

1. "A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations [MacKenzie et al., 2006].

2. "A software architecture that uses loosely coupled software services to support the requirements of business processes and software users" [Wikipedia, 2005].

3. "The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a simple, standards-based form of interface" [Sprott and Wilkes, 2004].

4. "Information technology approach or strategy in which applications make use of (perhaps more accurately, rely on) services available in a network such as the World Wide Web" [Ort, 2005].

From these definitions we can extract the defining principles of SOA: reuse, interoperability, abstraction, loose coupling, and different ownership domains. A more detailed summary of the specific architectural principles of SOA can be found in [Wikipedia, 2005]:

1. Service encapsulation: services encapsulate a given piece of functionality or, more generally, a value.

2. Service loose coupling: services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.

3. Service contract: services adhere to a communications agreement, as defined collectively by one or more service description documents.

4. Service abstraction: beyond what is described in the service contract, services hide business logic from the outside world.

5. Service reusability; logic is divided into services with the intention of promoting reuse.

6. Service composability: collections of services can be coordinated and assembled to form composite services.

7. Service autonomy: services have control over the logic they encapsulate.

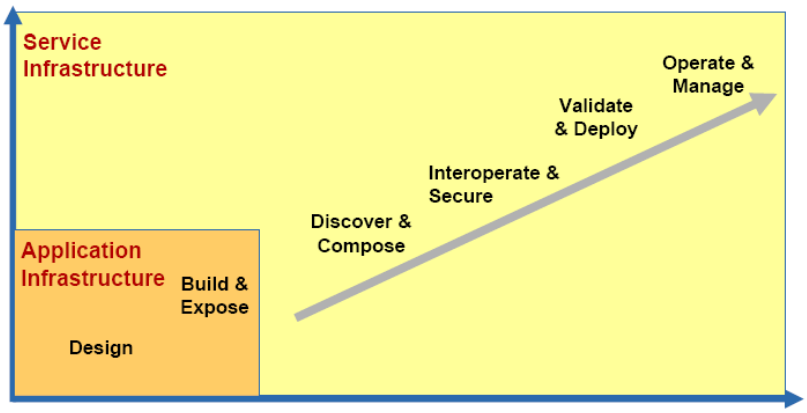

Figure 2.1: Services lifecycle

8. Service statelessness: services minimize retaining information specific to an activity.

9. Service discoverability: services are designed to be outwardly descriptive so that they can be found and accessed via discovery mechanisms.

The principles summarized above enable the development and exposition of services which can be reused across platforms, ownership domains, and protocols, and which can be composed in order to build complex functionalities from available services, that is, they enable the exposition of functionalities provided by heterogeneous and possibly distributed systems as reusable, platform-independent, interoperable, and meaningful (from a business point of view) services, thereby improving the reuse of IT assets and their principled integration into more complex services, both within and across organizational boundaries. In this way, previously isolated and heterogeneous pieces of functionality are given visibility and exposed so that they can be seamlessly located and accessed by other parties (see Figure 2.1). In this document, we will focus on the enhancement of the location (discovery) task depicted in the Figure, necessary for an efficient composition and exploitation of services.

SOA is receiving considerable attention and its adoption is spreading (see [Cearley et al., 2005] and [Heffner et al., 2006]). The wide-spread adoption of SOA principles is, though, still in progress, and the level of application and exploitation of this kind of architectures is at different stages in different organizations (see Figure 2.2).

2.1.2 Web services

SOA, or service-oriented computing, is an architectural paradigm and, as such, it is technology independent. However, there exist languages, technologies and infrastructure which can be used to realize a service-oriented architecture. Popular technologies for the realization of SOA include

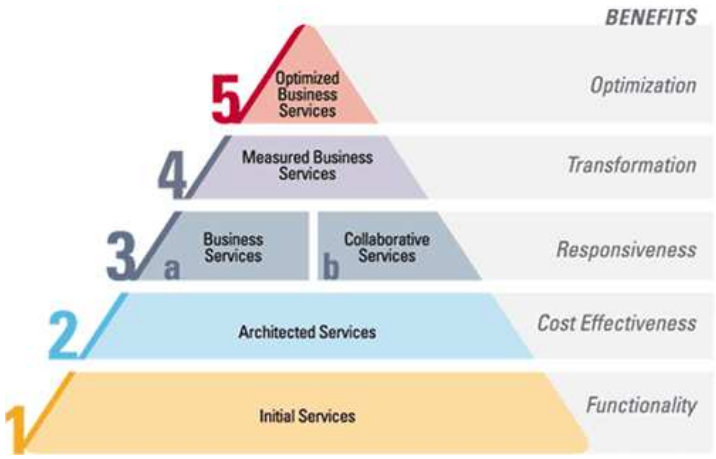

Figure 2.2: SOA maturity pyramid

web services [Alonso et al., 2003], which allow for the exposition of functionalities in an explicit, interoperable way, Business Process Modelling (BPM) [Smith and Fingar, 2003], which allows for the principled combination of services into business processes, and Enterprise Service Buses (ESBs) [Chappell, 2004], which aim at providing a middleware layer with features such as (reference and temporal) decoupling, data mediation, or intelligent routing of messages. This middleware layer can be convenient in different scenarios for actually building a service-oriented architecture.

From these key technologies, we will in the following focus on the description of what web services are and on the languages and interfaces they commonly rely on.

2.1.2.1 Definition of web services

Several definitions of what a web service is can be found in the literature. As a starting point, three definitions can be used:

1. "A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a format that machines can process (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with XML serialization in conjunction with other web-related standards" [Haas and Brown, 2004].

2. "Loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols [Sleeper, 2001].

3. "Self-contained, self-describing, modular applications that can be published, located and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes" [Tidwell, 2000].

What these definitions have in common is their characterization of Web Services as components providing functionality, distributed and accessible using Web-related standards. No restriction is given about the kind of functionality a Web Service can provide. It can provide static or dynamic information or perform real changes in the world, but what is essential in web services is their capability of providing functionality in a distributed manner within or across organizational boundaries using widely accepted technologies.

Web services provide access to some component or functionality, but in an interoperable manner and without revealing how this component and functionality is actually implemented. Furthermore, they describe, in a stardard way, how to access this functionality using platform-independent, interoperable protocols. In general, they provide loose coupling, interoperability, reusability and encapsulation. However, keep in mind that SOA is not equivalent to the use of web services. While this is a common misconception, web services are only a particular means to describe and make accessible interoperable, loosely-coupled services.

As example web services, we can imagine a web service which encapsulates the functionality of providing a list of available flights with certain constraints e.g. origin, destination and date, or a web service enabling the purchase and shipping of a book given its title, a payment method, and a shipping address. These web services can internally use other web services to perform their functionality. For example, the book purchase web service may use another web service to get the ISBN given the book title, and a shipping web service offered by a third party to order the shipping of the book to the buyer.

2.1.2.2 SOAP, WSDL and UDDI

Web services encapsulate functionality and enable access to any component in an interoperable way. For this purpose, web services make use of three main specifications which enable the description of web services, the exchange of messages with web services, and the location of web services, namely: WSDL, SOAP and UDDI (see Figure 2.3). These specifications are briefly introduced in the following.

SOAP The Simple Object Access Protocol (SOAP) [W3C, 2003] is a message layout specification that defines a uniform way of passing XML-encoded data. It also defines a way to bind to HTTP as the underlying communication protocol for passing SOAP messages between two endpoints. It overcomes the problems with techniques such as DCOM, RMI and CORBA, which are successful on

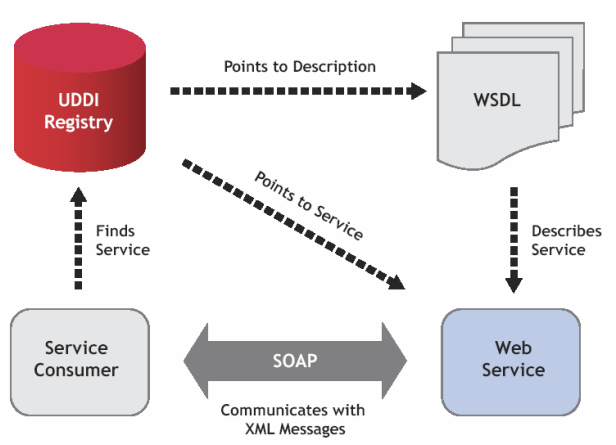

Figure 2.3: Relation among SOAP, WSDL and UDDI

the local network, while failing when transposed to a web environment. These techniques require a tight coupling between components and present security problems; firewalls and proxy serves will normally block RPCs between DCOM, RMI and CORBA components. Replacing this by a simple, lightweight RPC-like mechanism is the aim of SOAP. SOAP uses XML messaging over plain HTTP, thus avoiding firewall problems (asynchronous communication can also be accomplished via SMTP).

SOAP is platform and language independent, as well as simple and extensible, providing a way of communication between applications running on different operating systems with different technologies and programming languages. These features make SOAP a powerful way of exchanging messages in distributed and heterogeneous environments. In a nutshell, SOAP enables the exchange of messages in a platform and language-independent fashion. For more details on SOAP, we refer the reader to [W3C, 2003] and [Alonso et al., 2003].

WSDL The Web Services Description Language (WSDL) [Christensen et al., 2001] is an XML-based language for describing web services and how to access them. The description of services is written in XML, and it specifies the operations the service exposes. It provides a communication level description of the messages and protocols used by a web service. WSDL enables to separate the description of the abstract functionality offered by a service from concrete details of a service description such as "how" and "where" that functionality is offered.

The first component of a WSDL definition is the message component (see Figure 2.4). It describes the abstract format of a particular message that a web service sends or receives. The format of a message is typically described in terms of XML element information items and attribute information items. A message binding describes how the abstract content is mapped into a concrete format. A message consists of parts that describe a portion of the message that a web service sends

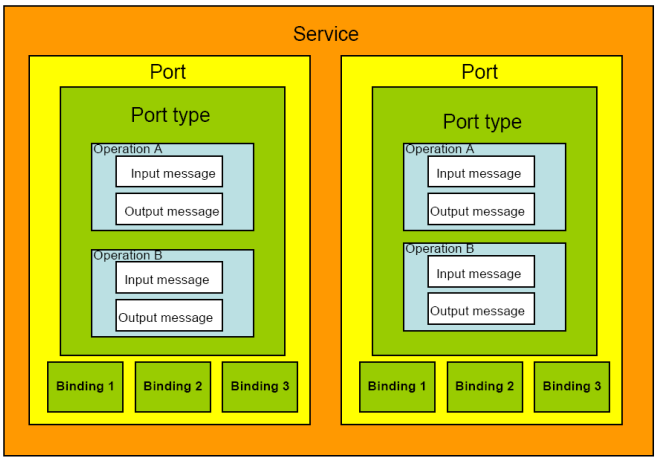

Figure 2.4: Graphical representation of a WSDL service.

or receives.

Once the messages to be interchanged by the service have been defined, they have to be related into operations the service is exposing, and these operations have to be grouped together. Operations are a set of message references to messages this operation accepts (input messages), or messages this operation sends (output or fault messages). A port type component groups operation together, thus grouping related operations into a wider message exchange.

The messages and port types defined, with their corresponding parts and operations, are kept abstract. A binding component describes a concrete binding of a port type component and associated operations to a particular concrete message format and transmission protocol. No concrete binding details are given in the WSDL specification, and a different specification defines such bindings for SOAP 1.1, SOAP 1.2, HTTP and MIME. A port component defines the particulars of a specific end-point at which a given service is available, that is, it defines the association of a port type with a binding. In this way, the operations exposed can be programmatically accessed.

The upper component WSDL defines is the service component, which describes the set of ports a service provides, thus exposing what operations can be invoked over the service and what messages must be exchanged.

The WSDL specification defines mechanisms to allow the separation of these components and thus their modularization. In addition, WSDL allows the inclusion of human readable documentation of any element.

UDDI Universal Description, Discovery and Integration (UDDI) [Bellwood et al., 2002] provides a mechanism for clients to find web services. Using a UDDI interface, businesses can dynamically look up as well as discover services provided by external business partners. A UDDI registry is similar

to a CORBA trader, or it can be thought of as a DNS service for business applications.

Central to UDDIs purpose is the representation of data and metadata about web services. A UDDI registry offers a standard mechanism to classify, catalogue and manage web services, so that they can be discovered and consumed. Businesses and providers can use UDDI to represent information about web services in a standard way, and queries can be issued to a UDDI Registry at design-time or run-time for finding web services which meet criteria of the following types: a) the web service has a given abstract interface definition, b) it is classified in a given way according to a known classification scheme or identification system, c) it supports certain security and transport protocols, or d) it contains certain keywords.

A UDDI registry has two kinds of clients: businesses that want to publish a service description (and its usage interfaces), and clients who want to obtain services descriptions of a certain kind and bind programmatically to them (using SOAP). UDDI itself is layered over SOAP and assumes that requests and responses are UDDI objects sent around as SOAP messages.

UDDI information contains three levels: the top level element is the Business entity, which provides general data about a company such as its address, a short description, contact information and other general identifiers. This information can be seen as the white pages of UDDI. Associated with each business entity is a list of Business services. These contain a description of the service and a list of categories that describe the service, e.g. purchasing, shipping etc. This can be considered as the yellow pages of UDDI. Within a business service, one or more Binding templates define the green pages: they provide the technical information about a web service.

An additional element involved in the general UDDI architecture is Technical Models, or tModels for short. They are used to represent unique concepts or constructs, providing a structure that allows reuse and, thus, standardization within a software framework. The UDDI information model is based on this notion of shared specifications and uses tModels to engender this behavior. For this reason, tModels exist outside the parent-child containment relationships between the businessEntity, businessService and bindingTemplate structures. Each distinct specification, transport, protocol or namespace is represented by a tModel. Examples of tModels that enable the interoperability of web services include those based on WSDL, XML Schema Definition (XSD), and other documents that outline and specify the contract and behavior, i.e., the interface that a web Service may choose to comply with.

To describe a web service that conforms to a particular set of specifications, transports, and protocols, references to the tModels that represent these concepts are placed in the bindingTemplate. In this way, tModels can be reused by multiple bindingTemplates. The bindingTemplates that refer to precisely the same set of tModels are said to have the same *technical fingerprint* and are of the same type. In this way, tModels can be used to promote the interoperability between software systems.

With this basic architecture, UDDI enables the publication and location of web services. Together, WSDL, SOAP and UDDI provide the basic infrastructure to describe, publish, locate and use web services in the way depicted by Figure 2.4.

Web services are widely supported by existing frameworks such as .NET[1] and J2EE[2], and interoperability between web services regardless of how they were created and their underlying technology is being enforced by organizations such as the Web Services Interoperability organizaion (WS-I)[3], which defines profiles interoperable web services must comply with.

Besides the core web service specifications presented above, other specifications aiming at covering aspects of web services not addressed by WSDL, SOAP and UDDI exist or are under development, such as WS-Security [Nadalin et al., 2004] or WS-Policy [WSP, 2006]. However, we will not cover them here; we refer the reader to the correspondent specification documents for details on these efforts.

2.1.3 Challenges

While service-orientation is widely regarded as the way IT systems must be conceived in order to improve reusability, interoperability and integration, there are challenges that still remain unsolved in order to fully exploit the potential benefits of the SOA paradigm. The service-oriented computing research roadmap elaborated by Papazoglou et al. in [Papazoglou et al., 2006] gives a very good overview of these challenges and discusses their implications and future steps. Research in service-oriented computing is subdivided into several areas or planes, namely: i) service foundations, ii) service composition, iii) service management, and iv) service engineering (Table 2.1).

Open challenges include an increase in the dynamics of service-oriented systems, in particular their dynamic reconfiguration and enhanced service discovery. These challenges are included as part of research in service foundations, and [Papazoglou et al., 2006] establishes that:

- The run-time service infrastructure should be able to configure itself and be optimized automatically in accordance with specific application requirements and high-level policies representing business-level objectives, for example that specify what is desired (such as particular security and privacy requirements) and not how it is to be accomplished.
- The main challenge of service discovery is the use of automated means for accurate discovery of services in a manner that demands minimal user involvement. Improving service discovery requires explicating the semantics of both the service provider and the service requester. Improving service discovery involves adding semantic annotations and including descriptions

[1] http://en.wikipedia.org/wiki/Microsoft_.NET_Framework
[2] http://en.wikipedia.org/wiki/Java_Platform%2C_Enterprise_Edition
[3] http://www.ws-i.org/

	State of the art		Grand challenges
Service foundations	*Enterprise Service Bus*	>Open standards-based message backbone >Implementation, deployment, management >Set of infrastructure capabilities implemented by middleware technology >Implementation backbone for SOA (applications as services)	>Dynamically (re)configurable run-time architectures >Dynamic connectivity capabilities >Topic and content-based routing capabilities >End-to-end security solutions >Infrastructure support for application integratino >Infrastructure support for data integration >Infrastructure support for process integration >Service discovery
Service composition	*Orchestration*	>Service interaction at message level >Perspective and control of single endpoint >Executable business process	>Composability analysis operators for replaceability, compatibility, conformance >Autonomic composition of services >QoS-aware service composition >Business-driven automated composition >Typing/syntactic conformance >Behavioural conformance >Semantic conformance
Service management	>Web Services Distributed Management (WSDM) >Management Using Web Services (MUWS) >Management of Web Services (MOWS)		>Self-configuring services >Self-healing services >Self-optimizing services >Self-protecting services
Service design and development (Service engineering)	>Port existing components using wrappers >Components-based development, object-oriented analysis and design >Do not address key elements: services, composition, components realizing services >Only address part of the requirements		>Design principles for engineering service applications >Associating a services design methodology with standard software development and business process modelling techniques >Flexible gap analysis techniques >Service Governance

Table 2.1: Overview of state of the art and grand challenges in service-oriented computing research [Papazoglou et al., 2006].

of QoS characteristics to service definitions in WSDL and then registering these descriptions in registries. The use of standard ontologies that support shared vocabularies and domain models for use in the service description also facilitates service discovery by making the semantics implied by structures in service descriptions explicit. To achieve automated discovery of services, the needs of service requesters have to be explicitly stated. We expect such needs to be expressed as goals, which correspond to the description of what services are sought, in some formal request language.

We can thus see that the location of services which can fulfill a given need is recognized as a key research challenge to be addressed in order to increase the dynamics of service-oriented architectures. Contributing to the resolution of this challenge is precisely the aim of the work we present in this document.

2.2 Logical Foundations and the Semantic Web

In [Papazoglou et al., 2006] and other works e.g. [Lara et al., 2003], the semantic description of services and of consumer needs using ontologies is identified as a promising path towards enhancing service discovery. In this Section, we will present what ontologies are and their logical foundations, as well as particular languages proposed for the description of ontologies. We will also briefly introduce the semantic web vision, as this vision has led to the establishment of an important research area in the engineering and use of ontologies, and it has been the origin of the developments in the definition of ontology languages and in the practical application of ontologies to the resolution of

different problems.

2.2.1 Ontologies and the semantic Web

The World Wide Web has made a huge amount of information electronically available, being an impressive success story in terms of both available information and the growth rate of human users [Fensel and Musen, 2001]. The Web has evolved from an in-house solution for around 1000 users in 1990 to more than 1 billion users (16.9% of the world's population)[4] and over 11.5 billion web pages in the publicly indexable Web as of the end of January 2005[5], not only world-wide but device-wide. This success has been based mainly on its simplicity, giving software developers, information providers and users easy access to new content. Nevertheless, the same simplicity that made the impressive expansion of the Web possible has brought important, and in some cases critical, drawbacks that are hampering a further development of the Web. These problems are experienced by users in their daily use of available contents. As an example, if we try to use any search engine to find on the Web airlines which offer flights from Munich to Madrid and we introduce a text query like "Airlines Munich Madrid", we will get thousands of results and we will be forced to navigate through some of the links to figure out whether the web sites listed are providing any useful information for our purpose.

The limitations of the current Web are summarized in [Ding et al., 2002]: searches are imprecise, often yielding matches to many thousands of pages; moreover, users face the task of reading the documents retrieved in order to extract the information desired. A related problem is that the maintenance of web sources has become very difficult. The burden on users to maintain consistency is often overwhelming. This has resulted in a vast number of sites containing inconsistent and/or contradictory information.

The problem current Web is suffering from is, as highlighted before, its simplicity. When an agent (either a human user or a machine) requests a web page, it only receives textual information together with some rendering (visualization) tags. The information received has neither structure nor explicit meaning, and the result is just a visualized version of the textual information based on the accompanying tags. Thus, it is not surprising that in a distributed source of information of such a big size and heterogeneity as the Web, finding and exploiting information becomes extremely hard.

In response to this problem, many new research initiatives have been set up in order to enrich available information with machine processable semantics. Tim Berners-Lee, Director of the World Wide Web Consortium[6], referred to the future of the current Web as the Semantic Web, *"an extension of the current Web in which information is given well-defined meaning, better enabling*

[4] http://www.internetworldstats.com/stats.htm
[5] http://www.cs.uiowa.edu/ asignori/web-size/
[6] http://www.w3.org/

computers and people to work in cooperation. It is based on the idea of having data on the Web defined and linked such that it can be used for more effective discovery, automation, integration, and reuse across various applications" [Berners-Lee et al., 2001].

The explicit representation of the semantics underlying information and other web resources can enable a knowledge-based web that provides a new level of service. Automated services will improve in their capacity to assist humans in achieving their goals by understanding more of the content on the Web and thus provide more accurate filtering, categorization, and search of information sources. This process will ultimately lead to an extremely knowledgeable system with various specialized reasoning services that will support us in our access to information on the Web [Ding et al., 2002].

To achieve the goal of the Semantic Web vision, means to provide explicit meaning to the information on the Web are required. Ontologies are introduced for this purpose, being a key enabling technology for the Semantic Web. They provide a source of shared and precisely defined terms that can be understood and processed by machines. An ontology is defined [Gruber, 1993] as a formal, explicit specification of a shared conceptualization, that is, an understanding that can be communicated across people and application systems [Fensel and Bussler, 2002]. A typical ontology consists of a hierarchical description of important concepts and their relations in a domain, task or service. Ontologies interweave human understanding of symbols with their machine-processability. They were developed in Artificial Intelligence to facilitate knowledge sharing and reuse. An example of a simple ontology of wines is given in Figure 2.5.

Ontologies have the following main components [de Bruijn et al., 2005a]:

- Concepts constitute the basic elements of the agreed terminology for some problem domain. From a high-level perspective, a concept, described by a concept definition, provides attributes with names and types. Furthermore, a concept can be a subconcept of several (possibly none) direct superconcepts. An example of concept is *Winery* in Figure 2.5.

- Relations are used in order to model interdependencies between several concepts (respectively instances of these concepts). An example is the *Maker* relation in Figure 2.5, which links wines to wineries that produce such wines.

- Functions are special cases of relations in which the n-th element of the relationship is unique for the n-1 preceding elements, e.g. the price of a wine can be defined as the price given by the maker plus a profit percentage for the deliverer.

- Instances represent specific elements of the concepts, in contrast with general concepts or classes. An example is the wine Chateau Lafite Rothschild Pauillac which is a Pauillac made by Chateau Lafite Rothschild (see Figure 2.5).

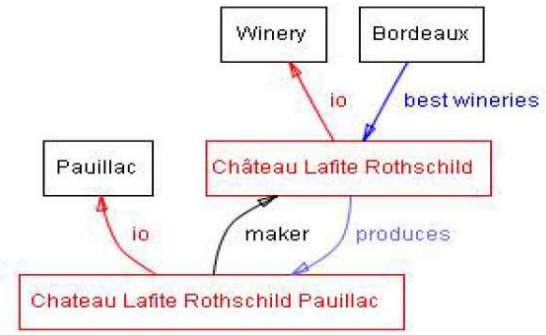

Figure 2.5: Sample ontology of wines (io means instance-of, concepts are in black and instances in red) [Noy and McGuinness, 2001].

- Axioms model sentences which must hold. For example, if a wine is produced by a maker from a given country e.g. Spain, then the wine is from the same country as the maker e.g. a Spanish wine.

If ontologies are defined and shared, and if information is described according to such ontologies, information on the Web is given an explicit, formal meaning which can be exploited to overcome the problems information on the Web currently has.

The use of ontologies has been and is being also investigated with other purposes besides the description of information on the Web. Real examples in which the PhD candidate has been involved include the integration of information based on ontologies in the automotive industry [Alexiev et al., 2005], the modelling and management of financial information [Castells et al., 2004], and the modelling of financial information for its automatic processing, analysis and exchange [Lara et al., 2006a].

The formal semantics of ontologies is what enables machine processability of information described in terms of these ontologies, as well as the derivation of new knowledge from existing knowledge [de Bruijn, 2007]. Similarly, the formal description of different aspects of services and of consumer's goals in service-oriented architectures can enable the automatic resolution of certain problems such as the dynamic location of services that can solve a given goal, the dynamic composition of services for accomplishing a given task, or the automated mediation of heterogeneous data exchanged by interacting parties, among others [Lara et al., 2003].

For the formal description of any artifact, either an information resource or a service, a language with formal semantics that allows for the use of associated formal rules of deduction has to be used, so that automated reasoning can be applied. Different (families of) languages exist, with different semantics, different expressivity, different computational complexity, and better suited for particular reasoning tasks. In the following sections, we introduce the main families of logical

languages that provide the foundations for the semantic Web and the semantic annotation of services in an SOA, and which have been used in our work.

First, the family of class-based knowledge representation formalisms known as description logics is presented partially based on the introduction given in [Volz, 2004]. We provide an introduction to the syntax and semantics of these languages, as well as to the most commonly used languages of the family. For an extended account of the core aspects of description logics we refer the reader to [Nardi et al., 2003].

Second, and based on the introductions to the matter provided by de Bruijn and Volz in [de Bruijn, 2007] and [Volz, 2004], respectively, we introduce logic programs, which are based on a subset of first-order logic [Fitting, 1996] but with different semantics (see [Lloyd, 1987] for an extended discussion).

Finally, Transaction Logic is introduced, as it will be used for formally characterizing services and goals, and we have made a limited use of it in our first attempt of semi-automating the location of relevant services (see Chapter 4). For a complete account of Transaction Logic, we refer the reader to [Bonner and Kifer, 1998; Bonner and Kifer, 1995].

2.2.2 Description Logics

Description Logics (DL) [Baader and Nutt, 2003] are a family of class-based knowledge representation formalisms based on first-order logic [Fitting, 1996]. They were first developed to provide formal, declarative meaning to semantic networks [Quillian, 1967] and frames [Minsky, 1981], and to show how such structured representations can be equipped with efficient reasoning tools.

Three main ideas have guided the development of DL:

- The basic syntactic building blocks are atomic concepts (first-order unary predicates), atomic roles (first-order binary predicates), and individuals (first-order constants). The cumbersome syntax of first-order logic is simplified, and a variable-free notation is used in DL.
- The expressive power of the language is restricted. In particular, most description logics are subsets or variants of $\mathcal{C}^2$, the fragment of first-order logic (without function-symbols) in which formulas may contain at most two variables, but with counting quantifiers allowed, and which is known to be decidable [Graedel et al., 1997].
- As a consequence of the restriction in expressivity imposed, the major reasoning tasks envisioned for description logics are kept decidable.

In the following, we introduce the syntax and semantics of most common description logics.

2.2.2.1 Syntax

Description Logics provide a set of atomic concepts and roles that, together with individuals, constitute the basic building blocks for constructing more complex concepts and roles.

Example 2.1 The following are examples of the basic building blocks of a DL:

- Atomic Concepts (Classes): $flight, flightSeat, city$
- Atomic Roles (Properties): $hasOrigin, hasDestination, hasDate, onFlight$
- Individuals: $flightA, myFlightSeat, rome, paris$

□

Intuitively, concepts represent sets of objects, i.e., a concept C corresponds to a first-order unary predicate $C(x)$ that is true for all objects x in the domain of discourse belonging to the set. Individuals are named elements belonging to some concepts [Colucci et al., 2005], i.e., elements of certain sets. Roles are binary relations, i.e., a role $\mathcal{R}$ corresponds to a first-order binary predicate $\mathcal{R}(x, y)$ that is true for objects x and y in the domain of discourse that are related by $\mathcal{R}$.

The basic description logic $\mathcal{AL}$ and common extensions. Complex concepts and roles are built inductively using a set of constructors which depends on the particular description logic at hand. In fact, description languages are distinguished by the constructors the provide. In Table 2.2, the constructors used by most common description logics are shown.

The description logic $\mathcal{AL}$ is considered to be the basic description logic, and it allows for the use of the constructors shown in the upper part of Table 2.2. Extensions of this basic description logic with additional constructors exist, and Table 2.2 shows in its lower part the most common extensions and how they are named according to the naming scheme proposed in [Schmidt-Schauss and Smolka, 1991]. According to this naming scheme, a letter or symbol is assigned to each extension of $\mathcal{AL}$ and written after the starting $\mathcal{AL}$, shown in the last column of the Table. For example, the extension of $\mathcal{AL}$ with full existential restriction is denoted with $\mathcal{ALE}$. Even though functionality (functional properties) is a special case of unqualified number restrictions, some description logics only allow for this limited form of number restrictions and, therefore, it is assigned the $\mathcal{F}$ symbol, which we list it explicitly in Table 2.2.

The $\mathcal{ALC}_{R^+}$ logic and its extensions, i.e., $\mathcal{AL}$ extended with full complement and transitive properties constitute a common base for prominent description languages, and is often abbreviated as $\mathcal{S}$. It has to be noted that such logic is equivalent to $\mathcal{ALUE}$, since both disjunction and full existential quantification can be expressed with the full complement in combination with other constructors in

Name	Syntax	Semantics	Symbol
Atomic named class	A	$A^{\mathcal{I}}$	$\mathcal{AL}(\mathcal{S})$
Universal class (Top)	$\top$	$\Delta^{\mathcal{I}}$	$\mathcal{AL}(\mathcal{S})$
Empty class (Bottom)	$\bot$	$\emptyset$	$\mathcal{AL}(\mathcal{S})$
Atomic complement	$\neg A$	$\Delta^{\mathcal{I}} \backslash A^{\mathcal{I}}$	$\mathcal{AL}(\mathcal{S})$
Conjunction (And)	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$\mathcal{AL}(\mathcal{S})$
Value restriction	$\forall R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \forall b((a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}})\}$	$\mathcal{AL}(\mathcal{S})$
Limited existential restr.	$\exists R.\top$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b((a,b) \in R^{\mathcal{I}})\}$	$\mathcal{AL}(\mathcal{S})$
Disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$\mathcal{U}(\mathcal{S})$
Full complement	$\neg C$	$\Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$	$\mathcal{C}(\mathcal{S})$
Full existential restr.	$\exists R.C$	$\{a \in \Delta^{\mathcal{I}} \mid \exists b((a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}})\}$	$\mathcal{E}(\mathcal{S})$
Unqualified number restr.	$\geq nR$	$\{a \in \Delta^{\mathcal{I}} \parallel \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\} \geq n\}$	$\mathcal{N}$
	$\leq nR$	$\{a \in \Delta^{\mathcal{I}} \parallel \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\} \leq n\}$	
Functionality	$\leq 1R$	$\{a \in \Delta^{\mathcal{I}} \parallel \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\} \leq 1\}$	$\mathcal{F}$
Qualified number restr.	$\geq nR.C$	$\{a \in \Delta^{\mathcal{I}} \parallel \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \geq n\}$	$\mathcal{Q}$
	$\leq nR.C$	$\{a \in \Delta^{\mathcal{I}} \parallel \{b \in \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \leq n\}$	
Enumeration	$\{i_1, \ldots, i_n\}$	$\{{i_1}^{\mathcal{I}}, \ldots, {i_n}^{\mathcal{I}}\}$	$\mathcal{O}$
Inverse property	R^-	$\{(b,a) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (a,b) \in R^{\mathcal{I}}\}$	$\mathcal{I}$
Transitive property	R^+	$\bigcup_{n \geq 1} (R^{\mathcal{I}})^n$	$\mathcal{R}^+(\mathcal{S})$

Table 2.2: DL common constructors [Volz, 2004]

$\mathcal{AL}$ and viceversa. In Table 2.2, the constructors included in the $\mathcal{S}$ logic are identified by the $\mathcal{S}$ in brackets in the symbol column.

Example 2.2 Given the atomic concepts, atomic roles and individuals from the previous example, the following expression denotes flights between Rome and Paris:

$$flight \sqcap hasOrigin.\{rome\} \sqcap hasDestination.\{paris\}$$

The following expression denotes all flights for which all origins and destinations are cities, and that have some origin and destination:

$$flight \sqcap \forall hasOrigin.city \sqcap \forall hasDestination.city \sqcap \exists hasOrigin.\top \sqcap \exists hasDestination.\top$$

The latter expression can be built with the constructors in $\mathcal{AL}$, while the former expression makes use of enumeration and, therefore, it cannot be expressed in the $\mathcal{AL}$ logic but only in $\mathcal{ALO}$ and its extensions.

□

DL Knowledge Bases. A DL knowledge base (KB) comprises two components: the TBox and the ABox:

- TBox. The TBox introduces the terminology of the application domain. It is constituted by a finite set of terminological axioms (see the upper part of Table 2.3) which define subsumption and equivalence relations on classes and properties, possibly built using the constructors of

the particular description logic. Terminological axioms make statements about how concepts or roles (properties) are related to each other, and they can be of the following types:

1. Equivalence axioms: They are of the form $C \equiv D (R \equiv S)$, where C and D are classes and R and S are properties, establishing the equivalence of certain classes or properties. If the left-hand side of the equality is an atomic concept (property), we call this axiom a *concept (property) definition*, and the atomic concept (property) on the left-hand side is interpreted as the complex description on the right-hand side. This allows for the introduction of *symbolic names* for complex descriptions.
2. Inclusion axioms: They are of the form $C \sqsubseteq D (R \sqsubseteq S)$, where C and D are classes and R and S are properties, meaning that a class (property) is more specific than (is subsumed by) another one. Furthermore, the set of axioms of the form $A \sqsubseteq B$, where both A and B are atomic classes (properties) is called a *class (property) hierarchy*. If the particular description logic allows for the definition of property hierarchies, i.e., property inclusions, a letter $\mathcal{H}$ is appended to the name of the particular description logic, as indicated in Table 2.3.

- ABox. In the ABox, individuals are introduced, given names, and properties of these individuals are asserted. Given individuals a, b, and c, a concept C, and a role R, assertions of the following kinds can be made in the ABox:

  1. Class (concept) assertions: They are of the form $C(a)$, denoting that a belongs to (the interpretation of) C, i.e., that the individual is member of a given class or set.
  2. Property fillers (role assertions): They are of the form $R(a, b)$, denoting that a is a filler of the role R for b, i.e., that individuals a and b are related to each other through property R.
  3. Individual equivalence: They are of the form $a = b$, denoting that both individuals are the same.
  4. Individual inequivalence: They are of the form $a \neq b$, denoting that the individuals are not the same one.

Concepts and Datatypes. Datatypes and predicates (such as $=$, $\geq$, $+$) defined over them can be used in the construction of concepts. Unlike concepts, datatypes and datatype predicates have obvious (fixed) extensions; e.g., the extension of ≥ 20 is all the integers that are greater or equal to 20. Due to the differences between classes and datatypes, there are two kinds of roles: (i) *object properties*, which relate objects to objects, and (ii) *datatype properties*, which relate objects to data

Name	Syntax	Semantics	Symbol
TBox			
Class equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$	
Class subsumption	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$	
Property equivalence	$R \equiv S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$	
Property subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$\mathcal{H}$
ABox			
Individual assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$	
Property filler	$R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$	
Individual equivalence	$a = b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$	
Individual inequivalence	$a \neq b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$	

Table 2.3: TBox and ABox axioms and assertions [Volz, 2004]

Name	Syntax	Semantics
Datatype property	T	$T^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_D^{\mathcal{I}}$
Datatype	d	$d^D \subseteq \Delta_D$
Datatype negation	$\neg d$	$\Delta_D \backslash d^D$
Datatype existential	$\exists T.d$	$\{x \mid \exists y.(x, y) \in T^{\mathcal{I}} \wedge y \in d^D\}$
Datatype universal	$\forall T.d$	$\{x \mid \forall y.(x, y) \in T^{\mathcal{I}} \rightarrow y \in d^D\}$

Table 2.4: Datatypes in DL [Volz, 2004]

values, which are instances of datatypes [Lara et al., 2004a]. In Table 2.4, the class constructors introduced for the use of datatypes are introduced.

Example 2.3 The following are examples of terminological axioms in a TBox:

$flightFromRomeToParis \equiv flight \sqcap hasOrigin.\{rome\} \sqcap hasDestination.\{paris\}$

$flightFromItalyToFrance \equiv flight \sqcap \exists hasOrigin \sqcap \forall hasOrigin.(city \sqcap inCountry.\{italy\}) \sqcap \exists hasDestination \sqcap \forall hasDestination.(city \sqcap inCountry.\{france\})$

$flightFromItalyToFrance \sqsubseteq transportationMeansFromItalyToFrance$

Examples of Abox assertions are the following:

$city(rome)$

$city(paris)$

$inCountry(rome, italy)$

$inCountry(paris, france)$

$flight(flightA)$

$hasOrigin(flightA, rome)$

$hasDestination(flightA, paris)$

Assuming we have a concrete domain Δ_D that provides natural numbers, a unary predicate $\leq_{10}$, a datatype property *hasDuration*, we can define in the TBox flights whose duration is less

than 10 hours as follows:

$$flightsOfLessThanTenHours \equiv flight \sqcap \exists hasDuration. \leq_{10}$$

□

2.2.2.2 Semantics.

The formal semantics of Description Logics are given by considering interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ with an interpretation function $\cdot^{\mathcal{I}}$ assigning to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to constructors by inductive definition, following the constructor semantics shown in Table 2.2. Interpretations are extended so that an interpretation $\mathcal{I}$ not only maps atomic concepts and roles to sets and relations, but also maps each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

Terminological axioms. The semantics of axioms is defined via set relationships, as shown in the upper part of Table 2.3. An interpretation is a model of a TBox $\mathcal{T}$ if it satisfies each axiom in $\mathcal{T}$.

Assertions. The semantics of assertions are defined in the lower part of Table 2.3 via set membership and (in)equivalence of objects. An interpretation $\mathcal{I}$ satisfies (is a model of) an ABox $\mathcal{A}$ if it satisfies each assertion in $\mathcal{A}$. $\mathcal{I}$ satisfies an assertion α or an ABox $\mathcal{A}$ with respect to a TBox $\mathcal{T}$ if in addition to being a model of α or of $\mathcal{A}$ it is a model of $\mathcal{T}$. A model of $\mathcal{A}$ and $\mathcal{T}$ is an abstraction of a concrete world where the concepts are interpreted as subsets of the domain as required by the TBox and where the membership of the individuals to concepts and their relationships with one another in terms of roles respect the assertions in the ABox.

Concrete domains. A possible way to give semantics to datatypes is described in [Horrocks and Sattler, 2001]. A DL is extended with a set D of concrete datatypes, and with each $d \in D$, a set $d^D \subseteq \Delta_D$ is associated, where Δ_D is the domain of all datatypes. Additionally, datatype properties are introduced and the datatype existential and universal constructors in Table 2.4, and given the semantics shown in this Table.

2.2.2.3 Reasoning services

A DL system not only stores axioms and assertions, but also offers services that reason about them. Typically, reasoning with a DL knowledge base is the process of discovering implicit knowledge entailed by the knowledge base. Reasoning services can be roughly categorized as basic services, which involve checking the truth value for a statement, and complex services, which can be built upon basic ones. Let $\mathcal{K}$ be a knowledge base with a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, $\mathcal{L}$ a Description

Logic, C, D concepts in $\mathcal{L}$, R a property in $\mathcal{L}$, and a and b individual names. In the following, we briefly describe the main TBox and ABox reasoning services:

- TBox services

  1. *Concept Satisfiability:*. Given a concept C, is the problem of checking whether there exists a model $\mathcal{I}$ of $\mathcal{T}$ in which $C^{\mathcal{I}} \neq \emptyset$.
  2. *Satisfiability:* is the problem of checking whether there exists a model $\mathcal{I}$ of $\mathcal{T}$. The satisfiability of the complete TBox can be reduced to checking whether $\top$ is satisfiable.
  3. *Subsumption:* Given concepts C and D, is the problem of verifying whether in every model $\mathcal{I}$ of $\mathcal{T}$ we have $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. In this case, we write $\mathcal{T} \models C \sqsubseteq D$.
  4. *Equivalence:* Given concepts C and D, is the problem of verifying whether in every model $\mathcal{I}$ of $\mathcal{T}$ we have $C^{\mathcal{I}} = D^{\mathcal{I}}$. In this case, we write $\mathcal{T} \models C \equiv D$, which is equivalent to $\mathcal{T} \models C \sqsubseteq D$ and $\mathcal{T} \models D \sqsubseteq C$.
  5. *Disjointness:* Given concepts C and D, they are disjoint with respect to $\mathcal{T}$ if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for all models $\mathcal{I}$ of $\mathcal{T}$.
  6. *Classification:* Given a new concept C, classification is the problem of putting the new concept in the proper place in a taxonomic hierarchy of *concept names*; this can be done by subsumption checking between each named concept in the hierarchy and the new concept. The location of the new concept C in the hierarchy will be between the most specific named concepts that subsume C and the most general named concepts that C subsumes. TBox classification, which computes the taxonomic hierarchy of concept names mentioned in a TBox, is a special case of classification, where $\top$ is chosen as the 'new' concept.

- ABox services

  1. *Consistency:* $\mathcal{A}$ is consistent with respect to $\mathcal{T}$ if there is an interpretation $\mathcal{I}$ that is a model of both $\mathcal{A}$ and $\mathcal{T}$. We say that $\mathcal{A}$ is consistent if it is consistent with respect to the empty TBox.
  2. *Instance Checking:* Given an individual a and a concept C is the problem of verifying whether in every model $\mathcal{I}$ of $\mathcal{A}$ we have that $a^{\mathcal{I}} \in C^{\mathcal{I}}$. In this case we write $\mathcal{A} \models C(a)$.
  3. *Retrieval:* Given a concept C, it is the problem of finding all individuals a such that $\mathcal{A} \models C(a)$; this can be done naively by instance checking between each named individual and the given concept [Lara et al., 2004a]. Similarly, we can find all named classes C for an individual a for which $\mathcal{A} \models C(a)$.

Description Logic	Complexity
$\mathcal{ALC}$	PSpace-complete
$\mathcal{S}$	PSpace-complete
$\mathcal{SI}$	PSpace-complete
$\mathcal{SH}$	ExpTime-complete
$\mathcal{SHIF}$	ExpTime-complete
$\mathcal{SHIQ}$	ExpTime-complete
$\mathcal{SHOIN}$	NExpTime-complete
$\mathcal{SHOIQ}$	NExpTime-complete

Table 2.5: Complexity of satisfiability for relevant description logics

4. *Property fillers:* Given a property R and an individual a, it is the problem of retrieving, with respect to TBox $\mathcal{T}$ and ABox $\mathcal{A}$, all individuals x which are related with a via R, viz. $\{x \mid (\mathcal{T}, \mathcal{A}) \models R(a, x)\}$. Similarly, we can retrieve all named properties R between individuals a and b, ask whether the pair (a, b) is a filler of R, or ask for all pairs (a, b) that are a filler of R.

It is important to notice that for description logics without full negation, all inference problems above can be reduced to subsumption. If the particular description logic allows both full complement and intersection as constructors, then all reasoning services can be reduced to satisfiability.

2.2.2.4 Complexity

The reasoning services introduced in the previous subsection have different complexity depending on the particular description logic. While decidability is kept by most common description logics, the complexity of reasoning depends on the constructors allowed by the particular logic.

In Table 2.5, the complexity of checking satisfiability for relevant description logics is summarized[7]. It can be seen how the addition of certain new constructors or the combination of certain constructors leads to higher complexity classes. For a discussion on the sources of complexity of different description logics we refer the reader to [Donini, 2003].

2.2.2.5 Reasoners

Modern DL reasoners have demonstrated that, even with expressive DL, highly optimized implementations can provide acceptable performance in realistic applications. In other words, thoughtful optimization techniques ([Horrocks, 1997; Horrocks and Patel-Schneider, 1998a; Horrocks and Sattler, 2002; Horrocks, 2003]) have moved the boundaries of tractability to somewhere

[7]It has to be noted that the complexity shown for $\mathcal{ALC}$ only holds for an empty or acyclic TBox, and the complexities shown for $\mathcal{S}$ and $\mathcal{SI}$ only hold for an empty TBox

very close to EXP-TIME-hard, or worse [Donini, 2003]. In the following, we will introduce the most prominent Description Logics reasoners available at the time of writing.

RacerPro RacerPro[8] is a commercial DL system which has its roots in the Racer system [Haarslev and Möller, 2001]. It implements the Description Logic $\mathcal{SHIQ}$, corresponding to the basic Description Logic $\mathcal{ALC}$ augmented with qualified number restrictions, role hierarchies, inverse roles, and transitive roles. It also includes limited support for concrete domains.

RacerPro has two limitations to deal with the $\mathcal{SHOIQ}(\mathcal{D})$ DL: individuals in class expressions (the so-called nominals) are only approximated, and user-defined datatypes given as external XML Schema specifications are not supported. RacerPro provides the reasoning services introduced in Section 2.2.2.3, and supports the specification of multiple TBoxes and ABoxes. For a complete account of the inference services provided by RacerPro, we refer the reader to [RAC, 2006].

RacerPro provides four different interfaces: the file interface, the TCP socket interface, the web service interface, and the HTTP Interface. The file interface allows the user for specifying the files where the TBox, the ABox, or both are described when starting the RacerPro server. The TCP socket interface allows for the communication with RacerPro using TCP sockets. There exists a Java API acting as a layer for accessing RacerPro services that makes use of the TCP sockets interface. The web service interface allows for the communication with RacerPro using web services, although such interface is not documented in [RAC, 2006]. Last, the HTTP interface uses HTTP as the communication protocol with RacerPro, in particular using the POST method. This interface supports the use of the Description Logics Implementation Group (DIG) standard [Bechhofer, 2003]. Such standard has been developed in order to provide a uniform way of communicating with DL systems. However, it lacks a number of features that are required by some applications e.g. on-demand classification of TBoxes which, as will be shown in Chapters 6 and 7, is a feature of great importance for the realization of our proposal for the location of services. The features supported by RacerPro but not included in the DIG standard are not available through the HTTP interface.

FaCT and FaCT++ FaCT [Horrocks, 1998] is a DL classifier also usable for modal logic satisfiability testing. FaCT includes two reasoners, one for the logic $\mathcal{SHF}$ ($\mathcal{ALC}$ augmented with transitive roles, functional roles and a role hierarchy) and the other for the logic $\mathcal{SHIQ}$ ($\mathcal{SHF}$ augmented with inverse roles and qualified number restrictions), both of which use sound and complete tableaux algorithms. FaCT is intended as a tool for conceptual schema design and ontological engineering and, therefore, does not provide an ABox but only a TBox; the use of an Instance Store [Bechhofer et al., 2005] has been proposed for enabling ABox reasoning.

FaCT++ [Tsarkov and Horrocks, 2003] is the new generation of the FaCT system, with

[8] http://www.racer-systems.com/

a new architecture and new optimizations. FaCT++ can handle $\mathcal{SHOIQ}$, and its last version can also handle the $\mathcal{SROIQ}$ Description Logic. FaCT comes in two flavours: a CORBA-FaCT system and a FaCT DIG servlet. The former, an OMG's Common Object Request Broker (CORBA) [COR, 1998] based client-server architecture, offers an Object Request Broker (ORB)[9] interface, while the latter offers an HTTP interface.

Pellet Pellet[10] [Sirin et al., 2006b] is an open-source DL reasoner based on the tableaux algorithms developed for expressive DL. As of version 1.4, Pellet supports the $\mathcal{SROIQ}(\mathcal{D})$ DL.

Pellet provides the different reasoning services introduced in Section 2.2.2.3. It also incorporates various optimization techniques described in the DL literature and contains several novel optimizations for nominals [Sirin et al., 2006a], conjunctive query answering [Sirin and Parsia, 2006], and incremental reasoning [Christian Halaschek-Wiener and Sirin, 2006]. Pellet offers different ways to access its reasoning capabilities, namely: a) a command line program, b) a programmatic API that can be used standalone or in conjunction with Jena[11] and the Manchester OWL-API[12] library, c) a DIG server that allows Pellet to be used with different clients, and d) integrated into the ontology editor SWOOP[13].

DLP The Description Logic Prover (DLP)[14] is an experimental description logic system, designed to investigate various options for checking satisfiability in expressive description logics and propositional modal logics. DLP is designed to allow various optimisations for description logic reasoning to be easily investigated. According to the project page, DLP is experimental because: 1) there are only minimal interfaces; 2) there is little optimization of the taxonomy code; and 3) the code is written in ML in a mostly-functional style. The biggest lack of DLP is that it does not handle individuals.

KAON2 KAON2[15] provides reasoning support for the $\mathcal{SHIQ}(\mathcal{D})$ DL, i.e., it cannot handle nominals. Contrary to the other DL reasoners introduced, KAON2 does not implement the tableaux calculus. Rather, reasoning in KAON2 is implemented by novel algorithms which reduce a $\mathcal{SHIQ}(\mathcal{D})$ knowledge base to a disjunctive Datalog program [Hustadt et al., 2004].

KAON2 provides an API for programmatic management of ontologies, a stand-alone server providing access to ontologies in a distributed manner using RMI[16], an inference engine for answering

[9] http://www.omg.org/gettingstarted/orb_basics.htm
[10] http://pellet.owldl.com/
[11] http://jena.sourceforge.net/
[12] http://sourceforge.net/projects/owlapi
[13] http://code.google.com/p/swoop/
[14] http://www.bell-labs.com/user/pfps/dlp/
[15] http://kaon2.semanticweb.org/
[16] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

conjunctive queries (expressed using SPARQL [Prud'hommeaux and Seaborne, 2006] syntax), a DIG interface, and a module for extracting ontology instances from relational databases.

2.2.3 Logic Programming

Logic Programming (or LP for short) is based on a subset of FOL, called Horn Logic, but it has a semantics that is slightly different from first-order semantics (see [Fitting, 1996]); the semantics of logic programs is based on *minimal Herbrand models* [Lloyd, 1987], rather than first-order models.

In the following, we define the syntax and semantics of logic programs, briefly introduce the prominent Datalog language, discuss the extension of logic programs with negation, and present common reasoning tasks and the complexity of these tasks.

2.2.3.1 Syntax

Logic Programming makes use of the Horn logic fragment of FOL. Formulas in the Horn fragment of FOL are defined as follows:

Definition 2.1 *(Horn formula) A first-order formula is in the Horn fragment of first-order logic, called horn formula, if it is a disjunction of literals with at most one positive literal, in which all variables are universally quantified:*

$(\forall)h \vee \neg b_1 \vee \ldots \vee \neg b_n$

which can be rewritten as:

$(\forall)h \leftarrow b_1 \wedge \ldots \wedge b_n$

A Horn formula with one positive literal and at least one negative literal is called a *rule*. The positive literal h is called the *head* of the rule, and the conjunction of negative literals $b_1 \wedge \ldots \wedge b_n$ is called the *body* of the rule. A rule without a body is called a *fact*, and a rule without a head is called a *query*.

A different notation is usual for rules, being a rule written as:

$h1 : -b_1, \ldots, b_n.$

A fact is written as:

$h_1.$

A query is written as:

$? - b_1, \ldots, b_n.$

Definition 2.2 *(Logic program) A logic program P is a set of rules, facts, and queries.*

2.2.3.2 Semantics

Different style of semantics can be given to logic programs. Among these styles, we will in the following concentrate in semantics based on minimal Herbrand models. An example of an alternative semantics for logic programs is fixpoint semantics, based on the use of an immediate consequence operator T_P. Remarkably, both semantics are equivalent [Lloyd, 1987].

Herbrand theory imposes a syntactic restriction on the admissible structures that can be a model. In the following we introduce the domain of Herbrand interpretations, Herbrand interpretations, Herbrand bases and Herbrand models.

Definition 2.3 *(Herbrand universe) The Herbrand universe is the domain Δ^H of Herbrand interpretations, and for a first-order signature $\Sigma = (C, F, P, V)$ it is inductively defined as follows:*

- $c \in \Delta^H$ *for all* $c \in C$*;*
- $f(t_1, \ldots, t_n) \in \Delta^H$ *if* $f \in F$ *and each* $t_i \in \Delta^H$*.*

Therefore, the Herbrand universe of a logic program P is the set of all ground terms which can be formed using the constant and function symbols in the signature Σ of P (in case P has no constants, some constant c is added).

Definition 2.4 *(Herbrand base) The Herbrand base B_H of a logic program P is the set of all ground atomic formulae which can be formed with the predicate symbols in Σ of P and terms in Δ^H, i.e., all formulae of the form:*

$p(t_1, \ldots, t_n)$

With p an n-ary predicate symbol and $t_1, \ldots, t_n \in \Delta^H$.

Definition 2.5 *(Herbrand interpretation) A Herbrand interpretation $\mathcal{H}$ of a logic program P is a subset of the Herbrand base B_H, and it corresponds to a first-order interpretation $\mathcal{I} = (\Delta^H, \cdot^I)$ such that $\cdot^I$ satisfies the following conditions:*

- $c^{\mathcal{I}} = c$ *for every constant symbol* $c \in C$*,*
- $(f(t_1, \ldots, t_n))^{\mathcal{I}} = f(t_1, \ldots, t_n)$ *for every function symbol* $f \in F$*, and*
- $(p(t_1, \ldots, t_n))^{\mathcal{I}}$ *is true for* $p(t_1, \ldots, t_n) \in \mathcal{H}$*.*

Example 2.4 Given the logic program P:

$p(a).$
$q(b).$
$p(X) :- q(X).$

$p(X) : -p(f(X))$.

where p, q are predicate symbols, f is a function symbol, and a, b are constants, the Herbrand universe is $\Delta^H = \{a, b, f(a), f(b), f(f(a)), f(f(b)), f(f(f(a))), \ldots\}$.

The Herbrand base is $B_H = \{p(a), p(b), q(a), q(b), p(f(a)), q(f(a)), p(f(b)), \ldots\}$.

Examples of Herbrand interpretations are $\mathcal{H}_1 = \{p(f(a)), q(b), q(f(b))\}$, $\mathcal{H}_2 = \{p(a), p(b), q(b)\}$, $\mathcal{H}_3 = \{p(a), p(b), q(a), q(b), p(f(a))\}$.

□

It must be noted that the Herbrand universe becomes infinite as soon as function symbols are used in a logic program, as it happens in the example above. This implies an also infinite Herbrand base.

The grounding of a program P, denoted $Ground(P)$, is the union of all possible ground instantiations of P, obtained by, for each rule $r \in P$, replacing each variable with a term in the Herbrand universe Δ^H. Next, we define herbrand models.

Definition 2.6 *(-Minimal- Herbrand model) Let P be a positive logic program i.e. a program without negation. A Herbrand interpretation $\mathcal{H}$ of P is a model of P if, for every rule $r \in Ground(P)$, if $b_1, \ldots, b_n \in \mathcal{H}$ then $h \in \mathcal{H}$.*

The intersection of all Herbrand models of P is also a model of P and is called the minimal Herbrand model*.*

Example 2.5 Among the interpretations given in the previous example, only $\mathcal{H}_2$ and $\mathcal{H}_3$ are Herbrand models of P. Furthermore, $\mathcal{H}_2$ is the minimal Herbrand model of P.

□

Definition 2.7 *(Ground entailment) A logic program P entails a ground atomic formula ϕ, denoted $P \models \phi$, iff ϕ is included in the minimal Herbrand model of P.*

A conjunction of ground formulae $\phi_1, \ldots, \phi_n$ is entailed by a program P iff $P \models \phi_1$ for 1 $\leq i \leq n$.

2.2.3.3 Datalog

Datalog [Ullman, 1988] is a prominent LP language that introduces certain restrictions on the signature and on the type of rules used in programs. In particular, function symbols are not included in the signature and therefore they cannot appear in Datalog programs, and all rules must be safe, as defined in the following.

Definition 2.8 *(Safe rule) A safe rule is a rule in which all variables that appear in the rule head must also appear in the rule body.*

By introducing these restrictions, it is ensured that all models of Datalog programs are finite.

2.2.3.4 Negation in logic programs

In order to discuss the extension of logic programs with negation, we start by introducing the concept of the dependency graph of a logic program and the notion of recursive program.

Definition 2.9 *(Dependency graph) A dependency graph is a directed graph where the predicates in the logic program are represented as nodes in the graph, and there is an arc from (the node of) some predicate p to (the node of) some predicate q if they occur in a rule with p in the body and q in the head.*

Definition 2.10 *A logic program is recursive iff there is a cycle in its dependency graph.*

Now, we are ready for discussing the extension of logic programs with negation. We start by defining normal logic programs.

Definition 2.11 *(Normal logic program) A normal logic program P consists of a set of rules of the form:*

$h : -b_1, \ldots, b_k, notn_1, \ldots, notn_m$

where $h, b_1, \ldots, b_k, n_1, \ldots, n_m$ *are atomic formulae.* $b_1, \ldots, b_k$ *are said to occur positively in the body of the rule, and* $n_1, \ldots, n_m$ *to occur negatively.*

As we can see, normal logic programs allow negation in the body of rules, denoted *not* as it differs from first-order (classical) negation, denoted with $\neg$. Negation in logic programs is also called *default negation*, as facts are assumed to be false (by default) unless we can infer otherwise, while under first-order semantics the negation of a formula is only true if it can be explicitly inferred.

The safety rule introduced for Datalog is extended, and all variables which occur in negative literals in the body of a rule must also occur in some positive literal.

We extend de definition of Herbrand models to logic programs with negation.

Definition 2.12 *(Herbrand model of normal logic programs) A Herbrand interpretation* $\mathcal{H}$ *of a normal logic program P is a model of P if for every rule* $r \in Ground(P)$*, we have that* $h \in \mathcal{H}$ *if:*

- $b_1, \ldots, b_i \in \mathcal{H}$*, i.e., all literals that occur positively in the rule body are in* $\mathcal{H}$*, and*
- $b_{i+1}, \ldots, b_n \notin \mathcal{H}$*, that is, all literals that occur negatively in the rule body are not in* $\mathcal{H}$*.*

A normal logic program may have more than one minimal model. However, there is a class of normal logic programs with only one minimal model, called *stratified logic programs*. The predicates in a stratified program can be divided into a number of strata such that there is no negative dependency between predicates in the same stratum. Stratifications fulfill the following conditions:

- If some predicate q is at stratum i and depends positively on some predicate p, then p must also be in stratum i, and
- if some predicate q is at stratum i and depends negatively on some predicate p, then p must be in stratum i-1.

We can extend the dependency graph of a program P so that if there is a rule with head h and with a negative body literal *not* p, there there is an arc between p and q and this arc is marked with "not". If P is a recursive program, and there are cycles in the graph which include a negative arc, then the program is not stratifiable.

For stratified logic programs, a single minimal Herbrand model can be computed by the intersection of all Herbrand models of the program. Therefore, their semantics can be defined based on minimal Herbrand models. For logic programs which are not stratifiable, other semantics exist, such as stable model semantics (also called answer set semantics) [Gelfond and Lifschitz, 1988] and well-founded semantics [Gelder et al., 1991].

2.2.3.5 Reasoning tasks

By the Definition 2.7 of ground entailment above, used in logic programming, it is possible to check whether particular facts follows from a logic program, but not whether some rule or formula follows from the program. However, the most prominent reasoning task for logic programs is *query answering*, which can be formulated in terms of ground entailment.

Definition 2.13 *(Query) A query q is a rule without a head, i.e., a conjunction of atomic formulae with a finite number of free variables (including 0). A query is written as:*

$?- \phi_1, \ldots, \phi_n.$

Answering a query q is the problem of determining all substitutions for all variables in q, and each such substitution is an *answer* to the query. If query q has no variables, we have a special case of query answering which boils down to checking whether $P \models q$, i.e., whether q is included in the minimal Herbrand model of P.

Other reasoning tasks for logic programs exist, such as query containment [Calvanese et al., 1998], but they will not be discussed here.

2.2.3.6 Complexity of reasoning

The complexity of logic programming naturally depends on the particular variants and extensions considered (see [Dantsin et al., 2001]). In general, three main kinds of complexity can be distinguished:

1. Data complexity. It is the complexity of checking whether $EDB \cup P \models A$ when logic programs P are fixed whereas input databases EDB and ground atoms A are an input.

2. Program complexity. It is the complexity of checking whether $EDB \cup P \models A$ when input databases EDB are fixed whereas P and A are an input.

3. Combined complexity. It is the complexity of checking whether $EDB \cup P \models A$ when all elements (EDB, P, A) are inputs.

Plain Datalog, i.e., without extensions, is data complete for the complexity class P and program complete for ExpTime. Its combined complexity is also ExpTime. If we add negation, and since the stratification algorithm is a polynomial algorithm, stratified Datalog with negation has the same complexity as plain Datalog. For a full account of the complexity of logic programming we refer the reader to [Dantsin et al., 2001].

2.2.3.7 Reasoners

In the following, we introduce a number of Logic Programming implementations presented in [de Bruijn et al., 2005b]. In general, these implementations deal very well with query answering or instance retrieval, but they are not meant to be used as subsumption reasoners. Still, as shown in [Grosof et al., 2003], subsumption reasoning can be reduced to query answering for the subset of DL that intersects LP, named DLP (Description Logic Programs). Thus, LP reasoners can be used for performing subsumption with DLP, although they are not specially designed and optimized for this reasoning task.

SWI-Prolog Prolog is a logical programming language based on first-order predicate calculus, restricted to allow only Horn clauses. The execution of a Prolog program is an application of theorem proving by first-order resolution. SWI-Prolog[17] [Wielemaker, 2003] is a free Prolog environment, licensed under the Lesser GNU Public License.

Among the advantages of SWI-Prolog are that it is portable to many platforms. Binary distributions for most popular platforms (Windows, Linux and MacOS X) are regularly released. One of its advantages over other LP reasoning engines is that it offers fast and flexible libraries for

[17] http://www.swi-prolog.org

parsing SGML (HTML) and XML, RDF, store and query RDF, RDFS, and OWL, and an extended DIG interface which can be used for performing DL reasoning by calling external DL reasoners. As a disadvantage it can be mentioned the fact that it does not deal with non-stratified negation.

XSB XSB[18] is a Logic Programming and Deductive Database system with support for different kinds of negation such as stratified negation and negation under the well-founded semantics [Gelder et al., 1991]. It also provides packages for evaluating F-Logic [Kifer et al., 1995] or a HiLog [Chen et al., 1993] implementation. The developers of XSB regard it as beyond Prolog because of the availability of SLG resolution [Chen and Warren, 1996] and the introduction of HiLog terms. SLG resolution enables the resolution of recursive queries that SLD resolution cannot deal with, and it also enables the use of well-founded semantics for non-stratified negation.

XSB also offers interfaces to other software systems, such as C, Java, Perl, ODBC, SModels, and Oracle. These interfaces also allow easy integration of new built-in predicates.

$\mathcal{FLORA}$-2 $\mathcal{FLORA}$-2[19] is a rule-based knowledge representation formalism and a reasoner for this formalism. It is based on F-Logic [Kifer et al., 1995], HiLog [Chen et al., 1993], and Transaction Logic [Bonner and Kifer, 1995], unifying these languages in a formalism which inherits important features from each of them: object-oriented features from F-Logic, and reification and meta-information processing capabilities from HiLog, and declarative programming of procedural knowledge from Transaction Logic[20].

The underlying inference engine of the reasoner is XSB Prolog. In fact, $\mathcal{FLORA}$-2 is often viewed as syntactic sugar on top of XSB. For more details on $\mathcal{FLORA}$-2 we refer the reader to the $\mathcal{FLORA}$-2 manual[21].

TRIPLE TRIPLE[22] is the name of both a language and the corresponding reasoning system. The language TRIPLE is a layered and modular language for the Semantic Web (especially for the querying and transformation of RDF [Brickley and Guha, 2004a] models) which bases on Horn Logic extended by F-Logic features and RDF constructs. Because these extensions are only syntactical, the language can be translated to Prolog and processed by a Prolog system. Correspondingly, the TRIPLE inferencing engine bases on the XSB Prolog system.

OntoBroker, SILRI AND MINS Ontobroker is a reasoner for reasoning with ontologies formalized in F-Logic or in Datalog/Prolog. It can be described as a main memory deductive, object

[18] http://xsb.sourceforge.org/
[19] http://flora.sourceforge.net/
[20] However, notice that it has features of Transaction Logic but it cannot be regarded as a full Transaction Logic reasoner.
[21] http://flora.sourceforge.net/docs/floramanual.pdf
[22] http://triple.semanticweb.org/

oriented database system. The Ontobroker system is developed and distributed by the company Ontoprise[23].

SILRI[24] is a less efficient version of OntoBroker with less features, which has been made public by Ontoprise.

MINS[25] is a reasoner for Datalog programs with negation and function symbols, which supports the well-founded Semantics. MINS is a re-implementation of SILRI tailored for the WSML family of languages, in particular for WSML-Rule and WSML-Flight [de Bruijn et al., 2005e] (see Section 2.2.5.3).

DLV The DLV system[26] (as well as the SMODELS system which will be discussed next) distinguishes from the above-mentioned Prolog-Based systems in that these systems implement fully declarative logic programming in a purer sense than PROLOG-like systems. These systems compute models rather than answering queries in a top-down fashion [de Bruijn et al., 2005b].

The DLV system is an efficient engine for computing answer sets (i.e. stable models for logic programs with negation under the extension with classical negation and disjunction [Gelfond and Lifschitz, 1988]. DLV uses as core input language safe Datalog programs (cf. [Ullman, 1988]) with disjunction in rule heads and default negation in rule bodies. Programs are safe if every variable occurring in head literals or default negated body literals also occurs in at least one non-default-negated body literal. Note that in DLV head and body literals may be classically negated. A logic Program is safe if all of its rules are safe. Note that the safety restriction is only syntactical but does not really affect the expressive power of the language in any way. DLV computes answer sets, i.e. it follows the stable model semantics but not the well-founded semantics.

DLV provides some preliminary APIs such as a wrapper for using DLV from Java and ongoing work on an ODBC interface connecting to relational databases.

SMODELS and GNT SMODELS[27] allows for the computation of stable models and well-founded models for normal logic programs, that is, for Datalog programs with non-disjunctive heads. However, there is an extended prototype version for the evaluation of disjunctive logic programs as well, called GNT[28]. Syntactically, SMODELS imposes an even stronger restriction than rule safety in DLV by demanding that any variable in a rule is bounded to a so-called domain predicate in the rule body which is, intuitively, a predicate which is defined only positively. Again, this restriction does not affect the expressive power of the language itself, but in some cases the weaker safety

[23] http://www.ontoprise.de/home_en
[24] http://ontobroker.semanticweb.org/silri/
[25] http://tools.deri.org/mins/
[26] http://www.dlvsystem.com/
[27] http://www.tcs.hut.fi/Software/smodels/
[28] http://www.tcs.hut.fi/Software/gnt/

restriction of DLV allows for more concise problem encodings.

KAON2 The KAON2 reasoner introduced in Section 2.2.2.5 can deal with Disjunctive Datalog with stratified negation, along with basic built-in predicates to deal with integers and strings.

Summarizing, all the Logic Programming implementations above agree on Datalog with stratified negation. In addition, XSB, $\mathcal{FLORA}$-2, and OntoBroker all support function symbols and support unstratified negation under the well-founded semantics. Furthermore, Frame-based modelling is supported by $\mathcal{FLORA}$-2 and OntoBroker.

2.2.4 Transaction Logic

Transaction Logic (or $\mathcal{TR}$ for short) accounts in a declarative fashion for the phenomenon of state change. It has a model theory and a sound and complete proof theory, as well as a *Horn* version which has a procedural as well as a declarative semantics. In the Horn fragment, transaction programs can be specified and executed, and in the full logic users can express properties of programs and reason about them.

The most salient features of $\mathcal{TR}$ are that it supports *both* hypothetical and committed updates, dynamic constraints on transaction execution, nondeterminism, and bulk updates. Furthermore, $\mathcal{TR}$ separates the specification of elementary operations from the logic of combining them, which has two benefits [Bonner and Kifer, 1998]: a) it allows to develop a language for state-changing procedures without committing to a particular theory of elementary updates, and b) it allows $\mathcal{TR}$ to accommodate a wide variety of state semantics, from classical to non-monotonic to various other non-standard logics.

2.2.4.1 Elementary updates, the state data oracle, and the state transition oracle

The emphasis of $\mathcal{TR}$ is on the logical combination of elementary updates, not on the definition of elementary updates themselves. In fact, the set of elementary updates is orthogonal to Transaction Logic, which can accommodate any such set. Therefore, $\mathcal{TR}$ is not committed to a particular theory of elementary updates.

Transaction Logic theories are not only parameterized by a language for constructing well-formed formulas, but also by a *state data oracle* and a *state transition oracle*. The data oracle specifies a set of primitive queries, i.e., the static semantics of states, while the transition oracle specifies a set of primitive updates, i.e., the dynamic semantics of states. Together, they encapsulate elementary operations, and separate the specification of elementary operations from the logic of combining them.

State data oracles. A state data oracle species the semantics of states, i.e., the formal meaning of states. Such an oracle, assuming a countable set of symbols serving as state identifiers, is a mapping $\mathcal{O}^d$, from state identifiers to sets of closed first-order formulas; if i is a state identifier, then $\mathcal{O}^d(i)$ is the set of formulas considered to be the truths about that state [Bonner and Kifer, 1995]. Therefore, the state data oracle can be intuitively regarded as an oracle that tells what is true at a particular state.

Example 2.6 Let us consider a classical state data oracle. In this case, a state s can be defined by a consistent set of variable-free first-order formula. For example, a possible state can be $s_1 = person(john) \wedge inCity(john, london)$. The mapping defined by the state data oracle is the following:

$\mathcal{O}^d(s) = \{\phi \mid s \models \phi\}$

That means that the state data oracle maps a state (a variable-free first-order formula) to its set of entailments, i.e., $\mathcal{O}^d(s) = Ent(s)$, where $Ent(s)$ denotes the set of classical entailments of a formula.

For example, we have that $\mathcal{O}^d(s_1) = \{person(john), inCity(john, london)\}$.

□

State transition oracles. The specification of elementary state transitions is a parameter of $\mathcal{TR}$, and is given by a state transition oracle. Such an oracle is a function $\mathcal{O}^t$ that maps pairs of states into sets of ground atomic formulas, being these ground atomic formulas the *elementary transitions* or *elementary updates*. Intuitively, for an initial state s_1 and a final state s_2, $\mathcal{O}^t(s_1, s_2)$ gives the set of elementary transitions that can perform the transition from the initial to the final state.

Example 2.7 Let us consider a classical state transition oracle. In this case, the oracle defines primitives for adding and removing formulas from the database, i.e., for changing states, resolving any conflicts between the new formulas and existing ones.

For example, the state transition oracle can define that $\mathcal{O}^t(s_1, s_2) = inCity(john, london).del$, where s_1 is the state from the previous example and $s_2 = person(john)$, i.e., we can go from state s_1 to state s_2 by applying the elementary transition $inCity(john, london).del$, which removes $inCity(john, london).del$ from the current state.

□

2.2.4.2 Execution paths, transaction bases and executional entailment

In $\mathcal{TR}$ a transaction execution is not only characterized by the initial and final states of the transaction, but also by a sequence of intermediate states. As Transaction Logic supports non-deterministic transactions, the final state and the intermediate states may not be uniquely determined at the start of the execution.

The sequence of states a given transaction execution passes through is called the *execution path* of the transaction, and it is represented explicitly. This allows to express constraints on such paths, such as forbidding certain initial, intermediate, or final states.

Example 2.8 An example of an execution path is the following:

$P, s_1, s_2 \models inCity(john, london).del$

where P is a transaction base, introduced in the following, and s_1, s_2 are the states from the previous example.

□

Elementary updates are combined into more complex transactions. A *transaction base* defines transactions and, as a special case, queries by combining elementary updates using the set of allowed $\mathcal{TR}$ operators which will be introduced later in this section. Formulas in a transaction base may use the full syntax of $\mathcal{TR}$, while formulas defining the semantics of states are limited to the syntax of first-order logic, a subset of $\mathcal{TR}$. Transactions are combinations of queries, which do not change the state and can be expressed in classical logic, and updates, which do change the state and are expressed in an extension of classical logic.

Example 2.9 We can define a transaction base P for financial transactions as follows [Bonner and Kifer, 1998]:

$transfer(Amt, Act, Act') \leftarrow withdraw(Amt, Act) \otimes deposit(Amt, Act')$

$withdraw(Amt, Act) \leftarrow balance(Act, B) \otimes B \geq Amt \otimes balance.change(Act, B, B - Amt)$

$deposit(Amt, Act) \leftarrow balance(Act, B) \otimes balance.change(Act, B, B + Amt)$

$balance.change(Act, B, B') \leftarrow balance.del(Act, B) \otimes balance.ins(Act, B')$

where $balance(Act, B)$ and $B \geq Amt$ are queries, and the rest are considered elementary updates; $balance.del(Act, B)$ means that we remove the formula $balance(Act, B)$, $balance.ins(Act, B')$ means that the we add the formula $balance(Act, B')$ to the current state, and $balance.change(Act, B, B - Amt)$ means that we update formula $balance(Act, B)$ to $balance(Act, B - Amt)$. The symbol $\otimes$ is called serial conjunction and, as we will see later in this section, an expression $a \otimes b$ can be intuitively read as "do a and then b"

□

A $\mathcal{TR}$ can be regarded as having two parts: a transaction base P provided by the programmer, and a single current database s or state, which he wishes to access and possibly modify. While the transaction base cannot be changed by other transactions, the database is updated when the user executes transactions defined in P.

The formal description of transactions is given in terms of the initial, intermediate, and final states of the transaction, relative to a transaction base that defines a set of complex state transitions. Such formalization is a form of logical entailment, called *executional entailment*:

$P, s_0, \ldots, s_n \models \psi$

where P is a transaction base, ψ is a transaction invocation, and $s_0, \ldots, s_n$ is a sequence of states representing all the states of the transaction execution. The formula above means, given P, that $s_0, \ldots, s_n$ is an execution path of ψ, i.e., if the current state is s_0, and if the user issues the transaction ψ, then the database will go from state s_0 to state s_n, passing through a set of intermediate states which may be $s_1, \ldots, s_{n-1}$. The proof theory of Transaction Logic can derive each possible execution path of ψ, but only one of them will be (non-deterministically) selected as the actual execution path, and the final state s_n will become the new state.

It is important to notice that $\mathcal{TR}$ treats transactions and queries uniformly: any transaction that does not cause a state change can be viewed as a query. $P, s_0 \models \psi$ is a special case of the statement above in which $n = 0$. In fact, all formulas in $\mathcal{TR}$ can have both a truth value and a side effect on the database; pure queries and pure updates are extreme cases of formulas with no side-effect or no truth value, respectively.

2.2.4.3 Syntax

The alphabet of a language of $\mathcal{TR}$ consists of the following symbols [Bonner and Kifer, 1995]:

- A signature $\Sigma = (C, F, P, V)$. As in first-order languages, C, F, P, and V are countable sets of constant, function, predicate and variable symbols, respectively, and function and predicate symbols have associated an arity n, which is a non-negative natural number.
- Logical connectives, $\wedge$, $\vee$ (classical disjunction and conjunction), $\otimes$, $\oplus$ (serial disjunction and conjunction), $\neg$ (classical negation). Additional connectives can be defined in terms of these.
- Quantifiers $\forall$, $\exists$.
- Auxiliary symbols, such as "(", ")", and ",".

Terms are defined as usual in first-order logic.

As it can be seen, $\mathcal{TR}$ extends the syntax of first-order logic with two binary connectives, $\otimes$ and $\oplus$. The resulting logical formulas are called transaction formulas.

Transaction formulas are defined recursively as follows [Bonner and Kifer, 1995]. First, an atomic transaction formula is an expression of the form $p(t_1, \ldots, t_n)$, where $p \in P$ is a predicate symbol, and $t_1, \ldots, t_n$ are terms. Second, if ϕ and ψ are transaction formulas, them so are the following expressions:

- $\phi \vee \psi$, $\phi \wedge \psi$, $\phi \oplus \psi$, $\phi \otimes \psi$, and $\neg\phi$.
- $(\forall X)\phi$ and $(\exists X)\phi$, where X is a variable.

2.2.4.4 Serial conjunction and serial disjunction

As presented before, $\mathcal{TR}$ extends classical predicate logic with two new connectives, $\otimes$ (serial conjunction), and $\oplus$ (serial disjunction).

A formula $\phi \otimes \psi$ intuitively means "Do ϕ and then do ψ" [Bonner and Kifer, 1995], and a formula $\phi \otimes \psi$ means "Do ϕ now or do ψ later". Formally, they are dual i.e. $\neg(\phi \oplus \psi)$ is equivalent to $\neg\phi \otimes \neg\psi$. As in classical logic, $\phi \leftarrow \psi$ is an abbreviation for $\phi \vee \neg\psi$.

2.2.4.5 Hypothetical reasoning

Transaction Logic supports both committed and hypothetical updates i.e. it can both execute programs (like e.g. Prolog) and reason about them (like e.g. logics of action).

To express hypothetical updates, $\mathcal{TR}$ introduces a modal operator $\diamond$ and a related operator $\Box$. $\diamond\phi$ means that execution of ϕ is *possible* starting at the present state, and $\Box\phi$ means that the execution of ϕ is *necessary* at the present state. Necessary means that ϕ is the only transaction that can succeed from the present state. Formally, it means that ϕ is executable along every path leaving the current state s. Possibility means that ϕ is executable along *some* path leaving the current state [Bonner and Kifer, 1995].

For example, the execution of $p \otimes \diamond q \otimes r$ means that first p is executed, then q is executed hypothetically, and finally r is executed. That q is executed hypothetically means that all its updates are rolled back before r starts executing. Therefore, $\diamond q$ acts as a test that have to be satisfied before r is executed, while p and r are committed and have a permanent effect on the state.

2.2.4.6 Semantics

The semantics of Transaction Logic is given by its model theory, which is based on execution paths and states. Truth is defined on execution paths, not on states, as we will see in the following.

We first define path structures. Satisfaction on paths will then be defined, followed by the definition of models of transaction formulas and of execution as entailment.

Definition 2.14 *(Path structures) Let $\mathcal{L}$ be a language of $\mathcal{TR}$ with its signature containing a set $\mathcal{F}$ of function symbols and a set $\mathcal{P}$ of predicate symbols. A* path structure*, $\boldsymbol{M}$, over $\mathcal{L}$ is a triple $(U, \mathcal{I}_{\mathcal{F}}, \mathcal{I}_{path})$, where*

- *U is a set, called the domain of $\boldsymbol{M}$.*

- *$\mathcal{I}_{\mathcal{F}}$ is an interpretation of function symbols in $\mathcal{L}$. It assigns a function $U^n \to U$ to every n-ary function symbol in $\mathcal{F}$.*

  Given U and $\mathcal{I}_{\mathcal{F}}$, let $Struct(U, \mathcal{I}_{\mathcal{F}})$ denote the set of all classical first-order semantic structures over $\mathcal{L}$ of the form $(U, \mathcal{I}_{\mathcal{F}}, \mathcal{I}_{\mathcal{P}})$, where U is the domain of the structure, $\mathcal{I}_{\mathcal{P}}$ is a mapping that interprets predicate symbols in $\mathcal{P}$ by relations on U of appropriate arity, and U and $\mathcal{I}_{\mathcal{F}}$ are the same as in $\boldsymbol{M}$. We also assume that $Struct(U, \mathcal{I}_{\mathcal{F}})$ contains the special structure $\top$, which satisfies every first-order formula.

- *$\mathcal{I}_{path}$ is a total mapping that assigns to every path a first-order semantic structure in $Struct(U, \mathcal{I}_{\mathcal{F}})$, such that:*

  - *$\mathcal{I}_{path}(< s >) \models^c \phi$ for every formula $\phi \in \mathcal{O}^d(s)$, where $< s >$ denotes a path of length 1 (Compliance with the data oracle).*
  - *$\mathcal{I}_{path}(< s_1, s_2 >) \models^c b$ for every atom $b \in \mathcal{O}^t(s_1, s_2)$ (Compliance with the transition oracle).*

As discussed in [Bonner and Kifer, 1995; Bonner and Kifer, 1998], $\mathcal{I}_{path}$ is the semantic link between transactions and paths: given a path and a transaction formula, $\mathcal{I}_{path}$ determines whether the formula is true on the path, as we will see in the definition below.

Intuitively, the compliance with the data oracle restriction says that path $< s >$ is a window over state (database) s. The second restriction says that elementary updates do what the transition oracle claims they do.

We now define a path split: given a path $< s_1, \ldots, s_n >$, any state s_i on the path defines a split of the path into two parts, $< s_1, \ldots, s_i >$ and $< s_i, \ldots, s_n >$. If path π is split into two parts γ and δ, then we write $\pi = \gamma \circ \delta$. Using the notion of path split, we define satisfaction as follows.

Definition 2.15 *(Satisfaction) Let $\boldsymbol{M} = (U, \mathcal{I}_{\mathcal{F}}, \mathcal{I}_{path})$ be a path structure, let π be a path in $\boldsymbol{M}$, and let v a variable assignment. Then,*

1. *$\boldsymbol{M}, \pi \models_v b$ iff $\mathcal{I}_{path}(\pi) \models_v^c b$, where b is an atomic formula.*
2. *$\boldsymbol{M}, \pi \models_v \neg\phi$ iff $\boldsymbol{M}, \pi \not\models_v \phi$.*
3. *$\boldsymbol{M}, \pi \models_v \phi \wedge \psi$ iff $\boldsymbol{M}, \pi \models_v \phi$ and $\boldsymbol{M}, \pi \models_v \psi$.*
4. *$\boldsymbol{M}, \pi \models_v \phi \vee \psi$ iff $\boldsymbol{M}, \pi \models_v \phi$ or $\boldsymbol{M}, \pi \models_v \psi$.*
5. *$\boldsymbol{M}, \pi \models_v \phi \otimes \psi$ iff $\boldsymbol{M}, \gamma \models_v \phi$ and $\boldsymbol{M}, \delta \models_v \psi$ for some split $\gamma \circ \delta$ of path π.*
6. *$\boldsymbol{M}, \pi \models_v \phi \oplus \psi$ iff $\boldsymbol{M}, \gamma \models_v \phi$ or $\boldsymbol{M}, \delta \models_v \psi$ for every split $\gamma \circ \delta$ of path π.*

7. $\boldsymbol{M}, \pi \models_v \forall x.\phi$ *iff* $\boldsymbol{M}, \pi \models_u \phi$ *for every variable assignment,* u*, that agrees with* v *on all variables except* x*.*

8. $\boldsymbol{M}, \pi \models_v \exists x.\phi$ *iff* $\boldsymbol{M}, \pi \models_u \phi$ *for some variable assignment,* u*, that agrees with* v *on all variables except* x*.*

Given the definition above, we can now define models of transaction formulas.

Definition 2.16 *(Models of Transaction Formulas) A path structure* $\boldsymbol{M}$ *is a model of a* $\mathcal{TR}$*-formula* ϕ*, denoted* $\boldsymbol{M} \models \phi$*, iff* $\boldsymbol{M}, \pi \models \phi$ *for every path* π *in* $\boldsymbol{M}$*. A path structure is a model of a set of formulas if and only if it is a model of every formula in the set.*

We now define executional entailment.

Definition 2.17 *(Executional entailment) Let* P *be a transaction base,* ϕ *a transaction formula, and* $s_0, s_1, \ldots, s_n$ *a sequence of databases. Then, the following statement*

$P, s_0, s_1, \ldots, s_n \models \phi$

is true iff $\boldsymbol{M}, < s_0, s_1, \ldots, s_n > \models \phi$ *for every model* $\boldsymbol{M}$ *of* P*. Related to this is the following statement:*

$P, s_0 - - \models \phi$

which is true iff there is a database sequence $s_1, \ldots, s_n$ *that makes* $P, s_0, s_1, \ldots, s_n \models \phi$ *true.*

Intuitively, the first statement means that a successful execution of transaction ϕ can change the database from state s_0 to $s_1 \ldots$ to s_n. Formally, it means that every model of P satisfies ϕ along the path $s_0, s_1, \ldots, s_n$. The second statement means that transaction ϕ can execute successfully starting from database s_0.

2.2.4.7 Reasoners

We are not aware of any Transaction Logic reasoner available. However, Prolog is very close in spirit to $\mathcal{TR}$ [Bonner and Kifer, 1995]. Although Prolog is not a logic of action or time, transactions can be defined in Prolog by using the operators *assert* and *retract*, which assert some new statement or retract some existing statement from the database.

Prolog transactions are close to Transaction Logic transactions in the following ways: a) updates are real, b) named procedures can be composed from simpler procedures, c) all predicates have both a truth value and a side effect on the database, and d) the frame problem is not an issue. However, updates in Prolog are not logical operators, and are always committed and they are not rolled back. For example, it is not possible in Prolog to perform an update tentatively, test its outcome, and then commit the update only if some condition is met.

In addition, updates in Prolog are not integrated into a complete logical system, and it is not clear how *assert* and *retract* interact with other logical operators. For example, it is not clear what $assert(X) \vee assert(Y)$ means.

However, $\mathcal{FLORA}$-2, a Prolog-based inference system which supports part of Transaction Logic (see Section 2.2.3.7), can be used to simulate the evaluation of some limited proof obligations expressed in Transaction Logic. How $\mathcal{FLORA}$-2 is used to automate the location of services based on Transaction Logic is presented in [Kifer et al., 2004] and it will be described in Chapter 4.

2.2.5 Languages and layering

We have introduced above logical formalisms which are related in different ways. We will discuss in the following how the formalisms introduced relate and, afterwards, we will present the most salient languages proposed for DL and LP and how they are layered.

2.2.5.1 Relation among formalisms

Description Logics are based on first-order logic and they have identical semantics. They restrict the expressivity of first-order languages so that the resulting language has desirable computational properties. In particular, most description logics are subsets or variants of $\mathcal{C}^2$, the fragment of first-order logic (without function symbols) in which formulas may contain at most two variables, but with counting quantifiers allowed, and which is known to be decidable [Graedel et al., 1997]. As a consequence of the restriction in expressivity imposed, the major reasoning tasks envisioned for description logics are kept decidable, which is a distinguishing property of Description Logics with respect to general first-order languages. Syntactically, Description Logics introduce simplifications in order to avoid the often cumbersome syntax of first-order logic. However, this does not have any impact on the expressivity and semantics of the language.

Logic Programming, as discussed in Section 2.2.3, is based on the Horn subset of first-order logic, and it has a semantics that is different from first-order semantics. In particular, the semantics of logic programs is based on minimal Herbrand models [Lloyd, 1987], rather than first-order models. Furthermore, the Closed-World Assumption (CWA), that is, assuming that what is not currently known to be true is false, is always made in Logic Programming and the type of negation used is negation as failure, i.e., facts are assumed to be false (by default) unless we can infer otherwise. On the contrary, under first-order semantics both the CWA or the Open World Assumption (OWA) (lack of knowledge does not imply falsity) can be made, although the OWA is usually made and the negation of a formula is only true if it can be explicitly inferred.

While the semantics of classic first-order logic and logic programming are not compatible in general, there is a fragment, which corresponds to the intersection of first-order logic and Horn

logic, for which ground entailment coincides under both semantics, i.e., the set of ground facts entailed under both first-order and LP semantics are equivalent. This fragment, studied in [Grosof et al., 2003] and [Volz, 2004], is called Description Logic Programs (DLP) or Description Horn Logic (DHL)[29], and it can also be as the intersection of the $\mathcal{SHOIN}$ description logic and Datalog.

Finally, Transaction Logic is complementary to both Logic Programs and Description Logics, as it is concerned with state change and it does not fix any state semantics. In fact, different fragments of Logic Programs and Description Logics could be used to establish the semantics of states, i.e., for the state data oracle.

2.2.5.2 OWL

The Web Ontology Language (OWL) [Bechhofer et al., 2004] is a World Wide Web Consortium (W3C) recommendation for a semantic web language, based on Description Logics. OWL consists of three species with increasing expressiveness, namely: OWL Lite, OWL DL, and OWL Full. All OWL species have different syntaxes, being their RDF/XML syntax the most prominent and, in fact, used in the language reference. However, the normative syntax for OWL is its abstract syntax, described in [Patel-Schneider et al., 2004]. In the following, we summarize the features of each of these sublanguages.

OWL Lite is a notational variant of the Description Logic $\mathcal{SHIF}(\mathcal{D})$. This Description Logic corresponds to $\mathcal{ALC}$ augmented with transitive roles, inverse roles, functional roles, a role hierarchy, and concrete domains. According to the OWL Language Overview [McGuinness and van Harmelen, 2004], OWL Lite supports those users primarily needing a classification hierarchy and simple constraints.

OWL DL is a notational variant of the Description Logic $\mathcal{SHOIN}(\mathcal{D})$ [Horrocks et al., 2003]. This Description Logic corresponds to $\mathcal{ALC}$ augmented with transitive roles, inverse roles, functional roles, a role hierarchy, nominals, arbitrary number restrictions, and concrete domains. According to the OWL Language Overview, OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computed) and decidability (all computations will finish in finite time).

OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. However, it turns out that OWL DL adds very little expressiveness to OWL Lite [Horrocks et al., 2003]. In fact, the only feature really added to OWL Lite by OWL DL is the use

[29] The term DLP was introduced in [Grosof et al., 2003] to denote Logic Programs based on DHL ontologies. In fact, the difference between DHL and DLP is that DLP as a Logic Program only allows for the entailment of ground facts and not the entailment of formulae. Horn Logic, as a subset of First-Order Logic, does allow the entailment of formulae. However, with respect to the entailment of ground facts, DHL and DLP are equivalent. [Volz, 2004] further develops DHL and DLP, but just uses the acronym DLP to indicate both [de Bruijn et al., 2004].

of nominals, i.e., the use of individuals in class descriptions; all other features added by OWL DL are only syntactic sugar, and they can be written down in OWL Lite using relatively complicated syntactic constructions.

It is important to notice that both OWL Lite and OWL DL pose several restrictions on the use of RDF and redefine the semantics of the RDFS primitives [Brickley and Guha, 2004b]. Therefore, OWL Lite and OWL DL are not properly layered on top of RDFS (see [de Bruijn et al., 2005f] for details).

OWL Full is the most expressive OWL species. It does not have a direct correspondence with any Description Logic. According to the OWL Language Overview, OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full. When compared with OWL DL, OWL Full does not pose any restriction on the use of OWL and RDFS constructs. While OWL DL requires a separation of types (a class cannot be an individual or property, and a property cannot be an individual or class), and requires that properties are either *ObjectProperties* or *DatatypeProperties*, OWL Full does not pose any of these restrictions. This results on an important increase on the complexity of reasoning with OWL Full. In fact, OWL Full does not offer computational guarantees.

OWL Full layers on top of RDFS and OWL DL and, because these languages are so different, the semantics of OWL Full is not straightforward and is not a proper extension of the OWL DL semantics [de Bruijn et al., 2005f]. In particular, entailment under OWL DL semantics is not equivalent to entailment under OWL Full semantics for the same ontology: OWL Full allows additional inferences. This discrepancy is caused by the incompatibility between the model-theoretic semantics of OWL DL and the axiomatic semantics of and syntactical freedom of RDFS. This raises doubts about the level of interoperability between the different species of OWL.

Reasoning support exists for OWL Lite and OWL DL, but not for OWL Full. In particular, all DL reasoners introduced in Section 2.2.2.5 support reasoning with OWL Lite ontologies, and Pellet and FaCT++ can handle OWL DL (the other reasoners do not provide complete and correct reasoning for nominals).

2.2.5.3 WSML

WSML [de Bruijn et al., 2005e; de Bruijn et al., 2005c] is a family of representation languages based on Description Logics, Logic Programming, and First-Order Logic, and with influences from F-Logic and frame-based representation systems. The motivation for WSML is to provide a

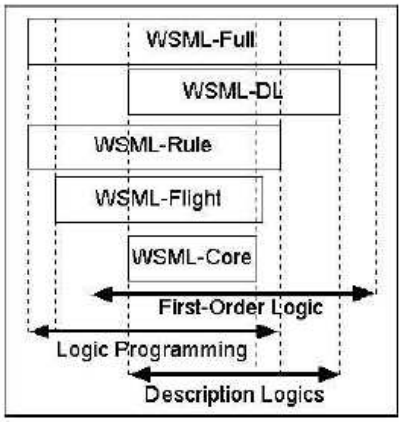

Figure 2.6: WSML Layering

family of languages not restricted to Description Logics, incorporating the benefits of Logic Programming and providing a core language that allows for a certain degree of interoperability among the different languages of the family.

WSML has five variants, namely: WSML-Core, WSML-Flight, WSML-Rule, WSML-DL, and WSML-Full. WSML has two alternative layerings, namely, WSML-Core → WSML-DL → WSML-Full and WSML-Core → WSML-Flight → WSML-Rule → WSML-Full. In both layerings, WSML-Core is the least expressive and WSML-Full is the most expressive language (see Figure 2.6).

All WSML variants are specified in terms of a human-readable syntax with keywords similar to the elements of the WSMO conceptual model [de Bruijn et al., 2005a]. Furthermore, WSML provides XML and RDF exchange syntaxes, as well as a mapping between WSML ontologies and OWL ontologies for interoperability with OWL-based applications. In the following we briefly introduce each of the WSML variants.

The two layerings are to a certain extent disjoint in the sense that interoperation between the Description Logic variant (WSML-DL) on the one hand and the Logic Programming variants (WSML-Flight and WSML-Rule) on the other, is only possible through a common core (WSML-Core) or through a very expressive (undecidable) superset (WSML-Full) [de Bruijn et al., 2005e].

WSML-Core is defined by the intersection of Description Logic and Horn Logic, based on Description Logic Programs (DLP), which is that subset of the Description Logic logic $\mathcal{SHIQ}(\mathcal{D})$ which falls inside the Horn logic fragment of First-Order Logic without equality and without existential quantification.

Two different types of reasoning can be done with WSML-Core, namely: a) subsumption reasoning and b) query answering. Subsumption reasoning is equivalent to checking entailment of

non-ground formulae and can thus be reduced to checking satisfiability using a First-Order style or a Description Logic-style calculus. Query answering is equivalent to checking entailment of ground facts. Thus query answering can be reduced to satisfiability checking. However, using a First-Order or Description Logic calculus for query answering is not very efficient [de Bruijn et al., 2004]. Fortunately, there are well-known techniques for query answering in the area of logic programming and deductive databases [Ullman, 1988]. Furthermore, WSML-Core is extended with datatype support in order to be useful in practical applications.

WSML-Core is fully compliant with a subset of OWL. As WSML-Core is based on plain (function- and negation-free) Datalog, thus, the decidability and complexity results of Datalog apply to WSML-Core as well. The most important result is that Datalog is data complete for P, which means that query answering can be done in polynomial time (see Section 2.2.3.6).

WSML-Flight extends WSML-Core with the full expressive power of Datalog rules, default negation, the full-blown use of integrity constraints (constraints are already in WSML-Core; however, they are only used for datatype predicates), (in)equality for abstract individuals, and meta-modeling. WSML-Flight allows to write down any Datalog rule, extended with inequality and (locally) stratified negation, and the semantics of WSML-Flight is grounded in Logic Programming. Since there exist no efficient implementation of query containment and since this problem is undecidable in general, the only reasoning task envisioned for WSML-Flight is query answering (i.e. entailment of ground facts). Still, subsumption reasoning can be done for the WSML-Core subset of a WSML-Flight ontology.

WSML-Rule extends WSML-Flight to a fully-fledged Logic Programming language, including function symbols and unsafe rules. WSML-Rule no longer restricts the use of variables in logical expressions.

WSML-DL (+). WSML-DL extends WSML-Core to an expressive Description Logic, namely, $\mathcal{SHIQ}$. The motivation for restricting this variant to $\mathcal{SHIQ}$ was that it was the Description Logic which could be efficiently handled by existing reasoners. However, there now exist effcient implementations which can deal with nominals, as we have discussed in Section 2.2.2.5, and, furthermore, it is desirable to be compatible with OWL DL. Therefore, we will consider a new variant of WSML, denoted WSML-DL+, which is semantically equivalent to the $\mathcal{SHOIN}(\mathcal{D})$ Description Logic, adding nominals to and disallowing number restrictions in WSML-DL.

WSML-Full unifies WSML-DL and WSML-Rule under a First-Order umbrella with extensions to support the nonmonotonic negation of WSML-Rule. It is yet to be investigated which kind of

formalisms are required to achieve this. Possible formalisms are Default Logic, Circumscription and Autoepistemic Logic.

Reasoning support for the DL part of WSML (WSML-Core and WSML-DL) is available, as it is provided by reasoners such as Pellet and FaCT++. Regarding WSML-Flight, it is supported by any of the LP reasoners introduced in Section 2.2.3.7. Support for WSML-Rule is provided by XSB, FLORA-2, OntoBroker, SILRI and MINS.

Syntax In the following, we briefly introduce the human-readable syntax of WSML logical expressions, as it will be used later in this document. For details on the conceptual syntax of WSML e.g. namespace declarations, importation of ontologies, or declaration of ontology elements, we refer the reader to the WSML specification [de Bruijn et al., 2005e]. This conceptual syntax can be mapped to the logical expression syntax, as presented in the specification.

It has to be noticed that the syntax which will be presented is the syntax of WSML-Full logical expressions; other variants define certain syntactic restrictions derived from their restrictions in the expressivity of the language (see the WSML specification for details).

Variables. Variable names start with a question mark *(?)* e.g. *?myVar*. The scope of a variable is always defined by its quantification. If a variable is not quantified inside a formula, the variable is implicitly universally quantified outside the formula[30].

Vocabulary. A vocabulary V of a WSML language $\mathcal{L}(V)$ consists of:

- A set of identifiers V_{ID}.
- A set of object constructors $V_O \subseteq V_{ID}$.
- A set of function symbols $V_F \subseteq V_O$.
- A set of datatype wrappers $V_D \subseteq VO$.
- A set of data values $V_{DV} \subseteq V_O$ which encompasses all string, integer and decimal values.
- A set of anonymous identifiers $V_A \subseteq V_O$ of the form $\_\#, \_\#1, \_\#2, \ldots$
- A set of relation identifiers $V_R \subseteq V_{ID}$.
- A set of variable identifiers $V_V \subseteq V_{ID}$.

Terms. Given a vocabulary V, the set of terms $Term(V)$ in WSML is defined as follows:

[30] Unless the formula is part of a capability description and the variable is explicitly mentioned in the *sharedVariables* block (see Chapter 3, Section 3.4.1).

- Any $f \in V_O$ is a term.
- Any $v \in V_V$ is a term.
- If $f \in V_F$ and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.
- If $f \in V_D$ and $d_{v1}, \ldots, d_{vn}$ are in $V_{DV} \cup V_V$, then $f(d_{v1}, \ldots, d_{vn})$ is a term.

Atomic formulae. Given a set of terms $Term(V)$, the set of atomic formulae in $\mathcal{L}(V)$ is defined by:

- If $\alpha, \beta \in Term(V)$ and $\gamma \in Term(V)$ or γ is of the form $\{\gamma_1, \ldots, \gamma_n\}$, with $\gamma_1, \ldots, \gamma_n \in Term(V)$, then:
  - α *subConceptOf* γ is an atomic formula in $\mathcal{L}(V)$. Here, α and γ both identify concepts (unary predicates), and this expression states that α is a subconcept of γ, i.e., that $\alpha(x) \rightarrow \gamma(x)$.
  - α *memberOf* γ is an atomic formula in $\mathcal{L}(V)$. Here, α identifies an instance (constant) and γ identifies (a) concept(s), meaning that α is an instance of γ, i.e., that $\gamma(\alpha)$ holds.
  - $\alpha[\beta$ *ofType* $\gamma]$ is an atomic formula in $\mathcal{L}(V)$. Here, α identifies an instance, β identifies an attribute (binary predicate) and γ identifies (a) concept(s). The expression means that α has a property β whose type is *constraint* to γ.
  - $\alpha[\beta$ *impliesType* $\gamma]$ is an atomic formula in $\mathcal{L}(V)$. Here, α identifies an instance, β identifies an attribute and γ identifies (a) concept(s). This expression means that α has a property β whose type is restricted to γ[31].
  - $\alpha[\beta$ *hasValue* $\gamma]$ is an atomic formula in $\mathcal{L}(V)$. Here, α identifies an instance, β identifies an attribute and γ identifies (an) instance(s). This expression means that the property β of instance α has value γ.

  These atomic formulae are also called molecules.
- If $r \in V_R$ (r is an n-ary predicate with $n > 1$) and $t_1, \ldots, t_n$ are terms, then $r(t_1, \ldots, t_n)$ is an atomic formula in $\mathcal{L}(V)$.
- If $\alpha, \beta \in Term(V)$ then $\alpha = \beta$, $\alpha :=: \beta$ and $\alpha! = \beta$ are atomic formulae in $\mathcal{L}(V)$.

Formulae. The set of formulae in $\mathcal{L}(V)$ is defined by:

- Every atomic formula in $\mathcal{L}(V)$ is a formula in $\mathcal{L}(V)$.

[31] See [de Bruijn et al., 2005g] for a discussion on restrictions and constraints.

- Let α, β be formulae which do not contain the symbols : − and !−, and let $?x_1, \ldots, ?x_n$ be variables, then:
  - α *and* β is a formula in $\mathcal{L}(V)$.
  - α *or* β is a formula in $\mathcal{L}(V)$.
  - *neg* α is a formula in $\mathcal{L}(V)$.
  - naf α is a formula in $\mathcal{L}(V)$.
  - $forall$ $?x_1, \ldots, ?x_n(\alpha)$ is a formula in $\mathcal{L}(V)$.
  - *exists* $?x_1, \ldots, ?x_n(\alpha)$ is a formula in $\mathcal{L}(V)$.
  - α *implies* β is a formula in $\mathcal{L}(V)$.
  - α *impliedBy* β is a formula in $\mathcal{L}(V)$.
  - α *equivalent* β is a formula in $\mathcal{L}(V)$.
  - α *:-* β is a formula in $\mathcal{L}(V)$. This formula is called an LP (Logic Programming) rule. α is called the head and β is called the body of the rule.
  - $! - \alpha$ is a formula in $\mathcal{L}(V)$. This formula is called a constraint. We say α is a constraint of the knowledge base.

Examples of logical expressions in WSML human-readable syntax, taken from [de Bruijn et al., 2005e], are:

"No human can be both a male and a female"

```
!− ?x[gender hasValue {?y, ?z}] memberOf Human and ?y = Male and ?z = Female.
```

"A human who is not a man is a woman"

```
?x[gender hasValue Woman] impliedBy neg ?x[gender hasValue Man].
```

"The brother of a parent is an uncle"

```
?x[uncle hasValue ?z] impliedBy ?x[parent hasValue ?y] and ?y[brother hasValue ?z].
```

"Do not trust strangers:"

```
?x[ distrust  hasValue ?y] :− naf ?x[knows hasValue ?y].
```

Please remember that the syntax presented above is just a way of writing formulas which will have the expressive power and semantics defined for each variant. For example, logical expressions within WSML-DL are a syntactical variant of the $\mathcal{SHIQ}(\mathcal{D})$ Description Logic presented in Section 2.2.2. Details of mappings from WSML human-readable syntax to other syntaxes, as well as the detailed definition of the semantics of each variant can be found in [de Bruijn et al., 2005e].

2.3 Summary

The SOA paradigm is gaining increasing acceptance, and its uptake is already underway. Basic technologies for the design, description and exposition of services, such as web service technologies, are also available and widely supported, and languages for complementary aspects such as security and policies have been developed. Furthermore, infrastructure such as that provided by Enterprise Service Buses can facilitate building real service-oriented architectures.

However, some challenges remain unsolve for the effective exploitation of the potential benefits of the SOA paradigm, mainly related to the limited dynamics of service-oriented computing. One of these challenges is the increase in the level of automation of service discovery, i.e., of the task of locating appropriate services for solving a given goal.

Describing services and goals using ontologies and languages with formal semantics is identified as a promising path towards achieving an enhancement in current service discover practices; using formal ontologies and formal languages, the precise semantics of the value of services and of consumer goals can be captured and exploited for enhancing service discovery. For this reason, we have introduced not only what SOA is, what technologies are mainly used for designing and exposing services, and what challenges remain open, but also the relevant field of semantic web research, which aims at describing information on the web based on explicit, shared and formal models (ontologies) so that information is given a well-defined meaning and current limitations of the web are overcome.

Besides presenting introducing the semantic web vision, we have presented in more detail the logical foundations of semantic languages, which will be used in our work on enhancing the location of services, and we have discussed what these languages offer, what reasoning support already exists, and how different formalisms relate. We have also introduced transaction logic, which can be used to model and deal with state change and which has been used in one of our attempts to automate the location of services, as it will be presented in Chapter 4.

In the next Chapter, we concentrate on clearly defining the core conceptual elements involved in the location of services. For this purpose, we will provide an explicit model of services and goals which will be used as a basis for the discussion in the following Chapters on the design and realization of an approach for enhancing the location of services. Furthermore, we will formally characterize core elements of this model using Transaction Logic, as it enables capturing the dynamics of states and services.

Chapter 3

Conceptual model of services and goals

3.1 Introduction

The relatively early stage of SOA development and adoption has created some terminology confusion. While terms like "SOA" and "service", among others, are increasingly used in all kinds of written publications and product descriptions, certain ambiguity exists around the core conceptual elements that interplay in a service-oriented architecture and the terms used to refer to them. For example, services are often assimilated to Web services, while some authors also use the term "service" in the sense of a business value e.g. [Baida et al., 2004; Preist, 2004].

A remarkable step towards reaching a consensus on the definition of core elements in an SOA, as well as on the terminology used for referring to such elements, has been the recent definition of a reference model for service oriented architecture by OASIS [MacKenzie et al., 2006], aiming at providing an abstract framework which *consists of a minimal set of unifying concepts, axioms and relationships within the SOA domain, and which is independent of specific standards, technologies, implementations, or other concrete details*, thereby reducing the confusion created by the proliferation of conflicting terminologies.

In this Chapter, and heavily based on the reference model defined by OASIS, we provide a conceptual view of the core elements of an SOA that are particularly relevant to our work, such as services and goals. We further elaborate the reference model where we deem necessary, and briefly explore the relationship between the concepts and terminology introduced by such model and those used by existing frameworks for modelling services such as WSMO [Roman et al., 2005] or OWL-S [Martin et al., 2004]. The conceptual model is presented in Section 3.2.

Besides introducing a (partial) reference model for SOA, we investigate further in this Chapter the nature of elements such as services and goals. In particular, we provide a deeper insight into the type of artifacts we aim at describing for their semi-automatic location, and formally characterize them based on Transaction Logic in Section 3.3, as it is crucial to provide a clear and explicit model of these artifacts so that we can properly understand what types of descriptions can be provided, what aspects they capture, and what level of confidence can be expected from the results of the location process based on the different types of descriptions.

In Section 3.4 we briefly introduce the frameworks proposed so far for the description of services and goals, and discuss how our model relates to the model of services and goals used by these frameworks. We will pay special attention to the WSMO framework [Roman et al., 2005; de Bruijn et al., 2005a] and to how the WSMO model of services and goals relates to our model. Finally, in Section 3.5, we provide a summary of the contents of this Chapter.

3.2 Conceptual model

In [MacKenzie et al., 2006], a reference model for SOA is presented with the purpose of defining core elements of SOA in a technology-independent manner. This model defines SOA as a *paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains*, and attributes the value of SOA to the fact that *it provides a powerful framework for matching needs and capabilities and for combining capabilities to address those needs.*

The central concepts introduced by the model are: *service, real world effect, capability*[1]*, service description, visibility, interaction, contract and policy,* and *execution context.* We will focus in the following on those concepts that are particularly relevant to the process of locating appropriate services and refine some of the concepts defined by the OASIS reference model.

3.2.1 Capabilities

Definition 3.1 *(Capability) A capability is the ability of performing some action or actions with a perceived value, in the sense that they can constitute a (perhaps partial) solution to some problem.*

Capabilities are associated to entities (either people or organizations), who can offer their capabilities to interested parties. The actual way such capabilities are brought to bear is independent of the capability itself.

Example 3.1 Let us imagine a travel agency which can book seats on flights operated by airlines which are not low-cost airlines. In this case, the capability offered by the travel agency is the booking

[1] Although this concept is not explicitly enumerated in [MacKenzie et al., 2006] as a core element, it is repeatedly used and we consider it central for the automatic location of services.

of this type of flights. This capability is independent of how it is put at disposal of interested customers. For example, customers can visit one of the offices of the travel agency network, or use the travel agency web site to actually get a seat booked on a given flight, i.e., there exist alternative ways to bring the capability to bear.

□

A capability has associated certain effects, which we define below; the purpose of using a capability is to realize all or part of such effects [MacKenzie et al., 2006].

Definition 3.2 *(Effect) An effect is a result of using a capability, and it can be of one of the following two types:*

1. *Some information is made visible to the party using the capability, or*
2. *There is a change to some state shared by (at least) the party providing the capability and the party using the capability.*

We call the former type of effect information effect *or* information provision*, and the latter* real world effect*.*

A capability must have associated at least one effect, and there is no upper bound for the number of effects that can be associated to a capability. From these effects, zero or more can be information effects, and zero or more can be real world effects. The perceived value of a capability actually corresponds to its effects, and the decision of a party of using a certain capability will depend on these effects.

General effects, independently of their type, are called real world effects in [MacKenzie et al., 2006]. However, we change this terminology in order to emphasize the distinction between the different types of effects and in order to align it more with the terminology used by proposals such as [de Bruijn et al., 2005a] or [Martin et al., 2004]. The name *real wold effect* given to the latter type of effects is used to reflect that it corresponds to some change beyond the provision of information, i.e., it refers to an effect on a "real world" e.g. the actual booking of some seat on a flight, as opposed to an "information world", where only the information known by the consumer changes but this does not affect any state shared with other parties.

How the information associated to an information provision is made visible to the interested party is not part of the definition of the effect.

Example 3.2 Let us consider the capability introduced in the previous example, and let us suppose the payment of the booking must be done with a credit card. The effects associated to such capability are:

- Seats are booked on some flight operated by an airline which is not a low-cost airline,
- the customer credit card is charged by the price of the seats, and
- a confirmation of the booking is provided to the customer.

From the effects above, the first two are real world effects, while the last one is an information effect.

□

3.2.2 Services

Different means can be available for using a capability and thus, for achieving its associated effects. This leads to the concept of service, which is defined in the following.

Definition 3.3 *(Service) A service is a mechanism by which capabilities are brought to bear, enabling access to them.*

A service is thus the means by which a certain capability is accessed and its effects achieved; there is no constraint on how this access is actually implemented. The capability a service enables access to is called the *service capability*.

Example 3.3 The capability of example 3.1 can be accessed in different ways. One such way could be to enable a network endpoint to which a SOAP message can be sent over secure HTTP with information about the desired booking and credit card details for the payment of the booking. This will cause the actual booking of a seat on the desired flight and the charge of the credit card provided, as well as a SOAP message being returned to the sender of the first message, containing the booking locator. This particular mechanism of providing the capability is a service.

□

As it can be seen from the example, the service is only a means to access a capability, but not the capability itself. Furthermore, a capability can be made simultaneously accessible in different ways, i.e., through different services. In this document, we will focus on electronic services, i.e., on services that provide electronic access to capabilities such as Web (WSDL) services (see Chapter 2).

Every service is provided by a particular party which we call the *service provider*, and it is used by a *service consumer*. It must be noted that service providers can also act as service consumers and vice versa. For example, the service above might make use of other services provided by different airlines in order to provide the booking of flights, acting as a service consumer for the airlines and as a service provider for some end user.

Providers of a service often require certain information to be provided by the consumer and certain conditions to hold on some shared state for enabling access to the underlying capability associated to a particular service. We call these requirements *preconditions* for service provision, and they are defined next.

Definition 3.4 *(Precondition) A precondition for service provision is a condition imposed by a given service for actually enabling access to its underlying capability, i.e., for bringing its capability to bear. It can be of one of the following two types:*

1. *Some information is required to be provided by the consumer of the service, or*
2. *Some conditions must hold on some state shared by (at least) the service consumer and the service provider.*

We call the former type of precondition information precondition*, and the latter* real world precondition*.*

A service can have associated zero or more preconditions, and there is no upper bound for the number of preconditions for the provision of the service. From these, zero or more can be information preconditions, and zero or more can be real world preconditions.

Example 3.4 The service from the previous example requires information about the desired booking for enabling access to the capability of booking seats on flights not operated by low-cost airlines. This is an example of an information precondition.

The service provider also requires a seat to be available on the flight requested in order to actually provide the booking. This is an example of a real world precondition.

□

We can see, therefore, that a service enables access to a certain capability only under certain conditions. Furthermore, different services might impose different conditions for enabling access to the same capability, i.e., they might require different information to be provided by the service consumer and different conditions to hold on some shared state.

For the service consumer to use any service in order to access its underlying capability, some *interaction* must take place. If such interaction succeeds, which will happen if the preconditions of the service are fulfilled, some effects associated to the capability exposed by the service are achieved. This leads to the concept of service execution.

Definition 3.5 *(Service execution) A service execution is a particular interaction between a consumer and the service, in which the preconditions of the service are fulfilled, and which leads to the achievement of some effects of the service capability.*

Notice that the above definition refers to successful executions of the service. However, the interaction will fail if the preconditions of the service are not fulfilled. Furthermore, other reasons might also lead to failed interactions, such as technical problems, although we will generally ignore these other possible sources of failure of an execution.

The effects achieved after a particular service execution are usually only a subset of the effects of the underlying capability, as illustrated by the following example.

Example 3.5 Let us consider the service in Example 3.3, and let us assume the real world precondition mentioned in Example 3.4 is fulfilled. Now, let us imagine that information about credit card *myCC* is sent to the service, and that the flight to be booked is a flight $flight4321$ from Madrid to La Havana, on August 1st, 2007, and the name of the passenger Ruben Lara. The effects achieved will be that a seat is booked for the passenger and on the flight given by the service consumer, that credit card *myCC* will be charged by the price of such seat, and that a confirmation of the booking of this particular seat will be sent to the service consumer.

□

It can be seen that, from all possible effects of the capability related in Example 3.2 (booking of seats on flights which are operated by a non-low-cost airline), only the booking of a particular seat on a particular flight is achieved from the set of all possible bookings associated to the capability. In this sense, we can see the effects of the capability as *abstract* or *potential effects*, which refer to general effects associated to the capability, and the effects achieved after a service execution as *realized* or *concrete effects*. It has to be noted that realized effects will always be part of the abstract effects of the capability the service enables access to, and that realized effects constitute the real value obtained by the service consumer from service usage.

Moreover, the fulfillment of preconditions is not only a pre-requisite for a service execution, but how they are fulfilled also conditions what concrete effects will be achieved. The relation between the information provided by the service consumer, some state shared by the service consumer and provider, and the effects achieved by using the service is what we call the *service functionality*.

Definition 3.6 *(Service functionality) The service functionality is the (possibly non-deterministic) relation between how service preconditions are fulfilled and the particular effects that will be achieved by its usage.*

If we consider deterministic services, the service functionality can be defined as a function that maps particular information provided by the service consumer and some shared state to particular realized effects, i.e., it defines possible service executions. Therefore, we can see the capability associated to a service as the general effects that can be achieved by using the service, while the

service functionality defines what particular effects associated to the capability will be achieved under different situations that fulfill the service preconditions. As a consequence, different services can enable access to the same capability but with a different functionality, i.e., the same set of effects is achievable by using the services but under different conditions.

Example 3.6 Let us consider the service from Example 3.3. The function it provides is the booking of seats on a flight that fulfills the criteria given by the service consumer, paid with the credit card given, and the provision of a confirmation of the booking for the flight, passenger and credit card given.

Another service might enable access to the same capability, but requiring the name of a registered user of the service and always assuming that the passenger is the person associated to such user. It can be seen, thus, that the capability associated to both services is the same, but they provide a different mapping from initial information and real-world conditions to the effects associated to such capability.

□

It must be noted that the effects achieved by a service execution will be those dictated by the service functionality, i.e., the service functionality describes the general mapping of preconditions to effects, while the service execution is the realization of a particular mapping.

As introduced before, for a service to be executed some interaction must take place between the service consumer and the service provider, and such interaction can only be carried out in some explicitly ways prescribed by the service provider. The *service interface* defines how interaction with the service must take place.

Definition 3.7 *(Service interface) The service interface is the means for interacting with a service. It defines the specific protocols, commands, and information exchange by which a service is executed. Information required by the service is represented by interface* inputs*, and information provided by the service is represented by interface* outputs*.*

The service interface includes the definition of how information will be provided to the service, how the service will return information to the consumer, in which order this interaction will happen, what particular information will be exchanged (via message exchange or in any other way) and, in general, how the interaction with the service must take place. This interaction model can be arbitrarily complex. However, we will assume, unless stated otherwise, that interactions with services are always in one shot, i.e., some information is provided to the service, and some effects will be realized as a consequence of the service execution (possibly including some information being returned in response), abstracting from multi-step interactions and from technical details associated to this interaction such as communication protocols used. In this way, we generally see interfaces

as the definition of a one-shot information exchange between the service consumer and the service provider, i.e., the interface only defines the inputs and outputs of the service.

Interface inputs and outputs only represent information elements, but do not define the conditions such information elements must fulfill. As we will see in Section 3.3, they are treated as names that identify information elements that have to be provided by the consumer and fulfill the service preconditions (inputs), or information elements that are provided by the service as part of the effects of the execution.

Example 3.7 A given service can have an interface with inputs i_1, i_2 and output o_1. Information preconditions of the service constrain possible values assigned to these inputs e.g. establishing that i_1 must be the description of a flight between European countries, and that i_2 must be a user name. The information effect of the service is e.g. that a booking locator is provided to the consumer, and such information will be identified by the output o_1.

□

In a nutshell, the interface is seen as only a declaration of information elements required and provided. However, real interfaces will generally incorporate details on how inputs and outputs have to be provided, in what order, etc.

3.2.3 Visibility

Before a service consumer can interact with a service in order to achieve some effects associated to its capability, such service must be located, i.e., the service consumer and the service provider must be able to see each other and to interact via the service interface. This leads to the concept of visibility.

Definition 3.8 *(Visibility) Visibility is the relationship between service consumers and providers that is satisfied when they are able to interact with each other.*

As detailed in [MacKenzie et al., 2006], requisites for visibility are that the service consumer and provider must be aware of each other's existence, they must be willing to interact, and they must be mutually reachable, i.e., they must be able to communicate with each other. Actually, this document focuses on how visibility is resolved, assuming the willingness of the parties involved to interact and their mutual reachability, and *concentrating on how service consumers can become aware of the existence of services, offered by some service providers, that enable access to certain capabilities.*

3.2.4 Goals

Service consumers are willing to use a certain service because it provides access to a certain capability that in turn has associated some effects fulfilling some consumers' needs. This means that a service consumer is actually not interested in using or consuming a service per se, but in *using a service to consume a certain capability associated to the service because it fulfills some particular consumer's needs.* This leads to the introduction of the concept of goal.

Definition 3.9 *(Goal) A goal is a set of needs a service consumer expects to resolve by using a service.*

Goals correspond to the objectives consumers have when using services, and they thus drive the decision on what services to use. The concept of goal is symmetric to the concept of capability: while capabilities express offers, goals express requirements that can be potentially met by such offers. In fact, goals can be also defined as a set of effects required, and they can include both information effects and real world effects. Therefore, the decision on what services to use will depend on the capabilities different services offer, particularly on whether their effects can resolve the needs defined by a goal, i.e., whether their effects can fulfill the requirements defined by such goal.

Example 3.8 An example of a goal is the desire of booking a seat on a flight from Madrid to Manchester, on a given date, and paid with a particular credit card.

□

In the example above, the consumer has the need of booking a seat on a flight with the characteristics given, and he expects to solve it by using some services that enable access to capabilities whose effects fulfill such need.

We have discussed above how there can be some preconditions for the usage of a given service, and how the concrete effects achieved by using the service will depend on the way these conditions are fulfilled. In particular, information preconditions refer to information *the consumer* must provide to the service. Therefore, it is not only relevant to know what effects the consumer expects from using a service, but also consumer knowledge and what particular knowledge the consumer is able and willing to provide to the service. This leads to the concept of consumer knowledge and consumer knowledge available for a given service.

Definition 3.10 *(Consumer knowledge -available-) Consumer knowledge is the set of all information known by a certain consumer at a given point in time. Given a service serv, the consumer knowledge available for serv is the set of information the customer is willing to disclose* to this particular service *from the set of information it knows.*

Consumer knowledge will determine whether a consumer will be able at all to satisfy the information preconditions of a service which can be used for fulfilling a given goal. Furthermore, consumers might be willing to disclose certain information only to certain services. For example, a consumer might be willing to disclose credit card details only to services certified by a given set of authorities.

Finally, we define the global state of the world as follows:

Definition 3.11 *(-Global- state of the world) The (global) state of the world is a state shared by all parties in a given system. This state determines what is true in the* real world.

It must be noted that the state of the world, unlike consumer knowledge, is not dependent on the service consumer; it does not depend on the service provider either, but it is a global state shared by all parties. Service real-world preconditions and real-world effects actually refer to this state of the world or to some subset of it.

3.3 Formal Characterization

In the previous Section, the core conceptual elements used in our model have been defined. In this section, we provide a deeper insight into these concepts and a formal characterization of them. This formal characterization will allow us to better understand what types of descriptions of these elements can be provided, what aspects they capture, and how they can be used to achieve visibility between consumers and appropriate services for solving a goal.

The formal characterization given is based on Transaction Logic (see Chapter 2, Section 2.2.4), which accounts in a declarative fashion for the phenomenon of state change, central in the formalization of services and related concepts. The major reason for using $\mathcal{TR}$ in the formal characterization of the elements previously introduced is that $\mathcal{TR}$ separates the specification of elementary operations from the logic of combining them, which makes $\mathcal{TR}$ a language for specifying and possibly executing state-changing procedures without committing to a particular theory of elementary updates, and allows $\mathcal{TR}$ to accommodate a wide variety of state semantics, from classical to non-monotonic to various other non-standard logics.

Other logics such as Process Logic [Harel et al., 1982], Dynamic Logic [Harel, 1979; Harel et al., 2000] or Temporal Logic [Emerson, 1990], are also candidates for formally characterizing the core concepts of our model, as they verse over states, dynamics and time. An approach as the one presented by Keller et al. in [Keller and Lausen, 2006; Keller et al., 2006b] could also be followed, defining a formalism-independent model for characterizing these artifacts. However, we have chosen to use Transaction Logic as it is a well-defined logic for specifying state-changing procedures which grants us the flexibility of being able to accommodate alternative state semantics. Furthermore, in

[Kifer et al., 2004] we have presented a first attempt to automate the location of services based on Transaction Logic reasoning which is described in Chapter 4.

In the following, we will start by presenting some initial choices we make for formally characterizing the concepts in our model. Then, we formally characterize services and related concepts. Capabilities are then characterized, and we end with the formal characterization of goals.

It has to be kept in mind, though, that the formalization which will be introduced in this Section is meant to precisely capture the nature of the artifacts we will deal with for enhancing the discovery of services, not to be directly usable for automating discovery. The reason is that there is no sufficient reasoning infrastructure for Transaction Logic (see Chapter 2, Section 2.2.4) and that the types of descriptions required are often too complex to be manageable by most users. In this setting, the formalization below is useful to comprehend what services, goals, capabilities, etc. are and how they can be modelled, but simplifications will be introduced in future Chapters in order to obtain manageable descriptions of relevant artifacts which can be efficiently and flexibly exploited for enhancing the discovery of services.

3.3.1 Language

As mentioned before, we will use Transaction Logic to formally characterize the core concepts used in our model of services and goals. In particular, we will use a $\mathcal{TR}$ language $\mathcal{L}$, with signature $\Sigma = (C, F, P, V)$. Formulae will be interpreted over a universe $\mathcal{U}$.

We will assume in the following the existence of a set O of domain ontologies which define the domain vocabulary and provide knowledge about such domain. The set F of function symbols and the set P of predicate symbols of signature Σ will be given by this set O of domain ontologies, i.e., domain ontologies introduce the function and predicate symbols that can be used.

3.3.1.1 State data oracle

We will assume that a state is defined by a set of ground atoms, i.e., by a set of first-order, variable-free atomic formulas. This means that the state database contains a set of ground atoms.

Remember from Chapter 2, Section 2.2.4 that a state data oracle was defined as a mapping from state identifiers to closed first-order formulas stating what is true at those particular states; if s is a state identifier, then $\mathcal{O}^d(s)$ is the set of formulas considered to be true about that state. In our setting, this mapping simply returns all the ground atoms in the database, i.e., $\mathcal{O}^d(s) = s$. This corresponds to a relational oracle as described in [Bonner and Kifer, 1995; Bonner and Kifer, 1998]. However, notice that the state data oracle and, thus, the semantics of states could be changed if necessary. In this way, we are formally characterizing the nature of the artifacts introduced, but we leave open the possibility of changing the semantics of states.

Different parties involved in a service interaction might have different knowledge of what is true at a given state, as shown in the following example.

Example 3.9 Let us imagine a state s and a relational oracle. Let us further imagine that $\{inCity(john, london)\} \in \mathcal{O}^d(s)$, i.e., this ground atom is contained in the state database, as told by the state data oracle, at current state s.

Still, there can be a party which does not know that $incity(john, london)$ holds at the current state. Precisely, one of the reasons of a service consumer for using a service can be to have access to some knowledge about the current state. □

For this reason, we will consider not only a single, global state and its corresponding database, as described in [Bonner and Kifer, 1995; Bonner and Kifer, 1998]; besides this global database, we will assume the existence of a database defining a state s^p for every party p in a given system. A state s^p is defined by a set of ground atoms corresponding to *what is known about the current state by party* p, i.e., the party database is a partial view of the global state database.

Intuitively, we have a single current, global state s, and we have different partial views, one per party, of such state. What holds at the current state is given by $\mathcal{O}^d(s)$, and what is known to hold by party p is given by $\mathcal{O}^d(s^p)$. In this way, we can see that we have a global database which stores what holds at the current state, and a database per party which stores a partial view of the global database. We assume these partial views are always consistent with the global database.

In a transaction formula, we will denote $[q]^p$ the evaluation of ground atom q at the database of party p, i.e., the evaluation of whether $q \in \mathcal{O}^d(s^p)$, where s^p is the current state as viewed by party p. We will simply write q when we evaluate predicate q at the global database, i.e., when we evaluate whether $q \in \mathcal{O}^d(s)$, where s is the current global state.

3.3.1.2 State transition oracle

Remember from Section 2.2.4 that a state transition oracle is a function $\mathcal{O}^t$ that maps pairs of states into sets of ground atomic formulas, being these ground atomic formulas the elementary transitions or elementary updates. Intuitively, for an initial state s_1 and a final state s_2, $\mathcal{O}^t(s1, s2)$ gives the set of elementary transitions that can perform the transition from the initial to the final state

We will use a state transition oracle which defines, for each predicate p, two new predicates $p.ins$ and $p.del$, as described in [Bonner and Kifer, 1995], representing the insertion or deletion of single atoms, respectively. Formally,

$$p.ins(\overline{a}) \in \mathcal{O}^t(s_1, s_2) iff s_2 = s_1 \cup \{p(\overline{a})\}$$
$$p.del(\overline{a}) \in \mathcal{O}^t(s_1, s_2) iff s_2 = s_1 - \{p(\overline{a})\}$$

where p is some predicate in Σ of arity n and $\overline{a}$ is an array of n constants.

Example 3.10 If we consider state s in the previous example, a possible elementary operation would be $inCity.del(john, london)$, which will go from state s to a state s' where $inCity(john, london)$ does not hold anymore, i.e.,

$$\mathcal{O}^d(s') = \mathcal{O}^d(s) - \{inCity(john, london)\}$$

Another possible elementary operation would be $inCity.add(john, lisbon)$, which, if executed at state s' would lead to a state s'' such that

$$\mathcal{O}^d(s'') = \mathcal{O}^d(s') \cup \{inCity(john, lisbon)\}$$

□

Notice that, as with the state data oracle, we leave open the possibility of replacing this oracle by a different type of oracle. The reason for using this pair of state data and state transition oracles is two-fold: a) we believe it suffices for representing the state of most service-oriented systems, and b) it avoids the creation of inconsistencies when updating states (see [Bonner and Kifer, 1995] for a more detailed discussion).

Elementary transitions, as defined in [Bonner and Kifer, 1998; Bonner and Kifer, 1995], refer to the update of a single, global current state. However, and as we have discussed above, we consider partial views of this state, i.e., partial databases for each party. We will therefore extend the primitives defined by the state transition oracle to primitives that add or delete formulas from a particular party database. These primitives will not cause a change of the global state, but a change of the knowledge of a party about the current state. We write $[q.del]^p$ and $[q.ins]^p$ for deleting (adding) a ground atom q to the database of party p.

Example 3.11 Let us assume the current state is the state s'' described in the previous example, and let us consider a party p and its associated current state s^p such that $\{inCity(john, lisbon)\} \notin \mathcal{O}^d(s^p)$. The execution of the elementary transition $[inCity.ins(john, lisbon)]^p$ will make party p know that this atom holds at the current state, i.e., it will cause a change in the current knowledge of party p, given by state s^p, to a new state s'^p where p knows that $inCity(john, lisbon)$ holds:

$$\mathcal{O}^d(s'^p) = \mathcal{O}^d(s^p) \cup \{inCity(john, lisbon)\}$$

Notice that the execution of this primitive will not cause any change in the global state s'', but only in the knowledge of party p of the current state.

□

3.3.1.3 Domain ontologies and the state data oracle

Remember we assume the existence of a set $\mathcal{O}$ of domain ontologies which provide domain knowledge. This knowledge can include the definition of axioms that can enable the derivation of new knowledge (including restrictions), and we assume these axioms are bound first-order formulas without negative literals.

In the following example, we show how the consideration of axioms defined by domain ontologies enables the derivation of new knowledge not captured by the set of ground atoms which define a state.

Example 3.12 If we consider an ontology containing the following formulas:

$$\forall x.\exists y.inCity(x, y) \rightarrow inWorld(x)$$
$$\forall x, y, z.inCity(x, y) \wedge inCity(x, z) \rightarrow y = z$$
$$city(london)$$

and consider the states from the previous examples, we can derive that $inWorld(john) \in \mathcal{O}^d(s)$ and that $city(london) \in \mathcal{O}^d(s)$. If we would also have that $inCity(john, ukCapital) \in \mathcal{O}^d(s)$, we could also infer that $london = ukCapital$, i.e., that both constants refer to the same domain object.

□

We will therefore extend the definition of the mapping function of our relational state data oracle in the following way:

$$\mathcal{O}^d(s) = \{\phi \mid s, \mathcal{O} \models^c \phi\}$$

where $\mathcal{O}$ is the set of domain ontologies considered, $\models^c$ denotes classical entailment, and ϕ is some bound first-order formula with only positive literals. Intuitively, this means that the set of truths at a given state will not only be the set of truths stored in the state database, but also the

set of truths that can be entailed from the combination of the knowledge contained in such database and in the domain ontologies used.

3.3.2 Services

In the following, we will formally characterize services, their execution, and other related concepts using Transaction Logic.

Definition 3.12 *(Formal characterization of service execution)*

A particular execution of a service by a service consumer c is defined by a pair of execution paths (χ, χ^c)*, where:*

1. $\chi = s_0, \ldots, s_i, \ldots, s_f$ *is an execution path which corresponds to the transition from a global state* s_0 *to a global state* s_f*, passing through any number of intermediate states, and*
2. $\chi^c = {s_0}^c, \ldots, {s_j}^c, \ldots, {s_g}^c$ *is an execution path which corresponds to the transition from state* ${s_0}^c$ *to state* ${s_g}^c$ *passing through any number of intermediate states, i.e., to a change in the knowledge of the service consumer of the global state.*

It can be the case that either $s_0 = s_f$ *or* ${s_0}^c = {s_g}^c$*, but not both. The former case means that there is no change in the global state, and the latter that there is no change in the knowledge of the service consumer caused by the service execution.*

We remind from Section 3.2.2 that for a service execution to happen certain conditions called preconditions must be fulfilled. In particular, a precondition for service provision was defined as a condition imposed by the service for actually enabling access to its underlying capability. This condition can be a condition over the shared state (real world precondition), or a requirement of information that must be satisfied by the service consumer (information precondition).

Regarding the provision of information by the consumer to the service, the selection of what information will be provided to the service will be defined through an *input binding*. Given a service *serv*, whose interface defines a set of input variables $i_1, \ldots, i_n$, an input binding is a total function, defined by the service consumer, $\beta : \{i_1, \ldots, i_n\} \rightarrow \mathcal{U}$. This function assigns objects in the universe $\mathcal{U}$ to every input variable of the service [Keller et al., 2006b].

Considering the formal characterization of a service execution, the definition of preconditions in Section 3.2.2, and the definition of input binding above, we can formally characterize preconditions as follows:

Definition 3.13 *(Formal characterization of service preconditions)*

Given a service serv with input variables $i_1, \ldots, i_n$*:*

- *Information preconditions of a service serv are formalized by an n-ary first-order predicate* $pre^{inf}_{serv}(i_1, \ldots, i_n)$ *in* $\mathcal{L}$*, where n is the number of input variables of the service interface.*
- *Real world preconditions of a service serv are also formalized by an n-ary first-order predicate* $pre^{rw}_{serv}(i_i, \ldots, i_n)$ *in* $\mathcal{L}$*.*

Given a service consumer c, an input binding $\beta : \{i_1, \ldots, i_n\} \rightarrow \mathcal{U}$ *defined by c, and a pair of execution paths* (χ, χ^c) *of service serv, with* $\chi = s_0, \ldots, s_i, \ldots, s_f$ *and* $\chi^c = s^c_0, \ldots, s^c_j, \ldots, s^c_g$*, we have that*

$$\mathcal{O}^d(s_0) \models pre^{rw}_{serv}(\beta(i_i), \ldots, \beta(i_n))$$
$$\mathcal{O}^d(s^c_o) \models pre^{inf}_{serv}(\beta(i_i), \ldots, \beta(i_n))$$

Intuitively, preconditions define conditions that must hold at the initial state of a service execution so that the service can be actually executed. Real world preconditions are conditions on the current, global state, and they must be fulfilled at the initial state of an execution path in order to be a valid execution path. Information preconditions are conditions on the knowledge of the service consumer, and they must be satisfied by the consumer database when service execution starts.

Preconditions will, therefore, constraint the set of possible initial states of the pair of execution paths that define a service execution; if service preconditions are not satisfied at the current global and consumer states for the input binding given, the service cannot be executed.

The evaluation of information preconditions at the consumer database, i.e., relative to the consumer knowledge of the current state, reflects the fact that the service consumer must not only assign some objects to the service input variables through an input binding, but also provide certain information about these objects. This information must be obviously known by the service consumer, which is what is evaluated by the formalization of information preconditions.

Example 3.13 Let us imagine a service has input variables f, d, p, cc and its information preconditions are formalized as follows (in the following, free variables are assumed to be universally quantified outside the formula):

$$pre^{inf}(f, d, p, cc) \leftrightarrow \exists co, cd, name, id, h, n, ed. flight(f) \wedge hasOrigin(f, co) \wedge city(co) \wedge$$
$$inContinent(co, europe) \wedge hasDestination(f, cd) \wedge city(cd) \wedge inContinent(cd, europe) \wedge$$
$$date(d) \wedge person(p) \wedge hasName(p, name) \wedge hasId(p, id) \wedge dummyCard(cc)$$
$$\wedge hasHolder(cc, h) \wedge hasNumber(cc, n) \wedge hasExpirationDate(cc, ed)$$

meaning that the objects from universe $\mathcal{U}$ assigned to f, d, p, and cc must be a flight between European cities, a date, a person, and a *dummyCard* credit card, respectively, and that the consumer must know the origin and destination of f, the name and id of p, and the holder, number and expiration date of cc, as this knowledge will have to be provided to the service in some way.

Now, let us imagine the following real world preconditions are defined by the service:

$$pre^{rw}(f, d, p, cc) \leftrightarrow \exists s, c.freeSeat(s) \wedge onFlight(s, f) \wedge onDate(s, d) \wedge valid(cc) \\ \wedge hasCredit(cc, c) \wedge c > 0$$

meaning that there must be a free seat on the flight and on the date given, and that the credit card given must be valid and have credit.

The service will only be executable if the consumer defines a binding for input variables f, d, p, cc such that $pre^{inf}(f, d, p, cc)$ is satisfied by current consumer knowledge, and such that $pre^{rw}(f, d, cc)$ holds at the current global state.

□

After formalizing preconditions, we continue with the formal characterization of a service.

Definition 3.14 *(Formal characterization of a service) A service serv is defined by a transaction formula of the following form, where $i_1, \ldots, i_n$ denote the input variables defined by the service interface, $o_1, \ldots, o_m$ its output variables, and c denotes a service consumer:*

$$serv(i_1, \ldots, i_n) \leftarrow [pre^{inf}_{serv}(i_1, \ldots, i_n)]^c \wedge pre^{rw}_{serv}(i_i, \ldots, i_n) \otimes \\ \Upsilon(i_1, \ldots, i_n, o_1, \ldots, o_m) \otimes \\ \Upsilon^c(i_1, \ldots, i_n, o_1, \ldots, o_m)$$

where $\Upsilon(i_1, \ldots, i_n, o_1, \ldots, o_m)$ and $\Upsilon^c(i_1, \ldots, i_n, o_1, \ldots, o_m)$ are transaction formulas in a transaction base P which might depend on the values assigned to input variables $i_1, \ldots, i_n$. In particular, $\Upsilon(i_1, \ldots, i_n, o_1, \ldots, o_m)$ defines a transaction formula which will cause a transition in the global state and which assigns some objects to output variables, and $\Upsilon^c(i_1, \ldots, i_n, o_1, \ldots, o_m)$ defines a transaction formula which will cause a transition in the service consumer knowledge.

From the definition above, we can see that a service defines a transaction, in the global state, in the service consumer state, or in both, which always imposes preconditions to be fulfilled at the initial state. Either $\Upsilon(i_1, \ldots, i_n, o_1, \ldots, o_m)$ or $\Upsilon^c(i_1, \ldots, i_n, o_1, \ldots, o_m)$ can consist only of queries or be empty, not causing any state transition, but not both. In the former case, the execution

of the service will not cause any update on the global state. In the latter case, no change in the service consumer knowledge will take place as a consequence of executing the service.

Example 3.14 Let us consider the service from the previous example, and let us suppose it defines two output variables s and n. The service could be defined as follows:

$$serv(f,d,p,cc) \leftarrow [pre^{inf}(f,d,p,cc)]^c \wedge pre^{rw}(f,d,p,cc) \otimes \Upsilon(f,d,p,cc,s,n) \otimes \Upsilon^c(f,d,p,cc,s,n)$$

where

$$\Upsilon(f,d,p,cc,s,n) \leftarrow booking(f,d,p,cc,s) \otimes hasNumber(s,n),$$
$$booking(f,d,p,cc,s) \leftarrow freeSeat(s) \wedge onflight(s,f) \wedge onDate(s,d) \wedge price(s,pr) \otimes$$
$$freeSeat.del(s) \otimes bookedSeat.ins(s) \otimes forPerson.ins(s,p) \otimes charge(cc,pr)$$

and

$$\Upsilon^c(f,d,p,cc,s,n) \leftarrow [bookedSeat.ins(s)]^c \otimes [onFlight.ins(s,f)]^c \otimes$$
$$[onDate.ins(s,d)]^c \otimes [forPerson.ins(s,p)]^c \otimes [hasNumber.ins(s,n)]^c$$

The transaction $\Upsilon(f,d,p,cc,s,n)$ books a free seat, for the passenger given, on the flight and on the date requested, and charges the credit card provided. The output variable s is assigned the particular seat booked when the transaction is executed, and the output variable n is assigned the number of such seat. The transaction $\Upsilon^c(f,d,p,cc,s,n)$ inserts into the service consumer database some information relative to input and output variables, in this case, relative to the seat booked.

Let us imagine an input binding which makes the service execute as follows:

$serv(flight123, tomorrow, me, myCC)$

First, the precondition $pre^{inf}(flight123, tomorrow, me, myCC)$ is evaluated on the service consumer database, and the precondition $pre^{rw}(flight123, tomorrow, me, myCC)$ is evaluated on the global database. If both preconditions are fulfilled, the transaction

$\Upsilon(flight123, tomorrow, me, myCC, s, n)$

will be executed, which will query the global database for some seat e.g. $mySeat$ on the flight and date given, will query for its price e.g. $mySeatPrice$, will delete from the global database the ground atom $freeSeat(mySeat)$, will insert into the global database the atoms $bookedSeat(mySeat)$ and $forPerson(s,p)$, and will issue the transaction $charge(myCC, mySeatPrice)$. This transaction has to be defined in the transaction base, but it is not shown here for simplicity. Furthermore,

the number of *mySeat* e.g. $mySeatNumber$ will be retrieved from the global database. Then, the transaction

$\Upsilon^c(flight123, tomorrow, me, myCC, mySeat, mySeatNumber)$

will be executed, which will insert into the consumer database the ground atoms $bookedSeat(mySeat)$, $onFlight(mySeat, flight123)$, $onDate(mySeat, tomorrow)$, $forPerson(mySeat, me)$, and $hasNumber(mySeat, mySeatNumber)$.

□

Example 3.15 Let us consider now a service which gives the current temperature of European cities. It can be defined as:

$$serv(ci) \leftarrow [pre^{inf}(ci)]^c \otimes \Upsilon(ci, t, dg) \otimes \Upsilon^c(ci, t, dg)$$

where

$$pre^{inf}(ci) \leftrightarrow city(ci) \wedge inContinent(ci, europe)$$

that is, a city in continent Europe is required from the service consumer,

$$\Upsilon(ci, t, dg) \leftarrow temperature(t) \wedge inCity(t, ci) \wedge forMoment(t, now) \wedge degreesCelsius(t, dg)$$

and

$$\Upsilon^c(ci, t, dg) \leftarrow [temperature.ins(t)]^c \otimes [inCity.ins(t, ci)]^c \otimes [forMoment.ins(t, now)]^c \otimes [degreesCelsius.ins(t, dg)]^c$$

As it can be seen, the service will not cause any state transition in the global database, i.e., if the current global state is s_0 then the execution path of the service will be $P, s_0 \models serv(city)$ for any city given. However, the consumer database will change from an initial state e.g. s_0^c to a state s_f^c where some ground atoms, relative to the current temperature of a city given to the service, have been inserted.

□

Given the definition of a service as a transaction formula, which is part of a transaction base P, this formula will have associated certain possible execution paths. In particular, the form of the transaction formula constraints what input bindings are valid input bindings for the service and what

initial (global and service consumer) states are valid for service execution. Given a valid input binding and initial states, the transaction formulas $\Upsilon(i_1, \ldots, i_n, o_1, \ldots, o_m)$ and $\Upsilon^c(i_1, \ldots, i_n, o_1, \ldots, o_m)$ will be executed, and this execution will have associated (possibly non-deterministically) a given pair of execution paths.

We can now formally define possible service executions.

Definition 3.15 *(Possible service execution) Given a transaction base P, a pair of execution paths ($\chi = s_0, \ldots, s_i, \ldots, s_f$, $\chi^c = s_0^c, \ldots, s_j^c, \ldots, s_g^c$) is a possible execution of a service serv, denoted $(\chi, \chi^c) \in \Xi_{serv}$, if and only if*

1. $P, s_0^c \models pre_{serv}^{inf}(\beta(i_1), \ldots, \beta(i_n))$,
2. $P, s_0, \ldots, s_f \models pre_{serv}^{rw}(\beta(i_1), \ldots, \beta(i_n)) \otimes \Upsilon(\beta(i_1), \ldots, \beta(i_n), \gamma(o_1), \ldots, \gamma(o_m))$, *and*
3. $P, s_0^c, \ldots, s_f^c \models \Upsilon^c(\beta(i_1), \ldots, \beta(i_n), \gamma(o_1), \ldots, \gamma(o_m))$

for some input binding $\beta : \{i_1, \ldots, i_n\} \rightarrow \mathcal{U}$ and some assignment of objects to output variables $\gamma : \{o_1, \ldots, o_m\} \rightarrow \mathcal{U}$.

This means that the initial state of the consumer satisfies information preconditions, the initial global state satisfies real world preconditions and the execution of transaction $\Upsilon(\beta(i_1), \ldots, \beta(i_n), \gamma(o_1), \ldots, \gamma(o_m))$ causes the state transitions specified by the execution path χ, and the execution of transaction $\Upsilon^c(\beta(i_1), \ldots, \beta(i_n), \gamma(o_1), \ldots, \gamma(o_m))$ causes the state transitions specified by the execution path χ^c.

The functionality of a service is given by its definition as a transaction formula; this transaction formula defines what pair of execution paths will take place given a particular input binding and a particular pair of initial states, thus establishing a (possibly non-deterministic) mapping of input bindings and initial states to final states. If the transaction formulas $\Upsilon(i_1, \ldots, i_n, o_1, \ldots, o_m)$ and $\Upsilon^c(i_1, \ldots, i_n, o_1, \ldots, o_m)$ are deterministic and the service has n input variables, this mapping can be expressed as a function:

$\mathcal{F} : \mathcal{S}x\mathcal{S}x\mathcal{U}^n \rightarrow \mathcal{S}x\mathcal{S}$

where $\mathcal{S}$ denotes the set of all possible states, i.e., possible sets of ground atoms in language $\mathcal{L}(\Sigma)$ and consistent with domain ontologies $\mathcal{O}$. Therefore, the function maps a pair of states (global and consumer initial states) and of n objects from the domain of discourse (the objects assigned to input variables of the service), to a pair of states (the final global state and the final consumer state).

Finally, we can define effects of a service execution, both real world and information effects.

Definition 3.16 *(Formal characterization of effects) Given an execution of a service by a consumer c, defined by the pair of execution paths ($\chi = s_0, \ldots, s_i, \ldots, s_f$, $\chi^c = s_0^c, \ldots, s_j^c, \ldots, s_g^c$),*

- *real world effects are given by the transition from state s_0 to state s_f, i.e., by the change of the global state caused by the service execution, and*
- *information effects are given by the transition from state s_0^c to state s_g^c, i.e., by the change of the service consumer knowledge caused by the service execution.*

Intuitively, real world effects correspond to the global state change caused by the service execution, and information effects correspond to some change in the knowledge of the consumer about such global state.

3.3.3 Capabilities

A capability is the ability to perform some action with some value; this value can be a state change or the provision of some information. We have seen that a service enables access to a certain capability, i.e., that the service is a means to access a capability and, furthermore, it enables such access in a particular way: requiring some information from the consumer and some conditions to hold in a shared state, and defining a particular mapping from the information provided to effects.

In this setting, we can define a capability as actions, i.e., as a transaction formula which will be incorporated in different ways to the definition of services enabling access to this capability.

Definition 3.17 *(Formal characterization of a capability) A capability is defined by a transaction formula of the form:*

$$\mathcal{C}(x_1, \ldots, x_n, y_1, \ldots, y_m) \leftarrow \Upsilon'(x_1, \ldots, x_n, y_1, \ldots, y_m) \otimes \Upsilon'^c(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

where $\Upsilon'(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a transaction formula which causes some change in the global state, and $\Upsilon'^c(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a transaction formula which causes some change in the knowledge of party c accessing the capability. $x_1, \ldots, x_n$ are parameters of the transaction formula, and $y_1, \ldots, y_m$ are output parameters of the formula, i.e., some objects are assigned to $y_1, \ldots, y_m$ when the transaction is executed.

Let us illustrate the definition above with an example.

Example 3.16 Let us consider the service from example 3.14. The capability this service enables access to would be defined as:

$$\mathcal{C}(x_1, x_2, x_3, x_4, y_1, y_2) \leftarrow \Upsilon'(x_1, x_2, x_3, x_4, y_1, y_2) \otimes \Upsilon'^c(x_1, x_2, x_3, x_4, y_1, y_2)$$

where

$$\Upsilon'(x_1, x_2, x_3, x_4, y_1, y_2) \leftarrow pre^{inf}(x_1, x_2, x_3, x_4) \otimes booking(x_1, x_2, x_3, x_4, y_1) \otimes hasNumber(y_1, y_2),$$
$$pre^{inf}(x_1, x_2, x_3, x_4) \leftrightarrow \exists co, cd.flight(x_1) \wedge hasOrigin(x_1, co) \wedge city(co) \wedge$$
$$inContinent(co, europe) \wedge hasDestination(x_1, cd) \wedge city(cd) \wedge$$
$$inContinent(cd, europe) \wedge date(x_2) \wedge person(x_3) \wedge dummyCard(x_4)$$

and

$$\Upsilon'^{c}(x_1, x_2, x_3, x_4, y_1, y_2) \leftarrow [bookedSeat.ins(y_1)]^c \otimes [onFlight.ins(y_1, x_1)]^c \otimes$$
$$[onDate.ins(y_1, x_2)]^c \otimes [forPerson.ins(y_1, x_3)]^c \otimes [hasNumber.ins(y_1, y_2)]^c$$

The capability is thus given by a transaction formula that consists of the action *booking*, for which some restrictions on the parameters to which the action can be applied have been defined, and by a transaction formula which describes the insertion of certain ground atoms in the database of the party accessing the capability, i.e., the provision of some information.

As it can be seen, the capability definition does not refer to input and output variables of a service interface, but to general parameters of the action associated to the capability whose possible values are constrained by predicate pre^{inf}.

A service like the one in Example 3.14 enables access to this capability by linking the parameters of the capability to input and output variables of the service, thereby defining how the input binding provided when executing the service will condition what effects are obtained. Furthermore, the service defines particular information requirements (input values) and conditions to hold in the global state in order to enable access to the capability. In particular, the service in example 3.14 can be defined in terms of the capability formalized above in the following way:

$$serv(f, d, p, cc) \leftarrow [pre^{inf}(f, d, p, cc)]^c \wedge pre^{rw}(f, d, p, cc) \otimes \mathcal{C}(f, d, p, cc, s)$$

□

Intuitively, a capability is given by a transaction formula which defines what actions are associated to the capability, and possibly for what parameters (if any) they can be applied. Particular services enabling access to this capability will impose real world and information preconditions for the execution of the service. Furthermore, services will determine what values are given to the parameters of the capability, either by linking these parameters to input and output variables or in any other way e.g. assigning some fixed value to them.

Therefore, a capability is defined by a transaction formula which can result, if executed, in the realization of a particular set of effects.

3.3.4 Goals

Goals define the needs consumers have, which they expect to get resolved by using some service. Therefore, given the current state, goals define what final state has to be reached and what information has to be known by the consumer, which leads to the following definition.

Definition 3.18 *(Formal characterization of a goal) A goal of a party c is defined as a pair* (s_f, s_g^c) *where:*

- s_f *is a global state which has to be reached, and*
- s_g^c *is a state of party c which has to be reached.*

We give an illustrative example in the following.

Example 3.17 Let us consider the goal in Example 3.8. Given a current global state s_0 and a current party state s_0^c, this goal can be defined by a pair of states (s_f, s_g^c) such that:

- $s_f = s_0 + \{bookedSeat(se), forPassenger(se, rubenLara)\}$
- $s_g^c = s_0^c$

for some seat se such that:

$$s_f \models seat(seat) \wedge onFlight(se, flight4321)$$

This means that the existence of a booked seat for the person desired and on the flight given must be entailed at state s_f reached after the execution of a service, and that no change in the knowledge of party c is required.

□

Example 3.18 Let us now consider the same party as in the previous example, from the same current states, wants to know the current temperature in Madrid. In this case, his goal is defined by a pair of states (s'_f, s'^c_g) such that:

- $s'_f = s_0$
- $s'^c_g = s_0^c + \{temperature(t), inCity(t, madrid), forMoment(t, now)\}$

for some temperature t such that:

$$s_0 \models temperature(t) \wedge inCity(t, madrid) \wedge forMoment(t, now)$$

□

3.4 Frameworks for the description of services

Different frameworks exist for the modelling of services, namely: WSMO [de Bruijn et al., 2005a], OWL-S [Martin et al., 2004], SWSO [Battle et al., 2005d], and WSDL-S [Akkiraju et al., 2005]. All these frameworks have been submitted to the W3C for their consideration as a standard for the semantic (formal) description of services. Furthermore, the Semantic Annotations for Web Services Description Language Working Group of the W3C [2] have worked on building the SAWSDL [Farrell and (editors), 2007] specification, which defines mechanisms for adding semantic annotations to WSDL 2.0 descriptions.

From these frameworks and specifications, we will focus in the next section on WSMO, of which the PhD candidate is a co-author. Afterwards, we will more briefly summarize other frameworks and how they differ from WSMO.

3.4.1 The WSMO Framework

The Web Service Modeling Ontology (WSMO) aims at describing all relevant aspects related to general services which are accessible through a Web service interface with the ultimate goal of enabling the (total or partial) automation of the tasks (e.g., discovery, selection, composition, mediation, execution, monitoring, etc.) involved in both intra- and inter-enterprise integration of services. It must be noted, though, that WSMO can be used for the description of general services in the sense introduced in Section 3.2, as its underlying model is not in general influenced by the particular technology used, i.e., WSMO, despite its name, does not have a strong bias towards Web service technologies, but it can be used to describe general services.

WSMO has its conceptual basis in the Web Service Modeling Framework (WSMF) [Fensel and Bussler, 2002], refining and extending this framework and developing a formal ontology and set of languages. In the following, we describe the core elements of the WSMO framework; WSML, which serves both as a general purpose ontology language and as the language for providing WSMO descriptions, has been introduced in Chapter 2, Section 2.2.5.3.

3.4.1.1 Core elements

The core, top-level elements of WSMO are: ontologies, (web) services, goals and mediators (see Figure 3.1), which we introduce next. The WSML [de Bruijn et al., 2005c] family of languages, introduced in the previous Chapter, is used for writing down WSMO descriptions of Web services, goals, ontologies, and to some extent mediators.

[2] http://www.w3.org/2002/ws/sawsdl/

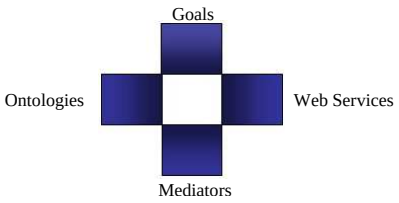

Figure 3.1: WSMO core elements

Ontologies provide the terminology used by other WSMO elements to describe the relevant aspects of the domain of discourse, and they constitute an agreed common terminology. This use of ontologies coincides with the use of the set $\mathcal{O}$ of domain ontologies our descriptions refer to in the formal characterization of our conceptual model (see Section 3.3.1).

Ontologies in WSMO include non-functional properties, used to describe non-functional aspects such as creator, creation date, natural language descriptions, etc. The elements defined by the Dublin Core Metadata Initiative [Weibel et al., 1998] are used as a starting point, and other properties such as the version of the ontology are added. Furthermore, ontologies imported for the definition of a particular ontology following a modular approach [Roman et al., 2005] are declared, as well as mediators necessary for the alignment of imported ontologies (mediators will be later explained in more detail).

Ontologies, as described in Chapter 2, Section 2.2.1, define concepts, relations, functions, instances and axioms. For details on how these elements are modelled in WSMO see [Roman et al., 2005]. An excerpt of an example ontology definition is shown in Listing 3.1.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-rule"

namespace { _"http://www.example.org/ontologies/example#",
      dc      _"http://purl.org/dc/elements/1.1#",
      foaf    _"http://xmlns.com/foaf/0.1/",
      wsml    _"http://www.wsmo.org/wsml/wsml-syntax#",
      loc     _"http://www.wsmo.org/ontologies/location#",
      oo      _"http://example.org/ooMediator#" }

ontology _"http://www.example.org/ontologies/example"
      nfp
            dc#description hasValue "fragments of a family ontology to provide WSML examples"
            dc#language hasValue "en-US"
      endnfp

      usesMediator _"http://example.org/ooMediator"

      importsOntology { _"http://www.wsmo.org/ontologies/location",
            _"http://xmlns.com/foaf/0.1" }
```

```
concept Human
        nonFunctionalProperties
                dc#description hasValue "concept of a human being"
        endNonFunctionalProperties
        hasName ofType foaf#name
        hasParent inverseOf(hasChild) impliesType Human
        hasChild impliesType Human
        hasAncestor transitive impliesType Human

relation ageOfHuman (ofType Human, ofType _integer)
        nfp
                dc#relation hasValue {FunctionalDependencyAge}
        endnfp

concept Man subConceptOf Human
        nfp
                dc#relation hasValue ManDisjointWoman
        endnfp

concept Woman subConceptOf Human
        nfp
                dc#relation hasValue ManDisjointWoman
        endnfp

axiom ManDisjointWoman
        definedBy
                !- ?x memberOf Man and ?x memberOf Woman.

concept Parent subConceptOf Human
        nfp
                dc#description hasValue "Human with at least one child"
        endnfp
        hasChild impliesType (1 *) Human

concept Girl subConceptOf Woman
        nfp
                dc#relation hasValue CompletenessOfChildren
        endnfp

concept Boy
        nfp
                dc#relation hasValue {ABoy,CompletenessOfChildren}
        endnfp

/*
 * Boy is a Man and a Child, and every Man which is also a Child is a Boy
 */
axiom ABoy
        definedBy
                ?x memberOf Boy equivalent ?x memberOf Man and ?x memberOf Child.

/*
 * Every child has to be either a boy or a girl (or both).
```

```
 */
axiom CompletenessOfChildren
        definedBy
                !– ?x memberOf Child and naf (?x memberOf Girl or ?x memberOf Boy) .

instance Mary memberOf {Parent, Woman}
        nfp
                dc#description hasValue "Mary is parent of the twins Paul and Susan"
        endnfp
        hasName hasValue "Maria Smith"
        hasChild hasValue { Paul, Susan }

instance Paul memberOf { Parent, Man }
        hasName hasValue "Paul Smith"
        hasChild hasValue George
```

Listing 3.1: Example ontology [de Bruijn et al., 2005e]

(Web) services are defined in WSMO as computational entities which are able (by invocation) to achieve users' goals. Services are understood in WSMO as the actual value provided by the invocation of a Web service. Thereby, a Web service might provide different services, such as for example Amazon can be used for acquiring books as well as to find out an ISBN number of a book.

```
webService _"http://example.org/Germany/BirthRegistration"
        nfp
                dc#title hasValue "Birth registration service for Germany"
                dc#type hasValue _"http://www.wsmo.org/TR/d2/v1.2/#services"
                wsml#version hasValue "Revision : 1.9"
        endnfp

        usesMediator { _"http://example.org/ooMediator" }

        importsOntology { _"http://www.example.org/ontologies/example",
                _"http://www.wsmo.org/ontologies/location" }

        capability
                sharedVariables ?child
                precondition
                        nonFunctionalProperties
                                dc#description hasValue "The input has to be boy or a girl
                                        with birthdate in the past and be born in Germany."
                        endNonFunctionalProperties
                        definedBy
                                ?child memberOf Child
                                        and ?child [hasBirthdate hasValue ?brithdate]
                                        and wsml#dateLessThan(?birthdate,wsml#currentDate())
                                        and ?child [hasBirthplace hasValue ?location]
                                        and ?location [locatedIn hasValue oo#de]
                                        or ( ?child [hasParent hasValue ?parent]
                                          and?parent[ hasCitizenship hasValue oo#de] ) .
```

```
    assumption
        nonFunctionalProperties
            dc#description hasValue "The child is not dead"
        endNonFunctionalProperties
        definedBy
            ?child memberOf Child
                and naf ?child [hasObit hasValue ?x].

    effect
        nonFunctionalProperties
            dc#description hasValue "After the registration the child
                is a German citizen"
        endNonFunctionalProperties
        definedBy
            ?child memberOf Child
                and ?child [ hasCitizenship hasValue oo#de].

interface
    choreography _"http://example.org/tobedone"
    orchestration _"http://example.org/tobedone"
```

Listing 3.2: Example WSMO Web service [de Bruijn et al., 2005e]

An example service description can be seen in Listing 3.2. The main elements of a WSMO Web service are:

- Non-functional properties, which are aspects of the Web service that are not directly related to its functionality e.g. network-related Quality of Service (QoS), the owner of the web service, etc. (see [Toma and Foxvog, 2006] for details).
- Imported ontologies and mediators. This element is a declaration of what particular ontologies are imported, providing the domain terminology used in the description of the Web service. When mediation is needed, the mediators used for resolving the alignment of ontologies are also declared.
- Capability, which in WSMO describes the functionality offered by a given Web service; it is expressed by the state of the world before the Web service is executed and the state of the world after successful Web service provision. The description of a capability is given in WSMO in terms of the following elements:
  - Non-functional properties, imported ontologies and mediators specific for the capability.
  - Preconditions specify the required state of the information space before the Web service execution; i.e., they specify what information a Web service requires, in order to provide its value. Preconditions constrain the set of states of the information space such that each

state satisfying these constraints can serve as a valid starting state (in the information space) for executing the Web service in a defined manner.

  - Assumptions describe the state of the world which is assumed before the execution of the Web service. Otherwise, the successful provision of the Web service is not guaranteed. Unlike preconditions, assumptions are not necessarily checked by the Web service. This distinction is made in order to allow an explicit notion of conditions which exist in the real world, but which exist outside the information space.
  - Postconditions describe the state of the information space that is guaranteed to be reached after the successful execution of the Web service; it also describes the relation between the information that is provided to the Web service and the information results of its execution.
  - Effects describe the state of the world that is guaranteed to be reached after the successful execution of the Web service i.e., if the preconditions and the assumptions of the Web service are satisfied.
  - Shared variables represent the variables that are shared between the logical expressions describing preconditions, postconditons, assumptions and effects.

- Interfaces. An interface describes how the functionality of the Web service can be achieved (i.e., how the capability of a Web service can be fulfilled) by providing a twofold view of the operational competence of the Web service: a) the choreography, which decomposes a capability in terms of interaction with the Web service, and b) the orchestration, which decomposes a capability in terms of functionality required from other Web services. This distinction reflects the difference between communication and cooperation. The choreography defines how to communicate with the Web service in order to consume its functionality. The orchestration defines how the overall functionality is achieved by the cooperation of more elementary Web service providers. The Web service interface is meant primarily for behavioral description purposes of Web services and is presented in a way that is suitable for software agents to determine the behavior of the Web service

It can be seen that the conceptual elements used by WSMO do not always coincide with the conceptual model presented in this Chapter. However, we can establish the following relations:

- A WSMO web service corresponds to a service in our model, which is in agreement with the SOA reference model defined by OASIS [MacKenzie et al., 2006]. What the WSMO models calls a service actually corresponds to the concept of realized effects in our model, i.e., to the different effects that can be realized by service execution.

- The definition of a general WSMO capability actually corresponds to the service functionality in our model; a capability, as understood in our model, can be seen as equivalent to what WSMO calls a capability but without references to the state of the world and the information state before a particular service is executed. Therefore, WSMO does not explicitly distinguishes between the functionality of a service and the underlying service capability (see Section 3.2).
- The elements that are introduced by WSMO for defining a capability can be mapped to concepts in our model. In particular: a) WSMO preconditions correspond to information preconditions, b) WSMO assumptions correspond to real world preconditions, c) WSMO postconditions correspond to information effects, and d) WSMO effects correspond to real-world effects.

Goals are representations of objectives for which fulfillment is sought through the execution of Web services; they can be descriptions of Web services that would potentially satisfy user desires.

```
goal _"http://example.org/Germany/RegisterGeorge"
    nfp
        dc#title hasValue "Goal of getting a Registration for Paul's son George"
        dc#type hasValue _"http://www.wsmo.org/TR/d2/v1.2/#goals"
        wsml#version hasValue "$Revision$ : 1.9"
    endnfp

    usesMediator { _"http://example.org/ooMediator" }

    importsOntology { _"http://www.example.org/ontologies/example",
        _"http://www.wsmo.org/ontologies/location" }

    capability
        effect havingRegistrationForGeorge
            nfp
                dc#description hasValue "This goal expresses Paul's desire
                    to register his son with the German birth registration board."
            endnfp
            definedBy
                George[ hasCitizenship hasValue oo#de] .
```

Listing 3.3: Example goal [de Bruijn et al., 2005e]

An example goal is given in Listing 3.3. The elements that constitute a goal are:

- Non-functional properties, similar to the ones used for Web services.
- Imported ontologies, which provide the domain vocabulary for describing the goal.
- A declaration of the mediators used. They can be *ooMediators* or *ggMediators*; we will explain below what exactly these mediators are.

- A capability, which is a description of the capability the sought service must offer.
- An interface, which represents the interface sought.

WSMO goals are essentially the same conceptual element introduced by our model.

Mediators. Mediation is concerned with handling heterogeneity, i.e., resolving possibly occurring mismatches between resources that ought to be interoperable [Roman et al., 2005]. WSMO mediators aim at resolving data-level mediation, protocol-level mediation, i.e, mediation between heterogeneous communication protocols, and process-level mediation, i.e., mediation between heterogeneous business processes.

The main elements of a mediator are the source and target resources mediated, non-functional properties of the mediator, and the mediation service, which defines the mediation facility applied for resolving mismatches. A mediation service is comprised of mediation definitions that resolve mismatches, and a facility of executing this mappings.

Four types of mediators are used in WSMO: i) *ooMediators*, which resolve mismatches between ontologies and provide mediated domain knowledge specifications to the target component, ii) *ggMediators*, which connect goals allowing the creation of a new goal from existing goals and thus defining goal ontologies, iii) *wgMediators*, which link a Web service to a goal, resolve terminological mismatches, and states the functional difference (if any) between both, and iv) *wwMediators*, which establish interoperability between Web services that are not interoperable a priori.

The concept of mediators is not introduced in our conceptual model, as we assume heterogeneity problems are transparently resolved. While in a real setting the definition and usage of mediators are necessary, we do not explicitly include them in our model as resolving data, protocol and process-level heterogeneity is beyond the scope of our work.

In general, the model of services and goals used by WSMO is quite in line with our conceptual model. However, there are some differences which have to be kept in mind when WSMO is used, as it will be the case, for describing services following the conceptual model elaborated in this Chapter. We can summarize as follows how elements in our conceptual model can be mapped to WSMO elements, keeping in mind the differences outlined above:

- A capability can be mapped to a WSMO capability without preconditions and assumptions.
- Information effects can be mapped to WSMO postconditions, and real-world effects to WSMO effects.
- A service can be mapped to a WSMO Web service.

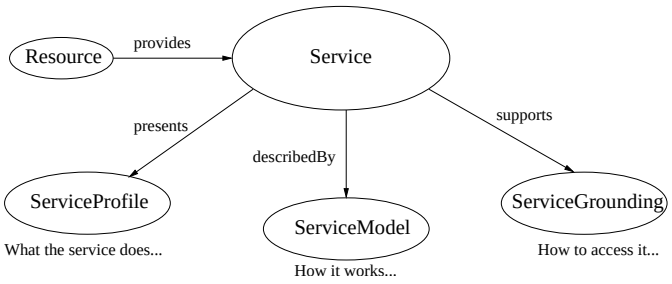

Figure 3.2: OWL-S upper ontology

- Information preconditions can be mapped to WSMO preconditions, and real-world preconditions to WSMO assumptions.
- A service execution can be mapped to a WSMO web service execution, and realized effects to the understanding of a (concrete) service in WSMO.
- A service functionality can be mapped to a general WSMO capability.
- A service interface can be mapped to a WSMO interface.
- A goal can be mapped to a WSMO goal whose capability does not have preconditions or assumptions.
- Consumer knowledge available cannot be directly mapped to any WSMO element. However, preconditions in WSMO goals can be seen as a declaration of consumer knowledge available for achieving this particular goal.
- The global state of the world is not captured by any WSMO elements. However, this was expected as such global state is external to services and goals.

3.4.2 Other frameworks for the semantic description of services

3.4.2.1 OWL-S

OWL-S [Martin et al., 2004] is an effort to define an ontology for the semantic markup of services, intended to enable automation of service discovery, invocation, composition, interoperation and execution monitoring by providing appropriate semantic descriptions of services. The upper ontology for services defined by OWL-S can be seen in Figure 3.2. In the following, we briefly outline the core elements of OWL-S.

Service. The concept of service serves as an organizational point of reference for declaring services; every service is declared by creating an instance of the service concept that links the profile, service model and grounding of a given service through the properties *presents*, *describedBy*, and *supports*, respectively.

Service profile. The profile describes what the service does at a high level, describing its functionality and other non-functional properties that are used for locating services based on their semantic description. A service profile is used both to describe the service offered by the provider and the service desired by the requester.

The profile of a service can be positioned in a hierarchy of profiles. However, this is optional, and a concrete profile can be directly defined as an instance of the profile class.

The OWL-S service profile includes human-readable information, contained in the properties *serviceName*, *textDescription* and *contactInformation*. A service categorization can also be given.

The value of a service is given by the service profile in terms of the *information transformation* performed by the service and the *state change* as a consequence of the service execution. The former is captured by defining the *inputs and outputs* of the service, and the latter is defined in terms of *preconditions and effects*. Inputs, outputs, preconditions and effects are normally referred to as IOPEs.

Service model. The model of a service describes how the service achieves its functionality, including the detailed description of its constituent processes (see [Martin et al., 2004] for further details).

Service grounding. The grounding describes how to use the service, i.e. how a client can actually invoke the service (see [Martin et al., 2004] for further details).

The OWL-S model roughly coincides with the conceptual model introduced in this Chapter with respect to services. In particular, the concept of service used by OWL-S basically coincides with our notion of service (see [Martin et al., 2004]), inputs with information preconditions, outputs with information effects, preconditions with real-world preconditions, and effects with real-world effects. However, the service profile can coincide with either the service capability (if no inputs and preconditions are defined) or the service functionality, if IOPEs are fully defined.

As for goals, OWL-S does not have a separate notion of goal, but a service profile can be used to describe services or consumer's goals. However, if a service profile is used to describe a goal, the outputs and effects of such profile coincide with the information and real-world effects of our model, respectively.

Regarding the language used by OWL-S descriptions, it must be noted that OWL-S is an ontology specified in OWL. Actual OWL-S service specifications are created by sub-classing and instantiating the classes of OWL-S. Thus, one can say that the OWL language together with the OWL-S vocabulary makes up the OWL-S Web Service specification language.

Furthermore, OWL-S allows the user a choice of different languages for the specification of preconditions and effects for the specification of preconditions and effects e.g. SWRL [Horrocks et al., 2004], KIF[3] or DRS [McDermott, 2004].

A detailed comparison of the differences and similarities between WSMO and OWL-S can be found in [Lara et al., 2004c] and [Lara et al., 2005].

3.4.2.2 SWSF

The Semantic Web Services Framework (SWSF) [Battle et al., 2005a] is a relatively recent attempt towards a Semantic Web Service annotation framework that greatly benefits from previous work with its roots in OWL-S and the Process Specification Language (PSL), standardised by ISO 18269. This framework is a joint proposal by the Semantic Web Services Language Committee and was submitted to the W3C [Battle et al., 2005d]. SWSF includes the Semantic Web Services Language (SWSL) [Battle et al., 2005b] and the Semantic Web Services Ontology (SWSO) [Battle et al., 2005d]. SWSO[4] can be seen as an extension or refinement of OWL-S. There are many similarities with the OWL-S ontologies, but the important difference is the expressiveness of the underlying language which is, instead of OWL, a richer language called the Semantic Web Service Language (SWSL).

Conceptually SWSF can be seen similar to OWL-S and thus we will not discuss it in more detail but refer the reader to the correspondent W3C submission [Battle et al., 2005a]. We will only note here that SWSO is the only approach which explicitly allows for giving different views of the value of the same service, which is an interesting feature but elaborated in little detail in the specification.

3.4.2.3 WSDL-S and SAWSDL

WSDL-S [Akkiraju et al., 2005] is a rather minimalist approach which aims at a direct extension of existing traditional Web Service descriptions in WSDL with Semantics (indicated by the last letter of the acronym). WSDL-S augments Web Service descriptions in WSDL with semantics by adding respective annotation tags to the XML schema of WSDL, the proposal picks aspects similar to those in WSMO capability definitions or OWL-S profiles, such as pre-condition and effects. This method keeps the semantic model outside WSDL, making the approach impartial to any ontology

[3] http://logic.stanford.edu/kif/kif.html

[4] The PhD candidate has been a reviewer of the SWSO submission to the W3C.

representation language. Hence WSDL-S does not fix a specific formalism for semantic descriptions and accordingly also does not claim to be a fully-fledged description framework/ontology, but rather simply adds some useful attributes to WSDLs XML tags in order to reference semantic annotations.

SAWSDL [Farrell and (editors), 2007] follows a very similar approach to (and has a considerable part of its roots in) WSDL-S, defining mechanisms for adding semantic annotations to WSDL 2.0 descriptions.

Both approaches do not aim at providing a model of services, but rather to semantically annotate certain aspects of a service, most of which are syntactically described by WSDL. Therefore, a mapping to our conceptual model cannot be established.

3.5 Summary

SOA is an increasingly used concept, and both the meaning of the term itself and of the concepts involved in this paradigm are sometimes unclear as different works and articles use them differently. In this Chapter, we have, based on the SOA reference model defined by OASIS, presented a conceptual model of the main elements and concepts which will be relevant for the location of services based on the value they provide. This model has the purpose of clarifying and assigning an unambiguous meaning to the concepts we will refer to in the following Chapters.

We have both introduced the concepts that compose our conceptual model, and formally characterized using Transaction Logic the nature of the artifacts relevant to our work. This formalization is not meant to be directly usable for automating the location of services, but to serve as a consistent basis for understanding what services, capabilities, goal, etc. are, which will help us to comprehend the problem we are facing and how we resolve it. We will in the following Chapters introduce simplifications over the formalization of the model introduced in this Chapter, so that we can explicitly describe the relevant concepts of the model necessary for (semi)automating the location of services in a way that is manageable and exploitable using existing reasoning infrastructure.

Finally, we have briefly introduced the major frameworks and efforts for adding formal semantics to services and related concepts, and we have outlined how the model underlying these frameworks relates to our conceptual model.

Chapter 4

A framework for service discovery based on F-Logic and Transaction Logic

4.1 Introduction

The nature of services and goals, presented in Chapter 3 and formally characterized using Transaction Logic, suggests the exploitation of Transaction Logic reasoning for the location of appropriate services for a goal at hand. In particular, a modelling and description of services amenable to their hypothetical execution (see Chapter 2, Section 2.2.4) can enable the hypothetical obtention of the effects of the execution of a given service and their evaluation with respect to the effects and the final state expected by the consumer as described by his goal.

In this Chapter, we summarize the work we carried out to automate the location of services based on Transaction Logic[1], the results obtained, and the limitations found. This work has been presented in [Kifer et al., 2004] and later adapted by Michael Kifer in [Kifer, 2005] to illustrate the possible use of the SWSL-Rules language [Battle et al., 2005b] for automating service discovery. It must be kept in mind that this work was a first attempt we made for enhancing service discovery; limitations were found which motivated the initiation of our efforts to find a more comprehensive model which can properly work in different situations and which grants users more flexibility. However, we think it is interesting to present this first work here, as it has some interesting features and it has served to motivate the work which will be presented in the following chapters.

[1] This work has been done in cooperation with Michael Kifer, Axel Polleres, Chang Zhao, Uwe Keller, Holger Lausen and Dieter Fensel.

The original work published in [Kifer et al., 2004] uses the model of services and goals given by WSMO, which is slightly different from our conceptual model (see the previous Chapter). Furthermore, the use of Transaction Logic was limited to the reasoning capabilities provided by $\mathcal{FLORA}$-2, which is not a real Transaction Logic reasoning engine and could only be used to simulate hypothetical executions. In this Chapter, we summarize the original work presented in [Kifer et al., 2004], which refers to WSMO concepts, and later relate it to the conceptual model (and its formal characterization) presented in Chapter 3.

While the work presented has been based on the conceptual model provided by WSMO of goals, services, mediators, and ontologies (see Chapter 3, Section 3.4.1), we have relied not on WSML but on F-Logic for describing these elements. The reason was that, by the time this work was accomplished, WSML was still under development and, furthermore, there existed reasoning infrastructure which enabled the (limited) simulation of Transaction Logic reasoning for F-Logic. F-Logic is a frame-based logic that provides higher-order features based on HiLog [Chen et al., 1993], and whose Horn version has been used.

$\mathcal{FLORA}$-2 has been used as an F-Logic reasoner which, furthermore, supports enough of Transaction Logic reasoning to realize the approach studied. An additional reason for using $\mathcal{FLORA}$-2 has been its support for the reification of complex formulas and rules, which allows us to treat descriptions of preconditions and of effects as values of some attribute of a service. Although reification (or self-reference) is capable of producing logical paradoxes, it is shown in [Yang and Kifer, 2003] that reification of queries does not cause paradoxes in $\mathcal{FLORA}$-2, and [Kifer et al., 2004] shows that the reification of rules is also free of paradoxes in $\mathcal{FLORA}$-2.

We start by introducing the formalization and proof obligations used in [Kifer et al., 2004], followed by a prototype realization of the framework proposed based on the $\mathcal{FLORA}$-2 reasoner (see Chapter 2, Section 2.2.4, and http://flora.sourceforge.net/). Finally, our conclusions, with special emphasis on discussing: a) how this work relates to the model presented in the previous Chapter, and b) the limitations of this approach which motivated the general model which will be presented in the next chapters, are presented.

4.2 Proof obligations and formalization

Logic has long been used for precise representation of statements about real-world objects or abstract artifacts. A suitable logic can be used to formalize WSMO goals, capabilities, mediators, as well as the proof obligations that must be established in order to determine whether a match exists between a user request and the value of available services, i.e., to determine whether a given service can be used to achieve a particular goal. However, there exist problems with the usability of logical descriptions and, therefore, with the scalability in terms of available human resources of

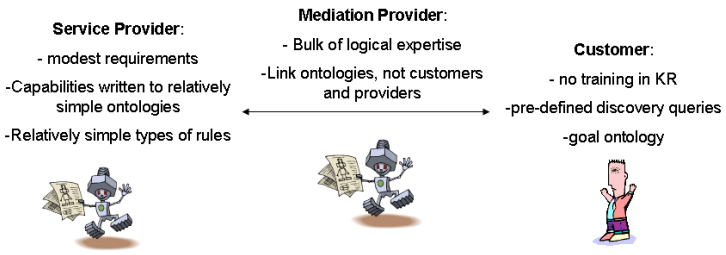

Figure 4.1: Categories of actors in contact with the logical mechanism for discovery

using this type of descriptions, as described next.

4.2.1 Formalization and scalability issues

Dealing with logical expressions is beyond the skills of most users and, therefore, in our work we envisioned three categories of people who would be in direct contact with the logical mechanisms used for automatically locating suitable services (see Figure 4.1):

1. *Consumers* who have no training in knowledge representation. These users will have access to pre-selected service discovery queries, which they can choose from a menu or construct using simple graphical tools. Such queries are the main components of consumer's goals.

   In general, this type of users is not expected to have a direct contact with logical descriptions, but they will instead make use of an ontology, specially suited for these users, which defines what goals can be expressed abstracting users from the underlying logical apparatus. This ontology is called the *goal ontology*, and we will see some examples of the type of goals it can define in Section 4.3.

2. *Service providers.* These users might not necessarily be more skilled logicians than the rest of the public, but they can hire skilled knowledge engineers. Still, the number of businesses who might want to share in the Semantic Web infrastructure can be potentially large, and it is unlikely that sufficient number of highly skilled engineers will be available to meet the demand. Therefore, in our work we tried to impose only modest requirements to the degree of sophistication of the engineers who might turn up in this type of labor market. The upshot of this is that the value of services should be written to relatively simple ontologies and use relatively simple types of rules.

   In general, service providers will make a limited direct use of logical descriptions, and also pre-defined ontologies will be used to ease the task of service providers of describing the value of their services. More details will be provided in Section 4.3.

3. *Mediation providers.* The bulk of logical expertise is expected to reside with companies whose business will be to provide ontology mediation. Mediators will bridge the gap between the ultimate simplicity of goal ontologies used by the clients of semantically described services and the relative simplicity of the service descriptions supplied by service providers. Since mediators link ontologies rather than customers and businesses, the number of skilled workers required to support such an infrastructure can be low enough to make the infrastructure scalable in terms of human resources.

In general, service providers and specially service consumers are not expected to have the skills to directly deal with complex logical descriptions. Thus, pre-defined and reusable concepts that capture common needs of consumers and common providers' offers will be used, defined by appropriate ontologies. Mediation providers will act as middle agents which will link common needs and common offers defined by such ontologies, as depicted in Figure 4.1. More detailed examples will be provided in Section 4.3.

4.2.2 Proof obligations

A proof obligation is a logical entailment that needs to be established in order for a service to be considered a *match* for a discovery goal, i.e., to be considered a candidate to fulfill the consumer's goal. A proof obligation is defined in terms of a set of imported ontologies O, a goal G, a service capability C, and a wgMediator wg, as defined by WSMO. Here, G and C are logical formulas for the goal and the service capability, respectively.

The effects and the precondition parts of the capability C are denoted as C_{eff} and C_{prec}. C_{eff} is a logical formula that states what the service guarantees to be true after the execution. C_{prec} is a formula that must be true before the service execution; typically it contains predicates on the input provided by the requester and predicates on the state of the world right before the execution. Notice that conditions over the information space, i.e., over the consumer knowledge and conditions over the state of the world has not been separated, for simplicity, in the descriptions of WSMO service capabilities. In this setting, C_{prec} can refer to both WSMO assumptions and preconditions, and C_{eff} can refer to both WSMO postconditions and effects.

A wgMediator wg performs two main functions:

- It takes a goal, G, and constructs a request, $In_{wg}(G)$, suitable for the services that are mediated by this particular mediator. This is needed because the goal ontology and the service ontologies might be very different.

- A mediator also needs to convert the goal into a postcondition expressed in the service ontology, which is to be tested in the after-state of the service against the effects of the service. This

expression is denoted as $Post_{wg}(G)$.

This means that the mediator will be responsible for the translation from the terminology used by the consumer into the terminology used by the provider. We will see particular examples of this type of translation in the following Section, in which a particular realization of mediators in $\mathcal{FLORA}$-2 is presented.

Translations performed by wgMediators can be quite complex, because goals can be expressed in a very high-level syntax in order to make them palatable to naive users and service capabilities can be rather simple in order to make it inexpensive to specify them by a knowledge engineer.

We consider two different notions of a match. In one, which we call *service discovery*, the user supplies a general goal G and wants to check if a service can execute in a way such that the requester goal will be achieved. This means that (after the appropriate translations) the goal is guaranteed to be true in the after-state of the service. This is formally stated as the following proof obligation:

$$O, In_{wg}(G), C_{eff} \models Post_{wg}(G) \tag{4.1}$$

The formula above means that, given the request expressed by the goal and translated into a terminology the service can understand, $In_{wg}(G)$, and given the value offered by the service, C_{eff}, the effects expected by the service consumer and translated into the service terminology, $Post_{wg}(G)$, hold.

Service contracting comes into play after a potentially suitable service has been discovered. In contracting, given an *actual* input to a *specific* service, we want to guarantee that this input does indeed lead to the results expected by the requester.

This goes beyond the proof obligation for discovery. First, at this stage concrete input may be required (e.g., a credit card number). Second, this input needs to be checked against the precondition specified in the service capability. Third, the specification of the effects of the service and the requester's goal might be more complex. Therefore, the following proof obligation has to be checked:

$$O, In_{wg}(G'), C_{eff'} \models C_{prec} \wedge Post_{wg}(G') \tag{4.2}$$

The difference between (4.2) and (4.1) is that more complex versions of the goal and effects might be used for contracting (denoted G' and $C_{eff'}$) and that the precondition is checked. The proof obligation (4.2) can also be used for more precise discovery, which takes precondition into

account. This may be appropriate in situations where the user is willing to provide complete input during the discovery process.

In general, the difference between discovery and contracting is that, at discovery time, we only check whether different services can in general provide the value required by the goal, while at contracting time we select a particular service and want to check whether the usage of this service with particular inputs will yield the desired results.

The discovery and contracting queries. In proof obligations (4.1) and (4.2) it is assumed that we are dealing with a *particular* service and just need to test if it matches the goal. In practice, we need to go over all the services and test which ones match. The problem with this is that neither $In_{wg}(G)$ nor C_{eff} are part of a global knowledge base, and C_{eff} is different for different services. Since the effects in (4.1) and (4.2) are different for different services being tested, we need to find a general discovery query that could yield all the matching services.

In this work, the answer has been provided by Transaction Logic and, in particular, by hypothetical assertions. This enables us to look at each service separately and hypothetically insert the effects into the knowledge base. The goal can then be tested in the new hypothetical state. If it is true, the service is declared a match. To be able to refer to different services in the same proof obligation, we change our notation to make service effects and goal postconditions relative to a service. Therefore, we will write $C_{eff}(Serv)$ and $Post_{wg}(G, Serv)$, where $Serv$ is a variable that represents a service. This idea is logically expressed as follows, where $\Diamond$ is the hypothetical operator in Transaction Logic (see Chapter 2, Section 2.2.4):

$$O \models \exists Serv \; \Diamond(insert\{In_{wg}(G), C_{eff}(Serv)\} \otimes Post_{wg}(G, Serv)) \tag{4.3}$$

The above query is looking for services such that $\Diamond(insert\{In_{wg}(G), C_{eff}(Serv)\} \otimes Post_{wg}(G, Serv))$ holds in the models of the imported ontologies O. The symbol $\otimes$ is a sequence operator, which says that first the effects and the input must be (hypothetically) asserted and then the goal must be tested. Since the assertion is hypothetical, it is rolled back after the test is done. Query (4.3) is the basis of the prototype realization of the framework, described in the next section.

A similar query can be constructed for (4.2), considering that the service we test is a particular service $serv$ and the contracting goal to be resolved is G':

$$O \models \Diamond(insert\{In_{wg}(G')\} \otimes C_{prec}(serv) \otimes insert\{C_{eff}(serv)\} \otimes Post_{wg}(G', serv)) \tag{4.4}$$

In the formula above, we first insert the request associated to G', then check whether the preconditions of service $serv$ are fulfilled, and only then assume the effects of the service hold and evaluate whether the consumer goal has been fulfilled.

4.3 Prototype realization

This Section shows fragments of a larger running example that illustrates the $\mathcal{FLORA}$-2 implementation of the proof obligations defined in the previous section. The complete example is given in Appendix A.

A geographic ontology. We start with a simple domain ontology that represents geographic regions and cities. In $\mathcal{FLORA}$-2, the symbols that begin with a lowercase letter are constants that represent objects, and capitalized symbols (and symbols beginning with a "_" are variables. In our taxonomy, `europe`, `germany`, `usa`, `america`, etc., denote classes of cities. Thus, `europe` is a class whose members are all the cities in Europe, `usa` is a class whose members are U.S. cities, and so on. The subclass relationship is denoted using "::", i.e., `austria` :: `europe` states that `austria` is a subclass of `europe` (which implies that all Austrian cities are also European cities). To specify that an object is a member of a class, we use the symbol ":". For instance, `paris` : `france` states that Paris is a city in France. A fragment of such a geographic taxonomy is shown below:

```
germany :: europe.    stonybrook : nystate.    frankfurt : germany.
austria :: europe.    innsbruck : tyrol.       paris : france.
france :: europe.     lienz : tyrol.           nancy : france.
tyrol :: austria.     vienna : austria.        usa :: america.
bonn : germany.       nystate :: usa
```

F-logic classes are also viewed as objects and therefore they can be members of other classes. For instance, `europe` is a region, and so is `america`. In the above statements these two symbols played the role of classes, but in the following statements they play the role of objects that are members of class `region`.

```
europe : region.    america : region.
```

USA, Austria, and Germany are also regions and so is Tyrol. Rather than listing all of them explicitly as members of class `region`, we use a rule to define all regions:

```
Region : region  :- AnotherRegion : region and Region :: AnotherRegion.
```

The rule above means that, if a given constant a is a subclass of another constant b and this constant b is a member of the `region` class, then a is also a member of the region class. For example, as we have that:

```
europe : region.  germany :: europe.
```

we can infer that:

```
germany : region.
```

4.3.1 Goals

We assume that goals have the form

goalId[requestId -> *someId*, query -> *someQuery*]

This means that goals are represented as objects with certain properties. In F-logic, a statement of the above form means that *goalId* is a symbol that represents the object Id of a goal (it can look, for example, like `g123`) and that goal-objects have attributes `requestId` and `query`. The attribute `requestId` represents the Id of the request in case it is desirable to have it separate from the Id of the goal (for instance, if goals are intended to be reused). The attribute `query` represents the query that corresponds to the goal. The symbol `->` means that these attributes are functional; the symbol `->>` (used in service descriptions below) means that the attribute is set-valued.

Our use case assumes four types of queries:

`searchTrip(from, to)`	`tripContract(servId, from, to, date, crCard)`
`searchCitipass(loc)`	`citipassContract(servId, city, date, crCard)`

The queries above are part of a goal ontology, i.e., they correspond to pre-defined and reusable objectives consumers might have, and service consumers will describe their goals in terms of such pre-defined queries, thus abstracting users from direct contact with complex logical expressions.

The two queries on the left-hand side are used to discover services that can sell tickets from one location to another and citipasses for various cities. The two queries on the right-hand side are used to make a contract with a specific service for purchase of a ticket or a citipass. This is why the Id of a concrete service is part of the query.

Some examples of goals, defined in terms of the pre-defined queries or requests shown above, are:

```
goal3[requestId->g123, query->searchTrip(france, austria)].
goal2[requestId->g321,
      query->tripContract(serv1, bonn, innsbruck, '1/1/2008', 12345)].
```

It can be seen that the query part of the goals above is defined by instantiation of the pre-defined queries in the example goal ontology given. Therefore, consumers will only have to instantiate pre-defined queries in order to define their goals.

The first goal is quite interesting, because none of the services expects regions as input. Thus, without mediation, `goal3` cannot be answered. Specifying mediators between this kind of queries and the input expected by services is quite nontrivial and cannot be expected of a common user. For this reason, this task will be accomplished by mediators.

4.3.2 Service descriptions

In accordance with the conceptual framework of WSMO, a service description includes a specification of the service capability and of the mediators used by the service. In our example, each service uses only one wgMediator to tell how to convert the goal ontology into the ontology used by the service. We also assume that there is a single goal ontology and two service ontologies (see Appendix A).

Descriptions of services `serv1` and `serv3` are shown below. Preconditions and effects are specified as reified formulas, which is indicated with ${...} in $\mathcal{FLORA}$-2. In addition, the effects of services are specified via rules, which tell how the particular input supplied at service invocation affects what will be true in the after-state of the service.

```
serv1[capability->
      // Request for a ticket from somewhere in Germany to somewhere
      // in Austria OR a request for a citipass for a city in Tyrol
      cap1[ precondition(Input)->${
                (Input = contract(_, From : germany, To : austria, Date, Card)
                  or Input = contract(_, City : tyrol, Date, _))
                and validDate(Date) and validCard(Card)  }
           effects(Input)->${
                (itinerary(Req)[from->From, to->To] : −
                     Input = search(Req, From : germany, To : austria))
                and
                (passinfo(Req)[city->City] : −Input = search(Req, City : tyrol))
                and
                (ticket(Req)[confirmation->Num, from->From, to->To, date->Date] : −
                       Input = contract(Req, From, To, Date, _CCard),
                       generateConfNumber(Num))
                and
                (pass(Req)[confirmation->Num, city->City, date->Date] : −
                     Input = contract(Req, City, Date, _CCard),
                     generateConfNumber(Num))  }
      ],
usedMediators->>med1  ].

serv3[capability->
      // request for a citipass for a French city
      cap3[ precondition(Input)->${
```

```
                Input = pay(_, City : france, Date, Card)
                and validDate(Date) and validCard(Card)  },
            effects(Input)->${
                (Req[location->City] : −Input = discover(Req, City : france))
                and
                (Req[confirmation->(Num, City, Date)] : −
                        Input = pay(Req, City, Date, _Card) and
                        generateConfNumber(Num))  }
        ],
usedMediators->>med2  ].
```

Notice the differences in the input the two services expect and in the form of their output, which is due to the fact that the two services use *different ontologies*. For instance, `serv1` expects `search(Req, City : tyrol)` as one of the possible inputs, while `serv3` wants `discover(Req, City : france)`. Likewise, `serv1` yields objects of the form `passinfo(Req)[city->City]` in response, while `serv3` yields objects of the form `Req[location->City]`. Due to the differences in the ontologies, `serv1` and `serv2` tell the world that different mediators must be used to talk to them. In the first case, this is mediator `med1` and in the second it is `med2`. Mediators are represented as objects that possess methods for performing the mediation tasks. The first mediator is shown in some detail next.

It must also be noted that both inputs (requests) required by services and their results are described in terms of particular predicates and concepts like `search(Req, From, To)` and `discover(Req, City)` for requests, and `ticket` or `passInfo` for results, i.e., the description of requests and offers is eased by making use of predicates and concepts in appropriate domain ontologies which are usable for consumers and service providers.

4.3.3 Mediators, discovery and contracting

Mediators. The job of a mediator in our scenario is to bridge between goals and services. More specifically, a wgMediator performs two functions:

1. It takes a goal and constructs the input to the service, which is appropriate for that goal; and
2. It takes the result produced by the service and converts it to the format specified by the goal ontology.

Part of the mediator `med1` is shown below.

```
med1[constructInput(Goal)->Input] : −
        Goal[requestId->ReqId, query->Query] and
```

```
            if Query = searchTrip(From, To)
            then ( generalizeArg(From, From1), generalizeArg(To, To1),
                   Input = search(ReqId, From1, To1) )
            else if Query = searchCitipass(City)
            then ( generalizeArg(City, City1), Input = search(ReqId, City1) )
            else if ... ... ...
            else fail.
med1[reportResult(Goal, Serv, Result)] : −
            Goal[query->searchTrip(From : region, To : region)] and
            not med1[doesNotServeCity(From, To)]
            and Result = ${Goal[result->>Serv]}.
```

The above rules define methods to perform the two main tasks mentioned above: constructing the input and converting the service results into the format suitable for the goal ontology. The definition of the method `constructInput` checks the form of the user goal and yields appropriate input for the service. The predicate `generalizeArg` (not shown here, but defined in the full example in Appendix A) replaces the arguments that are objects corresponding to geographical regions with universal variables, because the mediator "knows" that this corresponds to the query with the quantifier "for all cities in the region." The method `reportResult` is defined by several rules of which we show only the one that corresponds to region-level requests, i.e., requests for services that sell tickets from/to every city in a pair of regions. If the user query is a region-level request, the rule checks if the service serves every city in the specified regions and then constructs the result expected by the service ontology. This result is then inserted into the knowledge base by the discovery query — see next.

Discovery and contracting. The discovery query is shown below. It examines each available service one by one. For each service, it obtains the mediator specified by the service and uses that mediator to construct the input appropriate for the service. Next we can use the input to obtain the effects of the service. Then the effects are hypothetically assumed and the goal is tested in the resulting state. If the goal is true in that state, the result (which contains the identification for the service) is inserted into the knowledge base.

```
find_service(Goal) : −
        Serv[usedMediators->>Mediator[constructInput(Goal)->Input]],
        Serv.capability[effects(Input)->Effects],
        insertrule{Effects},        // hypothetically assume the effects
        if Mediator[reportResult(Goal, Serv, Result)] then insert{Result},
```

```
        deleterule{Effects}.        // Remove the hypothetical effects
```

The query for verifying a service contract is essentially similar except that it also tests the precondition:

```
contract_service(Goal) : −
        // get the service to invoke: contracting queries have 4 or 5 args
        (Goal.query = _(Serv, _, _, _, _)orGoal.query = _(Serv,, _, _)),
        Serv[usedMediators− >> Mediator[constructInput(Goal)->Input]],
        Serv.capability.precondition(Input) = Precond,
        Precond,
        Serv.capability[effects(Input)->Effects],
        insertrule{Effects},        // hypothetically assume the effects
        // Check if the goal is satisfied by the service and report result
        ifMediator[reportResult(Goal, Result)]theninsertResult,
        // Remove the hypothetically added facts and rules
        deleteruleEffects.
```

4.4 Conclusions

Summary and achievements. The work presented in this Chapter has been our first effort in the direction of enhancing current service discovery practices. In this work, we have explored the combined use of F-Logic and Transaction Logic to, with the help of the $\mathcal{FLORA}$-2 reasoner, automate not only discovery but also contracting of services, i.e., also the evaluation of particular services for particular inputs.

One of the distinguishing features of this work is its attention to the usability of the discovery and contracting model for users who might not have the sufficient proficiency in the use of logics to deal with the logical apparatus used to automate the location of services. In particular, the contact of service providers and specially of service consumers with logical expressions is limited thanks to the use of appropriate goal ontologies and appropriate domain ontologies for describing the effects of services.

The separation of discovery and contracting, i.e., the separation of a general evaluation of services which can potentially fulfill the goal and the evaluation of particular services with particular inputs has also been a novelty introduced by this work with respect to other existing works. The incorporation of mediation has also been a key feature of this work which cannot be found in other works, enabling the use of heterogeneous terminologies.

Furthermore, the description and consideration of the dependency between the inputs that can be provided by the service consumer and the effects that can be expected from service usage is novel, as most existing works, which rely on Description Logics reasoning, did not consider this relation. Some posterior works such as [Hull et al., 2006] have introduced this type of description, although in this particular case its realization was left open.

Finally, it must be stressed that the approach presented is realizable using existing reasoning infrastructure, in particular $\mathcal{FLORA}$-2. This reasoner can be used to hypothetically assume the effects of service usage and then evaluate whether the consumer goal has been fulfilled.

Relation to the conceptual model. In the work presented, the WSMO conceptual model has been used, which as discussed in the previous Chapter slightly differs from our conceptual model. Still, a mapping can be established between the concepts used by this work and the elements in our conceptual model:

1. The description of service capabilities include rules which describe the effects that can be obtained for particular discovery queries, and rules which correspond to contracting queries. From these, the former type, in which preconditions of any type are not considered, can be mapped to the concept of service capability introduced in Chapter 3. The latter type, in which the preconditions of the service (only input preconditions in the examples studied) are considered, as well as how service effects depend on how such preconditions are fulfilled, i.e., on the particular input values provided by the consumer, corresponds to the description of the service functionality in our conceptual model. Therefore, this work also considers the functionality of services and not only their capabilities.

2. Preconditions in this work correspond to real-world and information preconditions in our model, although real-world preconditions are not considered in the examples studied. Effects correspond to information and real-world effects in our model; however, their type is not explicitly distinguished in the work presented.

3. Consumer knowledge is not directly modelled, but it is assumed to be contained in the consumer goal, i.e, only knowledge explicitly given with the goal is considered as a source of possible input values to the service.

4. The queries included in goals roughly correspond to the concept of goal introduced in the previous Chapter, as they express the objectives sought by consumers. These objectives are described using goal ontologies which capture what "action" the service is expected to perform e.g. sell city passes. This means that the change in the current state expected, encoded in terms of a goal ontology, is described. The id of the request is not an essential part of the goal, but it is only required due to the particular implementation of the framework proposed.

5. The resolution of visibility is organized in two phases: in a first phase, relevant services in terms of their discovery capability are located, without considering their preconditions and particular input values; in a second phase, individual services are evaluated for particular input values, determining whether these values fulfill information preconditions of the service and how they condition the effects that will be obtained from the service.

If we consider the formalization of our conceptual model given in Chapter 3, Section 3.3 (the work presented was previous to this formalization) we can establish the following (rough) mapping:

1. Both the state data oracle and the state transition oracle are given by $\mathcal{FLORA}$-2 semantics. Only one global state is considered, reflected by the current $\mathcal{FLORA}$-2 knowledge base, without differentiating the consumer knowledge state from the global state. In fact, possible consumer knowledge is encoded in the description of goals, and no additional consumer knowledge is explicitly considered.

2. The rules that describe effects of a service for different consumer requests or queries can be seen as contained in the transaction formulas introduced in the previous Chapter, which capture what change in the state of the world and in consumer knowledge the service will produce for particular input values. However, these rules only describe what elements or truths will be added to the current state for particular inputs, but the real insertion of these effects into the $\mathcal{FLORA}$-2 knowledge base is performed by the discovery and contracting queries. Therefore, the combination of these queries and the rule describing service effects can be roughly seen as equivalent to the transaction formulas which describe the changes that will be operated to the current state, as described in the formal characterization of our conceptual model.

3. If we look at the contracting query proposed (4.4), and if we remove the hypothetical operator and the check of the goal at the end of the query, it can be seen that this query is very similar to the formalization of the service functionality given in the previous chapter: first, the preconditions of the service are checked against consumer knowledge, which in this case is limited to the knowledge contained in the goal definition, and if this check succeeds then the effects of the service for this input will be realized, which in the work presented in this Chapter means inserting certain elements in the reasoner knowledge base representing the current state.

4. The discovery query and rules that describe the effects obtained for discovery requests cannot be mapped to the formalization of our conceptual model, but they can be roughly seen as a simplified version of the service functionality in which preconditions are not checked and the dependency of effects on particular inputs values is not modelled in complete detail.

Limitations While the work presented has some interesting features, it also presents some limitations which are discussed next.

First, and while using a goal ontology greatly enhances usability, it also limits the type of goals that can be expressed and thus turns out to be a too rigid solution for general use cases. As the complete framework and proof obligations are articulated around the pre-defined goals given by these ontologies, the overall framework lacks flexibility. Therefore, and while the fundamental idea is of considerable interest, support to users must be provided while keeping flexibility so that skilled users (both consumers and providers) can refine their goals and make them more precise and complex than those given by goal ontologies.

Second, the usage of the Horn version of F-Logic and of $\mathcal{FLORA}$-2 implies using closed-world reasoning. While this type of reasoning is very appropriate when complete information can be assumed, it does not suffice in general scenarios with missing information. For example, a consumer might state that he wants to find a flight from Madrid to Paris for a maximum price of 150 euros. A service might provide flights with the required departure and arrival cities, but with different possible prices which will furthermore change over time. Therefore, the description of the expected effects of service usage will most likely not say anything about the price of offered flights. In this setting, if we assume service effects hold, which will not say anything about the price, and compare the consumer goal to the achieved effects using closed-world reasoning, the service will not be a match. However, in this case missing information in effects does not mean flights with an acceptable price cannot be obtained.

While this might not a problem if goals are restricted to pre-defined forms and mediators properly handle missing information, it will indeed become problematic if we want to grant more flexibility to users and, furthermore, it requires mediators to handle incomplete information in a different way the semantics of the formalism and reasoner used dictate. In fact, notice that the use of universal quantifications to say, for example, that we want a service which can sell city passes in a given region, is hard-coded in mediators, as the formalism used presents difficulties to deal with explicit universal quantification of variables in rule bodies which do not appear in rule heads, and Lloyd-Topor extensions are required [Lloyd, 1987], which are not fully supported by $\mathcal{FLORA}$-2.

In general, the restriction of this work to a single type of descriptions based on F-Logic and Transaction Logic, the restriction to closed-world reasoning, and the use of goal ontologies, does not grant enough flexibility for covering general use cases. Depending on the usage scenario, users might want to express more complex needs beyond a fixed goal ontology, and service descriptions might be incomplete and might require open-world reasoning. Therefore, a more general model, which can accommodate alternative types of descriptions (with different level of detail and possibly different semantics) as well as users with different profiles is needed.

Finally, the efficiency obtained when a big number of services is available might not be

sufficient, as $\mathcal{FLORA}$-2 does not support parallelism. This means that available services must be evaluated one by one at discovery time, as the hypothetical insertion of effects when evaluating one service might interfere with the evaluation of other services.

For all these reasons, we believe a comprehensive model for the location of services, which is not limited to a single way of describing services and of matching descriptions, and which can cover a wider range of usage scenarios and needs is required. In the next chapter, we present a general, abstract model for the location of services, for which a realization and a prototype implementation will be proposed in Chapters 6 and 7. This model is partially inspired by the limitations found in the work presented in this Chapter, as well as by the analysis of the discovery problem we have presented in [Keller et al., 2005] and [Lara and Olmedilla, 2005].

Chapter 5

Abstract model for the location of services

5.1 Introduction

In a service-oriented environment, parties can offer services whose execution results in some valuable effects (service providers), or they can have certain goals that might be resolved by using available services (service consumers). Parties can thus cooperate so that the goal of a a party is resolved by consuming a service offered by another party. However, for some party to act as a consumer of some available service in order to get some goal resolved, a suitable service must be first located, i.e., visibility among potentially cooperating parties (consumer and provider) must be resolved. The resolution of the service location (or discovery) problem is included in the Service-Oriented Computing research roadmap presented in [Papazoglou et al., 2006], especially the achievement of a higher level of automation so that an increased dynamics e.g. runtime system (re)configuration is possible, as discussed in Chapter 2, Section 2.1.3.

The value, i.e., the capability offered by a service or, more specifically, its functionality might not be the only criterion to decide the suitability of such service for solving a certain goal; other criteria such as the interaction model of the service, its technical details (e.g. protocols used for communication), or other general properties of the service provision or of the service provider such as reliability, cost, trust, etc. might also be considered for deciding upon the use of a particular service. In general, and as discussed by the OASIS reference model [MacKenzie et al., 2006], visibility not only requires parties to be able to see each other, which can be solved based on the description of the value offered by the service and of the goal at hand, but also on the possibility of communication between cooperating parties (interaction model, technical details) and on the willingness to interact,

possibly determined by factors such as cost of the service provision, QoS, etc.

In our work, we focus on the location of services based on to what extent the value they provide can solve a particular goal. While a number of works exist which try to address this problem (e.g. [Paolucci et al., 2002; Li and Horrocks, 2003; Benatallah et al., 2003; Grimm et al., 2004] or the work presented in Chapter 4), they almost exclusively concentrate on the matching of goals and services (requests and offers [Colucci et al., 2005]) assuming a single, particular way of describing these artifacts.

However, we believe the (automatic) location of services requires a comprehensive model which is not limited to the definition of how particular types of service and goal descriptions are matched, but also: a) identifies and covers different types of application scenarios for the location of services, offering a flexible solution in terms of the type of descriptions of services and goals expected and in terms of how efficient and precise the location process is, and b) addresses the complete location process, including how services and goals are described, how users are supported for accomplishing this task, how service descriptions are published, and how service and goal descriptions are matched. We believe such a model must be based on the analysis of its envisioned application scenarios, and on the consideration of requirements and practical aspects for its exploitation in a real-world setting. This includes, for example, setting realistic expectations on the accuracy of descriptions or considering information privacy issues during the location process.

Along these lines, we propose in this Chapter a flexible and general model for the location of services which is driven by practical requirements and considerations. In Section 5.2, we characterize the main groups of applications we envision for the partially or fully automated location of services and discuss their key characteristics. Our proposal for an abstract model for the semi-automatic location of services, of which an instantiation will be proposed in the following Chapters, will be described in Section 5.3, placing special emphasis on the principles and practical observations which have motivated the design of the model. The first part of the model proposed, which covers the description of services and the publication of these descriptions, will be discussed in Section 5.3.1; the second part of the model concentrates on the description of goals, and it will be presented in Section 5.3.2; the last part of the model, which covers the actual location and selection of services which can achieve an explicitly described goal, will be presented in Section 5.3.3. Finally, Section 5.4 summarizes the content of this Chapter and conducts a brief discussion on the model presented.

5.2 Applications

In this Section, we identify and illustrate with examples the main families of use cases which can benefit from an enhanced model for the location of services based on the value they provide. We will discuss what key features are common to applications within the same family, which will be

an important input for the design of our abstract model for the location of services.

5.2.1 Design-time location of services for their composition or integration into complex systems or processes

The paradigm of service orientation introduces changes on how systems are designed: the consumption of services in an standardized way is possible, so it is their composition and integration into more complex systems or business processes, as illustrated by the following examples:

Example 5.1 Let us imagine a pricing system for bank offices. This pricing system offers to bank office agents, given a customer, a product to be sold e.g. a mortgage, and the conditions of the operation e.g. what interest rate is applied to the mortgage: a) the expected profitability of selling this particular product under these conditions, b) possibilities for cross-selling, i.e., what other products can be proposed to the customer depending on his segment e.g. young person with average incomes, c) the global profitability of the operation including cross-selling, and d) the global profitability of the customer considering not only the current operation but also products he has already contracted. This information helps the employee of the bank office to properly adapt the operation offered to the customer so that a given profitability can be obtained.

The pricing system requires information about the customer (customer segment, products already contracted), the type, volume, and profitability of the products contracted by the average customer in a given segment, market interest rates, the product catalogue of the bank, or bank policies (solvency goal, Return on Equity (ROE), etc.). Furthermore, functionalities like profitability calculation or actual contracting of products (if the system allows for this possibility) are required.

Usually, the information and functionalities required by the pricing solution are already available at the bank (in data warehouses and existing systems) and, therefore, the solution must integrate these existing information and functionalities. If access to the bank information and systems is enabled by meaningful services, the pricing system can be designed and implemented in terms of goals which can be resolved by such services. This considerably eases the adaptation of the pricing system to different banks: parts of the pricing process are defined in terms of goals formulated by business experts, which have to be resolved by services of the bank adopting the pricing solution. Such services are located by IT professionals at design time, i.e., services that can resolve the goals defined are located and hard-wired into the pricing system so that a (static) instance of the system is obtained which uses the appropriate services of the bank and thus can be run at bank offices.

In this example, support for the location of appropriate services is desirable in order to ease the adaptation of the pricing system to different banks by IT professionals, especially when a big number of different services exists at the bank. The design of the system in terms of goals that can be solved by interoperable services offered by the bank adopting the system, together with the

existence of appropriate support for locating relevant services, can considerably reduce the effort required to adopt the system, thereby increasing the profitability of the product as well as reducing the cost of ownership.

□

Example 5.2 Let us imagine a telecommunications provider defines a new business process for informing their customers about special offers and promotions. This business process, defined by business experts at the corporation, might include activities such as notifying customers by different means (SMS, e-mail, etc.) or registering what offers and promotions each customer was informed of. These activities can be defined as general goals that will be resolved at design-time in order to obtain a business process that can be run to actually manage this marketing activity.

Goals will most likely be defined by business experts as part of the business process. However, the location and wiring of services into the process will be accomplished by IT professionals, who will make sure the selected services can be properly executed when an instance of the process has to perform the corresponding activity. In this way, the process can be defined in terms of business goals that can be used by IT professionals to locate appropriate services and incorporate them to the process so that it can be instantiated at run-time.

In this situation, support to IT professionals for locating services which can achieve the goals defined by business experts is desirable, especially if these services must be found among a big number of available services.

□

Example 5.3 [Hull et al., 2006] describes how, in the context of genome sequencing projects, many of the tools and databases for analyzing data are available as services, thereby allowing biomedical scientists to perform so-called *in silico* experiments. Large number of these experiments are carried out by choosing some of these services, composing them into a workflow, and running them.

In this application area, the structure of workflows is given by the biologist designing the experiment, who also knows the kind of services necessary at each step. Therefore, workflows can be defined in terms of goals given by the scientist designing an experiment, and these goals can be used to, at design-time, locate appropriate services and incorporate them into the workflow. Here, we assume this workflow can be defined once and executed multiple times, i.e., appropriate services are located once, incorporated into the workflow, and this workflow might be run any number of times.

As discussed in [Hull et al., 2006], there is a growing number of publicly available biomedical services (3000 as of February 2006). This motivates the need for appropriate support for the location of services that can solve a goal which is part of an experiment workflow.

□

The examples above illustrate a general type of application scenario we envision for the automatic location of services: the location at design time of services that can solve a given goal and their static incorporation into a system or process. This family of applications can be characterized by the following features:

1. There is human intervention in the location of services which can solve a given goal, i.e., a human user will participate in the location process. This means that no dynamism is required and/or desired in the resolution of goals, but only support for a more efficient location of services which will be statically incorporated to a process or system. This usually corresponds to cases where there are no strong reasons to change the service used to perform a certain activity on-the-fly, or where efficiency is a key factor and, thus, the time required to dynamically locate appropriate services cannot be afforded.

2. There might or might not be human intervention in the execution of the services incorporated to the system or process. This means that in some cases the particular activity of the process or system performed by the service located will be accomplished without human intervention, but in some other cases a human user will participate in the execution e.g. for choosing some particular input values for service execution.

3. We expect the location of services at design-time for their incorporation into a system or business process to be accomplished by IT professionals using the goals defined by domain experts. If goals are explicitly and properly defined by business experts, and if appropriate support for the location of services that can solve such goals is in place, IT professionals can concentrate on resolving the wiring of these services into the system or process, i.e., they will focus on resolving the technical details of the integration of services into the process or system. In this way, the communication of requirements from business experts to IT experts is improved, and both can concentrate on the tasks they are better prepared for: the definition of processes and systems from a business point of view, and the technical realization of these processes and systems, respectively. Of course, we do not exclude the possibility of business experts taking care of the location and/or wiring of services e.g. a biologist can design an experimental workflow, define different goals at different steps, find appropriate services, and finally execute the workflow. However, we believe that, especially in an industrial setting, this case will be less common than a separation of concerns in the definition and resolution of business goals.

4. The location of services is not a time-critical task, as it is done at design time and, thus, it does not directly affect the execution times of the process or system. Furthermore, the precision required from the results of the location process will not be especially high, as the participation

of a human user in this process implies that he can further filter the results obtained. Therefore, the majority of application scenarios in this family will require a moderate trade-off between efficiency of the location process and accuracy of results, i.e., moderately precise results will be acceptable if they are given in moderately low times. Still, we do not exclude the possibility of applications demanding a higher precision and tolerating higher response times, and viceversa.

In general, this family of applications demands support to IT professionals for the resolution of goals, and support to business experts in defining these goals. Therefore, the location process will have to ease the task of IT professionals of finding appropriate services, but no full automation is required, i.e., a human user will be involved in the location and selection of a particular service.

5.2.2 Location and execution of services by end human users

An improvement in how services are located will not only enable the resolution of goals for performing activities of a system or process. End human users, who usually have goals they want to get resolved, will naturally benefit from improved mechanisms for the location of services: they will formulate their objectives, and services which can be used to fulfill such objectives will be located. In the following, we give some illustrative examples of this type of scenarios for the application of a comprehensive service location model.

Example 5.4 Let us imagine an Austrian citizen moves to Madrid, Spain, to work as a researcher at one of the universities of the city. He will have to go through some bureaucratic work, such as registering at the city hall, applying for a city center parking permit, where he lives, registering with the Spanish Tax Agency, and changing the license plates of his car. Different tasks will require communicating with different public entities such as the city hall or different agencies of the regional or central government.

Currently, many of these processes cannot be done on-line. Furthermore, from the ones which can be done on-line, it is not always easy to figure out what particular agency offers what services. In this context, if e-Government services would be properly described and made available as interoperable services, new and old citizens could formulate the goals they want to achieve e.g. obtaining tax information and an appropriate service would be found. Found services could be then executed in order to carry out the desired task.

In general, if e-Government is further developed at different levels of public administration and interoperable services are properly defined, described and made public, the goals of citizens can be formulated and appropriate services for achieving these goals can be located and used.

□

Example 5.5 The field of e-tourism has considerably developed in the last years, and an increasingly number of users book their holiday or business trips on-line. Web sites such as Expedia[1] or Hotel Reservation Service[2] aggregate information from different flight, car rental or hotel providers and offer a centralized entry point to users for solving their needs. However, these sites only offer information from a fixed set of providers and, thus, users still have to use different sites and compare the options offered by each site in order to get more complete information about available options for solving their goals. Furthermore, users do not always know all sites which can offer this kind of information and, thus, they will usually have a partial view of available options.

If e-tourism providers would define, describe and publish services to obtain information about their offers and to request the provision of such offers, users could define their goals e.g. booking a seat on a flight from Madrid to Manchester on 30th June 2007, and services which can fulfill these goals could be located and executed.

As it can be seen, the publication of e-tourism offers as services would enable users to find, without any intermediary, the offer which best suits their needs from the set of available providers. However, this would require all e-Tourism providers to make their offers available as explicitly described services. An intermediate solution, in the presence of sites which aggregate information from different e-tourism providers such as Expedia, would be the publication of the functionalities of these sites as interoperable services and their explicit and proper description. In this way, users will formulate their goals and services from different e-tourism aggregators which can be used to achieve these goals will be located; e-tourism aggregators would act as e-tourism brokers, enabling access to a partial set of available e-Tourism offers. This type of brokers, such as Virtual Travel Agencies (VTAs) have been considered in [Lara et al., 2004b; Stollber et al., 2004; He et al., 2004] (see Figure 5.1). A similar type of broker, but limited to the location of offers part of a frequent flyer program, has been presented in [Lopez et al., 2005].

□

Similar examples to the ones above can be found in the literature e.g. purchasing furniture [Stollberg et al., 2004] or buying a bike [Haller, 2004]. All these examples, as well as the two examples above, refer to scenarios where an end user wants to locate services to achieve some goal, and can be regarded as a step beyond searching information on the Web: *services* offering some value are made public and can be located by end users for their consumption. We summarize the characteristics common to these application scenarios as follows:

1. Human users will be available during the process of locating appropriate services for the goal at hand, i.e., end users will be available for filtering services found, selecting the most appropriate

[1] www.expedia.com
[2] www.hrs.de

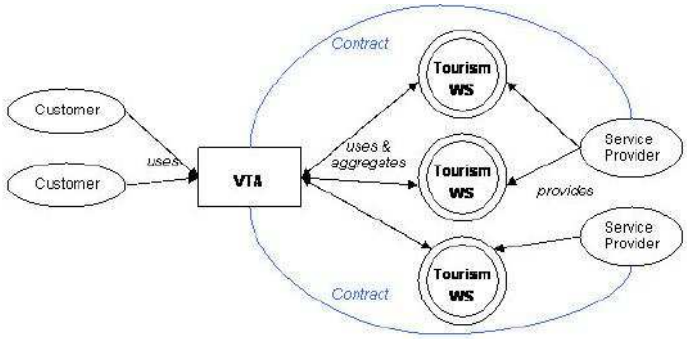

Figure 5.1: A Virtual Travel Agency which dynamically aggregates e-Tourism providers

one, and possibly revising their goals if necessary.

2. End users will participate in the execution of the services found. Therefore, support for such execution will be required e.g. some kind of automatic generation of user interfaces from service descriptions.

3. The definition of goals will be also accomplished by end users. Therefore, the end user will be participating in all the service location and execution life-cycle, starting from the definition of the goal to be achieved.

4. Once end user goal has been described, most scenarios will require the location of relevant services to be accomplished on-line, i.e., results must be returned to the user in relatively short times so that we can consider the location process an on-line process. That means that moderately low response times will be usually required, but we will hardly find any really time-critical use case. As the end user will be available for further filtering and selecting the most appropriate services from the set found, a moderate trade-off between efficiency of the location process and accuracy of results will be generally demanded.

In a nutshell, an end user will define some goal to be achieved and will be available during the process of locating appropriate services for the goal at hand, i.e., the end user will be available for filtering services found, selecting the most appropriate one, possibly revising his goal and, finally, executing the service.

5.2.3 Run-time location and usage of services

In the types of applications presented in the previous Sections, there is a human user involved in the service location process; appropriate services are located, an a human actor will then select the most appropriate service and either statically wire it into some system or process (Section

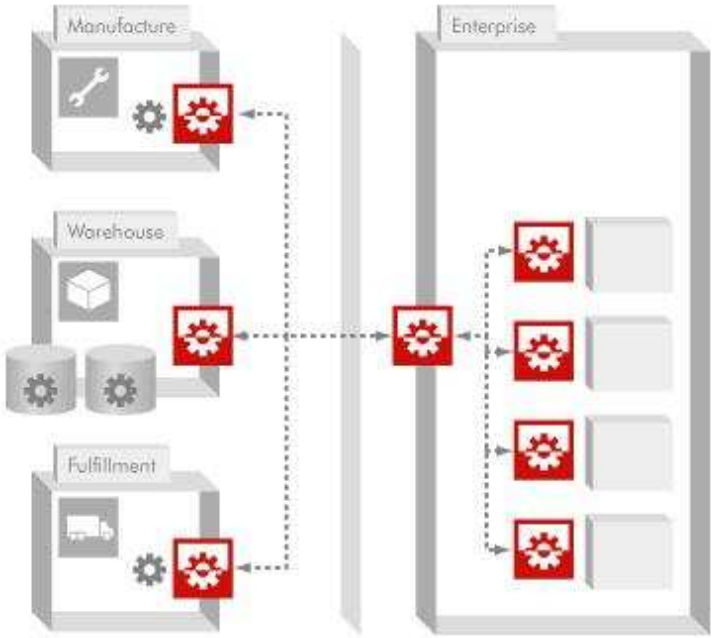

Figure 5.2: Generic supply chain

5.2.1) or directly execute it to get some goal resolved (Section 5.2.2). However, there are cases in which it is valuable to get some goal resolved without human intervention, as it will be illustrated by the following examples.

Example 5.6 Supply chains are common to a number of fields, and they can be defined as a coordinated system of organizations, people, activities, information and resources involved in moving a product or service in a physical or virtual manner from supplier to customer. Supply chain activities (aka value chains or life cycle processes) transform raw materials and components into a finished product that is delivered to the end customer[3] (see Figure 5.2[4]).

With current technologies, supply chains are completely determined when they are designed, i.e., the activities and providers involved are manually selected, probably as a result of prior agreements, and interactions among different actors hard-wired. In this context, the interaction with pre-selected providers to complete the supply chain is fixed, and the supply chain cannot be dynamically reconfigured to react to changes e.g. new providers entering the market or pre-selected providers going off-line [Lara et al., 2004b].

Dynamic, reconfigurable supply chains are a step beyond traditional supply chains. Instead of having a rigid configuration, they introduce dynamics by automatically locating the best providers

[3] http://en.wikipedia.org/wiki/Supply_chain
[4] Available at http://www.soa.com/index.php/section/solutions/supply_chain_management

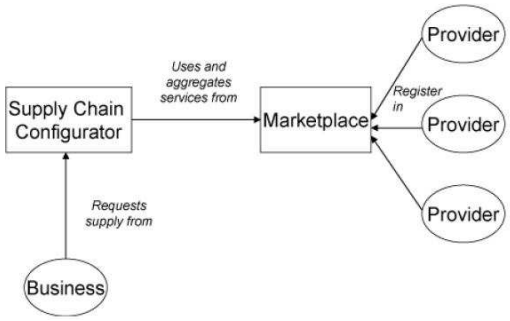

Figure 5.3: Supply chain application scenario [Lara et al., 2004b]

for a given need e.g. a certain raw material. This considerably decreases the effort required for configuring the supply chain and optimizes the supply process by always selecting an available provider which can solve some explicitly defined goals at different steps of the supply chain.

If a supply chain is defined in terms of explicit goals and appropriate facilities for the automatic location of services are in place, services which can fulfill these goals will be located and executed at run-time, without any human intervention. This yields a dynamic supply chain, where supply needs are dynamically resolved by selecting the most appropriate services of existing providers among available ones, thereby increasing the robustness of the supply chain; the supply chain is more hardly affected by some particular provider being unavailable as another provider can be dynamically found and incorporated to the supply chain.

In Figure 5.3, a given business has a supply request the supply chain configuration process will resolve by locating, aggregating and using appropriate services. In this case, the set of services considered is limited to those services offered by providers registered at the marketplace.

□

While the example above illustrates the run-time determination of the service to be used for performing a given activity, described by a goal, it must be noted that hybrid applications might also be found. For example, a service might be fixed in a business process, performing some activity of the process. However, an error-recovery strategy might be defined requiring the run-time location and execution of an alternative service: if the execution of the fixed service fails during the execution of the process, a service fulfilling a previously defined goal must be automatically located so that the process does not fail to execute, as illustrated by the following example.

Example 5.7 Let us come back to the pricing solution of Example 5.1. Configuring an operation so that a given profitability is obtained requires information about e.g. the product catalogue of the bank, the customer, or market conditions such as current interest rates. While information about the product catalogue of the bank or about the customer can usually only be provided by systems

controlled by the bank, information like market conditions is offered by a number of providers world-wide.

Now, let us imagine a bank adopting the pricing solution works with a given market information provider and, thus, the service offered by this provider will be used by the pricing solution for measuring the profitability expected from the operation. If the service or services of this provider go off-line for some reason, the pricing system will stop working and, thus, the possible simulations of operations being carried out for different customers at different bank offices will fail, which is not a desirable situation as attention to customers will be affected. In such a situation, the pricing system can try to dynamically find an alternative provider of market information required for performing the current simulation so that the system keeps working, i.e., the system can dynamically adapt so that a failure does not occur and bank offices can keep operating normally.

□

The examples above illustrate a type of application in which some activities in a system or process are accomplished by services located at run-time. In this way, the system or process can react to changes in the environment by dynamically, i.e., at run time selecting the service which can best fulfill the goal defined.

The main characteristics of this type of application scenarios can be summarized as follows:

1. No human user will participate in the location process, i.e., given a goal, the location of services which can achieve this goal will be completely automatic.

2. Human users might be involved in the execution of automatically located services. If this is not the case, we must be able not only to automatically locate an appropriate service, but also to automatically carry out the interaction with a dynamically found service.

3. We expect goals to be defined by domain experts, but there exists also the possibility of goals being automatically generated or parameterized at run-time. For example, the goal of shipping some goods to a given location might have to be achieved at some point during the execution of a business process, and this goal might be automatically generated by previous activities of the process or parameterized by these e.g. the location where the goods must be shipped to, as well as what particular goods must be shipped, might be dynamically bound as the result of previous activities, i.e., a generic goal can be parameterized at run-time.

4. The precision required from the results returned by the service location process will be usually higher than in other types of applications. The reason is that the selection of the service to interact with have to be accomplished automatically and, thus, the set of relevant services must be more accurately determined in order to avoid interactions with inappropriate services; failed interactions with services are expected to be generally more costly than a better selection of

	Phases of human intervention	*User profile (definition of goals)*	*Precision requirements*	*Efficiency requirements*
Design-time location	Location and possibly execution	Business experts	Moderate	Low
Location by end human users	Location and execution	End users	Moderate	Moderate
Run-time location	Possibly execution	Business experts or automatically generated	High	Moderate-High

Table 5.1: Main families of application scenarios and their main features

results. However, the particular trade-off between accuracy of results and response times will be chosen depending on the particular use case.

The run-time resolution of goals yields much more dynamic systems and processes, and dramatically increases their fault-tolerance, as they can reconfigure at run time if any problem is encountered. As discussed in [Trastour et al., 2002], in an e-commerce setting this means that relationships between businesses can be established dynamically and any possible lock-in in the relationships among trading partners is avoided.

The different types of application scenarios introduced above have different characteristics, thereby posing diverse requirements on the process of locating services for solving a given goal. We believe a comprehensive model for the location of services based on the value they provide must be designed in a way such that it can be applied to all these types of use cases. Therefore, the model must be flexible enough so that it can be adapted to the diversity of application scenarios envisioned.

The main dimensions used to characterize the families of application scenarios identified are the phases of intervention of human users in the location and execution of services, the kind of user defining the goal, and the precision and efficiency required (see Table 5.1). It must be noted, though, that while the families of application scenarios presented share some common features, there can be some degree of variance among particular application scenarios of the same type. A comprehensive model must, therefore, not only cover different types of application scenarios, but also the variety of scenarios within the same family.

In the next Section, and taking into account these factors, we propose an abstract model which is flexible in its conception and provides a basis for building concrete models for the location of services.

5.3 An abstract model for the location of services

The location of appropriate services for achieving a particular goal is not limited to the matching of a particular type of description of goals and services, but involves other tasks such as the description of these artifacts and their publication. In Figure 5.4, the usage of explicit (WSMO) descriptions of services for accomplishing the different tasks required to consume services achieving a certain goal is depicted. As it can be seen in the figure, the description of services and the advertisement of these descriptions is a key task for the service provider, so it is the description of goals by prospective service consumers.

When the descriptions of services and goals are provided and accessible (public), appropriate services can be located (discovered) for achieving the goal at hand. Once a number of services offering the desired functionality have been discovered, the actual service to interact with must be selected and Service Level Agreements (SLAs) must be established. This step often requires interaction with the service provider, being its outcome the selection of the actual service(s) to interoperate with, SLAs regarding the service being offered, and possibly a set of mediators required for seamless interoperation [de Bruijn et al., 2005d]. Finally, the service(s) selected can be used and the consumer and provider can interoperate to actually bring the capability the service offers to bear.

Besides the fact that the location of services involves a number of tasks or steps which together configure the service location process, we have seen in the previous Section that there exist usage scenarios with different characteristics in which support for the location of services which can fulfill a given goal is required. In this context, an abstract model which appropriately covers the different aspects of the service location process, and which is flexible enough to be applicable to different cases with different characteristics and requirements, is needed. In this Section, we present our proposal for such a model, covering the description of services and goals, the publication of service descriptions, the discovery of appropriate services for achieving a goal described, and the selection of the service(s) to interoperate with. The model has been designed having flexibility as a key principle, as it must adapt to different application settings.

In the following, we introduce the model and also discuss what practical considerations have motivated particular design decisions. An instantiation of the abstract model (excluding service selection), as well as a prototype implementation of this instantiation, will be presented in Chapters 6 and 7.

5.3.1 Description and publication of services

A service is a means to access a certain capability, where a capability is understood as the ability to perform some action with a perceived value, and this value can achieve a certain goal. Furthermore, different services can enable access to the same capability but presenting a different

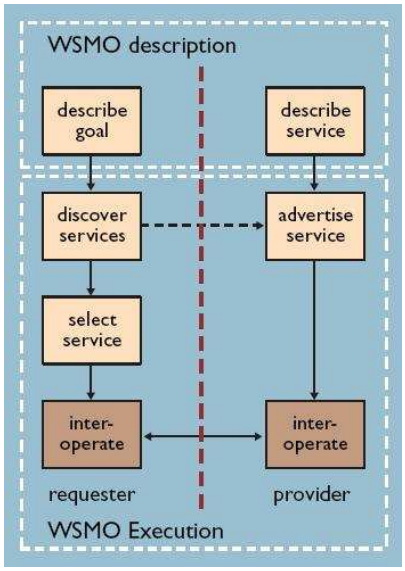

Figure 5.4: Service description and execution [de Bruijn et al., 2005d]

functionality, i.e., they can pose different conditions for service execution and provide a different mapping between valid initial conditions and the effects of the associated capability actually achieved (see Chapter 3).

For services to be located based on the capability and/or functionality they offer, these must be explicitly described. Otherwise, only by executing the service and observing the effects of its execution, we can have some knowledge of what value the service provides. However, a number of executions of different services, some of them perhaps failed, might be required for finding an appropriate service, which is expensive. Furthermore, there might be problems to roll back real world effects of executions which do not fulfill the consumer goal, and some of these real world effects might even not be directly visible to the service consumer.

Current practices in the description of services focus on the usage of WSDL, which describes the operations and input/output parameters of a service, and possibly give some textual documentation of what the service does (see Chapter 2, Section 2.1.2). While these types of descriptions usually suffice for the execution of services, they enable little degree of automation for their location. In general, current descriptions of services are mostly oriented to the execution of these services and to their manual inspection by technical users.

A number of frameworks have been proposed for enhancing how services are described, from which the most prominent ones have been presented in Chapter 3, Section 3.4. However, these frameworks do not completely determine how the value of services and goals has to be described; they only provide a general framework for describing services, but how to properly describe the value

of services for enhancing their location is left open.

Works such as [Li and Horrocks, 2003; Paolucci et al., 2002; Keller et al., 2004b; Verma et al., 2005; Grimm et al., 2004] have proposed different ways of describing the value of services, usually incorporating these descriptions into existing frameworks. However, they mostly concentrate on a single way of describing services and, furthermore, they do not discuss how these descriptions fit to different usage scenarios, how usable these descriptions are for different user profiles, and how users can be supported for providing these descriptions. The purpose of the first part of our model, covering the description of services and the publication of these descriptions, is to describe how the aspects not considered by most existing proposals can be solved.

5.3.1.1 Diversity of descriptions

The relevant aspects of a service which have to be taken into account for its location are its capability and, possibly, its functionality, as they define what value the service can provide and how this value is provided, respectively. Alternative types of descriptions of both a service capability and functionality might be provided. Furthermore, alternative views of these artifacts, with different accuracy, are possible, as illustrated by the following example.

Example 5.8 Let us imagine an international airline *IntAir* offers a service which enables the retrieval of information about flights operated by the airline. The service requires the main characteristics of the flight sought (origin, destination and date) and returns the departure time, flight number and seat availability on flights operated by *IntAir* matching these characteristics.

The capability of the service (the provision of information about flights operated by *IntAir*) can be described in alternative ways e.g. textually, by a first-order logic expression formalizing the set of flights about which the service can provide information, by explicitly listing the set of flights operated by the airline, or by categorizing the service using an e-Tourism services taxonomy. All these descriptions are alternative ways of describing the service capability, each being possibly of interest in different contexts. Furthermore, multiple descriptions of the same type are also possible, offering different views of the service capability with different degrees of detail. For example, one textual description of the service capability might say that information about flights operated by *IntAir* is provided by the service, while another textual description might list, in addition, all the itineraries operated by the airline. Both textual descriptions are possible views of the same service capability, given using the same type of description (a textual type).

Similarly, the functionality of the service can be described in different ways e.g. the transaction the service offers can be formalized using Transaction Logic, or a textual description of the service functionality can be given. Also different levels of detail are possible in these descriptions.

□

One may think that, from all the possible ways of describing a service we can find, we should choose the one which provides the highest accuracy and which has formal semantics. Or one may alternatively think that we should choose the type of description which is more usable by service providers while being accurate enough to unambiguously capturing the value of the service.

We believe the answer to what type of description of services is better is not unique, as it will strongly depend on the following factors:

1. User profile/Usability. Some service providers will be able to facilitate certain types of descriptions of their service capabilities and/or functionalities but not others, depending on their particular skills and resources. Furthermore, each possible type of description of services will usually require a similar type of description of goals to be provided by consumers so that the matching of services to goals is possible, which will pose certain requirements on consumers possibly with different profiles depending on the particular usage scenario.

2. Supported matching. The type of matching of services to goals which can be performed based on the description of these artifacts will obviously depend on the particular type of descriptions used. For example, textual descriptions can be very efficiently matched, but the accuracy of the results obtained is limited. On the contrary, matching of e.g. first-order descriptions might yield results with high accuracy, but the matching process can be time-consuming or even undecidable [Keller et al., 2004b]. In general, the type of description chosen will condition what matching mechanisms can be applied and, therefore, the trade-off between efficiency and accuracy of results.

Given these factors, which vary depending on the usage scenario considered, the type of description that can be provided by users (both providers and consumers) and the particular efficiency and accuracy expected from the location process cannot be completely anticipated. For this reason, in our model we enable the simultaneous usage of alternative types of descriptions of the capability and functionality of services, possibly with different accuracy, and we give the possibility of describing either the capability of the service, its functionality, or both (see Figure 5.5). In this way, we increase the coverage of different scenarios and we keep an appropriate level of flexibility.

In the abstract model, we do not impose the types of descriptions usable, and we leave open the possibility of incorporating any type of description. However, a particular instantiation of the model must define what types of descriptions are admitted, how they can be matched to goal descriptions, and what properties matching mechanisms over these descriptions present. One such instantiation will be presented in Chapter 6.

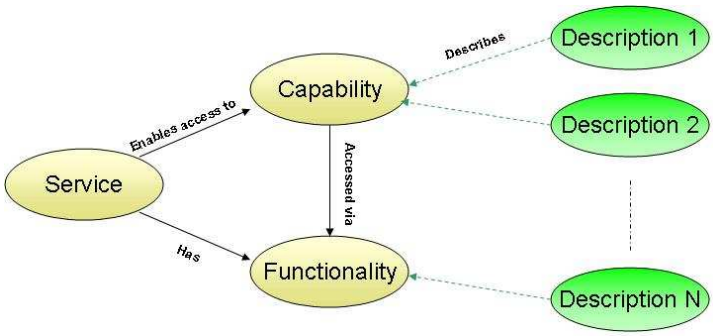

Figure 5.5: Diversity of descriptions of services

5.3.1.2 Nature of descriptions

The types of descriptions of a service we are interested in must reflect the value offered by the service, i.e., they must describe the capability and/or the functionality of the service. In principle, we could expect these descriptions to capture the value of a service with complete accuracy. However, some observations make this expectation rather unrealistic [Lara and Olmedilla, 2005; Keller et al., 2006a]:

1. Dynamics: while the capability a service enables access to might be static, there exist cases in which such capability has a dynamic component, i.e., the effects which can be realized by accessing the capability via a given service might vary over time. Similarly, what initial conditions are valid for using a service and how these initial conditions determine what capability effects will be realized after service usage, might also be dynamic. This is illustrated by the following example:

   Example 5.9 Let us imagine the airline from Example 5.8 also offers a service for booking flights it operates. The capability of the service can be textually described as: "booking of seats on flights operated by *IntAir*". Furthermore, the service requires as an input value a particular flight on which a seat must be booked, and a seat on that flight will be booked if available. However, what particular seat is booked will depend on seat availability, as well as the price charged for the booking, dependent on other factors such as the current level of occupation of the flight and the distance to the departure date.

   As we can see, what particular seat can be booked using this service, at what price, and even whether a seat on a particular flight can be booked at all, will change over time, i.e., both the service capability and functionality are dynamic.

   □

In situations like the one illustrated in the previous example, either direct communication with the service or the update of the service description every time some dynamic condition affecting the service capability and/or functionality changes would be required in order to have an accurate view of the service value.

2. Description effort: even if a service capability and/or functionality is static, the effort required to describe it with total accuracy can be too high. In fact, accurately describing the functionality of a service will imply making explicit the function implemented by the service, which generally requires translating the procedural function implemented into some declarative language. If we consider the example above, we must accurately and explicitly describe how the particular seat booked and the price of such seat depend on some conditions.

3. Sensitive information: accurately describing the functionality of a service might require making some strategic or sensitive information public. For example, accurately describing the functionality of the service in Example 5.9 would require making explicit and public the pricing strategy followed by the airline.

Given these observations, we can only realistically expect limited accuracy in the description of services. In fact, if we require descriptions to completely mimic the capability and functionality of the service, the descriptions used and the location process would actually make the service no longer necessary, at least in the case of information provision services: the description of the functionality itself could be used to determine the actual effects of using the service. Furthermore, we wonder whether, if formal descriptions are used, logical reasoning would scale under conditions where the execution of a service is replaced by reasoning over its semantic annotation [Keller et al., 2006a].

As a consequence, in our model we do not expect service descriptions to capture the value of services with complete accuracy, but we expect service descriptions to be *complete but not necessarily correct* [Preist, 2004; Keller et al., 2004a; Lara et al., 2004a; de Bruijn et al., 2005a; Keller et al., 2006a]. This means that all the effects achievable by using a service will be captured by a service description, but some effects might be described as achievable even though they might not be achievable by using the service (at a given point in time or permanently).

Example 5.10 Let us consider again the service in Example 5.9. We require any description of the service value, for example the description of its capability, to include all the possible effects of the usage of the service. This means that a description saying that seats on all flights operated by *IntAir* can be booked using this service is acceptable, even though some flights cannot be booked at certain points in time due to e.g. problems with seat availability or flight cancellations. This description is complete, as it includes all possible effects of service usage, but not correct, as it describes effects which might not be achievable.

On the contrary, a description declaring that the service can only book flights between Madrid and Manchester will not be acceptable as, if the airline operates flights between other destinations, it leaves out possible effects of the capability the service enables access to.

□

In general, our model requires descriptions of a service value to be complete but not necessarily correct. The maximum possible accuracy in the description while keeping it manageable and without disclosing sensitive information will be expected, but still correctness of descriptions will not be a requirement. In this setting, we expect service descriptions to be *a static, and as accurate as possible, characterization of the service value* [Lara and Olmedilla, 2005].

5.3.1.3 User support

Multiple descriptions of the value of a service, of different types, can be accepted by a particular instantiation of this abstract model, and the service provider is free to choose, from these types of descriptions, which ones will be provided for its service. However, it must be noted that a service can be matched only using the types of descriptions given, and that different types of descriptions enable different matching mechanisms with different properties, better suited for different usage scenarios. Therefore, it is desirable that a description of the service of each type admitted by the model instantiation is given so that the service can be matched in different ways and, therefore, in different scenarios.

In our model, we require two types of support for users describing services:

1. Support for providing alternative types of descriptions of the service.
2. Support for providing as accurate as possible descriptions of each type.

Different users, with different profiles and skills, might be able to provide more easily some types of descriptions but not others. The first type of support required by our model is oriented towards facilitating the provision of other types of descriptions of a service starting from the types of descriptions the user is comfortable with. For example, if the user is familiar with textual descriptions but has strong difficulties to provide formal descriptions, support must be provided for, from a textual description, obtaining a formal description of the service value.

This type of support must be based on the relation between the different types of descriptions considered by the particular instantiation of the abstract model. However, we acknowledge that, in some cases, it is a big challenge to establish a detailed relation among certain types of descriptions. For example, attempts to obtain a formal description from a textual description exist in the context of service discovery e.g. [Gomez et al., 2004], but this is a challenging task as the formalization of natural language, i.e., the definition of the relation between natural language and

formal languages is an open problem. In these cases, intermediate solutions such as establishing explicit links between the textual and formal description of pre-defined, reusable capabilities are possible, as it will be presented in Chapter 6.

Example 5.11 Let us consider the service in Example 5.9 and an instantiation of the service location model which admits textual descriptions and first-order descriptions of service capabilities. As the relation between textual and first-order descriptions is hard to establish, pre-defined, reusable capabilities can be introduced to link textual and fist-order descriptions. For example, a predefined capability $\mathcal{C}_{flightBooking}$, corresponding to the booking of seats on flights can be defined and described both textually ("booking of seats on flights") and formally.

When a user textually describes the service in Example 5.9 ("booking of seats on flights operated by *IntAir*"), this textual description can be used to match pre-defined capabilities. In this case, capability $\mathcal{C}_{flightBooking}$ will be matched using e.g. keyword matching and, thus, the formal description of $\mathcal{C}_{flightBooking}$ can be proposed to the user for formally describing his service. The user can directly associate the proposed description to his service or use this description as a starting point and refine it. This process is depicted in Figure 5.6.

□

In general, support must be provided to users for giving different types of descriptions of their services based on the relation between these alternative types. When such relation is difficult to establish, pre-defined capabilities can be used to relate different types of descriptions.

Support must also be provided to users for elaborating as accurate as possible descriptions of each type. This will generally be achieved through appropriate tool support e.g. user-friendly interfaces that ease the construction of service descriptions.

5.3.1.4 Publication of descriptions

Once services have been described, possibly in alternative ways, their descriptions must be made public so that prospective service consumers can have access to them. Currently, a widely accepted practice is to publish syntactic WSDL descriptions of services at UDDI repositories, which act as a common entry point for the location of services and provide keyword-based search facilities as well as search based on categories in taxonomies such as UNSPC through the UDDI inquiry API (see Chapter 2, Section 2.1.2).

In general, we require the publication of service descriptions at some registry accessible by the service location process. Such registry must enable the retrieval of services based on the different types of descriptions of services admitted by the instantiation of the abstract model, and using the matching mechanisms associated by the instantiation to such types of descriptions. In this sense, a

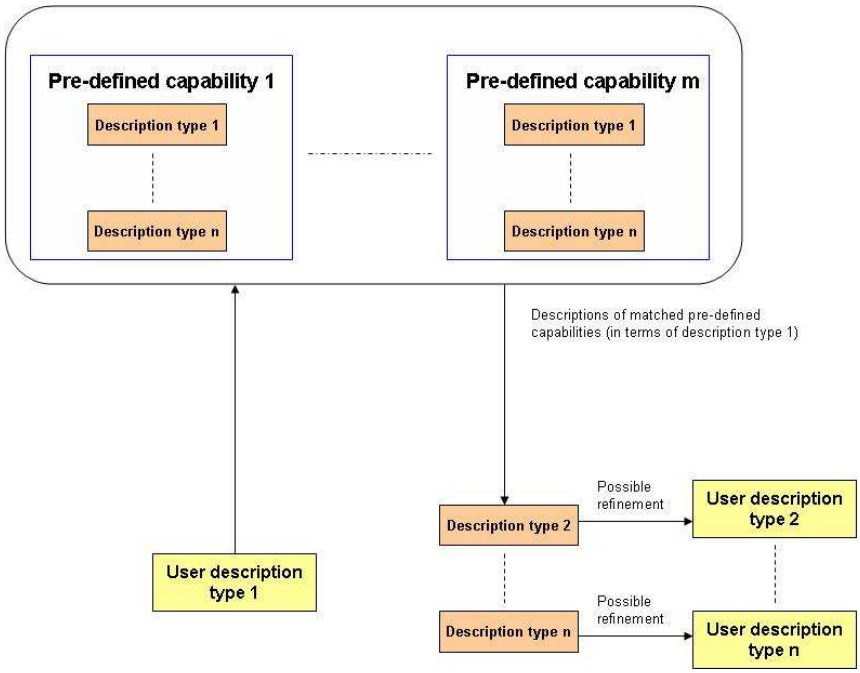

Figure 5.6: Usage of pre-defined capabilities

UDDI repository covers the requirements of an instantiation of the abstract model if the only types of descriptions of the value of services considered are textual descriptions and the classification of services in categories. However if, for example, formal descriptions of services are used, a UDDI repository will not fulfill the requirements we pose on the service registry, as such a repository does not enable per se the retrieval of services based on formal descriptions[5].

Our abstract model does not impose the architecture of the registry used. In fact, instantiations of the model can choose the particular architecture of the registry, as different architectures have different features which make them better suited for certain scenarios. Therefore, the architecture of the registry can range from a centralized registry e.g. [Lara et al., 2006b; Lara et al., 2007b] to a peer-to-peer registry e.g. [Toma et al., 2005], including hybrid approaches such as those described in [Verma et al., 2005; Sivashanmugam et al., 2004].

Finally, we require service registries to apply, at the time service descriptions are published, all possible pre-processing techniques over such descriptions which can speed up service retrieval. The reason is that our model does not consider service publication a time-critical task and, thus, it gives preference to an efficient service retrieval over an efficient publication.

[5]Strictly speaking, the registry is only required to enable the retrieval of services based on descriptions whose evaluation does not require direct access to consumer knowledge, i.e., the registry must enable the application of registry-side filters, as we will discuss in Section 5.3.3

In a nutshell, our model requires for service publication:

1. The existence of service registries which can store service descriptions and make them accessible to interested parties either directly or indirectly through a service location process. No particular registry architecture is assumed.

2. Service registries must enable the retrieval of services based on all different types of descriptions of a service admitted by the model instantiation.

3. Service registries must apply all possible pre-processing techniques at publication time so that retrieval times can be reduced to the minimum possible.

5.3.2 Description of goals

Goals correspond to the objectives prospective service consumers have, and they thus drive the decision on what services to use. In the same way the value of services must be described for enabling their location, the description of the value sought for achieving a goal must be described.

Similarly to the description of services, current approaches to enhance the location of services only consider a single way of describing goals. However, we have seen in Section 5.2 that different (families of) application scenarios pose different requirements on the location process and involve the definition of goals by users with different profiles. For this reason, in our model we will consider alternative descriptions of goals which can be used to match services in different ways, with different properties, and which can thus fit to different application scenarios.

In this Section, we will discuss not only the diversity of descriptions of goals expected, but also to what extent the knowledge of the prospective service consumer, which plays a role in the resolution of goals, can be expected to be disclosed and what assumptions our model makes to this respect. Finally, we will discuss the type of user support required for the description of goals. As we will see, this includes not only support for describing the objectives sought, but also in selecting what type of description of such objectives is required for achieving a given efficiency and accuracy of results.

5.3.2.1 Sought capability or functionality

The description of goals is the description of the value sought by a prospective service consumer. In our model, we require this description to capture the capability and/or functionality required, i.e., a goal description is a description of what value is required (the capability sought) and, possibly, of how this value must be provided (the functionality sought).

A goal can be described in different ways. In fact, as goal descriptions will be used to match service descriptions, the possible types of goal descriptions usually coincide or are closely related

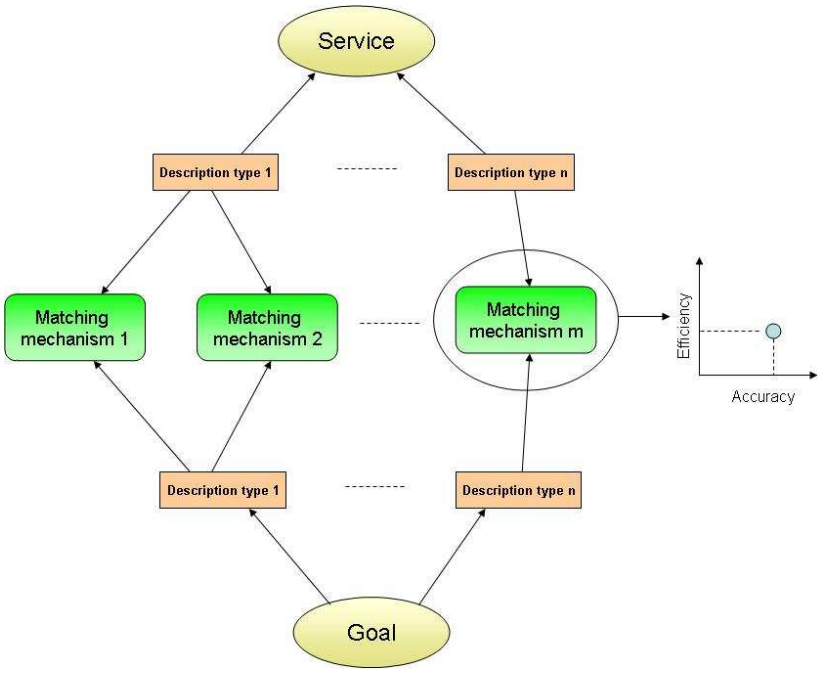

Figure 5.7: Different types of service and goal descriptions are linked by at least one matching mechanism with particular properties

to the types of service descriptions admitted. Furthermore, particular types of goal and service descriptions must have associated at least a matching mechanism with well-defined properties, i.e., goal and service descriptions are linked by at least one mechanism to establish a match between them which will depend on the particular type of description used, as depicted in Figure 5.7, and the properties (accuracy and efficiency) of each such matching mechanism must be made explicit by the instantiation of this abstract model.

Example 5.12 Let us imagine an instantiation of our abstract model admits, among others, the textual description of the value of services and of the value sought by consumers. In this case, at least a mechanism for matching textual descriptions of services and goals must be defined, but multiple mechanisms are also possible. For example, a simple keyword matching might be available, as well as a matching mechanism which also takes into account relations in WordNet [Fellbaum, 1998]. The former will probably be more efficient but less accurate, while the latter will offer better accuracy but higher response times.

In addition, at least one matching mechanism for all other types of descriptions admitted by the model instantiation must be defined, and their properties in terms of accuracy of results and efficiency of the matching process must be explicitly given.

□

Whether a goal will describe the capability or the functionality sought will mainly depend on the type of application scenario. We believe only the capability sought will be described when services are located either dynamically or by end human users (see Section 5.2); the description of the functionality of a service will be used to determine whether this particular service can be currently used to enable access to the desired capability, but it is unlikely that the goal will fix what particular functionality is sought. In these cases, the service consumer wants to locate a service offering a given capability, and will obviously require that the service can be used to access such capability, but no particular relation between current conditions and achieved effects will be generally required. However, if a service is located at design-time for its incorporation into a complex system or process, it might be the case that a particular functionality is required, as the system or process might expect a particular relation between current information and state-of-the-world conditions and the effects achieved. This is due to the fact that the service used will be always the same, i.e., the service is repeatedly used and, thus, its functionality must meet certain requirements.

Example 5.13 Let us consider the application scenario described in Example 5.1, in which parts of a pricing system are defined in terms of business goals for whose resolution appropriate services must be found at design-time and hard-wired to the system. In particular, let us consider the goal of retrieving information about products contracted by a given customer of the bank. In this case, the goal will be described by the particular functionality sought, as the system will have some particular constraints on what information can be provided to the service when the system executes and what information is expected in response. Therefore, the goal will describe the functionality expected: the provision of information about the products contracted by a bank customer, including the volume contracted and possibly other details, given the id of the customer.

However, if we consider the application scenario described in Example 5.5, the situation is slightly different. An end user wants to e.g. book a hotel in Madrid on a given date and in a particular price range, and he will describe the capability sought: booking of a hotel in Madrid with the user constraints given. However, the user will not require neither that the service accepts some particular information nor that the service establishes a particular relation between this information and the effects achieved; the user will require a service enabling access to a particular capability, and will only require that he can use the service to achieve the effects associated to such capability. In this situation, no particular functionality is sought, but only a capability; any service functionality will be fine for the user as long as there are valid initial conditions for the usage of the service and, given these conditions, the service realizes the effects required.

□

In a nutshell, a goal will describe the capability and/or functionality sought, and alternative types of descriptions of such capability and/or functionality are possible. These types of

descriptions will coincide or will be closely related to the types of service descriptions admitted by the instantiation of the model, and at least one mechanism for matching each type of description must be defined and its properties made explicit. As for services, the accuracy of goal descriptions is expected to be as high as possible. However, different users will be able to provide descriptions with different accuracy. As we will discuss later in this Section, user support will be required, and the particular type of description expected for goals will depend on the requirements on the accuracy of results and response times of the location process posed by the particular application scenario.

5.3.2.2 Consumer knowledge

In Chapter 3 we discussed how consumer knowledge plays a role in deciding whether a given service can be used to achieve a goal. In particular, services might define information preconditions, i.e., conditions over the knowledge the consumer must have and make available to the service for its execution.

In this context, the service location process must have access to consumer knowledge if we want to evaluate whether a particular service can be used to achieve the goal defined. However, there are privacy issues which might hamper the disclosure of consumer knowledge to the service location process. In general, consumers might be willing to disclose certain parts of their knowledge easily e.g. an end-user will generally disclose the country he lives in without much problem, but might be reluctant to disclose other parts unless some trust relation is established with the other party e.g. if a credit card or a passport number has to be provided. Furthermore, the description of goals might itself include some information deemed as private by the potential service consumer, as illustrated by the following example.

Example 5.14 Let us imagine an end, human user wants to book a particular flight and to pay such booking with his credit card. If the goal of this user is described in detail, it will include not only the information of the flight the user wants, but will also include the details of the credit card with which the payment must be made, such as the credit card type, holder, expiry date and number. For example, a quite detailed textual description of the goal will state: "Booking of a seat on the flight 123, flying from Madrid to New York on July 7th 2007, and payment with the Visa credit card with number 1234567".

However, and even though using this credit card to pay the booking is part of the consumer goal, credit card details should not be disclosed to third parties unless some trust relation has been previously established.

□

In this context, our model assumes that access to consumer knowledge for determining whether information preconditions are fulfilled will be limited. In particular, it is assumed that

consumer knowledge will be kept private by the consumer; its disclosure to third parties will be selective, and such disclosure will be guided by the definition of policies which define what information can be disclosed and under what circumstances, as proposed in the work we presented in [Olmedilla et al., 2004]. This corresponds to the concept of consumer knowledge available defined in Chapter 3.

Example 5.15 A business process might require at a given point in time the shipping of a book to a given address. This business goal will be described as part of the process and will be resolved at run-time by locating an appropriate service.

The process has access to some corporate databases, and there is no human intervention in the process, i.e., consumer knowledge corresponds to knowledge contained in the databases the process has access to. Furthermore, these databases contain some sensitive information such as payment methods generally used by the company running the process and employees information. Therefore, policies for the disclosure of this knowledge will be defined.

In this situation, the service location process will have limited access to consumer knowledge, as its disclosure is subject to certain policies. Therefore, the evaluation of whether a particular service can fulfill the goal defined by the process and, in particular, the evaluation of whether information preconditions of a candidate service are fulfilled, must take into account these factors.

□

As we can see, consumer knowledge is an important element for deciding upon the usability of a given service for achieving a goal. However, this consumer knowledge will only be selectively disclosed. For this reason, our model assumes consumer knowledge will be kept private by the consumer, and only disclosed to third parties under certain circumstances. Therefore, and as we will see in Section 5.3.3, we cannot assume the location process has access to the complete consumer knowledge unless the evaluation of information preconditions of candidate services is performed by the consumer itself, i.e, at the consumer side without disclosing any information.

Finally, it must be noted that if a human user participates in the execution of the service, he might provide information which actually increases the consumer knowledge which was available during the location phase.

5.3.2.3 User support

Our model requires support to prospective service consumers for describing their goals. While services were required to be described in multiple ways, ideally in as many ways as types of service descriptions were admitted by the instantiation of the abstract model, consumer goals are expected to be described only using the types of descriptions for which matching mechanisms with the desired efficiency and accuracy are available.

Example 5.16 Let us imagine that an instantiation of our abstract model for the location of services admits formal, first-order descriptions of the effects associated to a service capability, and textual descriptions of such effects. For the first type of description, a matching mechanism based on formal reasoning is available with high accuracy but very low efficiency e.g. the mechanism proposed in [Keller et al., 2004b], while for the second type of description a simple keyword matching mechanism is available, with high efficiency but low accuracy e.g. the keyword matching provided by UDDI repositories [Bellwood et al., 2002].

Now, let us imagine an end-user wants to quickly find services which might fulfill his goal of renting a car. In this case, the usage of textual descriptions and the application of keyword matching is more appropriate, as low response times are required and manual filter of candidate services by the end-user is possible. Therefore, the type of description of the consumer goal required will be a textual description, not a formal description.

□

In general, we require support to be provided to users for deciding what type of description of their goals must be given depending on their requirements. Such requirements will be generally described in terms of the efficiency of the location process and the accuracy of results desired, and suitable matching mechanisms (filters) with their associated types of descriptions will be proposed to consumers. This proposal might include not a single filter but combinations of them.

Once the type of description of the goal required for achieving the efficiency and accuracy expected has been determined, consumers must also be supported for providing such type of description with as much accuracy as possible. This type of support will be based on the same elements as the support offered to service providers for describing their services: the relation between the different types of descriptions will be exploited, possibly through the definition and matching of pre-defined goals, and appropriate tool support will be available for helping consumers in enhancing the accuracy of their descriptions.

In this way, consumers will be guided in choosing the matching mechanisms and goal descriptions better suited for their requirements. If the type of description required cannot be directly provided by the consumer, he will be required to provide another type of description he is familiar with, and the relation between different types of descriptions will be exploited to extract a description of the required type from the description of the type the consumer has provided. This can possibly be done through the definition of pre-defined goals, which will describe general goals in different ways and will be matched using the description provided by the consumer in order to obtain related descriptions of other types[6].

[6]This mechanism is similar to the concept of goal discovery proposed in [Lara et al., 2004a; Keller et al., 2004a; Keller et al., 2006a].

5.3.3 Discovery and selection of services

Once service descriptions have been published at one or more registries and a goal has been described, the location or discovery of services which can fulfill the goal given starts. This process is exclusively based on descriptions of goals and services, and there is no direct interaction between potential service consumers and available services. As we will see, interaction is possible and often necessary for the selection of services, but it is not allowed during the location of relevant services for achieving the goal. The reason is that the blind execution of services is much more expensive than working over descriptions, as it involves communication with each service considered.

In this Section, we present how the discovery and selection of services is organized in our model, what assumptions are made, and what principles guide the model. The discovery of services will be based on the application of different matching mechanisms to goal and service descriptions, called *filters*, and the application of these filters will be organized in two phases, namely: a) retrieval of relevant services from registries, and b) evaluation at the consumer side, where consumer knowledge is accessible, of the retrieved descriptions of services [Lara, 2006]. These two phases will be explained in the following, as well as the general type of filters considered in each case.

5.3.3.1 Registry-side filters

As discussed in Section 5.3.1, services can be and will ideally be described in multiple ways, as many as admitted by the particular instantiation of this abstract model, and these descriptions will be published at appropriate registries. The first phase in our discovery model is to retrieve service descriptions from these repositories. However, not all descriptions will be retrieved, but only descriptions of services which are relevant to a given goal.

The relevance of a service for achieving a particular goal is determined based on the description of the goal and the service. Furthermore, different mechanisms, with different properties and based on different types of descriptions of goals and services, can be used to determine this relevance. Therefore, and given that the potential service consumer can decide to provide different types of descriptions of his goal and that different matching mechanisms can be chosen by such consumer to find appropriate service descriptions, we want to retrieve from registries those services whose description is deemed as relevant under the matchmaking mechanism selected by the potential consumer and using the type of description of goals and services required for applying this mechanism.

In this context, we require service registries to implement all matching mechanisms defined by the instantiation of the model over the types of descriptions admitted. We will call these matching mechanisms *filters*, and they will be applied, given a particular goal, over all service descriptions published in the registry in order to filter which ones correspond to services relevant for the goal

at hand. These filters are *registry-side filters* as they are applied by the registry itself based on the goal description given and on the service descriptions available.

It must be noted that we allow for the application of multiple registry-side filters, i.e., for the usage of more than one matching mechanism for determining relevant service descriptions for the goal at hand. The possible benefits of combining multiple registry-side filters will depend fundamentally on the type of filters defined and whether they are to some extent complementary. Still, potential consumers can decide to apply multiple filters to obtain the description of relevant service for achieving their goal and, furthermore, some registry-side filters can be actually defined as combinations of other types of filters.

As different filters operate over different types of service descriptions, these filters will only be applicable to services for which the necessary type of description has been provided. This is an extremely important reason for service providers to describe their services in all admitted ways so that their services can be found when applying different types of filters.

In our model, registry-side filters operate exclusively over service and goal descriptions, i.e., they will not have access to consumer knowledge. Thus, this type of filters will not evaluate whether the potential service consumer has the required knowledge and/or will be willing to provide it to the service for execution.

After the application of filters, those services deemed relevant for achieving the goal provided will be known. Furthermore, the application of filters to service descriptions might not yield a boolean answer, but might distinguish among different degrees of relevance of available services for solving the goal at hand. In this case, not only what services are relevant, but also to what extent, will be obtained.

In a nutshell, a goal will be described and sent to the registry or registries where service descriptions are stored, together with the filters that must be applied. The selected filters will be applied by the registry to available service descriptions to determine which of them are relevant for achieving the goal received, and will return relevant service descriptions to the potential service consumer (or to an agent acting on behalf of the consumer) (step 1 in Figure 5.8). In this way, relevant services will be retrieved from repositories only based on goal and service descriptions, without access to consumer knowledge and without any actual communication with available services.

5.3.3.2 Consumer-side filters

After the application of registry-side filters, the descriptions of relevant services according to the filters applied are available at the service consumer side, where access to consumer knowledge is possible. This enables the application of filters which can evaluate whether the consumer has the necessary knowledge for using the service, and whether this knowledge can result on the effects required.

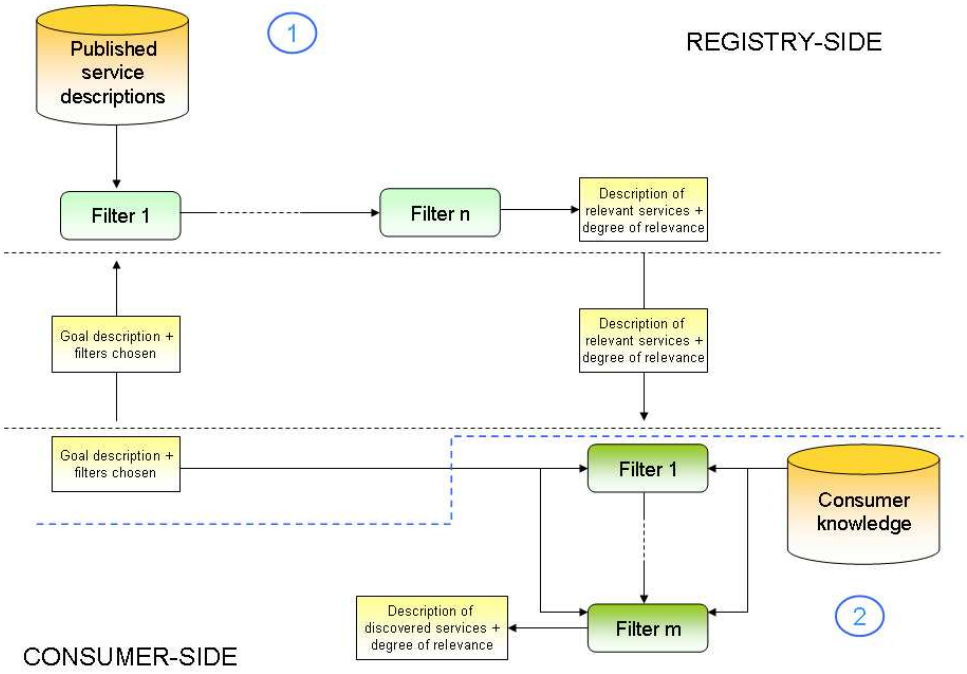

Figure 5.8: Discovery in two steps

Furthermore, filters which are not provided by service registries but which are of interest for a particular application can also be locally applied by the service consumer, thus offering flexibility on the set of filters available, as this set can be freely extended by the service consumer if new filters are required for covering his needs and as long as the descriptions of goals and services necessary for applying these filters are in place. Therefore, we can have two types of consumer-side filters:

1. Filters which perform a matching of services and goals taking into account consumer knowledge, accessible only on the consumer side.

2. Consumer-provided filters which are not implemented by service registries but which are of interest for the service consumer; these filters might also access consumer knowledge.

More than one consumer-side filter can be applied, as depicted in Figure 5.8 (step 2). The particular consumer-side filters to be applied will also be selected by the potential service consumer, so registry-side filters were.

Consumer-side filters will be applied only over the set of service descriptions retrieved from service registries, i.e., only over relevant services, and the application of these filters will result on a reduction of this set, yielding the final set of discovered services. The descriptions of these services can be then used for our next step: the selection of services.

Evaluation of real-world preconditions. It must be noted that at this step we have access to consumer knowledge, but we will in general not have a complete view of the state of the world. Our model assumes that the overall location process, either when registry-side or consumer-side filters are applied, will have only partial visibility over the state of the world. Therefore, whether the service can be used at the present moment cannot be generally assessed, i.e., only information preconditions can be evaluated by consumer-side filters, but not real-world preconditions, as illustrated by the following example.

Example 5.17 Let us imagine a consumer wants to book, with his credit card, a seat on a flight from Paris to London on a given date. He will describe this goal e.g. textually, will remove sensitive information from it e.g. his credit card details, and will send it to available repositories for retrieving the description of relevant services based on a keyword-based filter which also exploits WordNet relations [Fellbaum, 1998]. The selected filter will be applied at the registry, which will result on a set of service descriptions deemed relevant for achieving the goal described, possibly with different degree of relevance based on some scoring mechanism.

Relevant services will be returned to the potential consumer. From these services, some of them might require a particular type of credit card to be used or the name of a registered user of the service to be provided. A consumer-side filter can be applied to determine whether the consumer has the necessary knowledge to fulfill the preconditions of the retrieved services. Furthermore, the consumer might want to apply a custom filter for matching his goal to the description of the services retrieved.

After consumer-side filters have been applied, we have a set of services which are relevant to the goal given and usable by the service consumer in terms of the fulfillment of their information preconditions. However, there will most likely be services which define real-world preconditions stating that a seat on a flight will only be booked if such a is seat available, i.e., not already booked. For evaluating a condition like seat availability, we would need to have access to this information at discovery time, which will generally not be the case unless a service providing this information is in turn located and used to retrieve this particular knowledge of the state of the world. Even if such a service is available, which might not be the case, it can turn out that the service providing this knowledge also defines some real-world preconditions we have to evaluate; for this evaluation, we might need to again locate a service providing the required knowledge and execute it.

Summarizing, we can see that, for evaluating this type of real-world precondition, we need knowledge which might not be known by the service location process or by the consumer. This results in the need for locating services offering this knowledge, which requires the execution of services during discovery and can imply a recursive location problem [Lara, 2006].

□

In our model, we will not assume real-world preconditions of relevant services can be evaluated. In fact, we do not expect instantiations of this abstract model to include this evaluation as part of the discovery process, but they will instead leave it to the service selection step briefly described below.

5.3.3.3 Service selection

After the application of the two types of filters described above, we will have a set of service descriptions corresponding to services which might fulfill the goal defined. However, we must remember that: i) these descriptions have been retrieved based on the application of filters to complete but not necessarily correct descriptions of services (see Section 5.3.1), ii) real world preconditions of services were not evaluated as access to real-world knowledge is incomplete and, iii) the result of applying these filters might not be completely accurate depending on the efficiency requirements of the consumer and on the accuracy of filters themselves.

In this situation, and also considering that more than one service might be deemed relevant after the application of the selected filters, we have to go through a selection step where the actual service to be used is determined. If we want to have a full guarantee that the selected service can actually be used for achieving the consumer goal, the selection step will generally involve direct communication with discovered services in order to determine whether the required effects will result from service usage. After this selection, a particular value provision will be contracted [Preist, 2004], i.e., agreed between consumer and provider, SLAs will be established [de Bruijn et al., 2005d], and the service will be finally executed.

Regarding the communication with services to determine whether they can provide the value expected, it can be carried out by executing the service and evaluating whether the effects achieved are the ones sought. However, this is problematic for services having real world effects, as they might not be always visible and, if the effects achieved are not the ones expected, it might not always be possible to roll back these effects. Therefore, we expect the existence of a service contracting interface used to negotiate what effects will be provided by the service, and only after this negotiation succeeds actual execution will happen.

In the case of services which are fixed into a system or business process (see Section 5.2.1), this selection can be slightly different, as not only a guarantee that the service will provide *now* the value required is needed, but a guarantee that the service will provide this value for a given period of time. In this case, the selection step will sometimes require the establishment of long-lasting service provision contracts between consumer and provider.

In a nutshell, our model assumes the existence of a service selection process so that consumers can have a guarantee that a given candidate service will be actually able to offer a value satisfying his goal. This step will generally involve direct communication with candidate services,

which will be done through a service contracting interface at least for services whose usage results in real-word effects. After a service is selected, it will be actually used to achieve the effects which will resolve the goal defined.

5.4 Summary

The adoption of the SOA paradigm can expand different domains and application scenarios with different requirements. Therefore, the location of services which can achieve a given goal might be necessary in different scenarios and a comprehensive solution must take into account this diversity and offer an appropriate level of flexibility so that this variety of scenarios can be covered.

In this Section, we have presented an abstract model for the location of services which, based on an analysis of the main families of use cases which must be covered and of practical considerations which affect how the model can be articulated, keeps flexibility as a key principle. The model tries to be comprehensive and cover all aspects of service location, not only the matchmaking of a particular type of description of goals and services.

The abstract model presented is intended to provide general guidelines and design decisions for the location of services. However, particular instantiations of the model must concretize aspects which are left open by the abstract model, such as the particular types of descriptions of goals and services considered and the matching mechanisms applied to them. Such an instantiation, together with a prototype implementation, will be presented in the next two Chapters.

In the following, we briefly summarize the main aspects of our abstract model.

Description and publication of services. In our model, the way the value of services is described is not unique, but alternative types of descriptions of a service value can be provided in order to make the model usable for users with different profiles and in order to enable the application of matching mechanisms with different properties; particular instantiations of the abstract model will decide and make explicit what types of descriptions are admitted and how they are matched.

Alternative descriptions of services are possible, but none of them is expected to capture with complete accuracy the value of the service, as this is unrealistic in most cases. In this setting, we will assume service descriptions are complete but not necessarily correct, and that they are static in the sense that they will not be altered every time some dynamic condition which affects the set of effects achievable by using the service changes.

Providers must be supported for providing the types of descriptions of their services admitted by the model instantiation, both in improving the accuracy of their descriptions and in providing different types of descriptions. We believe strategies for providing this type of support are an important part of a model for the location of services and, thus, of instantiations of our abstract

model.

Finally, we require the existence of registries which store the descriptions of available services and enable their location based on the different types of descriptions admitted by the model instantiation and the associated matching mechanisms. Whether a single centralized registry, a completely distributed architecture in which any peer can register services, or a hybrid model is used is not dictated by our model, but it is up to particular instantiations to concretize it.

Description of goals Goals describe the capability or functionality sought, and alternative types of descriptions of such capability or functionality are possible. The types of descriptions admitted by a model instantiation will coincide or will be very closely related to the types of descriptions of the value of services admitted, and links must be established between types of goal descriptions and types of service descriptions, including the matching mechanisms available and their properties.

Consumer knowledge will also be relevant for the location of services, as it determines whether appropriate information can be provided to a candidate service. Therefore, this knowledge will be made explicit, but it will not be disclosed to other parties during the location of services; only after some service has been selected and a trust relation has been established such knowledge might be disclosed.

Finally, support must be provided to users for describing their goals, in the same way support was given to service providers for describing the value of their services. Support will be provided to service consumers not only for providing different types of descriptions of their goals, but also for choosing what type of description of their goals and matching mechanisms are better suited for their efficiency and accuracy requirements.

Discovery and selection of services The discovery of services is based on the possibly successive application of matching mechanisms, called filters. This application is split into two phases: the application of filters on the registry where service descriptions are stored and the retrieval of descriptions which can potentially resolve the goal, and the application of filters over the retrieved descriptions on the consumer side, where consumer knowledge is accessible. Consumer-side filters are filters which require access to consumer knowledge, or which are custom-defined by consumers. By splitting the discovery process in this way, we avoid the disclosure of consumer knowledge to the registry or registries, and we allow consumers to define custom filters over the descriptions retrieved.

After appropriate services have been discovered, a selection phase will be necessary as the results obtained from the discovery step might not be completely accurate. This selection requires communication with candidate services so that a full guarantee that the service will provide the effects required is obtained. For that purpose, the existence of a contracting interface which enables the establishment of provision contracts between service consumers and service providers, prior to

actual service execution, is assumed.

The model presented has been inspired by the work we have presented in [Kifer et al., 2004; Lara et al., 2004a; Keller et al., 2005; Lara and Olmedilla, 2005; Keller et al., 2006a], and by LARKS [Sycara et al., 2002], where also alternative descriptions of agents and alternative filters are considered. Posterior works such as OWLS-MX [Klusch et al., 2006] have followed a similar approach. A detailed discussion of these related works will be presented in Chapter 8.

Chapter 6

Model Instantiation and Prototype Implementation I: Description and Publication of Services

6.1 Introduction

In the previous Chapter, we have presented an abstract model for the location of services which can provide a certain value defined by a consumer goal. Such model was kept abstract, as it only defined a master line for the location process, leaving many aspects open for instantiations of the model to concretize them. This Chapter and the next one present a proposal for an instantiation of the abstract model and an associated prototype implementation, developed in the course of the SETA project[1], called the *SETA service location platform* after the name of the project (SETA platform for short), and whose core aspects have been published in [Lara et al., 2007b; Lara et al., 2006b]. In particular, this Chapter presents the instantiation of the first part of the abstract model, dealing with the description and publication of services. The next Chapter will present the instantiation of the part of the abstract model concerned with the process of actually locating services which can potentially fulfill a goal given, organized in two phases as described in Chapter 5.

The instantiation presented is not meant to be the only possible one, but it serves as a proof of concept of the abstract model proposed, enables an evaluation of the abstract model based on a particular instantiation, and demonstrates the feasibility of enhancing current service location practices. The instantiation proposed has the following salient features:

[1] http://www.tifbrewery.com/tifBrewery/writing.htm

1. It admits both formal and non-formal descriptions of services and goals, and of different types.
2. The descriptions used are integrated into an existing framework for the description of services (WSMO) and formal descriptions are expressed using existing languages with formal semantics (the WSML family of languages). Still, portability of descriptions to other frameworks is possible.
3. It admits formal descriptions with different semantics. In particular, it enables the combined use of descriptions with first-order and logic programming semantics for matching services and goals.
4. The reasoning support required over formal descriptions is mostly provided by existing reasoning infrastructure. In particular, the reasoning capabilities required are provided by the RACERPro [RAC, 2006; Haarslev and Möller, 2001] and Flora-2 [Yang et al., 2005] reasoners, among others.
5. The alternative types of descriptions proposed and their associated filters keep a balance between simplicity and coverage of application needs for different application types.
6. Support to users for describing their services and goals is included in the instantiation proposed.
7. A prototype implementation, as well as an evaluation of the model based on this implementation, has been accomplished.

In the following, we will present the first part of the instantiation of the abstract model, which will make concrete how services and goals are described (Section 6.2), how users are supported for providing these descriptions, and how service descriptions are published (Section 6.3). This will include a presentation of the parts of our prototype implementation concerned with these tasks.

6.2 Types of descriptions of services

In this Section, we present the types of descriptions of services considered by the SETA platform, as well as how they are integrated into the WSMO framework, i.e., how our service descriptions are encoded as part of a WSMO service description. The types of descriptions considered are split into two categories: syntactic or non-formal descriptions, and semantic or formal descriptions. Within each category, the details of the different types of descriptions considered will be introduced.

In general, our instantiation tries to, with a relatively small set of alternative descriptions, cover a wide range of user profiles and application scenarios; usability has been a criterium for choosing the types of descriptions of services and goals admitted, as well as fulfillment of heterogeneous requirements in terms of efficiency and accuracy.

6.2.1 Syntactic descriptions

Syntactic descriptions have no explicit formal semantics and, therefore, they are not amenable to the application of formal reasoning. While the lack of formal semantics will in general reduce the accuracy of matching this type of descriptions, they have other important features which makes their consideration worthwhile, namely: a) they are easy to provide by users with different profiles, and b) highly efficient filters can be applied to them.

In the following, we present the different types of syntactic descriptions we have included in the SETA platform. We will discuss the motivation for the use of each of these types of descriptions, as well as what user profiles they are targeted at, what part of the service value they capture, what type of filters are applicable to them, and how they are incorporated into WSMO service descriptions. It must be noticed that we will only outline the kind of filters applicable to each type of description and summarize their general properties; particular filters included in the SETA platform and their implementation will not be presented in detail until the next Chapter.

6.2.1.1 WSDL description

The most prominent and widely supported specification for describing the technical interface of a service for its actual provision or execution is the Web Service Description Language (WSDL) [Christensen et al., 2001; Alonso et al., 2003] (see Chapter 2, Section 2.1.2).

Beyond describing how a service can be technically accessed, WSDL descriptions capture information of services which can be used for their location. In particular, WSDL port types [Christensen et al., 2001] provide a signature view of the operations which can be accessed by consumers using the service, which in some cases gives information to prospective consumers of the actual value of the service. Furthermore, technical service users currently make use of this type of description to make a first decision on the suitability of a service for achieving a given task. This means that this is a type of description users with a technical profile are familiar with and which has proven somehow useful in the past. Last, but not least, current mechanisms for service location considerably rely on this type of description. In fact, a method for mapping WSDL descriptions to the UDDI model [Bellwood et al., 2002] has been defined, which in turn enables the location of services in UDDI repositories based on their WSDL descriptions [Colgrave and Rogers, 2004][2].

Summarizing, the WSDL description of a service sometimes offers information (although often imprecise) of what the service does, and it is currently widely accepted and used by technical users and service platforms, including registries. Therefore, if we want to ensure that our model is compatible with current practices so that a progressive enhancement of the service location process is

[2]However, the granularity allowed for querying for services based on their WSDL descriptions is limited if we only use core UDDI tModels. This issue will be discussed in more detail in the next Chapter.

possible, we must consider WSDL descriptions not only as a description of how to programmatically access a service but also as a possible description of the value of such service.

Modelling and aspects of the service captured. The modelling of services given by WSDL descriptions has been presented in Chapter 2, Section 2.1.2. WSDL port types and their operations are given names which might be descriptive, i.e., which might to some extent reflect the value associated to the port type and its operations, and operations define input and output messages whose names and syntactic structure might also reflect to some extent the service value. Furthermore, WSDL allows for the use of documentation elements which enable the incorporation of textual descriptions of any aspect of the service.

Listing 6.1 shows the WSDL description of a service offering the search of investment funds. As it can be seen, both in the syntactic description of the messages received and returned by the port type operation and in the description of the port type and its operation there is some information about what the service actually does.

Listing 6.1: WSDL description of a service for the search of investment funds

```
  <?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://www.afi.es/WSFundSearchEngine"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  targetNamespace="http://www.afi.es/WSFundSearchEngine"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://www.afi.es/WSFundSearchEngine">
      <s:element name="FundSearch">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="user" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="password" type="s:string" />
            <s:element minOccurs="1" maxOccurs="1" name="companyId" type="s:int" />
            <s:element minOccurs="1" maxOccurs="1" name="categoryId" type="s:int" />
            <s:element minOccurs="0" maxOccurs="1" name="fundName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="ISIN" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="FundSearchResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="FundSearchResult"
                type="tns:ArrayOfWSFund" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfWSFund">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="WSFund"
                nillable="true" type="tns:WSFund" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="WSFund">
        <s:sequence>
```

```
          <s:element minOccurs="0" maxOccurs="1" name="name" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="companyName" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="financialGroup" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="depositEntity" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="categoryAFI" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="categoryCNMV" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="currency" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="ratingAFI" type="s:string" />
          <s:element minOccurs="1" maxOccurs="1" name="registrationDate" type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="rankingPosition" type="s:int" />
          <s:element minOccurs="1" maxOccurs="1" name="rankingTotal" type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="investmentPolicy" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="rentabilityRisk" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="lastValues" type="tns:ArrayOfValueForDate" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="ArrayOfValueForDate">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="ValueForDate" nillable="true"
              type="tns:ValueForDate" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="ValueForDate">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="date" type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="value" type="s:decimal" />
          <s:element minOccurs="1" maxOccurs="1" name="rentability12M" type="s:decimal" />
        </s:sequence>
      </s:complexType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="FundSearchSoapIn">
    <wsdl:part name="parameters" element="tns:FundSearch" />
  </wsdl:message>
  <wsdl:message name="FundSearchSoapOut">
    <wsdl:part name="parameters" element="tns:FundSearchResponse" />
  </wsdl:message>
  <wsdl:portType name="WSFundSearchEngineSoap">
    <wsdl:operation name="FundSearch">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/"> Receives a user and password plus
        search criteria and returs investment funds meeting the criteria given
      </documentation>
      <wsdl:input message="tns:FundSearchSoapIn" />
      <wsdl:output message="tns:FundSearchSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="WSFundSearchEngineSoap" type="tns:WSFundSearchEngineSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <wsdl:operation name="FundSearch">
      <soap:operation soapAction="http://www.afi.es/WSFundSearchEngine/FundSearch"
        style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="WSFundSearchEngine">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/"> Search of investment funds
        commercialized in the Spanish market.
    </documentation>
    <wsdl:port name="WSFundSearchEngineSoap" binding="tns:WSFundSearchEngineSoap">
      <soap:address location="https://www.afi.es/FinancialWebService/WSFundSearchEngine.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

While the XML-based description of port types, operations and messages is purely syntac-

tic, the following correspondence can be established between the elements of this description and the elements of the conceptual model introduced in Chapter 5:

- Inputs and outputs of a port type correspond to input and output variables of a service interface. Actually, a WSDL description is primarily intended to describe the interface of the service in terms of operations and messages exchanged.
- The description of messages, their names and their structure can be seen as a syntactic version of domain models (ontologies). While no formal semantics is given to the messages used in WSDL descriptions, their structure and textual description can be used to identify the meaning of these messages.
- Operations can be seen as a syntactic description of the service functionality. However, it must be noted that the relation between initial conditions (before service usage) and final conditions (after service usage) can only be captured textually using WSDL documentation elements as shown in Listing 6.1, and that real world preconditions and effects can only be incorporated to the description of the operation in the same way, that is, as a textual documentation element. Furthermore, the description of input and output messages of an operation only contains the data type or syntactic structure of information preconditions and effects, not their precise semantics.
- Documentation elements can contain textual descriptions of any service element. While documentation elements can be placed anywhere in a WSDL description, the place in the WSDL description where documentation elements are found can be guide us in knowing what aspect of the service value they might capture. In particular, documentation elements associated to messages can be expected to be textual descriptions of the domain model used, documentation elements associated to port types, operations and input/outputs as textual descriptions of the service functionality, and documentation elements associated to the service element as descriptions of the service capability. However, note that this separation is not strict and it depends on the intention of the service provider when describing his service. Therefore, it serves as a guide but by no means as a robust rule for interpreting documentation elements in WSDL descriptions.

In a nutshell, WSDL descriptions capture the signature of services, i.e., their interface assuming a one-shot interaction and, to some extent, their functionality and/or capability.

Target user profile. The creation of WSDL descriptions is usually reserved to technical users, who can create them manually or generate them using existing development tools such as Eclipse[3],

[3]http://www.eclipse.org/

NetBeans[4] or Visual Studio .NET[5]. Non-technical users are not expected to deal with this type of description.

Encoding in WSMO. Services offered via a WSDL interface are expected to incorporate their WSDL description as part of a WSMO service grounding, so that a WSMO execution environment, following the choreography of the service, will be able to know which actual messages to send and/or expect to receive. The WSDL description will provide the networking details, for example that the data should be serialized as SOAP XML messages and sent over HTTP [Kopecky et al., 2005]. However, a stable and well-defined mechanism for incorporating WSDL descriptions for the grounding of WSMO services is still lacking. Therefore, and as a temporary solution until such a mechanism is available, we incorporate into WSMO service descriptions a non-functional property of the service capability with name *wsdlDescription* and whose value is a URI pointing to a location where the service WSDL description can be found. Listing 6.2 shows an example WSMO description of a service offering the search of investment funds commercialized in the Spanish market. The *wsdlDescription* property points in the example to a location where a WSDL file containing the description in Listing 6.1 can be found.

```
namespace { _"http://www.afi.es/services/finance#",
        loc _"http://www.afi.es/ontologies/ geopolitical /locations#",
        action _"http://www.afi.es/ontologies/general/actions#",
        funds _"http://www.afi.es/ontologies/finance/investmentFunds#",
        dc _"http://purl.org/dc/elements/1.1#",
        afi _"http://www.afi.es/WSDiscovery/Description#" }

webService _"http://www.afi.es/ services /FundSearch1"

  importsOntology { _"http://www.afi.es/ontologies/ geopolitical /locations",
    _"http://www.afi.es/ontologies/general/actions",
    _"http://www.afi.es/ontologies/finance/investmentFunds" }

  capability FundSearch1Capability
    nonFunctionalProperties
      afi#wsdlDescription hasValue "http://www.afi.es/FinancialWebService/WSFundSearchEngine.asmx?wsdl"
      dc#description hasValue "Search of investment funds commercialized in the Spanish market"
      dc#language hasValue "en-GB"
      afi#category hasValue "http://www.afi.es/Taxonomy1#InvestmentFundSearch"
    endNonFunctionalProperties

    sharedVariables {?c}

    postcondition
      nonFunctionalProperties
        afi#descriptionType hasValue "setBasedCapability"
        afi#intention hasValue "all"
```

[4] http://www.netbeans.org/
[5] http://en.wikipedia.org/wiki/Visual_Studio_.NET

```
  endNonFunctionalProperties
  definedBy
    ?x memberOf _"http://www.afi.es/services/FundSearch1" equivalent
    ?x memberOf action#InfoProvision and
        exists ?y(?x[action#ofInfoItem hasValue ?y]) and
        forall ?y(
            ?x[action#ofInfoItem hasValue ?y] implies ?y memberOf funds#InvestmentFund[funds#commercializedIn
                hasValue loc#Spain]).

precondition
  nonFunctionalProperties
    afi#descriptionType hasValue "LPInfoPreconditions"
  endNonFunctionalProperties
  definedBy
    ?c memberOf funds#FundsCategory and ?c[funds#definedByEntity hasValue funds#CNMV].

postcondition
  nonFunctionalProperties
    afi#postconditionType hasValue "inputDependentEffects"
    afi#intention hasValue "all"
  endNonFunctionalProperties
  definedBy
    ?x memberOf _"http://www.afi.es/services/FundSearch1" equivalent
    ?x memberOf action#InfoProvision and
        exists ?y(
            ?x[action#ofInfoItem hasValue ?y]) and
        forall ?y(
            ?x[action#ofInfoItem hasValue ?y] implies ?y memberOf funds#InvestmentFund[funds#commercializedIn
                hasValue loc#Spain, funds#hasCategory hasValue ?c]).
```

Listing 6.2: WSMO description of a service offering the search of investment funds

Applicable filters. The nature of WSDL descriptions does not enable the application of formal reasoning to the location of services providing a given value. We envision the application of the following types of heuristic filters to this type of description:

1. Filters which, given some particular elements of a WSDL description e.g. the name of a port type, find services with these characteristics. This type of filter is provided by the UDDI inquiry API [Bellwood et al., 2002], which, based on predefined tModels, enables querying for services based on elements of their WSDL descriptions, i.e., constraints can be expressed based on the elements and structure of WSDL descriptions and services fulfilling such constraints are matched.

2. Textual filters, which interpret WSDL descriptions as textual descriptions of the service and match textual requirements to these descriptions using different text matching techniques. This type of filter can be obtained e.g. if a WSDL description of a service is made public and

indexed by current search engines such as Yahoo![6] or Google[7].

3. Filters which perform keyword or textual matching but also take into account the structure of WSDL descriptions, i.e,. they take into account where the textual description appears e.g. as the name of an operation or as the name of an input message. A filter or matching mechanism which can be classified into this type is Woogle [Dong et al., 2004].

The type of filters enumerated above are expected to have a moderate precision, i.e., they will not be very accurate in finding services which can offer the value desired. However, they are expected to be quite efficient. The particular filters included in the SETA platform will be discussed in the next Chapter.

6.2.1.2 Textual description

Human users easily deal with information in natural language. Therefore, the textual description of the value of services, while sometimes imprecise due to the intrinsic characteristics of natural language and to the possible mistakes or limitations in the use of the language by users, is a useful and valuable type of description.

Textual descriptions incorporated into WSDL are coupled to a technical description, which we might want to avoid if we want different users to provide different types of descriptions they are more familiar with. For example, a business user might provide a textual description of the business value of a service, while textual descriptions included in a WSDL description of such service, provided by technical users, might have a technical bias and might lack the business point of view offered by a business user. Therefore, we consider a separate textual description of a service value, separate from textual descriptions encoded as WSDL documentation elements. However, this does not imply that these two types of textual descriptions are completely different. They will be usually related, and one type might be extracted, at least partially, from the other, as we will see in Section 6.3.

Besides the usability of textual descriptions by average users, there are other factors which motivate the usage of this type of description in the SETA platform: its suitability for manual filtering of discovered services, and the maturity and variety of text matching techniques. Depending on the application scenario, there might be a manual selection step over the services discovered (see Chapter 5, Section 5.3.3.3); textual descriptions of services are convenient for this task, as they help human users to perform a quick filtering of services before inspecting more complex descriptions. Additionally, there exists a wide range of mature techniques for the matching of textual descriptions, ranging from simple keyword matching to more complex techniques involving word stemming, sentence structure, etc. Furthermore, many of these techniques are built-in in common

[6] http://www.yahoo.com/
[7] http://www.google.com/

systems such as databases, and they have been successfully used by e.g. Web search engines. This enables the possibility of choosing among alternative, tested filters with different properties.

Modelling style and aspects of the service captured. We assume a single textual description of the value of the service is given. This description can capture the service capability and/or its functionality. We note that separate textual descriptions of the capability and the functionality of the service could be given, and even of its preconditions and effects. However, and for simplicity of usage, we will provide only one textual description element and leave the separation of these aspects of the service value to more complex, formal descriptions. The main reason is that we want to keep this type of description simple and highly usable for average users.

Support for textual descriptions in different languages e.g. English and Spanish is currently not supported by the SETA platform. This possibility is foreseen as an extension when the annotation extensions introduced in [Toma and Foxvog, 2006], and which easily allows for incorporating multilingual non-functional properties in WSMO descriptions, become supported by current WSMO parsers and tools.

Target user profile. Any user profile is expected to be able to provide a textual description of the value of a service. However, this type of description is specially well suited for business users, who can easily provide a description of the service value from a business point of view. On the contrary, technical users will most likely incorporate their textual descriptions into WSDL descriptions.

Encoding in WSMO. Textual descriptions are encoded in WSMO services as the value of the Dublin Core *description* non-functional property[8] of the capability, as shown in Listings 6.2 and 6.3. Additionally, the Dublin Core *language* non-functional property is used to indicate the language in which this description is provided.

```
namespace { _"http://www.afi.es/services/eTourism#",
        loc _"http://www.afi.es/ontologies/ geopolitical /locations#",
        action _"http://www.afi.es/ontologies/general/actions#",
        flights _"http://www.afi.es/ontologies/eTourism/flightsExample#",
        dc _"http://purl.org/dc/elements/1.1#",
        g _"http://www.afi.es/ontologies/general#",
        afi _"http://www.afi.es/WSDiscovery/Description#" }

webService _"http://www.afi.es/ services /LowCostFlightSearch1"

  importsOntology { _"http://www.afi.es/ontologies/ geopolitical /locations",
    _"http://www.afi.es/ontologies/general/actions",
    _"http://www.afi.es/ontologies/general",
    _"http://www.afi.es/ontologies/eTourism/flightsExample" }
```

[8] http://www.dublincore.org/

```
capability LowCostFlightSearch1Capability
  nonFunctionalProperties
    afi#wsdlDescription hasValue "http://www.afi.es/eTourism/LowCostFlightSearch1.asmx?wsdl"
    dc#description hasValue "Search of flights operated by low-cost airlines"
    dc#language hasValue "en-GB"
    afi#category hasValue "http://www.afi.es/Taxonomy1#LowCostFlightSearch"
  endNonFunctionalProperties

  sharedVariables {?co, ?cd, ?d}

  postcondition
    nonFunctionalProperties
      afi#descriptionType hasValue "setBasedCapability"
      afi#intention hasValue "some"
    endNonFunctionalProperties
    definedBy
      ?x memberOf _"http://www.afi.es/services/LowCostFlightSearch1" equivalent
      ?x memberOf action#InfoProvision and
          exists ?y(?x[action#ofInfoItem hasValue ?y]) and
          forall ?y(
              ?x[action#ofInfoItem hasValue ?y] implies (?y memberOf flights#Flight and
              exists ?z(?y[flights#operatedBy hasValue ?z]) and
              forall ?z(
                  ?y[flights#operatedBy hasValue ?z] implies ?z memberOf flights#LowCostAirline))).

  precondition
    nonFunctionalProperties
      afi#descriptionType hasValue "LPInfoPreconditions"
    endNonFunctionalProperties
    definedBy
      ?co memberOf loc#City and ?cd memberOf loc#City and ?d memberOf date.

  postcondition
    nonFunctionalProperties
      afi#postconditionType hasValue "inputDependentPostcondition"
      afi#intention hasValue "some"
    endNonFunctionalProperties
    definedBy
      ?x memberOf _"http://www.afi.es/services/LowCostFlightSearch1" equivalent
      ?x memberOf action#InfoProvision and
          exists ?y(?x[action#ofInfoItem hasValue ?y]) and
          forall ?y(
              ?x[action#ofInfoItem hasValue ?y] implies ?(y memberOf flights#Flight and
              exists ?z(?y[flights#operatedBy hasValue ?z]) and
              forall ?z(
                  ?y[flights#operatedBy hasValue ?z] implies (?z memberOf flights#LowCostAirline and
                  ?y[ flights#hasOrigin hasValue ?co] and ?y[ flights#hasDestination hasValue ?cd] and
                  ?y[flights#hasDate hasValue ?d])))).
```

Listing 6.3: WSMO description of a service offering the search of low-cost flights

```
namespace { _"http://www.afi.es/services/eTourism#",
        loc _"http://www.afi.es/ontologies/ geopolitical /locations#",
```

```
        action _"http://www.afi.es/ontologies/general/ETourismActions#",
        flights _"http://www.afi.es/ontologies/eTourism/flightsExample#",
        fc _"http://www.afi.es/ontologies/finance/common#",
        dc _"http://purl.org/dc/elements/1.1#",
        g _"http://www.afi.es/ontologies/general#",
        afi _"http://www.afi.es/WSDiscovery/Description#" }

webService _"http://www.afi.es/services/BookIntAirFlight1"

  importsOntology { _"http://www.afi.es/ontologies/geopolitical/locations",
    _"http://www.afi.es/ontologies/general/ETourismActions",
    _"http://www.afi.es/ontologies/finance/common",
    _"http://www.afi.es/ontologies/general",
    _"http://www.afi.es/ontologies/eTourism/flightsExample" }

   capability BookIntAirFlight1Capability
    nonFunctionalProperties
      afi#wsdlDescription hasValue "http://www.afi.es/eTourism/BookIntAirFlight1.asmx?wsdl"
      dc#description hasValue "Booking of flights operated by IntAir"
      dc#language hasValue "en-GB"
      afi#category hasValue "http://www.afi.es/Taxonomy1#FlightBooking"
    endNonFunctionalProperties

    sharedVariables {?f, ?p, ?cc}

    postcondition
      nonFunctionalProperties
        afi#descriptionType hasValue "setBasedCapability"
        afi#intention hasValue "some"
      endNonFunctionalProperties
      definedBy
        ?x memberOf _"http://www.afi.es/services/BookIntAirFlight1" equivalent
        ?x memberOf action#FlightBooking and
            exists ?s(?x[action#ofItem hasValue ?s]) and
            forall ?s(
              ?x[action#ofItem hasValue ?s] implies (?s memberOf flights#FlightSeat and
                  exists ?f(?s[flights#onFlight hasValue ?f]) and
                  forall ?f(
                    ?s[flights#onFlight hasValue ?f] implies ?f[flights#operatedBy hasValue flights#IntAir ]))) and
            exists ?cc(?x[action#withPaymentMethod hasValue ?cc]) and
            forall ?cc(
              ?x[action#withPaymentMethod hasValue ?cc] implies ?cc memberOf fc#CreditCard).

    precondition
      nonFunctionalProperties
        afi#descriptionType hasValue "LPInfoPreconditions"
      endNonFunctionalProperties
      definedBy
        ?f memberOf flights#Flight and ?f[flights#operatedBy hasValue flights#IntAir ] and
        ?cc memberOf fc#CreditCard and ?cc[fc#hasNumber hasValue ?n]
            and ?cc[fc#hasHolder hasValue ?h] and ?cc[fc#hasExpiryDate hasValue ?e] and
        ?p memberOf g#Person and ?p[g#hasId hasValue ?id] and ?p[g#hasName hasValue ?name].
```

```
postcondition
  nonFunctionalProperties
    afi#postconditionType hasValue "inputDependentEffects"
    afi#intention hasValue "some"
  endNonFunctionalProperties
  definedBy
    ?x memberOf _"http://www.afi.es/services/BookIntAirFlight1" equivalent
    ?x memberOf action#FlightBooking and
        exists ?s(?x[action#ofItem hasValue ?s]) and
        forall ?s(
            ?x[action#ofItem hasValue ?s] implies (?s memberOf flights#FlightSeat and
                    ?s[ flights#onFlight hasValue ?f] and ?s[ flights#forPerson hasValue ?p])) and
        ?x[action#withPaymentMethod hasValue ?cc].
```

Listing 6.4: WSMO description of a service offering booking of flights operated by airline IntAir

Applicable filters. We have already mentioned the availability of a wide variety of mature techniques and tools for text matching. While they might differ in the precision and recall levels offered, as well as on their response times, they are expected to offer matches with limited precision due to the inherent ambiguity of natural language and to the possible lack of precision of the textual descriptions given by users. However, these filters are expected to offer low response times, as shown by current database systems supporting textual matching and by current Web search engines. In the next Chapter, we will discuss what particular type of filter we apply to textual descriptions in the SETA platform.

6.2.1.3 Categorization

Textual descriptions are easy to provide by consumers and they allow great freedom: users can textually describe in any level of detail any aspect of a service value. However, the accuracy of such descriptions might be often limited and, furthermore, the precision of matching arbitrarily complex textual descriptions is limited itself. For these reasons, we have incorporated to our platform the categorization of services as it is a type of description which, while being convenient for average users, leaves less freedom to such users, supports more accurate matching mechanisms, and can serve to articulate mechanisms to support service providers for describing the value of the services they offer.

Taxonomies of categories are widely used in different settings. Examples of systems making use of this type of taxonomies are knowledge management systems, B2C sites such as Amazon[9], eBay[10], or Yahoo[11], and widely used information sources such as Yellow Pages. By browsing taxonomies and selecting particular categories, users can unambiguously and in a controlled manner

[9] http://www.amazon.com/
[10] http://www.ebay.com/
[11] http://www.yahoo.com/

narrow down their searches or contributions. In general, taxonomies of categories, if properly defined, are easy to understand by and convenient for average users, as they offer a close set of choices thus guiding users through the huge number of contents or services a given system might offer.

In this context, we find the categorization of services an interesting type of non-formal description of their value. If taxonomies of categories reflecting the capability of services are defined, these can be used to categorize available services in terms of the value they offer, to guide service providers in describing their services by first selecting general, predefined categories of services and, as we will see in Section 6.3, to link different types of descriptions referring to services which offer a similar value. In fact, the categorization of services is possible in the UDDI model and in existing UDDI repositories, either by using standardized taxonomies such as the United Nations Standard Product and Services Classification (UNSPSC)[12] or custom taxonomies of categories.

Modelling style and aspects of the service captured. We make use of taxonomies of categories which capture general capabilities different services can enable access to, i.e., we make use of categories which capture the value different services might offer, independently of the specifics of how such value is offered by particular services. Service categorization can be seen, thus, as the association of a service to a general, pre-defined capability this and possibly other services can offer e.g. flight booking.

Figure 6.1 shows some (partial) examples of taxonomies of categories which can be used to categorize services. As we can observe in the figure, there can be alternative taxonomies for categorizing services, but all of them are expected to contain categories which can be mapped to a certain service capability. For example, under the *Travel* category, the Yahoo! taxonomy contains more specific nodes for flight search, hotel search, etc., which can be used to capture the capability of a service.

While taxonomies of categories can in principle be arbitrarily complex, this complexity must be kept in levels that are compatible with simplicity of use and reusability. In general, we will assume that categories in taxonomies correspond to relatively general capabilities, i.e., categories in taxonomies have to keep a balance between accuracy and ease of use and they must be applicable to a wide set of services. For example, the UNSPSC taxonomy is, if considered in its entirety, too complex to be usable by an average user. Furthermore, we do not expect a taxonomy to define a too-specific category such as "Booking of flights in business class between Madrid and Rome operated by IntAir", as such a category is most likely only applicable to a single service and, if this level of detail is allowed, the size of taxonomies will hamper their use. More general categories such as "Flight booking" or even "Low-cost flight booking" are the types of categories expected to be defined by taxonomies.

[12] http://www.unspsc.org/

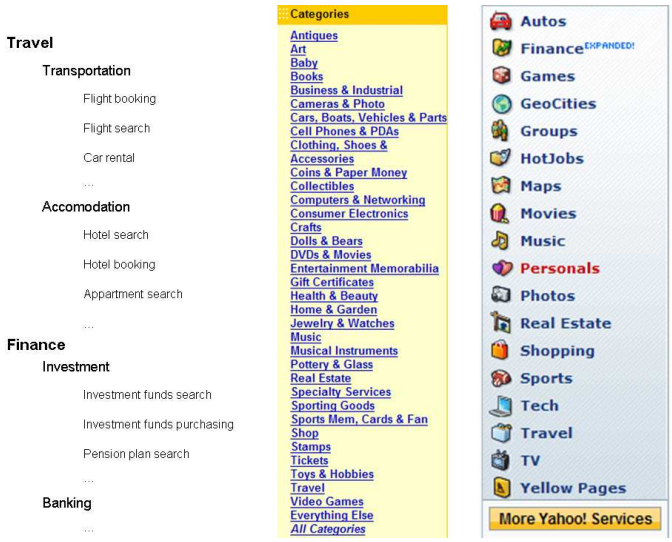

Figure 6.1: From left to right: 1. a custom defined taxonomy, 2. first level of the eBay taxonomy, and 3. first level of the Yahoo! taxonomy.

Finally, we assume that services are categorized using the most specific categories which fit the actual service capability. For example, if a service offers the booking of seats on flights operated by low-cost airlines, it will be categorized under a *low-cost flight booking* category, and not under a *flight booking* category, if both exist.

Summarizing, the categorization of a service is expected to capture the general type of capability offered by the service, probably without completely fitting the exact capability such service offers but providing a coarse-grained view of it.

Target user profile. This type of description, due to its ease of use, is well suited for any type of user. However, and like textual descriptions, the categorization of services is expected to be used especially by business users or users having a good understanding of the value of the service, without requiring technical skills.

Encoding in WSMO. We encode the categorization of a service using a *category* non-functional property in the WSMO capability element, as shown in Listings 6.2 and 6.3. If more than one value is given for this property, it means that the service belongs to *all* the categories given, i.e., we interpret this case as a logical conjunction of categories.

A category is identified by a URI. As we can see in Listings 6.2 and 6.3, the URIs used

are built by appending, with a # symbol, the id of the category to the URI of the taxonomy the category belongs to. Therefore, all taxonomies of categories must be referrable by a URI, and all categories in a taxonomy must have a unique identifier.

Applicable filters. The filters applicable to this type of description are expected to offer low response times and return results as precise as the granularity of the categories defined by available taxonomies allow. These will in general be simple filters which, given one or more categories, find services categorized under such categories. Optionally, filters can also exploit the structure of the taxonomy and consider parent-child relations for matching services given a set of categories. In the next Chapter, the filter implemented in the SETA platform for service categories will be presented.

In this Section, we have presented three alternative types of non-formal descriptions of the value of services. These types of descriptions are close to the types of descriptions users are currently familiar with and, therefore, we can achieve a smooth transition from current practices to more advanced descriptions and achieve certain backwards compatibility with widely spread systems and techniques such as web service registries, text matching techniques, or taxonomies of categories. Furthermore, and as we will see in Section 6.3, non-formal descriptions serve as an entry point to users for the provision of formal descriptions, as they can serve to suggest to users some pre-defined, formal descriptions related to their non-formal descriptions.

6.2.2 Semantic descriptions

While syntactic or non-formal descriptions provide us with features such as ease of use and efficient matching of descriptions, their lack of formal semantics does not allow for automated reasoning, which can improve the precision of search results. In this Section, we introduce semantic or formal descriptions, which will be exploitable by filters which, based on standard inference mechanisms, can help to locate results with a considerably high precision.

6.2.2.1 Ontologies

Formal descriptions of the value of services will refer to a set $\mathcal{O}$ of ontologies which provide the domain vocabulary such descriptions refer to. Furthermore, ontologies are themselves equipped with formal semantics and their shared nature can ensure usability of descriptions across systems and organizations[13]. Two main types of ontologies can be distinguished in our approach, namely: (application) domain ontologies, and ontologies of actions. They are explained in the following.

[13] To what extent the ontologies used are shared will of course determine whether they can be used across particular systems or organizations.

Domain ontologies. Services offer some value in a given application domain e.g. finance, eTourism, or eGovernment and, more particularly, they might be limited to a given sub-domain e.g. investment funds, transport means, or local eGovernment. These application domains and sub-domains will be modelled by one or more ontologies, and formal descriptions of the value of services will refer to them.

In Listing 6.5, a very simple eTourism ontology is described using WSML human readable syntax [de Bruijn et al., 2005e]. This ontology only defines some simple eTourism concepts for illustrative purposes and it imports, among others, an ontology of geographic locations shown in Listing 6.6. The other ontologies imported are not shown for simplicity. A simple ontology of investment funds is given in Listing 6.7, which will also be used for illustrative purposes[14].

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"

namespace {
        _"http://www.afi.es/ontologies/eTourism/flightsExample#",
        dc      _"http://purl.org/dc/elements/1.1#",
        foaf    _"http://xmlns.com/foaf/0.1/",
        wsml    _"http://www.wsmo.org/wsml/wsml-syntax#",
        loc     _"http://www.afi.es/ontologies/geopolitical/locations#",
        fc      _"http://www.afi.es/ontologies/finance/common#",
        g       _"http://www.afi.es/ontologies/general#" }

ontology _"http://www.afi.es/ontologies/eTourism/flightsExample"
        nfp
                dc#title hasValue "Simple ontology of eTourism"
                dc#subject hasValue {"eTourism", "flights"}
                dc#description hasValue "Simple eTourism ontology for illustrating the formal description of services"
                dc#contributor hasValue _"http://nets.ii.uam.es/~rlara/foaf.rdf"
                dc#date hasValue _date(2007,01,29)
                dc#format hasValue "text/html"
                dc#language hasValue "en-GB"
        endnfp

        importsOntology { _"http://www.afi.es/ontologies/geopolitical/locations",
                _"http://xmlns.com/foaf/0.1",
                _"http://www.afi.es/ontologies/finance/common",
                _"http://www.afi.es/ontologies/general" }

        concept Flight
                hasNumber ofType _string
                hasOrigin impliesType loc#City
                hasDestination impliesType loc#City
                hasDate ofType _date
                operatedBy impliesType Airline

        concept FlightSeat
                hasNumber ofType _string
```

[14] A much more complete ontology of investment funds has been defined in [Lara et al., 2006a] as part of our work in modelling this domain and comparing the use of XBRL and formal ontologies in finance.

```
            onFlight impliesType Flight
            onDate ofType _date
            price impliesType fc#Price
            forPerson impliesType g#Person

      concept ETourismOperator
            hasName ofType _string

      concept Carrier subConceptOf ETourismOperator

      concept Airline subConceptOf Carrier
            fromCountry impliesType loc#Country

      concept LowCostAirline subConceptOf Airline

      concept RegularAirline subConceptOf Airline

      instance IntAir memberOf RegularAirline
            fromCountry hasValue loc#Spain

      instance Vueling memberOf LowCostAirline
            fromCountry hasValue loc#Spain

      instance EasyJet memberOf LowCostAirline
            fromCountry hasValue loc#UK

      ...
```

Listing 6.5: Excerpt of a simple eTourism ontology

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"

namespace {
      _"http://www.afi.es/ontologies/geopolitical/locations#",
      dc      _"http://purl.org/dc/elements/1.1#",
      foaf    _"http://xmlns.com/foaf/0.1/",
      wsml _"http://www.wsmo.org/wsml/wsml-syntax#"}

ontology _"http://www.afi.es/ontologies/geopolitical/locations"
      nfp
            dc#title hasValue "Simple ontology of locations"
            dc#subject hasValue {"geopolitical", "locations"}
            dc#description hasValue "Simple locations ontology for illustrating the formal description of services"
            dc#contributor hasValue _"http://nets.ii.uam.es/~rlara/foaf.rdf"
            dc#date hasValue _date(2007,01,29)
            dc#format hasValue "text/html"
            dc#language hasValue "en-GB"
      endnfp

      importsOntology _"http://xmlns.com/foaf/0.1"

      concept Location
            hasName ofType _string
```

```
concept Continent subConceptOf Location

concept Country subConceptOf Location
        inContinent impliesType Continent

concept City subConceptOf Location
        inCountry impliesType Country

instance Europe memberOf Continent
        hasName hasValue "Europe"

instance America memberOf Continent
        hasName hasValue "America"

instance Africa memberOf Continent
        hasName hasValue "Africa"

instance Asia memberOf Continent
        hasName hasValue "Asia"

instance Antarctica memberOf Continent
        hasName hasValue "Antarctica"

instance Australia memberOf Continent
        hasName hasValue "Australia"

instance Spain memberOf Country
        hasName hasValue "Spain"
        inContinent hasValue Europe

instance UK memberOf Country
        hasName hasValue "United Kingdom"
        inContinent hasValue Europe

instance Madrid memberOf City
        hasName hasValue "Madrid"
        inCountry hasValue Spain

instance London memberOf City
        hasName hasValue "London"
        inCountry hasValue UK

...
```

Listing 6.6: Excerpt of a simple ontology of locations

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"

namespace {
        _"http://www.afi.es/ontologies/finance/investmentFunds#",
        dc      _"http://purl.org/dc/elements/1.1#",
```

```
        foaf _"http://xmlns.com/foaf/0.1/",
        loc _"http://www.afi.es/ontologies/geopolitical/locations",
        wsml _"http://www.wsmo.org/wsml/wsml-syntax#"}

ontology _"http://www.afi.es/ontologies/finance/investmentFunds"
    nfp
        dc#title hasValue "Simple ontology of investment funds"
        dc#subject hasValue {"finance", "investment funds"}
        dc#description hasValue "Simple ontology of investment funds for illustrating the formal description of
            services"
        dc#contributor hasValue _"http://nets.ii.uam.es/~rlara/foaf.rdf"
        dc#date hasValue _date(2007,01,29)
        dc#format hasValue "text/html"
        dc#language hasValue "en-GB"
    endnfp

    importsOntology {_"http://www.afi.es/ontologies/geopolitical/locations",
        _"http://xmlns.com/foaf/0.1"}

    concept InvestmentFund
        hasName ofType _string
        commercializedBy impliesType FinancialEntity
        managedBy impliesType ManagementEntity
        commercializedIn impliesType loc#Country
        hasCategory impliesType FundsCategory

    concept FinancialEntity
        hasName ofType _string
        hasLegalName ofType _string
        fromCountry impliesType loc#Country

    concept ManagementEntity subconceptOf FinancialEntity
        hasName ofType _string
        hasLegalName ofType _string
        fromCountry impliesType loc#Country

    concept FinancialMarketSupervisor subconceptOf FinancialEntity
        hasName ofType _string
        fromCountry impliesType loc#Country

    concept FundsCategory
        hasName ofType _string
        definedBy impliesType FinancialEntity

    instance CNMV memberOf FinancialMarketSupervisor
        hasName hasValue "CNMV"
        fromCountry hasValue loc#Spain

    instance FIAMM-CNMV memberOf FundsCategory
        hasName hasValue "FIAMM-CNMV"
        definedBy hasValue CNMV
```

...

Listing 6.7: Simple ontology of investment funds

Ontologies of actions. In Chapter 3, Section 3.3, we provided a formal characterization of services and goals with the purpose of capturing the nature of these artifacts. In this formal characterization, domain ontologies were used and both services and goals were defined as transactions over a real world state, a consumer knowledge state, or both. These transactions were defined as transaction formulas in the framework provided by Transaction Logic, which enabled the definition of services in terms of elementary transitions or updates, as well as the definition of arbitrarily complex transaction formulas modelling actions such as the booking of seats on a flight. In fact, a capability was captured by a transaction formula or action whose performance resulted in changes to the current state. The definition of this transaction formula can be in general given in terms of more simple transaction formulas, which can be in turn defined in terms of other transaction formulas until we reach the level of elementary transitions or updates.

While the definition of transaction formulas capturing service capabilities and/or functionalities is feasible, carrying out this task from scratch is rather cumbersome even for a relatively skilled user. Furthermore, and although transaction logic is a valuable framework for modelling services and goals, there is no sufficient reasoning infrastructure for conveniently reasoning over $\mathcal{TR}$ descriptions. For these reasons, we use the formal characterization of services and goals based on transaction logic given in Chapter 3 exclusively as a solid basis for understanding the nature of services and goals, but we pursue an additional abstraction layer over this formalization which makes descriptions usable and practically exploitable using existing reasoning infrastructure.

A step towards making descriptions usable and abstracting from the formalization in Chapter 3 is the introduction of ontologies of actions, i.e., of reusable terms which refer to common transaction formulas and which can be used in the description of services and goals to, with relative simplicity, express what value a service offers or what value is requested, respectively. In this way, usability is increased as the description burden is reduced. A mapping can be established if required between the actions defined in these ontologies and the definition of the transaction formula providing the accurate meaning of the action. However, transaction formulas will not be directly used in our platform, as illustrated by the example below.

Example 6.1 In Example 3.16, the capability of booking seats on flights between European cities is given by a transaction formula which defined what actions are associated to the capability, and possibly to what parameters (if any) they could be applied. This transaction formula was split into a transaction formula $\Upsilon'(x_1, x_2, x_3, x_4, y_1, y_2)$, which produced some changes in the global state, and a transaction formula $\Upsilon'^c(x_1, x_2, x_3, x_4, y_1, y_2)$, which produced some changes to the consumer knowledge

state. The former made reference to a generic action (transaction formula) $booking(x_1, x_2, x_3, x_4, y_1)$, and some constraints were placed over the values of variables x_1, x_2, x_3, x_4 to which the action could be applied. The latter defined what formulas were inserted into the consumer knowledge base.

The action of booking something is a common action which can be reused across capabilities and whose intuitive meaning is understood by users. Therefore, the action *booking* is a good candidate for being defined in an ontology of actions. Furthermore, the action of providing knowledge to a consumer, i.e., of providing some information is also common and can be incorporated to ontologies of actions.

□

Listing 6.8 presents a simple ontology of general actions or transactions which will be used for illustrating the different types of formal descriptions supported by the SETA platform. In this ontology, an action *InfoProvision* is defined representing the provision of information, i.e., a transaction which has an effect on consumer knowledge. Other actions, sub-concepts of *RealWorldEffect*, correspond to actions having an impact on the real world state., such as the action *Booking*.

Multiple ontologies of actions, and ontologies of refined actions applicable to a given domain might be defined. For example, Listing 6.9 shows a brief excerpt of a simple ontology of eTourism actions which refine some of the actions defined in the general ontology of actions.

In general, ontologies of actions define concepts which are an abstraction of commonly used transactions such as booking, purchasing, etc. These actions are intended to be reusable across capabilities, and the properties they define model those aspects of the effects obtained from performing the action which serve to characterize the value offered by the capability. Such properties roughly correspond to the output values of the transaction formulas introduced in Chapter 3. For example, action $FlightBooking$ in Listing 6.9 defines two properties $ofItem$ and $withPaymentMethod$, and $ofItem$ can only be a flight seat. Particular capabilities can refine this definition and constraint the kind of flight seat which can be booked e.g. on flights operated by only some airlines or with certain restrictions on the itinerary. Similarly, the type of payment method used could be restricted by particular capabilities offering the booking of flight seats.

In a nutshell, ontologies of actions provide an abstracted view of commonly used actions which correspond to transactions having an effect on the real world state or on consumer knowledge, and which are usable to concisely, and without requiring a full formalization of state change, characterize the value associated to capabilities and services. They provide a vocabulary of actions with some value that can be used by service consumers and providers to describe their needs and offers, respectively.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"

namespace {
```

```
        _"http://www.afi.es/ontologies/eTourism/ActionsExample#",
        dc          _"http://purl.org/dc/elements/1.1#",
        foaf  _"http://xmlns.com/foaf/0.1/",
        wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
        fc  _"http://www.afi.es/ontologies/finance/common"}

ontology _"http://www.afi.es/ontologies/general/actions"
        nfp
                dc#title hasValue "Simple ontology of actions"
                dc#subject hasValue "actions"
                dc#description hasValue "Simple actions ontology for illustrating the formal description of services"
                dc#contributor hasValue _"http://nets.ii.uam.es/~rlara/foaf.rdf"
                dc#date hasValue _date(2007,01,29)
                dc#format hasValue "text/html"
                dc#language hasValue "en-GB"
        endnfp

        importsOntology {_"http://xmlns.com/foaf/0.1",
                         _"http://www.afi.es/ontologies/finance/common"}

        concept InfoProvision

        concept RealWorldEffect

        concept Booking subConceptOf RealWorldEffect

        concept Rental subConceptOf RealWorldEffect

        concept Purchase subConceptOf RealWorldEffect

        concept Charge subConceptOf RealWorldEffect

        ...
```

Listing 6.8: Excerpt of a simple ontology of actions

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"

namespace {
        _"http://www.afi.es/ontologies/eTourism/ETourismActionsExample#",
        dc          _"http://purl.org/dc/elements/1.1#",
        foaf  _"http://xmlns.com/foaf/0.1/",
        wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
        fc  _"http://www.afi.es/ontologies/finance/common",
        ac  _"http://www.afi.es/ontologies/general/actions",
        pe  _"http://www.afi.es/ontologies/general/persons"}

ontology _"http://www.afi.es/ontologies/general/ETourismActions"
        nfp
                dc#title hasValue "Simple ontology of eTourism actions"
                dc#subject hasValue {"eTourism", "actions"}
                dc#description hasValue "Simple eTourism actions ontology for illustrating the formal description of services"
                dc#contributor hasValue _"http://nets.ii.uam.es/~rlara/foaf.rdf"
```

```
        dc#date hasValue _date(2007,01,29)
        dc#format hasValue "text/html"
        dc#language hasValue "en-GB"
endnfp

importsOntology {_"http://xmlns.com/foaf/0.1",
                 _"http://www.afi.es/ontologies/finance/common",
                 _"http://www.afi.es/ontologies/general/actions"}

concept FlightBooking subConceptOf ac#Booking
  ofItem impliesType FlightSeat
  withPaymentMethod impliesType fc#PaymentMethod

concept CarRental subConceptOf ac#Rental

...
```

Listing 6.9: Excerpt of a simple ontology of eTourism specific actions

Logical language. Both domain and action ontologies must be defined using a given logical formalism, with a given semantics and expressivity. The choice of the logical formalism is highly relevant, as it will condition what kind of formal descriptions of the value of services can use these ontologies.

In Chapter 2, Section 2.2.5, we introduced the most salient languages proposed for semantic modelling. OWL is the W3C recommendation for an ontology language, and the Lite and DL species of OWL are probably the most widely used languages. They are given a first-order semantics; in particular, the Lite and DL species are equivalent to certain Description Logics, as summarized in Chapter 2. However, using first-order logic semantics usually implies making the Open World Assumption (OWA), and restrictions are used instead of constraints, which might be counterintuitive for users and inconvenient in certain applications (see [de Bruijn et al., 2005f]). Furthermore, alternative types of descriptions and matching mechanisms might require and exploit different semantics, such as First-Order and Logic Programming semantics, as we will see in the next Chapter. For these reasons, it is desirable to make the minimum possible commitment to a particular formalism when choosing the semantics of domain and action ontologies in order to enable the usability of the same ontologies by different descriptions with different semantics and properties. Along this line, we find the use of WSML-Core [de Bruijn et al., 2005e] the most convenient choice, as it corresponds with the intersection of Description Logic and Horn Logic (without function symbols and without equality), based on Description Logic Programs (DLP) [Grosof et al., 2003] and extended with datatype support in order to be useful in practical applications; as discussed in Chapter 2, Section 2.2.5.3, in this fragment ground entailments coincide both under LP and DL semantics, which makes WSML-Core a basic interoperability layer.

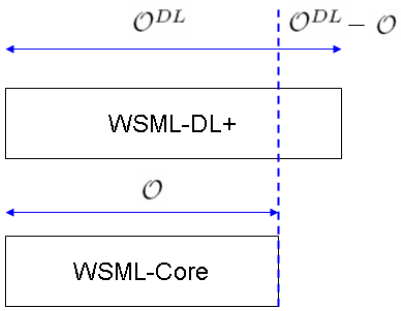

Figure 6.2: Expressivity allowed for domain ontologies

Therefore, by using WSML-Core we can make use of the same ontologies in descriptions of services both with DL semantics and LP semantics and, if required, we can extend these ontologies in either direction; as stated in [de Bruijn et al., 2005e], it is possible that definitions in WSML-Core are extended with definitions in a more expressive language such as WSML-DL (with first-order semantics) or WSML-Flight (with logic programming semantics).

The example ontologies discussed so far are WSML-Core ontologies. However, in Listing 6.10, the example ontology of locations given is extended in the direction of first-order logic to state that: a) members of the concept *Country* must have a value, instance of the concept *Continent*, for the property *inContinent* (*CountryHasAContinent* axiom), b) members of the concept *City* must have a value, instance of the concept *Country*, for the property *inCountry* (*CityHasACountry* axiom), c) the concept *City* is disjoint from the concept *Country* (*DisjointnessCountryCity* axiom), and d) concepts *Country* and *Continent* are disjoint (*DisjointnessCountryContinent* axiom). This extension will only be consistently usable in descriptions of services with first-order semantics, as all these axioms are beyond the intersection of first-order and horn logic.

In general, certain types of descriptions of services and goals we will use might require domain ontologies to formalize aspects which are beyond the expressivity of WSML-Core and which can be only captured if extensions of WSML-Core in the direction of Description Logics are used, such as disjunction axioms or certain restrictions on concept roles. In this setting, we will consider a set $\mathcal{O}$ of domain ontologies which contain that part of the domain vocabulary expressible in WSML-Core, and a set $\mathcal{O}^{DL}$ of domain ontologies which are an extension of the basic set $\mathcal{O}$ in the direction of Description Logics and for which the allowed expressivity is that of WSML-DL+, i.e., the $\mathcal{SHOIN}(\mathcal{D})$ Description Logic (see Chapter 2, Section 2.2.5.3). This is illustrated in Figure 6.2, where $\mathcal{O}^{DL}$ - $\mathcal{O}$ denotes that part of domain ontologies expressible in WSML-DL+ but not in WSML-Core.

wsmlVariant _"http://www.wsmo.org/wsml/wsml−syntax/wsml−dl"

```
namespace {
        _"http://www.afi.es/ontologies/ geopolitical /locations#",
        dc    _"http://purl.org/dc/elements/1.1#",
        foaf  _"http://xmlns.com/foaf/0.1/",
        wsml _"http://www.wsmo.org/wsml/wsml-syntax#"}

ontology _"http://www.afi.es/ontologies/ geopolitical /locationsExtended"
        nfp
                dc#title hasValue "Extended ontology of locations"
                dc#subject hasValue {" geopolitical", " locations"}
                dc#description hasValue "Extended locations ontology in the direction of first order logic"
                dc#contributor hasValue _"http://nets.ii.uam.es/~rlara/foaf.rdf"
                dc#date hasValue _date(2007,01,29)
                dc#format hasValue "text/html"
                dc#language hasValue "en-GB"
        endnfp

        importsOntology {_"http://xmlns.com/foaf/0.1", _"http://www.afi.es/ontologies/ geopolitical /locations"}

        axiom CountryHasAContinent
                definedBy
                    ?x memberOf Country implies
                        exists ?y (?x[inContinent hasValue ?y]) and
                        forall ?y (?x[inContinent hasValue ?y] implies ?y memberOf Continent).

        axiom CityHasACountry
                definedBy
                    ?x memberOf City implies
                        exists ?y (?x[inCountry hasValue ?y]) and
                        forall ?y (?x[inCountry hasValue ?y] implies ?y memberOf Country).

        axiom DisjointnessCountryCity
                definedBy
                        ?x memberOf Country implies neg ?x memberOf City.

        axiom DisjointnessCountryContinent
                definedBy
                        ?x memberOf Country implies neg ?x memberOf Continent.
```

Listing 6.10: Simple extension of the ontology of locations in the direction of first-order logic

6.2.2.2 Set-based modelling of service capabilities

In the following, we present the first type of description of the value of services with formal semantics we consider in our platform: the set-based modelling of the capability of services. This type of description is the result of the work we have presented in [Keller et al., 2004a; Lara et al., 2004a; Keller et al., 2005; Keller et al., 2006a] and it is based on modelling the possible effects of accessing a capability as a set, independently of the conditions the service poses to access such capability and of how initial conditions, i.e., conditions that hold before service usage, affect the

particular effects obtained from using the service.

The interest of the set-based modelling of service capabilities can be summarized by the following main points:

1. It enables the evaluation of the suitability of a service in cases where initial conditions cannot be known, as access to the state of the world is limited for the evaluation of real world preconditions and we might not have access to consumer knowledge to evaluate information preconditions.

2. In situations where the prospective consumer is interested in locating a service which enables access to a particular capability, without having particular requirements (at least that he can describe) on initial conditions and on how they affect the results obtained, it enables to skip or to delay the definition and evaluation of these requirements.

3. If the expressivity of the language used to describe the set of possible effects of the service is limited to a certain subset of first-order logic, this type of description enables classification of services in terms of the capability they provide at publication time and, as a consequence, an efficient matching of descriptions at discovery time.

4. Descriptions will be given in terms of pre-defined actions defined in ontologies of actions, for which restrictions on what kind of objects these actions apply to are introduced. This modelling style makes it relatively easy for users with basic skills in knowledge representation to provide this type of description.

5. General capabilities can be pre-defined and used as a starting point for users to describe their service capabilities by selecting appropriate pre-defined capabilities and refining them. Furthermore, these general capabilities can be naturally mapped to categories in taxonomies, which will be an important feature for being able to propose different types of descriptions to users. Details on how this is done will be presented in Section 6.3

Modelling style and aspects of the service captured. The set-based modelling of a service capability aims at formally describing, at an appropriate abstraction level so that descriptions can be relatively easily provided by users and so that an efficient matching of descriptions is possible, what capability a service enables access to regardless of initial conditions and of their influence on the particular effects of using the capability, i.e., we aim at modelling the set of abstract effects that can be achieved by using a service.

For this purpose, we will use the ontologies of actions introduced in the previous section and describe what actions can be performed when the capability is used, and what kind of objects these actions apply to. In particular, the set of abstract effects associated to a capability $\mathcal{C}$ service

serv enables access to will be modelled by a DL concept Eff_{serv} with the following form in DL syntax:

$$\begin{aligned} Eff_{serv} \equiv (Action_1 \sqcap Restrictions_1) \sqcup \\ (Action_2 \sqcap Restrictions_2) \sqcup \\ \ldots \sqcup \\ (Action_n \sqcap Restrictions_n) \end{aligned} \tag{6.1}$$

where $Action_1, \ldots, Action_n$ are actions defined in ontologies of actions and $Restrictions_1, \ldots, Restrictions_n$ are concepts denoting restrictions introduced over the properties of such actions, and which will refer to the set $\mathcal{O}^{DL}$ of domain ontologies. In a nutshell, concept Eff_{serv} defines the set of potential effects which can be achieved by accessing the capability, described in terms of the actions associated to the capability and the restrictions applied over such actions. This is illustrated by the examples below.

Example 6.2 If we consider the service in Listing 6.2, offering the search of investment funds commercialized in the Spanish market, the set of effects offered by its underlying capability can be formalized as[15]:

$$Eff_{FundSearch1} \equiv InfoProvision \sqcap$$
$$\exists ofInfoItem \sqcap \forall ofInfoItem.(InvestmentFund \sqcap commercializedIn.\{Spain\})$$

where $Infoprovision$ is the concept defined by the ontology of actions in Listing 6.8, concept $InvestmentFund$ and property *commecializedIn* are defined by the ontology shown in Listing 6.7, and *Spain* corresponds to the instance defined by the ontology in Listing 6.6.

Intuitively, the formalization above can be read as: *possible effects of using the service are the provision of information about any investment fund commercialized in Spain.* This description includes only one action $InfoProvision$, as there is no real-world effect of accessing this capability but only information effects. The restriction over this action says that the objects about which information will be provided are investment funds commercialized in Spain. If we consider the general form descriptions of capabilities take, given by Formula (6.3), we have that:

$$Action_1 \equiv InfoProvision$$
$$Restriction_1 \equiv \exists ofInfoItem \sqcap \forall ofInfoItem.(InvestmentFund \sqcap commercializedIn.\{Spain\})$$

[15] In the remainder of this document, and for reasons of readability, we will omit namespaces in formalizations, and we will not use full URIs in the names of services, goals and categories. Only descriptions in WSML human-readable syntax will include these details.

□

Example 6.3 If we now consider the service in Listing 6.3, which offers the search of flights operated by low-cost airlines, the set of effects offered by its underlying capability can be formalized as:

$$Eff_{LowCostFlightSearch1} \equiv InfoProvision \sqcap$$
$$\exists ofInfoItem \sqcap \forall ofInfoItem.(Flight \sqcap \exists operatedBy \sqcap \forall operatedBy.LowCostAirline)$$

□

Example 6.4 Now, let us consider the service in Listing 6.4, whose underlying capability is the booking of seats on flights operated by the airline $IntAir$, and payable with credit card. This capability can be formalized as:

$$Eff_{BookIntAirFlight1} \equiv FlightBooking \sqcap$$
$$\exists ofItem \sqcap$$
$$\forall ofItem.(FlightSeat \sqcap \exists onFlight \sqcap \forall onFlight.(Flight \sqcap operatedBy.\{IntAir\}) \sqcap$$
$$\exists withPaymentMethod \sqcap \forall withPaymentMethod.CreditCard$$

Intuitively, the formalization above states that the service has the capability of booking seats on flights operated by $IntAir$, and that such bookings are payable with credit card, i.e., that the set of possible effects of using the service (abstract effects) are the result of booking with credit card any seat on flights operated by $IntAir$. □

In general, we model the abstract effects associated to a capability by concepts which state what action or actions, from the set of pre-defined actions defined by available ontologies, are associated to the capability, and what restrictions apply to the performance of such actions. The disjunction of actions means that all the actions given by the disjunction can be performed, as illustrated by the following example.

Example 6.5 Let us imagine a service which enables access to the capability of both searching flights provided by IntAir and booking such flights. This capability would be described as:

$$Eff_{SearchAndBookIntAirFlight1} \equiv$$
$$(FlightBooking \sqcap$$
$$\exists ofItem \sqcap \forall ofItem.(FlightSeat \sqcap \exists onFlight \sqcap \forall onFlight.(operatedBy.\{IntAir\}) \sqcap$$
$$\exists withPaymentMethod \sqcap \forall withPaymentMethod.CreditCard)$$
$$\sqcup$$
$$(InfoProvision \sqcap$$
$$\exists ofInfoItem \sqcap \forall ofInfoItem.(Flight \sqcap operatedBy.\{IntAir\}))$$

The formalization above can be seen as the union of two sets of abstract effects: the set of effects corresponding to the booking of seats on flight operated by IntAir, and the set of effects corresponding to the provision of information about flights operated by IntAir.

□

This modelling style corresponds to the formal characterization of a capability presented in Chapter 3, where a capability was defined by a possible action or actions, given by transaction formulas, which could be performed, and some restrictions over the parameters of these actions were defined. Notice that the modelling style used does not establish any connection to the input and output variables of the service interface, as a capability is independent of how a particular service enables access to it. However, if compared to the formalization of capabilities presented in Chapter 3, our modelling style introduces variations, namely: i) an abstraction layer is added and actions already defined by appropriate ontologies are used; this is expected to improve usability, and ii) we describe capabilities by a concept so that the resulting description can be viewed as a set, which will enable efficient classification and matching of services based on their capabilities.

In [Keller et al., 2004a; Keller et al., 2005; Keller et al., 2006a], we considered the use of meta-annotations describing the *intention* of the description given, with the purpose of differentiating cases where a service can provide *all* the effects in the set of abstract effects described by the capability, from the cases where a service can provide only *some* of the effects in this set. For example, a service can exist which provides information about any flight operated by low-cost airlines, i.e, which enables full access to the capability in Example 6.3, while another service can exist which enables only partial access to this capability, not being able to provide information about certain low-cost flights. The latter case would correspond to situations in which the description of the exact capability the service enables access to is too complicated to be provided by the user, and a simplified version, not exact, is used.

In this setting, a service capability will be defined not only by a concept Eff_{serv} formalizing the set of effects achievable, but also the intention of this set, denoted I_{serv}, must be given. Therefore,

we model a service capability as a pair $\mathcal{C}_{serv} = (Eff_{serv}, I_{serv})$, where Eff_{serv} is a WSML-DL+ concept defined in terms of the set $\mathcal{O}^{DL}$ of domain ontologies and with the form presented above, and $I_{serv} \in \{some, all\}$, where *some* denotes an *existential intention* (only some of the effects in the set defined by Eff_{serv} are offered), and *all* denotes a *universal intention* (all the effects in the set defined by Eff_{serv} are offered).

Logical language So far, we have not restricted the expressivity allowed for describing the set of effects achievable by accessing a capability. However, in our platform we will restrict this expressivity to WSML-DL+, i.e., to the $\mathcal{SHOIN}(\mathcal{D})$ Description Logic. The reasons behind this choice can be summarized in the following main points:

1. The use of first-order semantics is convenient for the description of the abstract effects of capabilities, as it enables dealing with incomplete descriptions or missing information, as it will be demonstrated by the filters we will introduce in Chapter 7.

2. This fragment of First-Order logic is decidable, while full FOL is undecidable.

3. This fragment is the DL fragment underlying the DL species of OWL, as shown in [Horrocks et al., 2003], and it is very close to WSML-DL, which corresponds in expressivity and semantics to the $\mathcal{SHIQ}(\mathcal{D})$ description logic [de Bruijn et al., 2005e]. Therefore, we can benefit from the research done in the making of these languages, as well as from tools developed for them.

4. There exist reasoners, as presented in Chapter 2, Section 2.2, which can efficiently handle this subset of first-order logic, especially offering efficient subsumption reasoning. This enables the classification of services based on their capabilities at publication time and the efficient matching of services, already classified, at discovery time.

Target user profile. The abstract effects associated to a capability are expected to be formalized, in the way introduced above, by users with minimum skills in knowledge representation. This profile will most likely correspond to technical users but, if appropriate support is provided, we can make this type of description accessible also to business users, who will in most cases have a better view of the business capability offered. The type of support provided by our platform for the set-based modelling of service capabilities will be described in the next Section.

Encoding in WSMO. The set-based modelling of the service capability is encoded as a post-condition of the WSMO capability element. However, and as different types of descriptions will be encoded in our platform as WSMO postconditions, we differentiate this particular type of postcondition by the *setBasedCapability* value of the *descriptionType* non-functional property associated to

it, as shown in Listings 6.2 and 6.3. Notice that in these listings, WSML human readable syntax is used instead of common DL syntax. The concept formalizing the set of abstract effects offered will take the full name of the service it is associated to, i.e., the URI identifying the service. Whether a universal or existential intention is associated to this concept will be indicated by the values *all* or *some*, respectively, of the *intention* non-functional property of the WSMO postcondition.

Applicable filters. The type of filters expected to be applied to this type of description are filters that, based on the set-theoretic relations between the set of effects associated to the capability and the set of effects required by a consumer [Keller et al., 2006a], determine what services enable access to a capability which can provide the set of effects requested. Filters applicable to these descriptions will not take into account how a particular service enables access to the capability described, i.e., what initial conditions must be fulfilled for the service to be used and how they affect what abstract effects from the set described will be realized will not be considered.

These filters are expected to offer a considerably degree of accuracy, of course dependent on the accuracy of the descriptions given. Furthermore, and as they can benefit from off-line classification of services using subsumption reasoning, as we will see in Section 6.3, they are expected to be relatively efficient. For details, see Chapter 7.

6.2.2.3 Description of information preconditions with LP semantics

The second type of formal description of aspects of the value offered by a service focuses on the description of information preconditions, with the purpose of explicitly and formally describing what knowledge the potential consumer of a service must posses and make available to the service in order to access its underlying capability. The type of description of information preconditions we will use, and unlike most existing proposals in the area e.g. [Paolucci et al., 2002; Li and Horrocks, 2003; Colucci et al., 2005], will describe what knowledge the consumer has to provide to the service but under Logic Programming semantics, and it is motivated by the following main considerations:

1. We want this description to be usable to determine, *from the actual information the consumer has available*, whether valid input values can be provided for the execution of the service, and what particular combinations of valid input values can be provided from consumer knowledge (possibly more than one).

2. The evaluation of the availability of knowledge which satisfies information preconditions of candidate services must be performed efficiently, and using existing reasoning infrastructure.

3. In most situations, we do not expect consumers to explicitly describe what information they have available and willing to disclose for achieving a particular goal. Instead, we want to evaluate information preconditions on general consumer knowledge.

The type of description of information preconditions we will present in the following is different from most existing proposals, where the description of information preconditions is given first-order semantics, and where matching mechanisms implicitly expect potential consumers to describe, for each goal, what particular information can be expected by candidate services. In this sense, the type of description introduced by our framework is novel and we believe it complements existing proposals based on description logics and, consequently, on first-order semantics. We will discuss related work in further detail in Chapter 8.

Modelling style and aspects of the service captured. Information preconditions define conditions on the knowledge a potential consumer of the service must possess and disclose to the service for its execution. In particular, what knowledge will be disclosed by the consumer to the service is defined through a binding of certain objects known by the consumer to a set of input variables $i_1, \ldots, i_n$ defined by the service interface (see Chapter 3).

In this setting, and besides requiring some objects to be assigned to input variables, we will formalize what particular conditions these objects must fulfill to be a valid input binding. This formalization, given a service with input variables $i_1, \ldots, i_n$, will take the form of a Logic Programming query with at least n free variables:

$$? - conditions(i_1, \ldots, i_n) \tag{6.2}$$

where predicate $conditions(i_1, \ldots, i_n)$ is a (possibly complex) predicate which captures the conditions values assigned to input variables $i_1, \ldots, i_n$ must fulfill.

Example 6.6 The service enabling access to the capability of providing information about investment funds commercialized in Spain (see Listing 6.2), requires the consumer to provide information about a category of investment funds defined by the CNMV (the Spanish market supervisor), and only information about funds in this category will be provided. This is formalized by the following LP query, where C is an input variable:

$? - fundsCategory(C), definedBy(C, cnmv)$.

Another service might enable access to the same capability, but requiring the consumer to provide information about a management entity and providing information about funds managed by this entity. This would be formalized by the LP query:

$? - managementEntity(M)$.

where M is an input variable.

We can imagine yet another service which enables access to the same capability as the previous services, but requiring not only a management entity to be provided but also information about a user account (user name and password) for the $fundsOnline$ service:

$? - managementEntity(M), userAccount(A), hasUserName(A, U),$
$hasPassword(A, P), forService(A, fundsOnline).$

where both M and A are input variables. U and P do not correspond to input variables, as their values will be part of the information provided to the service through the binding of a user account to input variable A. Their presence means that possible substitutions for them must be known by the consumer, but they will actually be provided as part of a complete user account given through input variable A.

□

Given the examples above, answers to the queries must be found given the knowledge the consumer has available, so that the services defining these preconditions can be used. The LP queries described are intended to be evaluated over consumer knowledge, i.e., substitutions of variables in these queries by objects (constants) known by the consumer are sought. Intuitively, this means the following:

- The potential consumer of a service must have information available which is a possible answer to the query defining information preconditions.
- Substitutions of variables found in consumer knowledge will conform possible input bindings for using the service, i.e., will correspond to possible combinations of information which, if provided to the service, enables its execution.

This type of description on the one hand captures the conditions an input binding $\beta(i_1, \ldots, i_n)$ defined by a service consumer must fulfill, which correspond to the predicate $pre^{inf}_{serv}(\beta(i_1), \ldots, \beta(i_n))$ introduced in Chapter 3, and on the other hand it is usable to automatically determine possible input bindings. If a particular input binding is defined by the consumer, variables $i_1, \ldots, i_n$ in the description of information preconditions can be substituted by the values given by the binding and the query evaluated. Otherwise, the query can be used to automatically determine possible *valid* bindings of objects known by the consumer to input variables defined by the service interface. Of course, consumer knowledge must be available for performing this evaluation; how this knowledge is described will be presented in the next Chapter.

We give now one more example of the description of information preconditions as LP queries:

Example 6.7 Let us consider again the service used in example 6.4, whose underlying capability was the booking of flights operated by IntAir. This service might require the consumer to know what flight has to be booked, a credit card for the payment, and some details of the passenger for which the seat will be booked, which is formalized by the following query:

$?- flight(F), operatedBy(F, IntAir), creditCard(CC), hasNumber(CC, N),$
$hasHolder(CC, H), hasExpiryDate(CC, E), person(P), hasId(P, ID),$
$hasName(P, Name).$

where F, CC and P are input variables. Now, let us imagine the consumer declares he has the following knowledge (besides shared domain knowledge defined by ontologies):

$Flight(myFlight), operatedBy(myFlight, IntAir).$
$CreditCard(myCC1), hasNumber(myCC1, 123), hasHolder(myCC1, me),$
$hasExpiryDate(myCC1, myCCExpiryDate).$
$Person(me), hasId(me, 321), hasName(me, ruben).$

If we evaluate the query over this knowledge, an answer $\beta = \{F = myFlight, CC = myCC1, P = me\}$ will be found, which will constitute a valid input binding for the service.

□

Logical language. The formalization of information preconditions, as presented above, is given by a Logic Programming query. The purpose is to benefit from efficient query answering so that possible input bindings for the service can be automatically found from potentially big consumer knowledge bases. As the purpose of this description is to determine possible input bindings which can be provided by the potential consumer and which fulfill the conditions necessary to be valid input bindings for the service, or checking whether particular input bindings fulfill certain conditions, the Closed-World Assumption made in Logic Programs perfectly fits our purpose; only the information explicitly known by the consumer and accessible for the evaluation of preconditions will be considered, as we will see in Chapter 7.

Regarding the expressivity allowed for the LP queries described, we currently restrict it to WSML-Flight, i.e., to Datalog queries (Datalog rules with an empty head) with inequality and default negation. This choice is made for two main reasons:

1. WSML-Flight queries can consistently refer to the domain and action ontologies introduced before, as WSML-Flight is properly layered on top of WSML-Core [de Bruijn et al., 2005e].
2. There is reasoning infrastructure available which provides very efficient query answering for Datalog with inequality and default negation.

It must be noticed that, in our setting, extending the expressivity allowed from WSML-Flight to WSML-Rule has no effect. This becomes clear if we consider the extensions introduced by WSML-Rule and the type of description we consider: first, WSML-Rule allows unsafe rules, but this extension has no effect on the expressivity allowed for queries (rules with an empty head), which is the only type of rules we use; second, WSML-Rule allows the use of function symbols but, as our descriptions only refer to ontologies restricted to WSML-Core, which cannot define function

symbols, it turns out that there are no function symbols which can be used in the description of information preconditions. Therefore, in our setting there is no difference between using WSML-Flight or WSML-Rule.

Target user profile. We expect this type of description to be provided by technical users without much difficulties. Technical users are familiar with database systems and, therefore, they will not find difficult to provide a query with LP semantics as it is very close, especially in semantics, to the type of descriptions they commonly use for querying relational databases.

Business users are not expected to necessarily deal with this type of description, as it refers to the way the service must be accessed and not to the business value of the service. However, in some cases, this type of descriptions can have a business ingredient. For example, requiring the service consumer to be a registered user or not is a business decision reflected in the description of information preconditions. Still, we do not expect business users to give the complete description of information preconditions, but only to guide it.

Encoding in WSMO. This type of description of the information preconditions of a service will be encoded by a WSMO *precondition*, part of the WSMO capability element, as illustrated in Listings 6.2, 6.3 and 6.4 (where queries are described using WSML human-readable syntax). As other types of descriptions of information preconditions might be added in the future, we will distinguish this particular type of description of information preconditions by the *LPInfoPreconditions* value of the *descriptionType* non-functional property.

There might exist services which accept input bindings fulfilling different conditions. For example, a service might accept the user name and password associated to a valid user account or, alternatively, require a set of data about the (unregistered) user of the service such as name, e-mail address, etc. In these cases, we could allow for the description of alternative information preconditions, encoded by multiple WSMO precondition elements. However, we believe this case actually corresponds to two different services, offered by the same provider, which enable access to the same capability in different ways. This interpretation is coherent with the conceptual model presented in Chapter 3 and much in the line of common practices in the development of web services, which assume a web service has a single operation.

Finally, notice that if a service has no information preconditions, its WSMO capability should still include a *precondition* element but defined by a *true* value, i.e., defined by a query which is always fulfilled. This will serve to differentiate cases where the service has no information preconditions from cases where such information preconditions have not been described.

Applicable filters. The type of filter these descriptions are well suited for are filters which determine whether the consumer has knowledge which can be used to define a valid input binding for the service, and what particular input bindings can be defined.

Due to the LP semantics of this type of description and the existence of reasoners which provide very efficient query answering for the expressivity allowed, applicable filters are expected to be efficient even when knowledge the consumer has available is big in volume. Furthermore, the results of this filtering are expected to be quite accurate, as they can precisely and automatically determine whether the consumer can fulfill the information preconditions of a candidate service.

6.2.2.4 Set-based modelling of input-dependent effects

The last type of formal description of the value of a service we will use aims at capturing, from the set of abstract effects associated to the capability of the service, what subset of effects can be realized for a particular input binding, i.e., this type of description will capture how the provision of a particular input binding restricts the set of effects achievable by using the service. The modelling style used builds upon the set-based modelling of service capabilities presented above, but it introduces the dependency of the set of achievable effects on the values bound to input variables of the service.

Initial conditions include the particular shared state which holds at the moment a service is used, and what knowledge the consumer has of this state and provides to the service via an input binding. However, describing how the effects achievable by using a service depends on initial conditions will be of little use for the location of a service unless the location process has complete knowledge of this state. For this reason, we concentrate on the dependency of effects on the input binding provided by a consumer, as we can much more easily know, and even in some cases automatically determine, what input bindings can be provided by the consumer to the service.

The main reasons behind the choice of the modelling which will be presented in the following are:

1. It captures what effects can be realizable, from the set of abstract effects of the service capability, for a particular input binding. This enables the evaluation of, not only whether the service can provide the effects sought by a consumer, but also of *whether the service can provide these effects given the input bindings the consumer can provide.*

2. The modelling style used can be easily linked to the description of information preconditions so that we can use the formalization of preconditions to determine, given the knowledge the consumer has available, valid input bindings and what set of effects can be realized for such bindings.

3. The restriction in expressivity to a decidable subset of first-order logic, for which reasoning infrastructure exists, will guarantee that we can exploit this type of description for discovery using state-of-the-art systems.

In general, this type of description provides a more fine-grained view of the set of effects which can be provided by a service than the set-based modelling of capabilities, thereby enabling a more accurate filtering of candidate services.

Modelling style and aspects of the service captured. The functionality of a service was defined in Chapter 3 as the relation between how service preconditions are fulfilled and the particular effects, from the set of abstract effects of the service capability, that will be achieved by the usage of the service. In fact, the formal characterization given, based on transaction logic, defined a service (functionality) as a transaction formula which caused, given valid initial conditions, a transition in the consumer knowledge state and in the real-world state; the particular transition caused commonly depends on how initial conditions are fulfilled. However, and while transaction formulas like the ones given in Chapter 3 can accurately capture how transactions depend on initial conditions, this type of description is difficult to provide and difficult to handle, as no Transaction Logic reasoner is currently available and only approximations are possible.

In this setting, we will describe, in a simple and intuitive way, how the set of effects associated to a service capability is restricted when a particular (valid) input binding is given. In particular, we will revise the definition of the concept which was used for describing the abstract effects associated to a capability, yielding a concept of the form:

$$\begin{aligned} InputEff_{serv} \equiv (Action_1 \sqcap Restrictions_1(i_1, \ldots, i_n)) \sqcup \\ (Action_2 \sqcap Restrictions_2(i_1, \ldots, i_n)) \sqcup \\ \ldots \sqcup \\ (Action_m \sqcap Restrictions_m(i_1, \ldots, i_n)) \end{aligned} \tag{6.3}$$

where $i_1, \ldots, i_n$ are input variables of the service, and $Restrictions_i(i_1, \ldots, i_n)$ denote restrictions over the actions associated to the capability which depend on the values assigned to input variables of the service. Therefore, the set of effects modelled by the concept above and, more specifically, the restrictions over the actions associated to the service, are parameterized by the particular values bound to input variables.

Example 6.8 If we consider the service in Listing 6.2, which requires a category of investment funds as an input and provides information about investment funds commercialized in Spain and categorized under the category given, its input-dependent effects would be formalized by:

$$InputEff_{FundSearch1} \equiv InfoProvision \sqcap$$
$$\exists ofInfoItem \sqcap$$
$$\forall ofInfoItem.(InvestmentFund \sqcap commercializedIn.\{Spain\} \sqcap hasCategory.\{c\})$$

where c is the input variable of the service, which will be bound a particular investment funds category. If a particular input binding $\beta = \{c = Category1\}$ is given, variable c in the formalization above will be replaced by its value yielding the concept:

$$InputEff_{FundSearch1,\beta} \equiv InfoProvision \sqcap$$
$$\exists ofInfoItem \sqcap$$
$$\forall ofInfoItem.(InvestmentFund \sqcap commercializedIn.\{Spain\} \sqcap hasCategory.\{Category1\})$$

The formalization above describes the set of effects that can be provided by the service for input binding $\beta = \{c = Category1\}$.

□

Example 6.9 If we now consider the service in Listing 6.4, which requires a flight (input variable f), a credit card (input variable cc) and a person (input variable p) as inputs, and books a seat on the flight given, with the credit card given, and for the person given, its input-dependent effects can be formalized as:

$$InputEff_{BookIntAirFlight1} \equiv FlightBooking \sqcap$$
$$\exists ofItem \sqcap \forall ofItem.(FlightSeat \sqcap onFlight.\{f\} \sqcap forPerson.\{p\}) \sqcap withPaymentMethod.\{cc\}$$

□

In these examples, we can see how the concept defined to describe the set of abstract effects of the service capability is redefined including the input variables of the service ($serv$), yielding a concept $InputEff_{serv}$. These variables, when bound to particular values given by an input binding β, will restrict the global set of effects of the capability to the set of effects achievable for these values. This restricted set is described by the concept $InputEff_{serv,\beta}$ resulting from substituting input variables by their values as defined by β.

Notice that in the formalizations above it is assumed that the values bound to input variables are valid input bindings for the service, i.e., the conditions the values assigned to variables $i_1, \ldots, i_n$ must fulfill are not formalized here again, but they are only formalized by information

preconditions. It is also important to notice that the input variables that appear in the predicate above are the same input variables used in the description of information preconditions, and they will have the same name. However, only input variables that have an effect on the set of effects achievable by using the service will appear in the formalization of input-dependent effects. For example, the service in Example 6.8 might have an input variable u and require that a valid user account is bound to it. While this will be a precondition for the usage of the service, what particular value is bound to this variable will not affect the set of abstract effects achievable and, therefore, it will not appear in the formalization of input-dependent effects.

The intention of the modeler must also be made explicit, i.e., a meta-annotation declaring the (existential or universal) intention of the concept described will be given in the same way it was given for the set-based modelling of the service capability. In this way, providers will indicate whether all the effects in the set of input-dependent effects modelled will be achievable for a particular input binding or only some of them.

Therefore, input-dependent effects will be modelled as a pair $\mathcal{C}'_{serv} = (InputEff_{serv}, I'_{serv})$, where $InputEff_{serv}$ is a WSML-DL+ concept defined in terms of the set $\mathcal{O}^{DL}$ of domain ontologies describing input-dependent effects, and $I'_{serv} \in \{some, all\}$ is the meta-annotation which captures the (existential or universal) intention of such concept.

Logical language. We will use first-order semantics for the description of the restricted set of effects of a service for a particular input binding. In particular, we will require that the formalization of input-dependent effects corresponds in expressivity and semantics to the formalization of a $\mathcal{SHOIN}(\mathcal{D})$ concept after replacing input variables by instances in domain ontologies, i.e., we will restrict the expressivity of this type of descriptions to WSML-DL+ after input variables has been replaced by instances in $\mathcal{O}$.

Target user profile. We expect this type of description to be provided by technical users, as it is related to how the service enables access to a capability, which is more at the service than at the capabiliy level. However, this type of description can also have a business ingredient. For example, restricting the set of effects in different ways depending of, for example, the type of subscription of the user given as an input, is a business decision. For this reason, there is not a sharp distinction between what type of profile is expected to provide this type of description, and in many cases it is expected to result from the cooperation of business and technical users.

Encoding in WSMO. These descriptions will be encoded as a postcondition of the WSMO capability element, and distinguished from the set-based modelling of the service capability by the *inputDependentEffects* value of the *descriptionType* non-functional property associated to it, as

shown in Listings 6.2, 6.3 and 6.3 (where the description is given in WSML human-readable syntax). The name given to the concept formalizing the set of input-dependent effects will be the URI of the service, and the intention of this concept will be encoded by the *intention* non-functional property of the WSMO postcondition.

Finally, the input variables of the service the formalization of input-dependent effects refer to will be declared as shared variables, using the *sharedVariables* element of WSMO, as illustrated in the Listings referenced above. This means, according to the WSMO model [Roman et al., 2005], that variables with the same name appearing in the description of different preconditions and postconditions and declared as shared variables actually correspond to the same variable, and their values will be shared. Therefore, input variables used in both the description of information preconditions and input-dependent effects, and declared as shared variables, are actually the same variables.

Applicable filters. Once input variables in the description of service effects are replaced by particular input values, and based on the set-theoretic relations between the set of effects obtained and the set of effect required by a consumer [Keller et al., 2006a], filters can be applied which determine whether the service, for the input binding given, can provide the set of effects requested.

These filters are expected to be applicable only after possible input bindings have been found, either manually or automatically, and they will offer quite accurate results. However, they are not expected to be very efficient, as classification of descriptions using subsumption reasoning is not possible before particular input bindings have been determined, which will be most likely done at discovery time. Taking into account that classification is an expensive task, this will yield relatively high response times. For details, see Chapter 7.

6.3 Description and publication of services

For services to be located, they must be first described and such descriptions must be made accessible to prospective consumers. We will see in the next Chapter that different types of descriptions of services enable the application of different types of filters and, furthermore, they require different types of descriptions of goals to be provided so that the matching of descriptions is possible. Therefore, for a service to be locatable no matter what kind of description of his goal a prospective consumer can provide, and no matter what particular filter(s) are selected by the consumer, service providers must describe the value of their services in all the possible ways introduced in the previous Section.

While describing a service in all the ways introduced above is desirable, this might require a considerable effort from service providers. In this Section, we will discuss how users are supported by our platform for providing different types of descriptions of their services so that the difficulties

derived from the complexity of descriptions and from the possible lack of appropriate skills of the provider are mitigated.

We will also present in this Section the registry designed (and prototypically implemented) for the publication of service descriptions and taxonomies of categories. The architecture of this registry is depicted in Figure 6.3. The registry is partly based on [Srinivasan et al., 2004], and it is conceived as an extension of a UDDI repository with improved service location capabilities based on the types of descriptions introduced before. Still, we allow for the direct usage of the UDDI API of the UDDI repository [Bellwood et al., 2002] in order to keep backwards compatibility with current service infrastructure and practices. In this way, consumers can choose between locating services directly using the UDDI API, and using the new location interface and the enhanced capabilities added on top of the UDDI repository.

The main components of the registry, corresponding to green boxes in the Figure, are:

- The *taxonomy manager*, which enables defining, storing and managing taxonomies of categories, and accessible via a set of web (WSDL) services depicted by a blue box next to the taxonomy manager in Figure 6.3.
- The *publication manager*, in charge of handling requests for the publication of service descriptions and also accessible via web services.
- The *location manger*, which will receive a consumer goal and retrieve from the registry relevant the descriptions of services deemed relevant for achieving the goal.

Details on the taxonomy and publication managers and on how they are used for the description and publication of services will be given in this Section, while the location manager of the registry will be presented in detail in the next Chapter.

6.3.1 Pre-defined descriptions

The abstract model for the location of services presented in Chapter 5 required support for the provision of other types of descriptions of a service starting from the types of descriptions particular users are comfortable with. In particular, the relation between the different types of descriptions considered by the particular instantiation of the abstract model was proposed as the basis for offering this type of support. In our instantiation, we will exploit the relation among the types of descriptions of services presented in Section 6.2 in order to propose to service providers types of descriptions of their services they have not provided. For this purpose, we will make use of pre-defined service descriptions, corresponding to common capabilities and functionalities, which will link different types of descriptions.

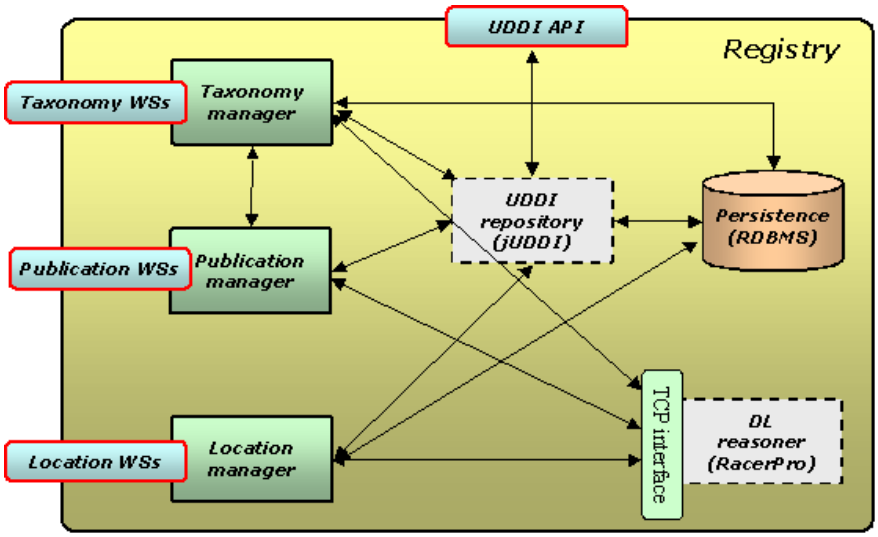

Figure 6.3: Registry architecture

In the following, we will briefly describe the relation among the types of descriptions of services considered in the SETA platform and then describe how pre-defined service descriptions, associated to categories, are built and published to the service registry.

6.3.1.1 Relation among descriptions.

All the types of descriptions introduced in Section 6.2 are related in some way, as they all try to capture the value offered by a service. We will briefly precise in the next paragraphs interesting relations among these types of descriptions:

WSDL - textual. The WSDL description of services can incorporate documentation elements which contain textual descriptions of different aspects of the service. These documentation elements can contain information which is common to the textual description of the value of services introduced in the previous Section; however, it is likely that WSDL documentation elements will have a technical bias as they are expected to be provided by technical users.

Category - textual. Taxonomies of categories are used to provide a coarse-grained view of the capability of services. These categories can be associated a textual description, which will actually be the description, using natural language, of common capabilities different services might enable access to.

Category - set-based modelling of capabilities. Categories in taxonomies represent common, pre-defined capabilities of services. In this setting, capabilities represented by categories can be given a formal meaning by providing a set-based description of them, i.e., categories can be associated the formal description of the abstract set of effects corresponding to the capability they represent. In this way, categories are given a precise meaning, yielding a taxonomy of common, pre-defined, and *formally described* capabilities.

Example 6.10 Let us imagine a category *LowCostFlightBooking* is defined at a given taxonomy. This category can be associated a textual description of the capability it represents: "Booking of seats on flights operated by low-cost airlines".

Furthermore, a formal description of the capability this category represents can be given:

$$\begin{aligned} Eff_{LowCostFlightBooking} \equiv FlightBooking \sqcap \\ \exists ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap \exists onFlight \sqcap \forall onFlight.(Flight \sqcap \exists operatedBy \sqcap \\ \forall operatedBy.LowCostAirline)) \end{aligned}$$

The formula above formalizes the set of abstract effects associated to the category, thereby providing it with a precise meaning.

□

Please note that, while most categories will have associated a formal meaning, some categories are only introduced in order to give an appropriate structure to the taxonomy, such as categories *Travel* or *Transportation* given in the custom taxonomy of Figure 6.1. These categories will not have a formal meaning, as they do not correspond to real capabilities but only to *topics* which serve to organize other categories.

Set-based modelling - input-dependent effects. The set-based modelling of a capability and the description of input-dependent effects of a service are related in that the former describes the complete set of effects that can be potentially obtained by accessing the capability, and the latter describes how this set of effects is parameterized by the values bound to certain input variables.

The following relation must hold between these two types of descriptions for any service *serv* and any valid input binding β assigning values to the input variables of the service:

$$InputEff_{serv,\beta} \sqsubseteq Eff_{serv}$$

The formula above means that the set of effects achievable by executing, with particular input values, a service which enables access to a certain capability, will be a (not necessarily strict) subset of the set of abstract effects associated to such capability.

LP information preconditions - input-dependent effects. The modelling of information preconditions we use will validate and possibly find input bindings for a given service. Such input bindings will be used to replace input variables in the description of input-dependent effects, thereby obtaining the restricted set of effects that can be achieved for such input binding. Therefore, these two types of descriptions will in many cases work together to offer a simplified view of the service functionality.

Category - LP information preconditions + input-dependent effects. The description of information preconditions and of input-dependent effects is particular of a service, while categories refer to general capabilities to which access can be enabled by multiple services. Still, we can associate to a category (and to its represented capability) a *prototypical way of accessing it*, i.e., we can associate to a category prototypical descriptions of information preconditions and input-dependent effects.

Example 6.11 It is usual for the booking of flights that the details of the flight to be booked must be given, as well as the details of the person for which a seat must be booked and a payment method. The effect will generally be the booking of a seat on the flight given, for the person given, and paid with the payment method specified. In this setting, we can associate to the category *LowCostFlightBooking* the following prototypical information preconditions (where F, P and CC are input variables):

$? - flight(F), person(P), paymentMethod(CC)$

and the following prototypical description of input-dependent effects:

$$InputEff_{LowCostFlightBooking} \equiv FlightBooking \sqcap$$
$$\exists ofItem \sqcap \forall ofItem.(FlightSeat \sqcap onFlight.\{f\} \sqcap forPerson.\{p\}) \sqcap withPaymentMethod.\{cc\}$$

□

6.3.1.2 Definition and management of taxonomies of categories

From the relations previously presented, those relations between categories pre-defined in taxonomies and other types of descriptions will be central in our platform for supporting providers for describing their services. The reason is many-fold:

1. Categories are expected to have an appropriate granularity for representing common, pre-defined capabilities providers can reuse for describing their services.

2. Given the relations above, different types of descriptions of categories can be statically associated to them, yielding a set of pre-defined descriptions, of alternative types, of reusable capabilities.

3. Taxonomies of categories are expected to be highly usable by service providers and, thus, they can serve as a good entry point to service providers for obtaining alternative types of descriptions of their services; once a service is assigned to a category, the alternative types of descriptions associated to this category can be also associated to such service.

In general, categories will act as common, pre-defined capabilities providers can use to easily provide a coarse-grained description of their services, and alternative types of descriptions of the capability categories represent will be associated to them. This scheme will be later exploitable for supporting providers for describing their services. For this purpose, taxonomies of categories must be defined, alternative descriptions of their categories appropriately linked, and such taxonomies and their associated descriptions published. In our platform we have included, as part of the registry for the publication of service descriptions, a taxonomy manager for the definition and management of categories, depicted in Figure 6.3. This manager allows for the following main operations, accessible as WSDL web services:

Taxonomy creation. For creating a new taxonomy of categories, a URI which identifies such taxonomy must be given and, optionally, a brief textual description of the taxonomy. Given this information, the taxonomy manager will store in the relational database used for persistency of the registry the taxonomy data. Furthermore, a new UDDI tModel [Bellwood et al., 2002] will be created representing the taxonomy at the UDDI repository of the registry; this tModel will have the value *categorization* for the *uddi-org:types* key [Bellwood et al., 2002]. In the current prototype implementation, the UDDI repository used is jUDDI[16], as it is open source, it integrates well with different relational databases, and it implements the UDDI API.

In Figure 6.4, the creation of a taxonomy using a simple user interface we have implemented for the management of taxonomies, is depicted. This user interface will also support other tasks such as the publication of service descriptions and the location of services, as we will see in this and the following Chapters. After the information of the taxonomy shown in the Figure is given, a request for the creation of the taxonomy will be submitted to the taxonomy manager of the registry using the WSDL service defined for this purpose.

[16] http://www.juddi.org/

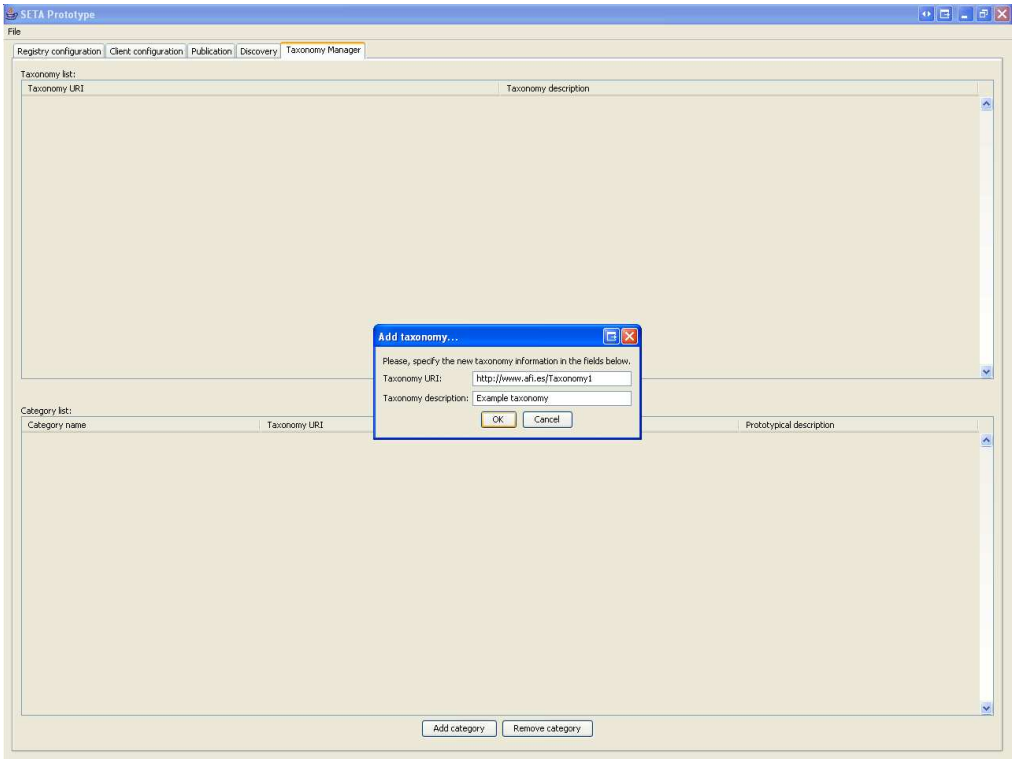

Figure 6.4: Creation of an example taxonomy

Category creation. For the creation of a category, a name for this category must be given, as well as what taxonomy the category belongs. Furthermore, the parent category of the newly created category must be provided, unless it is a root category. For reasons of simplicity and usability of taxonomies, we do not allow a category for having multiple parents.

We have discussed above that a category can be associated a set of alternative descriptions of the capability it represents, as well as a description of prototypical information preconditions and input-dependent effects for accessing such capability. In particular, when a category is created, we can associate to it a description of a prototypical service enabling access to the capability such category represents. The description of such prototypical service will include the textual description of the capability, its set-based modelling, and prototypical information preconditions and input-dependent effects.

In Listing 6.11, the prototypical description of a category *FlightSearch* defined at a taxonomy $http://www.afi.es/Taxonomy1$ is shown. The prototypical service is identified by the full URI of the category, and all types of descriptions considered in our platform but a WSDL description are given.

```
namespace { _"http://www.afi.es/services/eTourism#",
        loc _"http://www.afi.es/ontologies/ geopolitical /locations#",
        action _"http://www.afi.es/ontologies/general/actions#",
        flights _"http://www.afi.es/ontologies/eTourism/flightsExample#",
        dc _"http://purl.org/dc/elements/1.1#",
        g _"http://www.afi.es/ontologies/general#",
        afi _"http://www.afi.es/WSDiscovery/Description#" }

webService _"http://www.afi.es/Taxonomy1#FlightSearch"

  importsOntology { _"http://www.afi.es/ontologies/ geopolitical /locations",
    _"http://www.afi.es/ontologies/general/actions",
    _"http://www.afi.es/ontologies/general",
    _"http://www.afi.es/ontologies/eTourism/flightsExample" }

   capability FlightSearchCapability
    nonFunctionalProperties
      dc#description hasValue "Search of flights"
      dc#language hasValue "en-GB"
      afi#category hasValue "http://www.afi.es/Taxonomy1#FlightSearch"
    endNonFunctionalProperties

    sharedVariables {?co, ?cd, ?d}

    postcondition
      nonFunctionalProperties
        afi#descriptionType hasValue "setBasedCapability"
        afi#intention hasValue "all"
      endNonFunctionalProperties
      definedBy
        ?x memberOf _"http://www.afi.es/Taxonomy1#FlightSearch" equivalent
        ?x memberOf action#InfoProvision and
            exists ?y(?x[action#ofInfoItem hasValue ?y]) and
            forall ?y(
                ?x[action#ofInfoItem hasValue ?y] implies ?y memberOf flights#Flight).

    precondition
      nonFunctionalProperties
        afi#descriptionType hasValue "LPInfoPreconditions"
      endNonFunctionalProperties
      definedBy
        ?co memberOf loc#City and ?cd memberOf loc#City and ?d memberOf date.

    postcondition
      nonFunctionalProperties
        afi#postconditionType hasValue "inputDependentPostcondition"
        afi#intention hasValue "all"
      endNonFunctionalProperties
      definedBy
        ?x memberOf _"http://www.afi.es/Taxonomy1#FlightSearch" equivalent
        ?x memberOf action#InfoProvision and
            exists ?y(?x[action#ofInfoItem hasValue ?y]) and
            forall ?y(
```

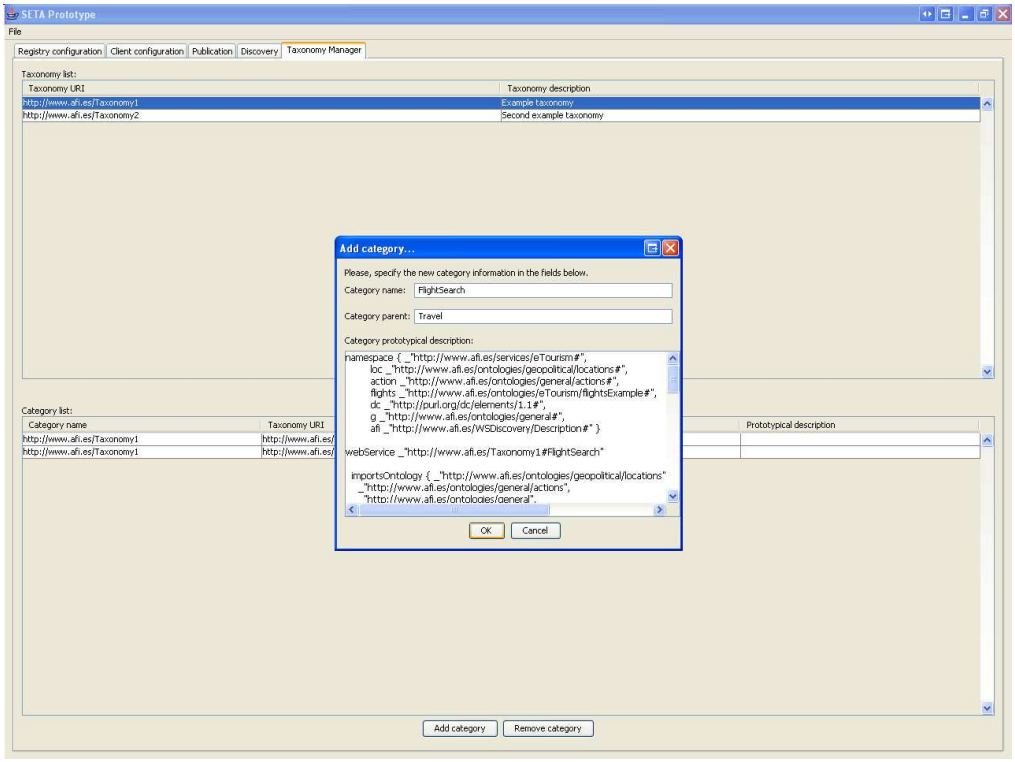

Figure 6.5: Creation of an example category

```
?x[action#ofInfoItem hasValue ?y] implies (?y memberOf flights#Flight[flights#hasOrigin hasValue ?co,
                                                                      flights#hasDestination hasValue ?cd,
                                                                      flights#hasDate hasValue ?d]).
```

Listing 6.11: Description of a prototypical service associated to a FlightSearch category

Categories are expected to be created by skilled users, and then reused by any service provider (or consumer, as we will see). Therefore, categories are expected to always have associated *complete* prototypical descriptions, i.e., all types of descriptions of services discussed in the previous Section will be included in the description of the prototypical service. In Figure 6.5, the creation of the *FlightSearch* category mentioned above is depicted.

When the creation of a category is requested, including in such request the name of the category, the taxonomy it belongs to, its parent category, and the complete description of a prototypical service enabling access to the capability the category represents, as depicted in Figure 6.5, the taxonomy manager will do the following:

1. First, the satisfiability of the concept defining the set of abstract effects of the service capability, encoded by the postcondition with value *setBasedCapability* for the *descriptionType*

non-functional property in Listing 6.11, will be checked. The DL reasoner of the registry, depicted in Figure 6.3, will be used to check the satisfiability of this concept wrt. a *Categories* TBox created when the registry is started. When the registry is started, the set $\mathcal{O}^{DL}$ of domain ontologies formal descriptions can refer to has also been loaded into this TBox and the TBox has been classified.

If the concept is not satisfiable, the creation of the category will fail and the user will have to correct this type of description of the category. Otherwise, we will proceed to the next step.

2. The details of the category will be stored at the relational database of the registry. In particular, we will store the name of the category, its parent category, the taxonomy it belongs to, the complete prototypical description given for the category and, for convenience, we will also store separately each type of description of the prototypical service given. Additionally, we will load in memory the new structure of the taxonomy the category belongs to, i.e., the parent-child relations of the taxonomy; in this way we will be able to explore the taxonomy tree faster at service location time (see the next Chapter).

3. The concept defining the set of abstract effects of the service capability, i.e., the set-based modelling of the capability the category represents, will be published to the *Categories* TBox of the DL reasoner.

4. After the concept is published to the appropriate TBox, this TBox will be classified, i.e., the subsumption relation between the categories of the taxonomy, in terms of their associated formal capabilities, will be computed.

Of course, the WSMO description of the prototypical service associated to the category must be valid. This will be checked using WSMO4J[17], which also enables the parsing of the WSMO description. Otherwise, an error will be returned.

The DL reasoner used is RacerPro [RAC, 2006], even though it does not provide complete and correct reasoning for nominals. Still, it has certain features, such as support for multiple TBoxes, a good efficiency, and especially a rich and well documented interface, which motivates its usage. The interface used to communicate with RacerPro is the TCP interface, which we use through the jRacer Java library[18]. While the DIG interface supported by RacerPro would enable the seamless replacement of RacerPro by other DL reasoner, it does not support certain operations, such as the on-demand classification of the TBox, which we require in our prototype. However, the possible use of other DL reasoners is envisioned as part of our future work (see Chapter 9).

[17] http://wsmo4j.sourceforge.net/
[18] http://www.racer-systems.com/products/download/nativelibraries.phtml

Example 6.12 Let us imagine we create the category *FlightSearch* mentioned above at a taxonomy *http://www.afi.es/Taxonomy1*. This category has associated the prototypical description in Listing 6.11, and its parent category is a *Travel* category.

We will first check the satisfiability of concept *http://www.afi.es/Taxonomy1#FlightSearch* wrt. the *Categories* TBox. The definition of this concept is given by the *setBasedCapability* post-condition of the prototypical service description. Given that this concept is consistent, we will store the category data into the relational database, publish such concept definition to the *Categories* TBox, and then classify this TBox.

Now, let us imagine we create a category *LowCostFlightSearch* at the same taxonomy, defining that its parent category is the *FligtSearch* category created before, and whose set-based modelling of its associated capability is given by:

```
?x memberOf _"http://www.afi.es/Taxonomy1#LowCostFlightSearch" equivalent
?x memberOf InfoProvision and
    exists ?y(?x[ofInfoItem hasValue ?y]) and
    forall ?y(
        ?x[ofInfoItem hasValue ?y] implies (?y memberOf Flight and
        exists ?z(?y[operatedBy hasValue ?z]) and
        forall ?z(
            ?y[operatedBy hasValue ?z] implies ?z memberOf LowCostAirline))).
```

We will create this category, and after checking the satisfiability of the concept above, we will also publish this concept to the *Categories* TBox and classify this TBox. The classification of the TBox will yield that category *LowCostFlightSearch*, according to the set-based modelling of its associated capability and given the example domain ontologies shown at the beginning of this Chapter, where concept *LowCostAirline* was defined as a subconcept of *Airline*, is more specific than category *FlightSearch*, i.e., concept *http://www.afi.es/Taxonomy1#FlightSearch* subsumes concept *http://www.afi.es/Taxonomy1#LowCostFlightSearch*.

□

In Chapter 8, we will discuss in detail the complexity of this process and the experimental times measured. For the moment, we can anticipate that the complexity of creating a category is high, as the TBox of the DL reasoner corresponding the taxonomy the category belongs to must be classified. Still, as this task is done off-line, i.e., without affecting service location times, it is not time-critical. Furthermore, modern DL reasoners are highly optimized, which enables the obtention of results in relatively low times.

Category search. Once taxonomies and their categories are created and available at the registry, we will enable searching for categories and retrieving their details in the following ways:

1. Textual: given a textual description, we will provide categories (with their details) which

match this description, no matter the taxonomy they belong to. In our prototype, we use simple keyword matching over the textual description associated to the category and stored separately in the relational database. In particular, we first remove from the textual description given words considered *noise* words. For this purpose, we use the *stop-list* given by Oracle Text[19] (descriptions in English are always assumed and, thus, the set of noise words defined for the English language is used). After noise words have been removed, we will simply match categories whose textual description contains any of the keywords remaining in the textual description given.

2. Set-based modelling of a capability: given a WSML-DL+ concept defining a set of abstract effects, we will retrieve categories whose associated set-based capability: a) is equivalent to, b) is subsumed by, c) subsumes, or d) intersects (but is neither equivalent to, nor subsumed by, nor subsumes) such concept. For this purpose, we will query the *Categories* TBox of the DL reasoner for concepts corresponding to categories for which these relations with the concept given hold. These categories and their details will be returned, grouped into four sets in terms of the relation between their formalized set of effects and the set of effects given: a) equivalent categories, b) more specific categories, c) more general categories, and d) categories with some effects in common.

   It must be noted, though, that existing DL reasoners, including RacerPro, do not allow for directly querying for *all* concepts which intersect a given concept, but we have to check the satisfiability of the intersection of such concept and each concept published one by one. In this setting, and in order to obtain intersecting concepts without requiring so many queries, we will query RacerPro for concepts which *do not intersect* the concept given, i.e., which are *equivalent to or subsumed by the negated concept*. Intersecting concepts will be obtained by taking all concepts in the TBox corresponding to categories but those which do not intersect, are equivalent to, subsumed by, or subsume the concept given. In other words, we will query for the set of concepts which are equivalent to (denoted $\mathcal{C}_{\equiv}$), subsumed by (denoted $\mathcal{C}_{\sqsubset}$), subsume (denoted $\mathcal{C}_{\sqsupset}$), or have an empty intersection with the concept given (denoted $\mathcal{C}_{\emptyset}$ and obtained as the union of concepts equivalent to -$\mathcal{C}_{not(\equiv)}$- or subsumed by -$\mathcal{C}_{not(\sqsupset)}$- the negated concept, i.e., $\mathcal{C}_{\emptyset} = \mathcal{C}_{not(\equiv)} \cup \mathcal{C}_{not(\sqsupset)}$). The set of categories which have some effects in common with the set given by the input concept, but are neither equivalent, nor more general, nor more specific than such concept, are those categories whose formalization of capability effects is in the set:

$$\mathcal{C}^{\sqcap} \equiv \mathcal{C} - \mathcal{C}_{\equiv} - \mathcal{C}_{\sqsubset} - \mathcal{C}_{\sqsupset} - \mathcal{C}_{\emptyset}$$

[19] http://www.oracle.com/technology/products/text/pdf/9ir2text_features_overview.pdf

where $\mathcal{C}$ denotes all categories part of the taxonomy being checked, and the minus sign denotes set difference.

3. Direct: given a taxonomy URI and a category name, we will return the details of this category.

The retrieval of categories information, either by directly selecting one category or by providing a textual description, is efficient. Extracting the details of a given category only requires executing a simple query against the relational database of the registry. Furthermore, in our prototype implementation the search of categories matching a textual description is reduced to finding categories whose textual description has some keyword in common with the text given, which is also efficient especially given that the number of available categories is not expected to be very high.

The set-based retrieval of categories is more complex. However, Modern DL reasoners incorporate important optimizations and, once a TBox has been classified, the response times obtained when checking subsumption are remarkably low despite the high complexity of this reasoning task. We will evaluate in more detail the theoretical complexity of this search and response times obtained from experimental evaluation in Chapter 8

Example 6.13 Let us consider the categories in the previous example, and let us imagine that only one taxonomy with these two categories exists and that we search categories based on the following set of abstract effects, defined by concept *SoughtCapability*:

```
?x memberOf SoughtCapability equivalent
?x memberOf InfoProvision and
    exists ?y(?x[ofInfoItem hasValue ?y]) and
    forall ?y(
        ?x[ofInfoItem hasValue ?y] implies ?y memberOf Flight[hasOrigin hasValue Madrid, hasDestination hasValue
            Barcelona]).
```

The concept above formalizes the provision of information about flights from Madrid to Barcelona. If this concept is given, we will query the *Categories* TBox for:

1. Concepts representing categories equivalent to *SoughtCapability*: no concept is found.
2. Concepts representing categories subsumed by *SoughtCapability*: no concept is found.
3. Concepts representing categories which subsume *SoughtCapability*: concept *"http://www.afi.es/Taxonomy1#FlightSearch"* is found, corresponding to the *FlightSearch* category. Notice that concept *"http://www.afi.es/Taxonomy1#LowCostFlightSearch"* will not be found, as it refers to flights operated by airlines which are *LowCostAirline* and not to general flights.
4. Concepts representing categories which are equivalent to or subsumed by the negation of *SoughtCapability*: no concept is found and, therefore, we can determine that concept *"http://www.afi.es/Taxonomy1#LowCostFlightSearch"* intersects the concept given.

Therefore, the sets of equivalent and more specific categories will be empty. The set of more general categories consists of the *FlightSearch* category, and the set of categories with effects in common with the concept given consists of the *LowCostFlightSearch* category. The details of these two categories will be returned.

□

Category update. When a category is updated, its new details must be given. If the set-based modelling of the category it represents does not change, we will only update its details on the relational database. Otherwise, we will check the satisfiability of the set-based modelling given by its prototypical service description, retract the previous concept associated to this category from the *Categories* TBox, publish the new concept, and reclassify the TBox. Furthermore, if the parent category of the updated category changes, we will also update the structure of the taxonomy loaded in memory.

Category deletion. For the deletion of a category, its name and the taxonomy it belongs to must be given, i.e., its complete URI is required. We will check whether any published service is associated to this category (we will see below how services are published and associated to categories) and, if this is the case, we will not allow for the deletion of this category as services associated to it must be reassigned first. Otherwise, we will delete the category information from the relational database, update the structure of the taxonomy loaded in memory, retract this concept from the *Categories* TBox, and reclassify this TBox.

Taxonomy deletion. For requesting the deletion of a taxonomy, its URI must be given. We will recursively delete the categories contained in this taxonomy (if any) using the operation above, and then delete the taxonomy information from the relational database and its structure from memory. It must be noted that the deletion of a taxonomy will only be possible if all its categories can be deleted, i.e., if no service is assigned to categories in the taxonomy.

Taxonomy retrieval. The taxonomy manager can be queried for all taxonomies published and their categories. This operation will return the list of all published taxonomies, together with the names and parents of all categories in such taxonomies.

6.3.2 Supporting users in the description of services.

Based on the relations among different types of descriptions of the value of a service presented above and, especially, on taxonomies of pre-defined categories to which a prototypical service description is associated, we will offer support to providers for describing their services.

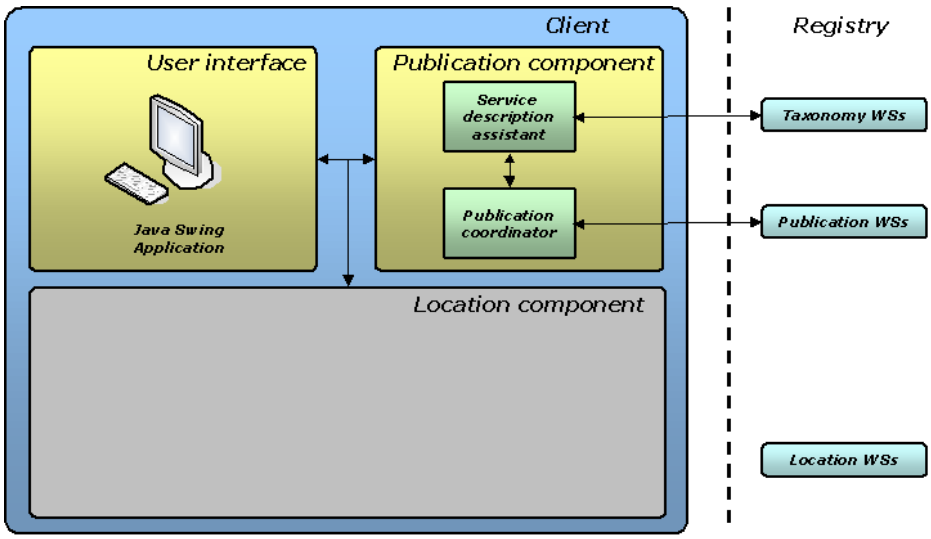

Figure 6.6: Client architecture (publication component)

We have implemented a publication component, part of the SETA client, which enables the description and publication of services to the registry depicted in Figure 6.3, and which supports users for providing different types of descriptions of their services. In Figure 6.6, the architecture of the client used in the SETA platform for publishing and locating services is shown. The publication component consists of a publication coordinator and of a service description assistant; the former will be presented in the next Section, while the latter is presented in this Section.

A service provider will use the user interface depicted in Figure 6.6 to provide the description of its service for publication. This user interface also includes tabs for the management of taxonomies as presented above, and for the location of services as we will see in the next Chapter.

The user interface, in the current prototype implementation, offers two options to the provider for describing his service:

1. Loading an existing WSMO description of the service from a file (Figure 6.7).

2. Creating a new description. In this case, the URI which identifies the service must be given and, optionally, the URL where a WSDL description of the service can be found can be provided. These data will be used to build a template WSMO description of the service (Figure 6.8).

In both cases, the description loaded or built can be updated to modify or to complete the description of the service, and support will be offered to users for providing types of descriptions of the service not yet available.

Example 6.14 Let us consider the service given in Listing 6.4, which offers the booking of flights operated by IntAir, and let us imagine we want to create now from scratch the description of this service. We will provide to the service description assistant the URI of the service (*http://www.afi.es/services/BookIntAirFlight1*), and the location of the WSDL description of the service (*http://www.afi.es/eTourism/BookIntAirFlight1.asmx?wsdl*). With this input, the assistant will build the following WSMO template description (see Figure 6.8):

```
namespace { dc _"http://purl.org/dc/elements/1.1#",
            afi  _"http://www.afi.es/WSDiscovery/Description#" }

webService _"http://www.afi.es/ services /BookIntAirFlight1"

  importsOntology { _"TO BE COMPLETED" }

  capability  BookIntAirFlight1Capability
    nonFunctionalProperties
      afi#wsdlDescription  hasValue "http://www.afi.es/eTourism/BookIntAirFlight1.asmx?wsdl"
      dc#description  hasValue "TO BE COMPLETED"
      dc#language hasValue "en-GB"
      afi#category  hasValue "TO BE COMPLETED"
    endNonFunctionalProperties

    sharedVariables  {?toBeCompleted}

    postcondition
      nonFunctionalProperties
        afi#descriptionType  hasValue "setBasedCapability"
        afi#intention  hasValue "TO BE COMPLETED"
      endNonFunctionalProperties
      definedBy
        ?x memberOf _"http://www.afi.es/services/BookIntAirFlight1"  equivalent
        ?x memberOf ToBeCompleted.

    precondition
      nonFunctionalProperties
        afi#descriptionType  hasValue "LPInfoPreconditions"
      endNonFunctionalProperties
      definedBy
        ToBeCompleted.

    postcondition
      nonFunctionalProperties
        afi#postconditionType hasValue "inputDependentEffects"
        afi#intention  hasValue "TO BE COMPLETED"
      endNonFunctionalProperties
      definedBy
        ?x memberOf _"http://www.afi.es/services/BookIntAirFlight1"  equivalent
        ?x memberOf ToBeCompleted.
```

□

Either after loading an existing description from a file or after obtaining the template

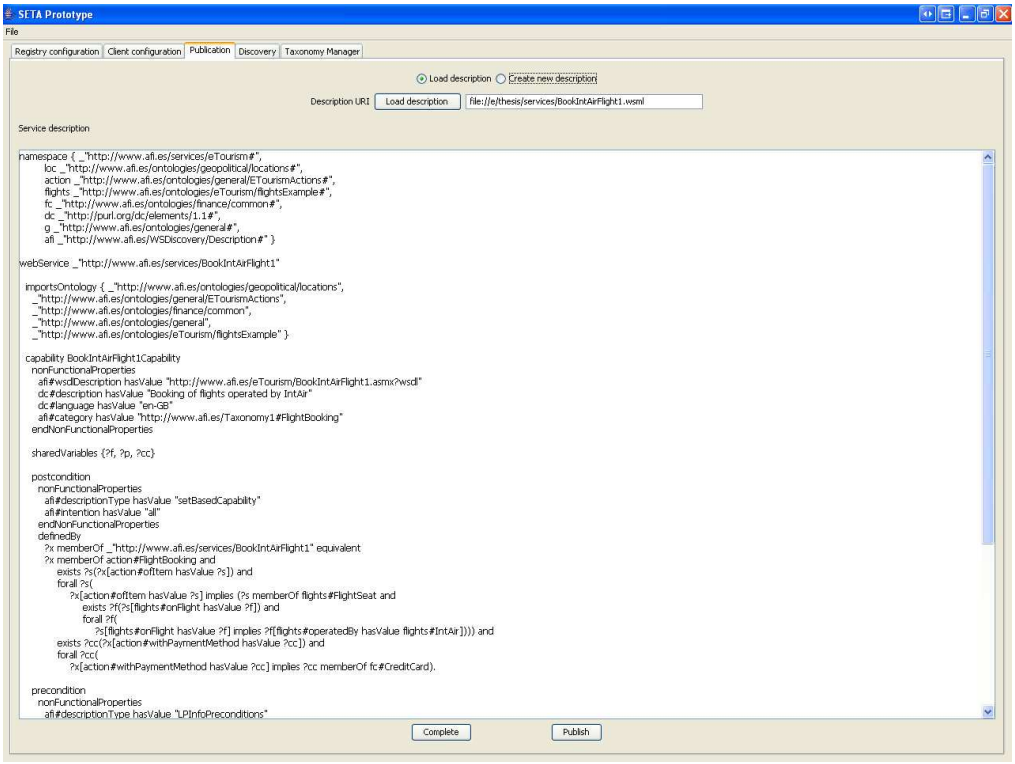

Figure 6.7: Loading a service description

description shown in the example above, we will support users for completing such descriptions, helping them in providing the types of descriptions of the service not yet available starting from the types of descriptions already known. In particular, if the user clicks the *Complete* button shown in Figures 6.7 and 6.8, he will receive proposals for completing missing descriptions of the service, in the order and following the mechanisms detailed in the following. In particular, at each step we will submit to the service description assistant the current service description and specify for what type of description we require a proposal, and the assistant will return its proposal (if any). Based on this proposal, the consumer will ultimately choose the description of each type to be incorporated to its service description.

6.3.2.1 Proposal for a textual description

If a textual description of the service has not been provided, we will submit the current service description to the description assistant. The assistant will have two possible proposals for a textual description:

- If a WSDL description of the service is available, i.e., the location of a WSDL description has

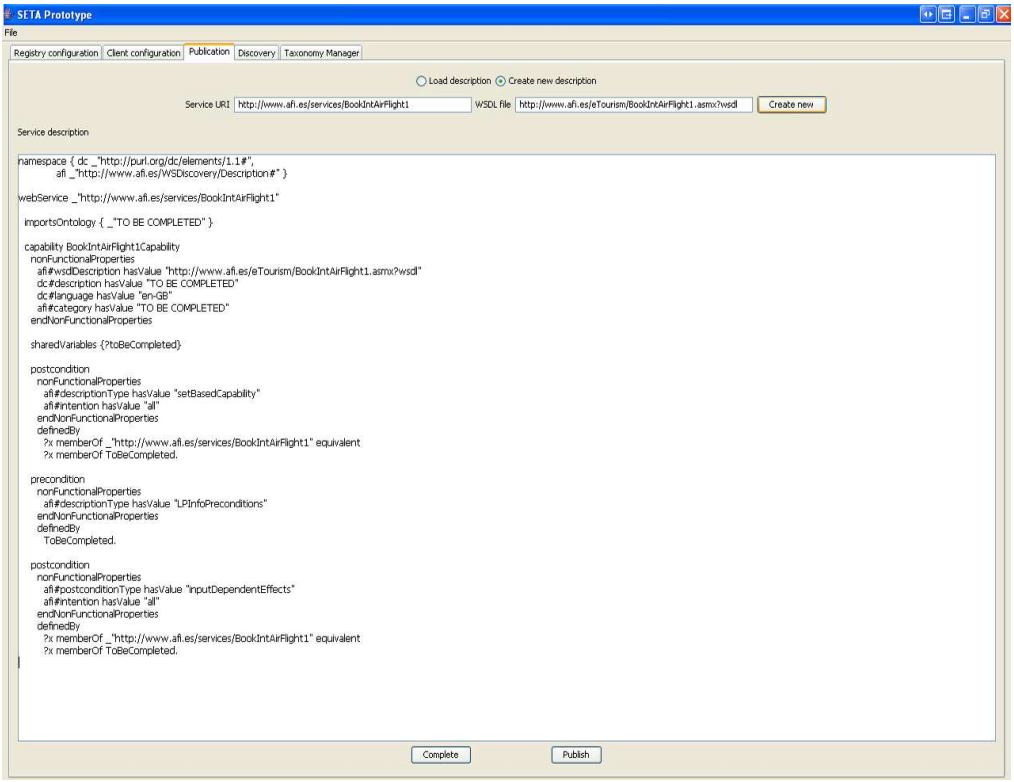

Figure 6.8: Creating a new service description

been provided with the WSMO description of the service, documentation elements present in the WSDL description will be concatenated to build a proposal for the textual description of the service.

- If the service has been categorized, i.e., if some category has been given as a value for the *category* non-functional property of the WSMO description, the assistant will communicate with the taxonomy manager of the registry in order to retrieve the details of these categories and the textual descriptions associated to such categories will be concatenated to build a proposal for the textual description of the service.

If both a WSDL description of the service and a categorization are available, both proposals of textual descriptions to the user will be given by the service description assistant, and we will provide both proposals to the user so that he can use them to build his own combined textual description. In any case, these proposals will only help the user in providing this type of description, but the user will have to ultimately choose what textual description is given to the service. After the user has provided the textual description based on the proposal given (an empty description is

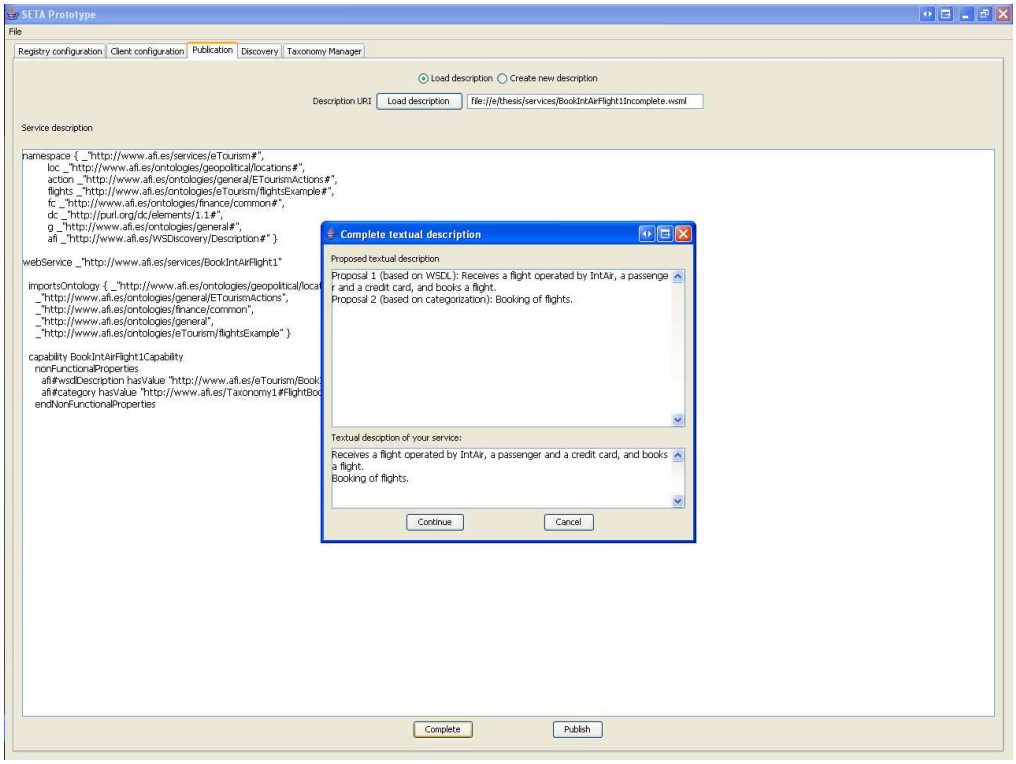

Figure 6.9: Proposal for a textual description

possible), we will add this textual description to the WSMO service description and continue to the next step.

In Figure 6.9, we show the proposal for a textual description if we load an incomplete description of the service in Listing 6.4, for which only the location of a WSDL description of it and its categorization under category *http://www.afi.es/Taxonomy1/FlightBooking* have been given.

6.3.2.2 Proposal for categorization

If the service has not been categorized, we will require a proposal from the service description assistant, which will distinguish two cases:

- If a set-based modelling of the service capability is available, the assistant will communicate with the taxonomy manager of the registry to search, given the concept formalizing the service capability, related categories. These categories will constitute the proposal for categorization and returned by the assistant, grouped into equivalent, more general, more specific, and intersecting categories.

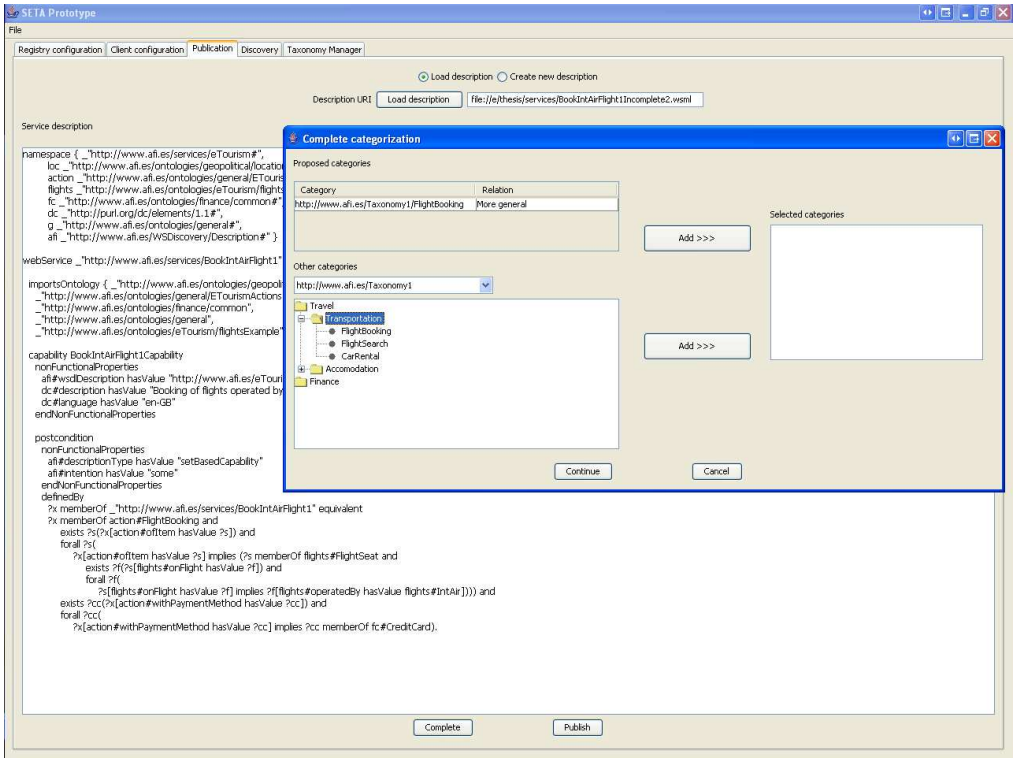

Figure 6.10: Proposal for categorization of a service

- If no set-based modelling of the service capability is available but a textual description of the service is available, the assistant will communicate with the taxonomy manager of the registry to search, given the textual description of the service, related categories.

Notice that, if both a set-based modelling of the service capability and a textual description of the service are available, the service description assistant will only use the first type of description to search related categories, as search results will be more accurate than those obtained using a textual description.

Proposed categories will serve to guide the user in categorizing his service. Still, the user will have to choose what categories the service will be assigned to and, for this purpose, he will be able to browse categories in all available taxonomies and choose which ones his service belongs to. After categories have been chosen by the user possibly based on the proposal given (if any), we will add these categories to the complete description of the service and continue. Choosing an empty categorization is also possible.

In Figure 6.10, we show an example proposal for the categorization of an incomplete version of the service in Listing 6.4, for which the set-based modelling of its capability but not a category

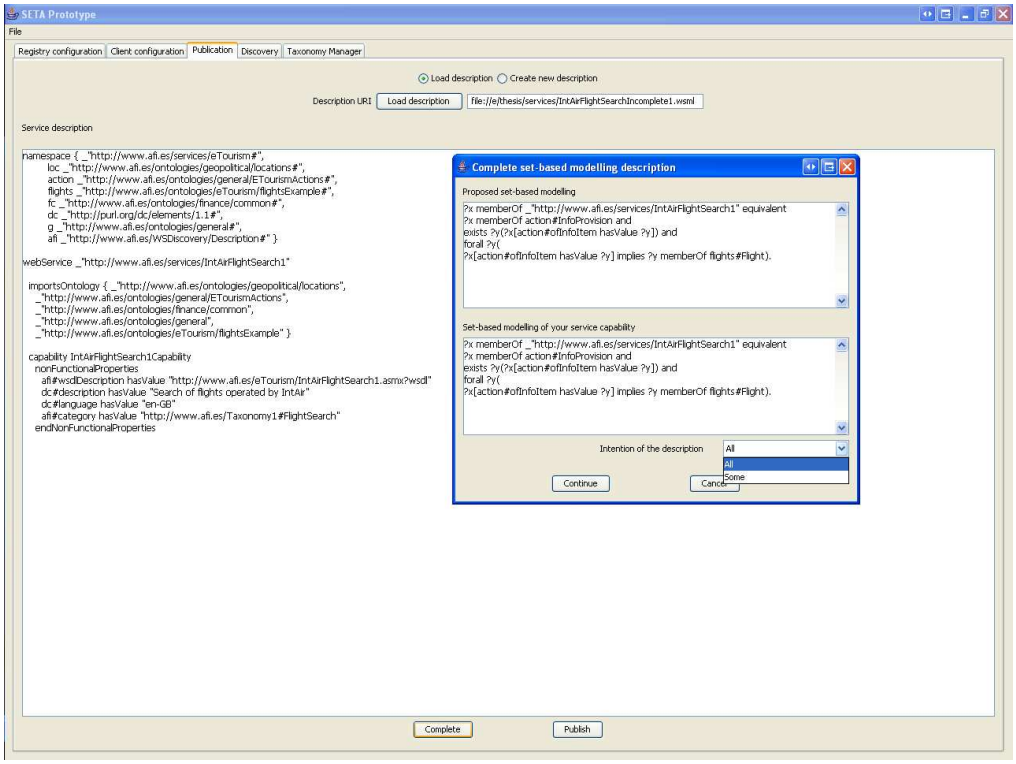

Figure 6.11: Proposal for the set-based modelling of the service capability

has been given.

In [Lara et al., 2006b] we used an alternative way of proposing categories to users. This method heuristically, and based on the semantic description of information preconditions and on the set-based modelling of capabilities, proposed the categories from existing taxonomies that best fit the service. This was done by measuring the similarity between the semantic description of the service preconditions and effects, and the preconditions and effects of services published at the registry and already assigned to a given category, using the method presented in [Corella and Castells, 2006b; Corella and Castells, 2006a]; however, only named concepts in domain ontologies used in the description of preconditions and effects were used.

We consider the method we have proposed more precise than the method presented in [Lara et al., 2006b], as it does not depend on how users have categorized so far their services, which might not always be correct, and as it uses the precise description of the effects associated to the service capability and not only what concepts in domain ontologies appear in the description of information preconditions and effects.

6.3.2.3 Proposal for the set-based modelling of capabilities

If the set-based modelling of its capability has not been provided, we will require from the assistant a set-based modelling of the service. The assistant, if the service has been categorized will propose to the user the capability formalizations associated to the categories the service has been assigned to. In particular, if the n categories a service $serv$ has been assigned to are denoted $c_1, \ldots, c_n$, and their associated capabilities are given by concepts $Eff_{c_1}, \ldots, Eff_{c_n}$, the concept proposed for the formalization of the abstract effects of the service will be:

$Eff_{serv} \equiv Eff_{c_1} \sqcup \ldots \sqcup Eff_{c_n}$

The definition above means that the abstract effects proposed will be the union of the effects associated to each category the service has been assigned to.

This proposal will be given to the user, who has the possibility of modifying it. If no categorization of the service has been given, no proposal can be built and we will simply ask the user to provide from scratch the set-based modelling of his service. In any case, the user can choose not to provide any set-based modelling of his service capability.

Finally, the consumer also has to choose what intention is associated to the description given. The set-based modelling and the intention provided will be added (if any) to the WSMO description of the service and we will proceed to the next step.

In Figure 6.11, we show the proposal for the set-based modelling of a service categorized under *http://www.afi.es/Taxonomy1/FlightSearch* for which a set-based modelling of its capability has not been given. The user can, taking this proposal as a basis, provide a more accurate formal description of the service capability.

6.3.2.4 Proposal for information preconditions

If the description of information preconditions of the service is not available, we will ask the description assistant for a proposal. The assistant, if the service has been categorized but neither the description of its information preconditions nor the description of its input-dependent effects have been provided, will propose the prototypical information preconditions associated to the categories the service has been assigned to.

If the n categories a service $serv$ has been assigned to are denoted $c_1, \ldots, c_n$, and their associated information preconditions are given by queries $q_{c_1}, \ldots, q_{c_n}$, the information preconditions proposed for the service will be:

$? - q_{c_1}, \ldots, q_{c_n}$

This means that we will require all information preconditions of all categories to be fulfilled in our proposed description of information preconditions. This proposal (if any) will be given to the user, who will possibly modify it (see Figure 6.12) or leave it empty. After the description has been

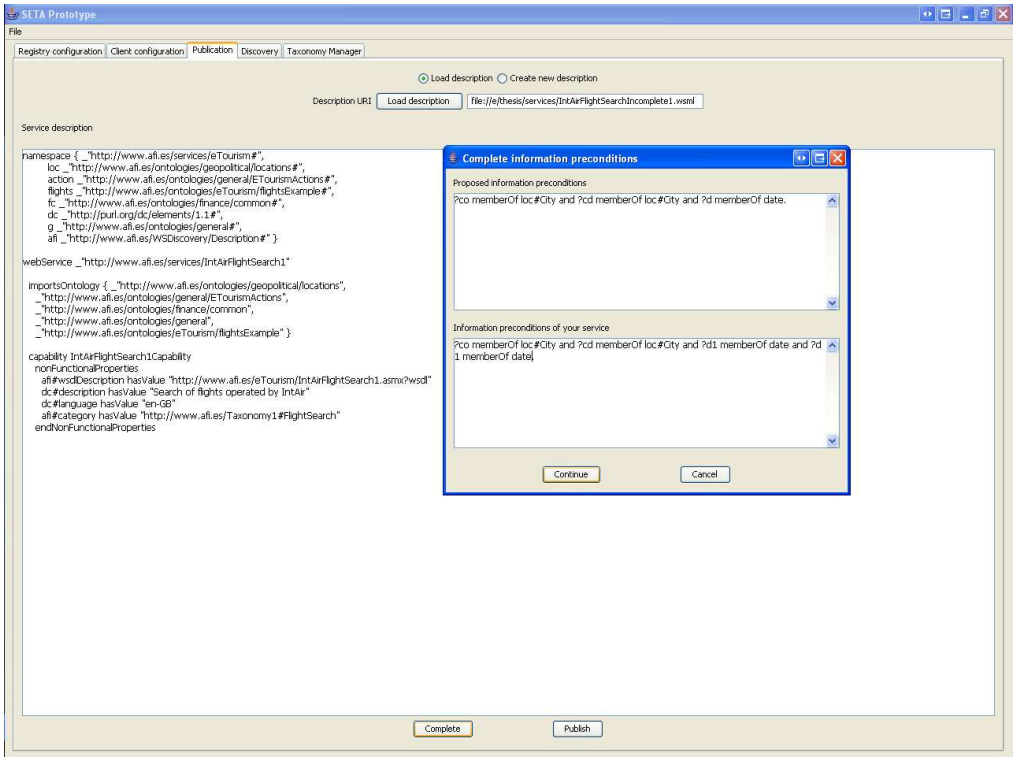

Figure 6.12: Proposal for information preconditions

completed, we will add it to the WSMO description of the service (if it is not empty) and continue with the description completion process.

6.3.2.5 Proposal for input-dependent effects

If input-dependent effects of the service have not been describe, the service description assistant will be asked for a proposal. If the service has been categorized but neither the description of its information preconditions nor the description of its input-dependent effects have been provided, the assistant will propose the prototypical input-dependent effects associated to the categories the service has been assigned to. In particular, if the n categories a service $serv$ has been assigned to are denoted $c_1, \ldots, c_n$, and their associated input-dependent effects are given by concepts $InputEff_{c_1}, \ldots, InputEff_{c_n}$, the concept proposed to the user for the formalization of the abstract effects of the service will be:

$$InputEff_{serv} \equiv InputEff_{c_1} \sqcup \ldots \sqcup InputEff_{c_n}$$

Furthermore, categories $c_1, \ldots, c_n$ declare shared variables used in the description of input-dependent effects. The union of the shared variables associated to each category will be proposed

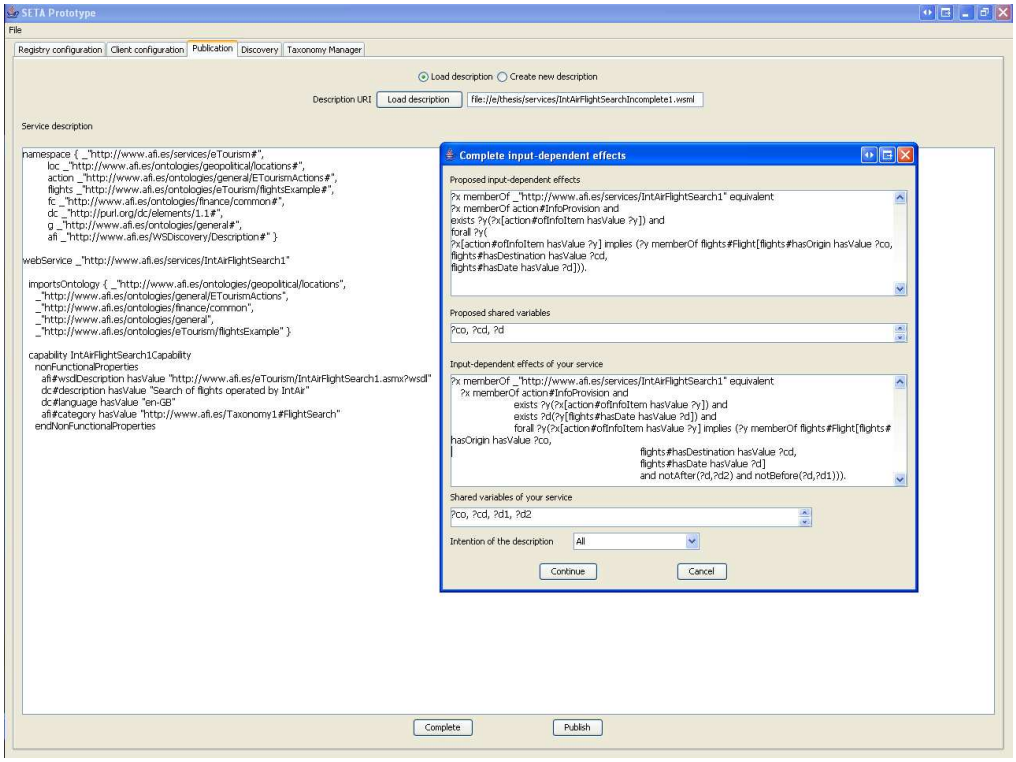

Figure 6.13: Proposal for input-dependent effects

as the set of shared variables of the service (see Figure 6.13). Based on this proposal, the user will provide the description of input-dependent effects of his service and of shared variables (or leave them empty), and will choose the intention of this description.

Helped by the proposals presented above, users will have the possibility of completing the description of their services. As presented above, these proposals will act as a guide for providers to describe their services, but they will have the final responsibility for selecting and modifying the different descriptions proposed. This description completion procedure can be executed any number of times.

6.3.3 Publication of descriptions

Once the user has available, possibly with the help of our service description assistant, the description of a service he wants to publish, we will initiate the process of publishing this description to the SETA registry. In particular, the description will be sent to the publication manager of the registry depicted in Figure 6.3 by the publication coordinator of the publication component in Figure

6.6, and this publication manager will: a) check the consistency of the service description provided, b) store the description of the service, and c) process the description so that its matching to a consumer goal can be evaluated at location time as efficiently as possible.

In the following, we will describe in more detail the main phases of the publication process.

6.3.3.1 Checking the validity of a description

When the user decides to initiate the actual publication of a service description to the registry, after completing missing types of descriptions of such service using the assistance provided by the platform, the publication coordinator of the SETA publication component will, as its name suggests, coordinate the submission of the description to the registry and the communication with the user if his intervention is required at some point for correcting the description provided.

The description of the service will be submitted to the publication manager of the service registry, which will first check the validity of the description provided. This validation will be two-fold: i) the WSMO service description will be parsed and its validity evaluated using WSMO4J, and ii) the validity and the consistency of the set-based modelling and the categorization of the service, encoded by the WSMO service, will be evaluated; how this second validation is performed is presented in the following.

Consistency of categories and set-based modelling of the service capability. If a set-based modelling of the service capability has been given, and possibly a categorization of the service, we will evaluate the consistency of these descriptions. In particular we will evaluate whether the effects given by the formal descriptions associated to service categories are compatible with the effects given by the set-based modelling of the service capability. For this purpose, the publication manager will communicate with the taxonomy manager to obtain the set-based modelling of the categories given, and use the DL reasoner to determine the satisfiability of the concept defined as the conjunction of: a) the concept which formalizes the service capability, and b) the union of the concepts formalizing the set of effects associated to the categories the service has been assigned to. However, if the service has not been categorized, we will only check the satisfiability of the concept formalizing the service capability.

Given a service $serv$, assigned to categories $c_1, \ldots, c_n$ and whose formalized effects are given by a concept Eff_{serv}, we will check the satisfiability of a concept Eff^T_{serv} defined as:

$$Eff^T_{serv} \equiv Eff_{serv} \sqcap Eff_C$$

where $Eff_C \equiv Eff_{c_1} \sqcup \ldots \sqcup Eff_{c_n}$ if the service has been categorized, and $Eff_C \equiv \top$ otherwise.

This is illustrated by the following example:

Example 6.15 Let us imagine we want to publish the description given in Listing 6.4 of a service which has the capability of booking seats on flights operated by IntAir. Now, let us imagine a category *LowCostFlightBooking* whose set-based modelling of the capability it represents is given by:

$$\begin{aligned} Eff_{LowCostFlightBooking} \equiv FlightBooking \sqcap \\ \exists ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap \exists onFlight \sqcap \\ \forall onFlight.(Flight \sqcap \exists operatedBy \sqcap \forall operatedBy.(LowCostAirline))) \end{aligned}$$

If the service is assigned to category *LowCostFlightBooking*, and if in domain ontologies $\mathcal{O}^{DL}$ we have a disjunction axiom stating that an airline cannot be both a low-cost airline and a regular airline, i.e., that $LowCostAirline \sqcap RegularAirline \sqsubseteq \bot$, we will have that concept $Eff^{T}_{BookIntAirFlightCapability}$ defined as:

$$Eff^{T}_{BookIntAirFlight1} \equiv Eff_{BookIntAirFlight1} \sqcap Eff_{LowCostFlightBooking}$$

is not satisfiable and, thus, the categorization of the service is not consistent with the set-based modelling of its capability. □

If the result of this test is negative, i.e., if concept Eff^{T}_{serv} defined as explained above is not satisfiable, the publication manager will return an error to the publication coordinator of the publication component. In this case, we will ask the user to revise the description of the service.

Checking the satisfiability of concepts for the $\mathcal{SHOIN}$ description logic can be time-consuming. While the response times obtained for this task can be relatively high, publication is not a time-critical task and, therefore, we consider these times acceptable. We will discuss in more detail the complexity and response times expected in Chapter 8.

Validity of information preconditions and of input-dependent effects. As discussed in Section 6.3.1, the following relation must hold for any capability $\mathcal{C}$ and any valid input binding β_i assigning values to the input variables of a service *serv* enabling access to $\mathcal{C}$:

$$InputEff_{serv,\beta_i} \sqsubseteq Eff_{\mathcal{C}}$$

However, as we cannot determine at publication time all possible valid input bindings β_i, we cannot evaluate whether this relation holds for the description provided. Still, we must keep in mind that this relation must hold. In this setting, we will only evaluate whether the description of information preconditions (if given) is a valid WSML-Flight query, and whether the description of input-dependent effects (if given) is a valid WSML-DL+ concept.

The consistency of other types of descriptions cannot be checked, in the sense that our platform cannot detect contradictory information in the remaining types of descriptions.

6.3.3.2 Storing and processing descriptions

Once the publication manager has checked the validity of the description of the service and no problem has been encountered, the description of the service will be stored and processed for its posterior retrieval.

UDDI storage The service description will be stored in the UDDI repository part of the registry. In particular, the publication manager will create a *UDDI business service* [Bellwood et al., 2002], whose name will be the service identifier extracted from the WSMO description. This business service will be associated:

- using a *UDDI category bag* [Bellwood et al., 2002], the categories the service belongs to. In particular, we will reference the tModel created for the taxonomy of the category using a *UDDI keyed reference*, and the value given to this category will be the name of the category;
- using keyed references, each to a particular tModel created for this purpose, the textual description, information preconditions, set-based modelling, and input-dependent effects of the service;
- using a keyed reference to a UDDI pre-defined tModel, the WSDL description of the service retrieved by the taxononomy manager from the URL given. This pre-defined tModel will be of type (*uddi-org:types*) *wsdlSpec*.

Description processing. In order to speed up the location of services based on their descriptions, we will process the categorization of the service and the set-based modelling of the service capability at publication time, as described next.

First, the publication manager will associate to categories the service belongs to (if any), in the structure of taxonomies we keep loaded in memory (see Section 6.3.1), the service identifier. In this way, each category c will have associated in memory the set $\mathcal{S}_c$ of published services which

have been categorized under it. If a service has been categorized under several categories, it will be a member of the set of services associated to each such category.

Second, the publication manager will publish the concept which provides the set-based modelling of the service capability (if available) to a TBox named *Services* of the DL reasoner, and this TBox will be classified. Published services are thus arranged in a subsumption hierarchy in terms of the effects they can provide. We will at location time be able to efficiently query this subsumption hierarchy, as we will see in Chapter 7. It must be noted that the *Services* TBox is created when the registry is started and the set $\mathcal{O}^{DL}$ of domain ontologies service descriptions can refer to loaded into this taxonomy. Furthermore, the TBox is loaded at this time.

The times required for computing the subsumption hierarchy of services based on the set-based modelling of their capabilities can be high. However, this task will be done only once for each service and at publication time and, therefore, it will not interfere with the times required for locating services. More details will be provided in Chapter 8.

After the publication of the concept formalizing the capability of the service and the classification of the *Services* TBox, and if no error occurred, we consider the publication of the service finished, and the publication manager will inform of the successful publication to the publication coordinator of the SETA client, which will in turn notify it to the user.

It must be noted that the publication manager of the registry is accessible via two WSDL services:

1. A service which, given a WSMO service description for publication checks its consistency, stores the description, and completely process it in order to make it ready for efficient matchmaking.

2. A service which does the same tasks as the previous service *but the classification of the Services TBox*. This service has been used for *bulk publications*, where we want to publish a (possibly big) set of n service descriptions but we do not want to classify n times the TBox, i.e., after the publication of each individual description. Instead, we want to save the classification time required by each individual publication and only classify the TBox when the last description is published and, therefore, all the concepts formalizing the capability of the different services have been already published to the TBox.

Furthermore, users can choose to directly use the UDDI publication API. However, services published in this way will only be locatable using the UDDI inquiry API, as they will only be handled by the UDDI repository.

6.4 Summary

In this Chapter, we have presented the first part of our proposed instantiation of the abstract model presented in Chapter 5.

First, we have introduced the types of syntactic and semantic descriptions of the value of services considered; this set of description types has been chosen with the purpose of covering the needs of different use cases while keeping the set as reduced as possible and, what is more, keeping compatibility with current practices in service-oriented computing.

Second, we have presented how users can describe their services using a publication component we have prototypically implemented, as well as how such descriptions are published to a prototype registry, conceived as an extension of a UDDI repository, which is able to store and process all the types of descriptions proposed. In particular, we provide support to users for providing alternative types of descriptions of their services based on associating formal and non-formal descriptions to categories, which can be seen as coarse-grained, pre-defined capabilities.

It must be noted that, although a centralized registry is assumed in our discussion, i.e., all service descriptions are published to a single registry, this is not imposed by our model and alternative architectures e.g. peer-to-peer or hybrid architectures could be used. This will, though, have implications on how the location process is articulated.

With the types of descriptions used and the support offered to users for describing their services, we believe we have achieved a flexible and usable basis for publishing the value of available services, which will be later exploitable for locating services with different accuracy and efficiency requirements. How the location process makes use of such descriptions and of the registry presented will be presented in the next Chapter, as well as how consumer goals are described.

Chapter 7

Model Instantiation and Prototype Implementation II: Description of goals and discovery of services

7.1 Introduction

In Chapter 6, we have presented alternative types of descriptions of the value of services, how users (service providers or other agents acting on their behalf) are supported for providing different types of descriptions of a particular service, and how the complete description of a service is published to a registry. Given that services have been described and their descriptions accessible to prospective consumers or to systems acting on their behalf, we will in this Chapter describe how, and following the guidelines provided by the abstract model in Chapter 5, we articulate the discovery of services based on their value, i.e., how we instantiate the second part of such abstract model in the SETA platform.

In Section 7.2 we discuss how consumers can describe their goals in our model instantiation, i.e., how consumers can describe the value they expect to obtain from using a service, which will guide what services are potential solutions (to different extents) for the consumer goal. In this Section we will also discuss how consumer knowledge is expected to be described, as it might play a role in deciding what services can be used to achieve the consumer goal. Finally, this Section will describe what type of support is offered to consumers for describing their goals; this support will be very similar to the type of support offered to service providers for describing their services, as the types of descriptions of goals considered are basically symmetric to the types of descriptions of

services presented in Chapter 6, Section 6.2.

Once the consumer goal has been described, we will initiate the service discovery process which, as dictated by the abstract model in Chapter 5, will be split into two phases: a first phase where filters are applied in order to retrieve from the registry relevant services but without having access to consumer knowledge, and a second phase where obtained service descriptions are evaluated at the consumer side, where access to consumer knowledge is possible and where custom filters can be defined by consumers. The first phase of the discovery process and the filters applied will be presented in Section 7.3, and the second phase and its filters will be presented in Section 7.4.

Finally, Section 7.5 provides a summary of the contents of the Chapter. An evaluation of our complete model instantiation will be presented in Chapter 8, where known limitations of this instantiation will also be discussed, related work analyzed, and possible extensions outlined.

7.2 Description of goals

Consumers are interested in locating and using certain services because they can provide effects that solve a given consumer's need. For example, a consumer might need to fly from Madrid to Munich, and he will be interested in locating a service that can provide the booking of a seat on an appropriate flight. For locating services, consumers' objectives must be explicitly described, as presented in this Section.

Two families of descriptions of the effects expected by consumers are distinguished: syntactic descriptions and semantic (formal) descriptions. The particular types of syntactic and semantic descriptions considered, which are mainly symmetric to the types of descriptions of services allowed, are presented in the following. Additionally, consumer knowledge available will be explicitly described, but not necessarily once per goal, as we will see.

7.2.1 Syntactic descriptions

7.2.1.1 Textual description

Consumers can describe the effects they expect from using a service using natural language. This type of description is symmetric to the textual description of the service value, and it has the following main properties:

- It describes, using natural language, what a service is expected to do for the consumer e.g. the purchase of books or investment funds.
- It is easy to provide by an average user.
- It often has limited precision, and it suffers from the ambiguity of natural language.

Goals in our platform will be encoded as WSMO goals (see Chapter 3, Section 3.4.1), and the textual description of the goal will be encoded as the value of the Dublin Core description non-functional property of the capability, as shown in Listings 7.1, 7.2, and 7.3. Additionally, the Dublin Core language non-functional property is used to indicate the language in which this description is provided.

```
namespace { _"http://www.afi.es/services/finance#",
        loc _"http://www.afi.es/ontologies/ geopolitical /locations#",
        action _"http://www.afi.es/ontologies/general/actions#",
        funds _"http://www.afi.es/ontologies/finance/investmentFunds#",
        dc _"http://purl.org/dc/elements/1.1#",
        afi _"http://www.afi.es/WSDiscovery/Description#",
        ck _" file://consumerknowledge/myKnowledge" }

goal _"http://www.afi.es/goals/FundSearch1"

  importsOntology { _"http://www.afi.es/ontologies/ geopolitical /locations",
    _"http://www.afi.es/ontologies/general/actions",
    _"http://www.afi.es/ontologies/finance/investmentFunds",
    _" file://consumerknowledge/myKnowledge" }

   capability FundSearch1Capability
    nonFunctionalProperties
      dc#description hasValue "Search of investment funds commercialized in the Spanish market by MyEntity"
      dc#language hasValue "en-GB"
      afi#category hasValue "http://www.afi.es/Taxonomy1#InvestmentFundSearch"
       afi#filter hasValue {"Capability", " InputAvailability ", "InputDependentEffects" }
    endNonFunctionalProperties

    postcondition
      nonFunctionalProperties
        afi#intention hasValue "all"
      endNonFunctionalProperties
      definedBy
        ?x memberOf _"http://www.afi.es/services/FundSearch1" equivalent
        ?x memberOf action#InfoProvision and
            exists ?y(?x[action#ofInfoItem hasValue ?y]) and
            forall ?y(
                ?x[action#ofInfoItem hasValue ?y] implies ?y memberOf funds#InvestmentFund[funds#commercializedIn
                    hasValue loc#Spain, funds#commercializedBy hasValue ck#MyEntity]).
```

Listing 7.1: WSMO description of the goal of searching investment funds commercialized in Spain by entity MyEntity

```
namespace { _"http://www.afi.es/services/eTourism#",
        loc _"http://www.afi.es/ontologies/ geopolitical /locations#",
        action _"http://www.afi.es/ontologies/general/ETourismActions#",
        flights _"http://www.afi.es/ontologies/eTourism/flightsExample#",
        fc _"http://www.afi.es/ontologies/finance/common#",
        dc _"http://purl.org/dc/elements/1.1#",
        g _"http://www.afi.es/ontologies/general#",
        afi _"http://www.afi.es/WSDiscovery/Description#",
```

```
        ck _" file ://consumerknowledge/myKnowledge" }

goal _"http://www.afi.es/goals/BookAFlight1"

  importsOntology { _"http://www.afi.es/ontologies/ geopolitical /locations",
    _"http://www.afi.es/ ontologies /general/ETourismActions",
    _"http://www.afi.es/ ontologies /finance/common",
    _"http://www.afi.es/ ontologies /general",
    _"http://www.afi.es/ ontologies /eTourism/flightsExample",
    _" file ://consumerknowledge/myKnowledge" }

   capability BookAFlight1Capability
    nonFunctionalProperties
      dc#description hasValue "Booking a seat on flight flight1234 from Madrid to Munich, paid with a credit card
          DummyCard"
      dc#language hasValue "en-GB"
      afi#category hasValue "http://www.afi.es/Taxonomy1#FlightBooking"
       afi#filter hasValue "Category"
    endNonFunctionalProperties

    postcondition
      nonFunctionalProperties
        afi#intention hasValue "some"
      endNonFunctionalProperties
      definedBy
        ?x memberOf _"http://www.afi.es/goals/BookAFlight1" equivalent
        ?x memberOf action#FlightBooking and
            exists ?s(?x[action#ofItem hasValue ?s]) and
            forall ?s(
              ?x[action#ofItem hasValue ?s] implies
                ?s memberOf flights#FlightSeat[flights#onFlight hasValue ck#flight1234, flights#forPerson hasValue
                    ck#rubenLara]) and
          ?x[action#withPaymentMethod hasValue ck#myDummyCreditCard].
```

Listing 7.2: WSMO description of the goal of booking a particular flight and paid with a particular type of credit card

```
namespace { _"http://www.afi.es/services/eTourism#",
        loc _"http://www.afi.es/ontologies/ geopolitical /locations#",
        action _"http://www.afi.es/ ontologies /general/ETourismActions#",
        flights _"http://www.afi.es/ ontologies /eTourism/flightsExample#",
        dc _"http://purl.org/dc/elements/1.1#",
        afi _"http://www.afi.es/WSDiscovery/Description#" }

goal _"http://www.afi.es/goals/SearchFlights1"

  importsOntology { _"http://www.afi.es/ontologies/ geopolitical /locations",
    _"http://www.afi.es/ ontologies /general/ETourismActions",
    _"http://www.afi.es/ ontologies /eTourism/flightsExample" }

   capability SearchFlights1
    nonFunctionalProperties
      dc#description hasValue "Search of flights from Madrid to Crete"
```

```
    dc#language hasValue "en-GB"
    afi#category hasValue {"http://www.afi.es/Taxonomy1#FlightSearch"}
    afi#filter hasValue {"Capability", " InputAvailability "}
  endNonFunctionalProperties

  postcondition SearchFlights
    nonFunctionalProperties
      afi#intention hasValue "all"
    endNonFunctionalProperties
    definedBy
      ?x memberOf _"http://www.afi.es/goals/SearchFlights1" equivalent
      ?x memberOf action#InfoProvision and
           exists ?y(?x[action#ofInfoItem hasValue ?y]) and
           forall ?y(
             ?x[action#ofInfoItem hasValue ?y] implies ?y memberOf flights#Flight[flights#hasOrigin hasValue loc#Madrid,
                                                                                  flights#hasDestination hasValue loc#Crete]).
```

Listing 7.3: WSMO description of the goal of searching flights from Madrid to Crete

7.2.1.2 Categorization

The second (and last) type of syntactic description of the value expected from using a service is the provision of the categories sought services must belong to, which is symmetric to the categorization of services discussed in Chapter 6. This type of description has the following properties:

- It describes what categories relevant services must belong to. Alternatively, we can see this description as a coarse-grained description of what capability sought services must enable access to.
- The usage of categories is intuitive for users, which makes this type of description of the value expected from services easy to provide.
- Categories enable a more precise matching than textual descriptions, as we avoid the inherent ambiguity of natural language.

More than one category can be provided, meaning that candidate services must ideally belong to all the categories provided. Such list of categories will be encoded using a category non-functional property in the WSMO capability element, as shown in Listings 7.1, 7.2, and 7.3.

Finally, notice that the categories used for describing goals will be the same ones used for describing the value of services, published as described in Chapter 6 and part of pre-defined taxonomies. Therefore, taxonomies of categories constitute a vocabulary shared by service consumers and providers, which will enable comparing the categorization of a service to the categories given by a goal in order to locate relevant services.

The textual description and the description based on pre-defined categories of the value expected by prospective service consumers are symmetric to the same types of descriptions of the value of services presented in the previous Chapter. One type of syntactic description considered for services has not been included in the WSMO description of goals: the WSDL description. However, this does not mean the WSDL description of a service cannot be used by a consumer to decide on the suitability of a service for resolving his goal; we simply do not encode any details of the WSDL description of a sought service in the WSMO goal, but we will allow for the direct usage by consumers of the UDDI inquiry API and, therefore, for the retrieval of services from the registry based on their WSDL description. We will see how this is done in Section 7.3.

7.2.2 Semantic descriptions

In a similar way we formally described the value of services, we will now formally describe what value is expected by consumers; this expected value will drive the process of locating relevant services. The formal description of the value a consumer expects will be provided in terms of the same set $\mathcal{O}$ of domain ontologies introduced in Chapter 6 and to which formal service descriptions referred. Furthermore, if necessary, they will also refer to existing extensions of these ontologies in the direction of WSML-DL+, i.e., to the set $\mathcal{O}^{DL}$ of extended domain ontologies.

In the following, we briefly explain how expected effects are modelled as a DL concept and encoded into WSMO goal descriptions, and how consumer knowledge is declared for its posterior use in the second phase of the service location process.

7.2.2.1 Set-based modelling of effects

Consumers can specify the goal they want to achieve by describing the capability they want to access. This description will be provided, as for services, by a DL concept $Eff_{\mathcal{G}}$ which will have the following form in DL syntax:

$$\begin{aligned} Eff_{\mathcal{G}} \equiv\ & (Action_1 \sqcap Restrictions_1) \sqcup \\ & (Action_2 \sqcap Restrictions_2) \sqcup \\ & \ldots \sqcup \\ & (Action_n \sqcap Restrictions_n) \end{aligned}$$

where $Action_1, \ldots, Action_n$ are actions defined in ontologies of actions and $Restrictions_1, \ldots, Restrictions_n$ are concepts denoting restrictions introduced over the properties of such actions, and which will refer to the set $\mathcal{O}^{DL}$ of domain ontologies. In a nutshell, concept $Eff_{\mathcal{G}}$ defines the set of effects which the consumer wants to achieve by using a service, described in

terms of actions pre-defined in ontologies of actions and restrictions over such actions which define with more precision what particular action candidate services must be able to perform.

The expressivity allowed for describing the concept above is, as for services, the $\mathcal{SHOIN}(\mathcal{D})$ Description Logic underlying WSML-DL+.

Example 7.1 If a consumer wants to book a flight *flight1234*, which is a flight from Madrid to Munich operated by *IntAir* and about which the consumer has previously obtained information by using a flight search service, for passenger *rubenLara*, and paid with a credit card of type DummyCard, the concept formalizing the set of effects required will be defined as:

$$\begin{aligned} Eff_{BookAFlight1} \equiv FlightBooking \sqcap \\ \exists ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap onFlight.\{flight1234\}) \sqcap forPerson.\{rubenLara\} \sqcap \\ withPaymentMethod.\{myDummyCreditCard\} \end{aligned}$$

Intuitively, the formalization above can be read as: *a service which can provide the booking of a seat on flight flight1234, for person rubenLara, and paid with myDummyCreditCard* is sought.

If we consider the general form descriptions of capabilities take, given by Formula (7.1), we have that:

$$\begin{aligned} Action_1 \equiv FlightBooking \\ Restriction_1 \equiv \exists ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap onFlight.\{flight1234\}) \sqcap forPerson.\{rubenLara\} \sqcap \\ withPaymentMethod.\{myDummyCreditCard\} \end{aligned}$$

This formalization corresponds to the goal in Listing 7.2, where the formalization is written in WSML human-readable syntax.

□

In general, the concept which formalizes the capability sought services must enable access to is encoded by the WSMO postcondition of the goal capability, as shown in Listings 7.1, 7.2, and 7.3. This type of description has the following main properties:

- It formally describes the capability the consumer wants to access, which will enable the matching of services with a considerable precision based on explicit domain models and exploiting formal reasoning. Furthermore, and given the expressivity allowed, precise results will be

obtained in relatively low times using state-of-the-art reasoners, as we will see in the next Section.

- It only refers to the capability the consumer wants to access, without requiring a particular way of accessing it by the service. This makes this type of description *value-driven*, in the sense that it concentrates on the value sought and not on how it will be accessed.
- Given the form these descriptions take, in terms of pre-defined actions in appropriate ontologies for which restrictions on what kind of objects these actions apply to are introduced, we expect users with basic skills in knowledge representation to be able to provide this type of descriptions. Still, users will be supported for providing this type of description in the way we will present in Section 7.2.4.

Intention. The type of description introduced above defines the set of effects, associated to a capability, candidate services must enable access to. However, as introduced in [Keller et al., 2004a; Keller et al., 2005; Keller et al., 2006a], the description of the set of expected effects can be interpreted in different ways and, thus, the description by means of a set is not semantically unique. In this setting, the *intention* of users when modelling their goals must be explicitly stated as a meta-annotation of the concept describing sought effects. The purpose is to differentiate cases where the consumer seeks a service which can provide *all* the effects in the set described, from cases where the consumer goal can be satisfied by a service which can provide only *some* of the effects in this set. This meta-annotation is encoded by the *intention* non-functional property associated to the goal postcondition; a value *all* of this property means that all the effects in the set are required (see Listings 7.1 and 7.3), while a value *some* means that only some of these effects are required (see Listing 7.2). We will denote $Effs_{\mathcal{G}}$ the concept formalizing the set of effects required for achieving a goal $\mathcal{G}$, and $I_{\mathcal{G}} \in \{\exists, \forall\}$ its associated intention.

7.2.2.2 Description of consumer knowledge

We have seen how a consumer can describe the capability he wants to access in different ways, without placing any specific requirement on the preconditions and functionality of the service enabling access to such capability. However, information preconditions must be fulfilled in order to use a service and, thus, they can play a role in deciding whether a service can be used to fulfill a given goal or not.

One may think that a consumer goal should not only describe the capability sought, but also what information the consumer has available and is willing to disclose for achieving the required effects. However, the input required by available services, some of which might solve the consumer's goal, is not known by the consumer before-hand, and it can greatly vary among them. For example,

any service might require authentication (user name), and a consumer defining a goal cannot anticipate whether his user details will be necessary for achieving this goal. In the best case the customer can only guess, when defining his goal, what input values will be required by services satisfying such goal.

In addition, the consumer might have a considerable volume of information from which input values to services can be obtained. As an example, consider a business process which acts as a service consumer and which has access to information in corporate databases: input values for a service can be extracted from these databases if this is required for achieving a goal of the process. Furthermore, part of this information might be sensitive and, therefore, it should not be disclosed to third parties before establishing some trust relation. For example, and even though a consumer trying to achieve a particular goal might be willing to disclose his credit card details, he will probably not do so before the specific service that can be used for this purpose has been identified and a trust relation has been established with it.

Finally, and while most existing works e.g. [Li and Horrocks, 2003] and [Benatallah et al., 2003] expect consumers to explicitly describe, for each goal they want to achieve, the type of input values they can provide for achieving this particular goal, this requires consumers to anticipate what type of input values might be required by relevant services. We believe, though, that in most cases consumers will want to find services *which can provide the effects required and which can be executed*, i.e., the primary concern of a consumer is to realize some effects by executing a service, which requires having appropriate input values for this execution; consumers will not require particular information preconditions to be defined by the service, as long as they can execute the service with the information they have available.

In this setting, following considerations above and the abstract model defined in Chapter 5, we assume in our model instantiation that a knowledge base KB_c exists containing consumers's knowledge, and that possible input values for candidate services enabling access to a sought capability can be obtained from it.

Definition of the knowledge base containing consumer knowledge. Consumer's knowledge will be described in terms of the set $\mathcal{O}$ of domain ontologies. In particular, consumer knowledge will correspond to instances of concepts in these ontologies and, therefore, this knowledge will be kept within the expressivity of WSML-Core.

In our prototype implementation, KB_c is realized by a $\mathcal{FLORA}$-2 knowledge base (see Chapter 2, Section 2.2.3.7), as depicted in Figure 7.1. Users will indicate to the platform what consumer knowledge is available by providing the location of a file containing all instances which conform consumer's knowledge and this file, together with domain ontologies in $\mathcal{O}$ which provide the domain vocabulary, will be loaded to $\mathcal{FLORA}$-2. Notice that this knowledge can be reused across

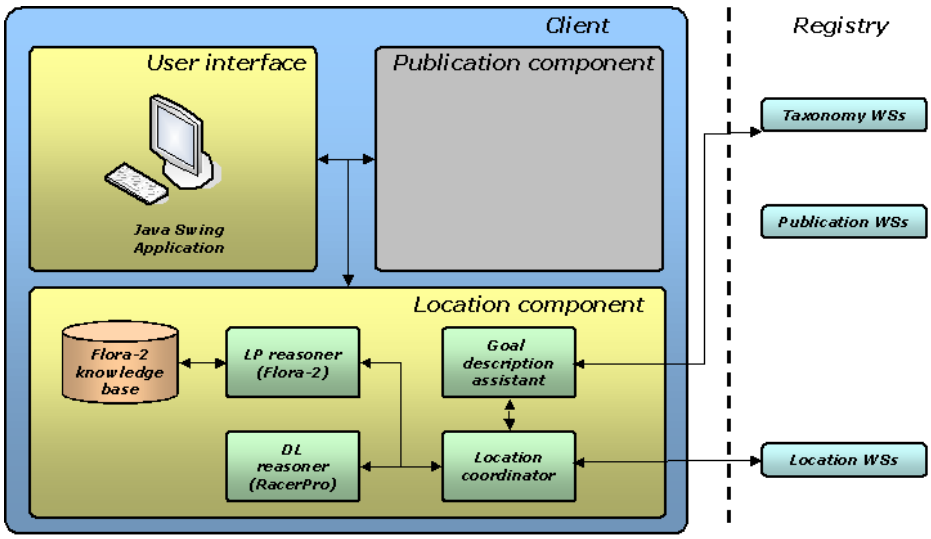

Figure 7.1: Client architecture (location component)

different requests to locate candidate services.

Furthermore, consumer knowledge will be loaded together with the set $\mathcal{O}$ of domain ontologies to a knowledge base, called *CandidateServices*, of the DL reasoner depicted in Figure 7.1 (RacerPro is used). This domain and consumer knowledge will be necessary for the matching of candidate services using one of the filters provided by our platform, as we will see in Section 7.4.2.

In Listing 7.4, an example set of instances modelling consumer knowledge such as a particular flight, personal details, a user registered for www.intair.com, or credit card details, is given. These instances are modelled as an ontology which imports the domain ontologies in $\mathcal{O}$ providing the necessary domain vocabulary. In addition, this knowledge base will also contain domain knowledge given by the set $\mathcal{O}$ of domain ontologies.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-core"

namespace {
        _"http://www.afi.es/KBExample#",
        dc          _"http://purl.org/dc/elements/1.1#",
        foaf  _"http://xmlns.com/foaf/0.1/",
        wsml _"http://www.wsmo.org/wsml/wsml-syntax#",
        loc  _"http://www.afi.es/ontologies/geopolitical/locations#",
        flights   _"http://www.afi.es/ontologies/eTourism/flightsExample#",
        fc  _"http://www.afi.es/ontologies/finance/common#",
        g  _"http://www.afi.es/ontologies/general#",
        users  _"http://www.afi.es/ontologies/users#"}

ontology  _" file://consumerknowledge/myKnowledge"
        nfp
                dc#title  hasValue "Example consumer knowledge base"
```

```
            dc#contributor hasValue _"http://nets.ii.uam.es/~rlara/foaf.rdf"
            dc#date hasValue _date(2007,01,29)
            dc#format hasValue "text/html"
            dc#language hasValue "en-GB"
        endnfp

        importsOntology { _"http://www.afi.es/ontologies/geopolitical/locations",
            _"http://xmlns.com/foaf/0.1",
            _"http://www.afi.es/ontologies/eTourism/flightsExample",
            _"http://www.afi.es/ontologies/finance/common",
            _"http://www.afi.es/ontologies/general",
            _"http://www.afi.es/ontologies/users" }

instance flight1234 memberOf flights#Flight
    hasNumber hasValue "1234"
    hasOrigin hasValue flights#Madrid
    hasDestination hasValue flights#Munich
    hasDate hasValue _date(2007,10,01)
    operatedBy hasValue flights#IntAir

instance rubenLara memberOf g#person
    hasName hasValue "Ruben"
    hasSurname hasValue "Lara"
    hasBirthDate hasValue _date(1979,09,03)
    hasId hasValue "4321"

instance myIntAirUser memberOf users#User
    hasUserName hasValue "ruben_lara"
    hasPassword hasValue "********"
    forAccessOf hasValue "www.intair.com"

instance myDummyCreditCard memberOf fc#DummyCard
    hasHolder hasValue rubenLara
    hasNumber hasValue "8987887"
    hasValidity hasValue _date(2010,01,01)

...
```

Listing 7.4: An example set of instances modelling consumer knowledge

7.2.3 Selection of filters

The last ingredient in the description of a goal is the level of accuracy expected from the results of the location process, i.e., we do not only describe the capability sought and possibly consumer's knowledge, but also requirements on the accuracy and efficiency of the location process. In particular, the consumer can choose to apply different filters to obtain services that can provide the effects required.

Applicable filters are split into two groups, following the abstract model in Chapter 5: 1) filters which are applied by the service registry, without considering what input values can be provided

Applied filter	*Type of description of goal required*	*Type of description of service required*
Textual	Textual description	Textual description
Category	Categorization	Categorization
Capability	Formalization of effects	Formal capability
Input availability	Consumer KB	Information preconditions
Input-dependent effects	Formalization of effects + Consumer KB	Input-dependent effects

Table 7.1: Descriptions required for the application of different filters

by the consumer and, therefore, without evaluating information preconditions and input-dependent effects, and 2) filters whose application must be done once service descriptions are retrieved from the registry, i.e., at the consumer side and considering what information the consumer has available for providing input values to the service.

Consumers will select what filters must be applied for determining candidate services for achieving a goal, and selected filters will be part of the goal and encoded by the *filter* non-functional property of the goal capability (see Listings 7.1, 7.2 and 7.3). In the current platform, the following filters can be applied: a) a textual filter, corresponding to the *Textual* value of the *filter* non-functional property, b) a category filter, indicated by the value *Category* of the *filter* non-functional property, c) a *Capability* filter, d) an *InputAvailability* filter, and e) an *InputDependentEffects* filter. Details of these filters can be found in Sections 7.3 and 7.4.

Different filters will be applied over different types of descriptions of services and goals and, thus, will require the provision of a particular type of description of the goal. These correspondences between filters and types of descriptions required, summarized in Table 7.1, will limit what filters can be applied depending on the type of description of the goal provided and, at the same time, will guide the assistance provided to users for describing their goals, as we will see in the next Section.

7.2.4 Support for the description of goals

Properly describing goals is a necessary task for locating services which provide a given value and, in our platform, we try to ease this task by proposing, when possible, descriptions of the appropriate type to users. The location component of the SETA client, depicted in Figure 7.1, includes a goal description assistant similar to the service description assistant introduced in Chapter 6, Section 6.3.2. The task of the goal description assistant is to support users for describing their goals. Similarly to the service description assistant presented in the previous Chapter, the goal description assistant will receive a WSMO goal description and the type of description required and,

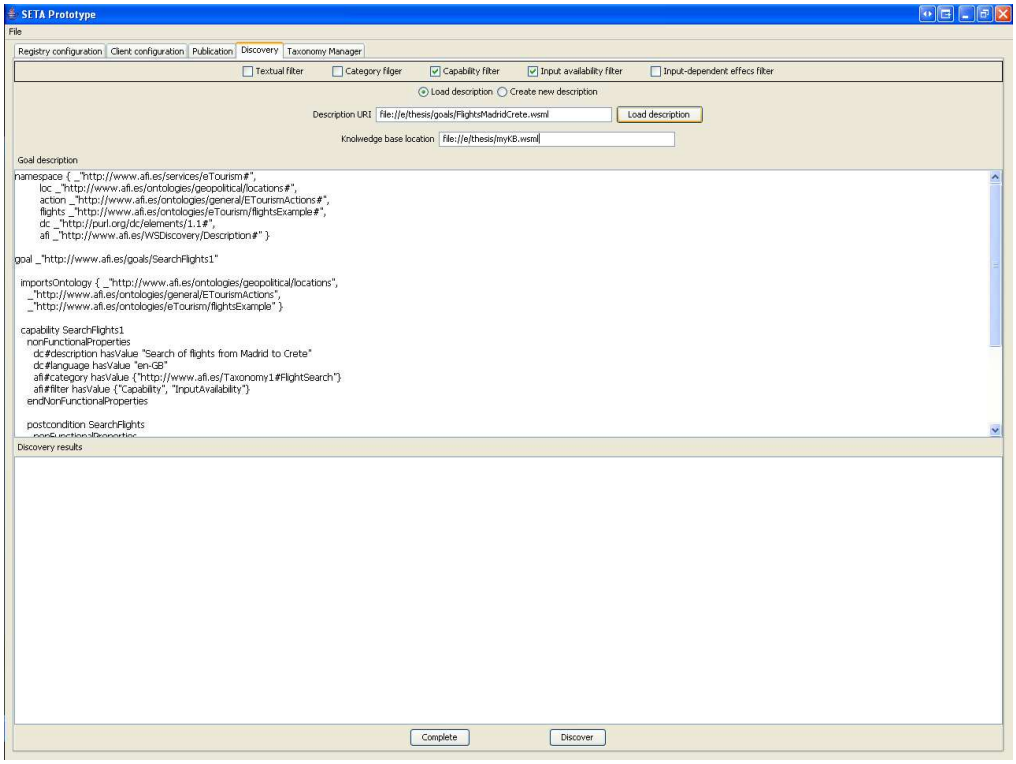

Figure 7.2: Loading a goal description

if possible, return a proposal for this type of description.

A consumer will use the user interface included in the client architecture of Figure 7.1 to provide the description of a goal for whose resolution appropriate services must be located. This user interface, in the current prototype implementation, offers two options to users:

1. Loading an existing WSMO description of the goal from a file (Figure 7.2).

2. Given a URI, creating a new goal description identified by such URI (Figure 7.3). In this case, a template goal description will be built, to be completed by the user.

In both cases, the filters which will be applied for locating relevant services must be selected. If a goal has been loaded from a file, we will check whether this goal includes the selection of filters to be applied and, if so, automatically check the appropriate filters in the user interface shown in Figure 7.2. If filters have not been selected, either because they are not included in the loaded description or because the goal description is created from scratch, the user will have to select what filters will be applied for locating services.

Once we know the filters to be applied, we will evaluate for each selected filter whether a

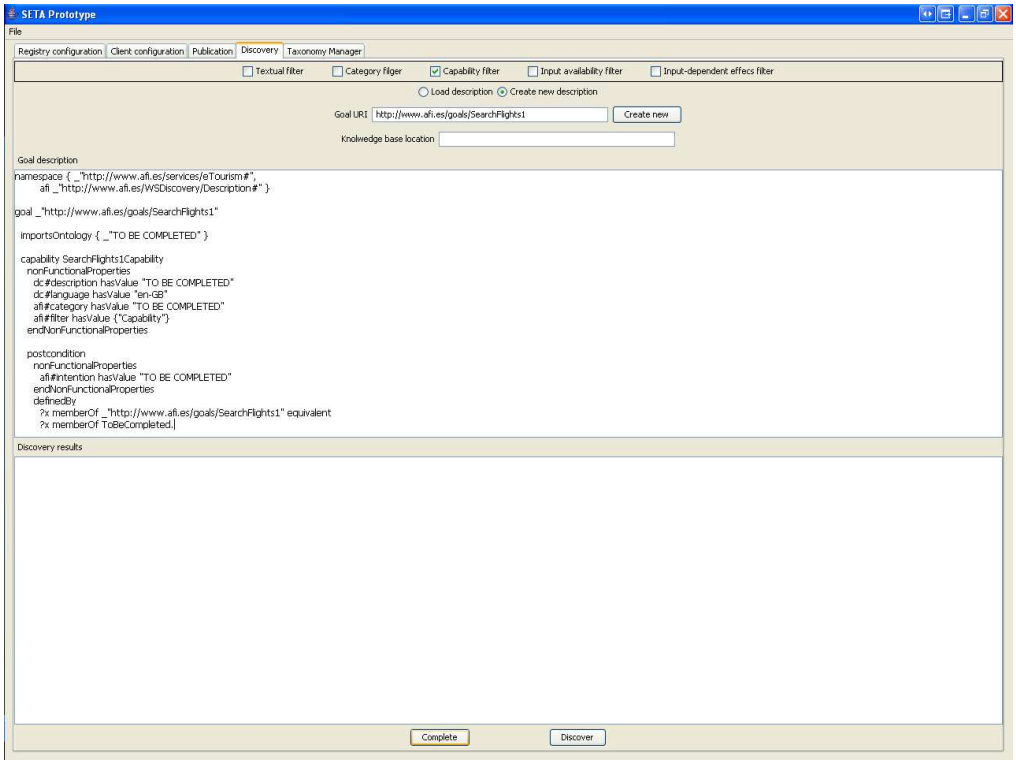

Figure 7.3: Creating a new goal description

description of the goal of the appropriate type for the application of this filter is available and, if not, we will try to assist the user for providing it in the order described below.

Textual filter. If the textual filter has been selected, we need a textual description of the sought capability. If such textual description has not been provided, we will ask the goal description assistant for a proposal for this type of description.

The assistant, if the categories sought services must belong to have been given, will communicate with the taxonomy manager of the registry in order to retrieve the details of these categories and the textual descriptions associated to such categories will be concatenated to build a proposal for the textual description of the goal. Possibly helped by this proposal, the consumer will have to, in any case, provide a textual description of his goal (see Figure 7.4).

After the textual description of the goal has been given, we will add it to the WSMO goal description and continue to the next step.

Category filter. If this filter has been selected, we require the specification of what categories sought services must belong to. If these categories have not been provided, we will ask the assistant

for a proposal, and the assistant will will distinguish two cases:

- If a set-based modelling of the sought capability is available, the assistant will communicate with the taxonomy manager of the registry to search, given the concept formalizing this sought capability, related categories (see Chapter 6, Section 6.3.1). These categories will be proposed to the user as the categories sought services must belong to, grouped into equivalent, more general, more specific, and intersecting categories. It must be noted, though, that this will only be possible if the set-based modelling given only refers to instances in the set $\mathcal{O}$ of domain ontologies shared with the registry, not to instances only in the knowledge base defining consumer knowledge. Otherwise, a pre-processing of the goal is necessary, as it will be discussed in Section 7.3.1, so that this formalization can be correctly interpreted by the taxonomy manager to retrieve relevant categories.
- If no set-based modelling of the sought capability is available but we have a textual description of the goal, we will communicate with the taxonomy manager of the registry to search, given this textual description, related categories.

If both a set-based modelling of the sought capability and a textual description of it are available, the assistant will only use the first type of description to search related categories, as search results will be more accurate than those obtained using a textual description.

Proposed categories will help users in specifying what categories sought services must belong to, but it also might be the case that no category can be proposed as we have no textual or formal description of the goal. Users will have to ultimately choose what categories will describe their goal and, for this purpose, they will be able to browse categories in all available taxonomies and choose the ones which better fit their goals (see Figure 7.5).

After categories have been selected by the consumer, we will update the WSMO goal description and continue to the next step.

Capability filter. If the capability filter has been selected, we require the consumer to provide a set-based modelling of the capability sought. If such formal description has not been provided, we will ask the assistant for a proposal.

If the categories sought services must belong to are described by the WSMO goal, the assistant can propose to the user the capability formalizations associated to the categories selected. If such categories are not provided, but at least a textual description of the goal is available, we will ask the assistant for a proposal of categories as described above so that the user can provide a set of categories for guiding the formal description of the capability required. Still, if categories cannot be proposed, we will enable the manual browsing of categories by users, i.e., for the proposal of the set-based modelling we will always give the chance to users to first select categories (if not given) so

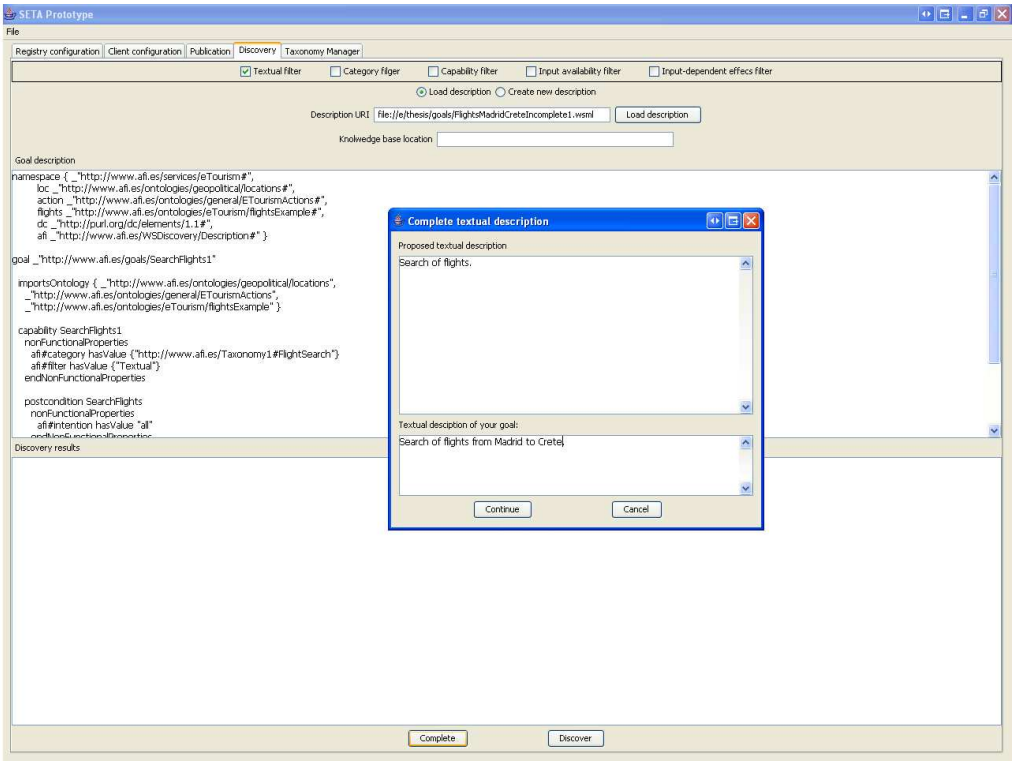

Figure 7.4: Proposal for a textual description of the goal

that we can use these categories, easy to provide by users, to guide the set-based modelling of the sought capability.

Once categories are given, we will eventually ask the assistant for a proposal for the set-based modelling of the effects sought by the consumer. The proposal for the formal description of the capability sought by a consumer will follow the same mechanism used in the previous Chapter for proposing formal capabilities of services starting from their categorization. In particular, if the n categories provided are denoted $c_1, \ldots, c_n$, and their associated capabilities are given by concepts $Eff_{c_1}, \ldots, Eff_{c_n}$ (obtained after communication of the goal description assistant with the taxonomy manager), the concept proposed for the formalization of the capability sought will be:

$Eff_{\mathcal{G}} \equiv Eff_{c_1} \sqcup \ldots \sqcup Eff_{c_n}$

This means that the abstract effects required will be the union of the effects associated to each category sought services must belong to.

Furthermore, users can decide to use the input-dependent effects associated to selected categories *and to parameterize them with particular values*. In this way, users can constrain the general capability associated to a selected category by providing assignments of values to shared variables

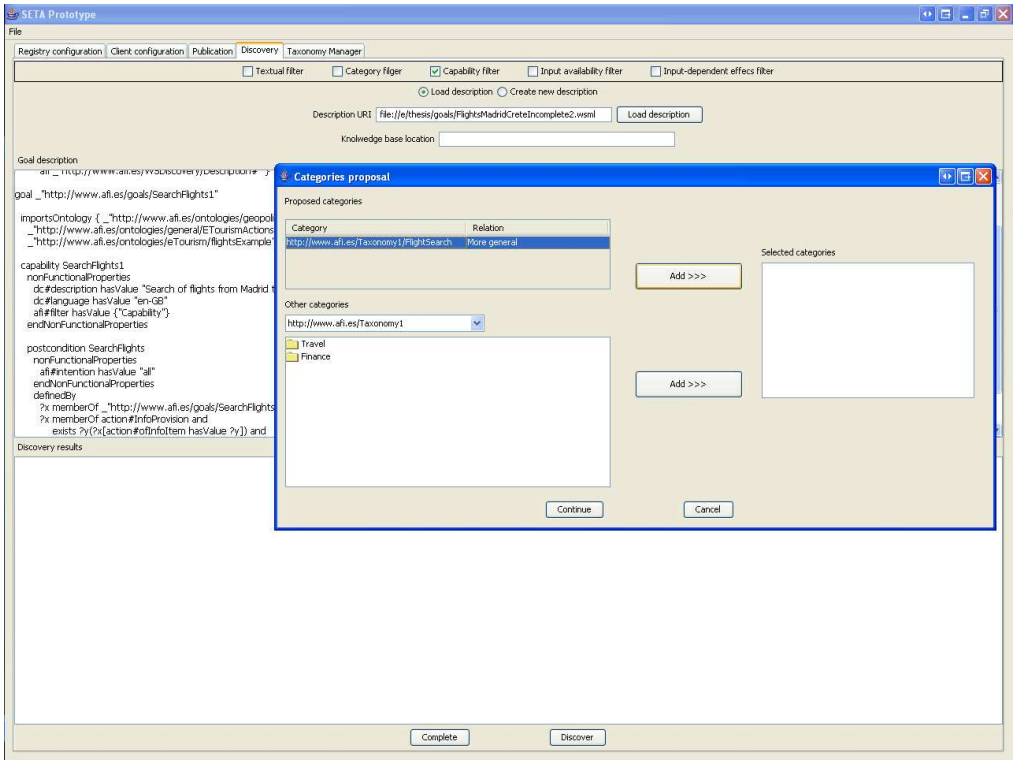

Figure 7.5: Proposal for categories of the goal

defined in the prototypical description of input-dependent effects of such category. Therefore, for any category c_i in the expression proposed above, the formalized capability Eff_{c_i} associated to the category can be replaced by input-dependent effects $InputEff_{c_i,\beta_i}$ of the category, where β_i is a particular input binding defined by the user which assigns values to the shared variables appearing in $InputEff_{c_i}$ (see Figure 7.6).

It can be seen, thus, that prototypical input-dependent effects associated to a category can also serve to help users in further constraining the set-based modelling of their services based on their categorization in a similar way goal ontologies were used in Chapter 4, where users referred to predicates such as $searchTrip(from, to)$ for expressing goals, giving values to the $from$ and to variables. However, in our model instantiation we use this option only as a help to users who might have difficulties in describing their goals, but flexibility for freely providing a set-based modelling of the set of effects achievable by using a service is granted.

Input availability filter. The input availability filter requires the location of a file containing the instances that define available consumer knowledge. This knowledge base will be the source of

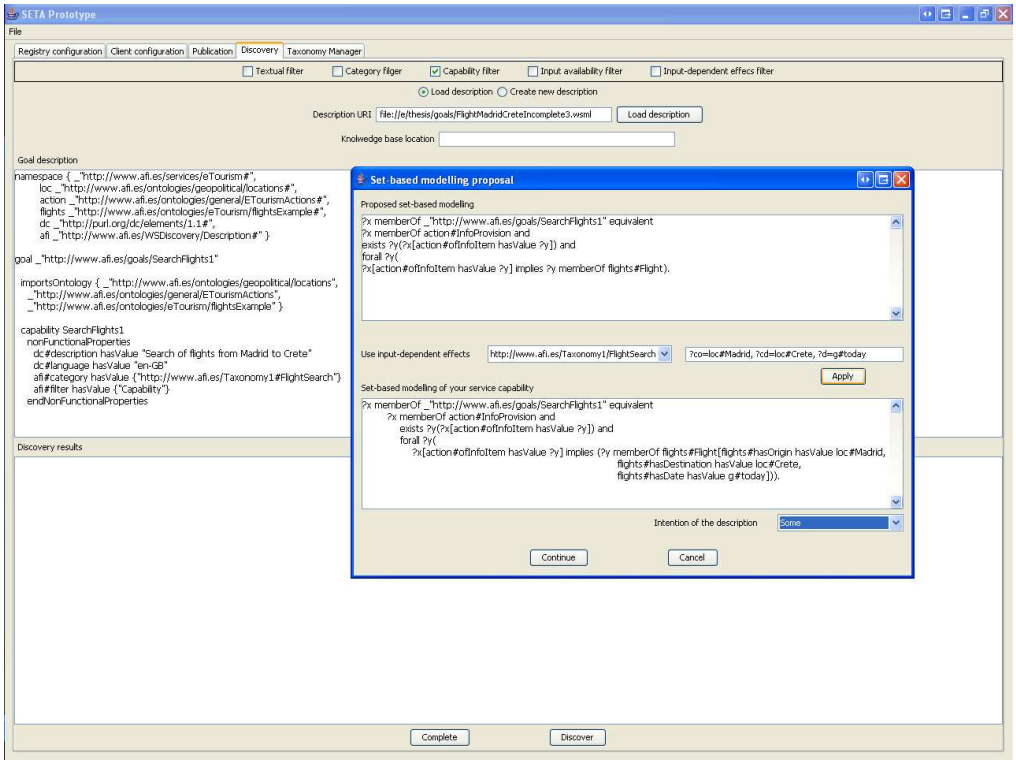

Figure 7.6: Proposal for the set-based modelling of expected effects

possible input values for candidate services and, thus, will be used to determine whether information preconditions of candidate services are fulfilled. In this setting, we will ask users to provide the location of a file defining consumer knowledge.

Input-dependent effects filter. This filter requires the provision of a formal description of the sought capability. Therefore, if this type of description is not available, we will propose a description in the same way we do when the capability filter is selected. Furthermore, if the knowledge base containing consumer knowledge has not been defined yet, we will require the user the location of a file providing such knowledge.

We have seen how consumers will be guided in what type of descriptions they must provide depending on the filters selected and, furthermore, we propose in some cases the description of the goal based on available descriptions of other types. The final result of this phase will be a properly defined goal, for which the types of description provided correspond to the types of descriptions required for the application of selected filters. In the following Sections, we present how this goal, and possibly consumer knowledge, will be used to locate relevant services.

7.3 Registry-side filters

Once a consumer's goal has been described, the filters to be applied selected, and the necessary types of descriptions of the goal for the application of such filters are available, we will start the service discovery process. This process, as described in Chapter 5, will be based on the application of filters to determine which available services, from those whose descriptions have been published, can be used to fulfill the consumer's goal, and it will be split into two phases: a) a first phase in which filters are applied at the registry where service descriptions are stored, and the descriptions of services which can potentially resolve the goal are retrieved, and b) the application of filters over the retrieved service descriptions on the consumer side, where consumer knowledge is accessible. We call the former type of filters registry-side filters, and they will be presented in this Section; the latter type of filters are consumer-side filters, and they will be described in the next Section.

The application of registry-side filters will be the responsibility of the location manager of the registry, depicted in Figure 6.3. In particular, we can communicate with this location manager via a WSDL service which receives the WSMO description of a goal and returns relevant services, grouped according to their degree of relevance as we will see below.

Note, though, that consumers can locate candidate services published to the registry by directly using the inquiry API offered by the UDDI repository used in the registry. This option will not require the consumer to provide a description of his goal in the way we have discussed above, but only the input required by the UDDI API will be required (see [Bellwood et al., 2002]). The purpose of enabling the direct use of the UDDI API is to keep backwards compatibility with current practices, as developers often use this means to search for suitable services in UDDI repositories in their organizations. This is possible as we have published service descriptions to the UDDI repository in a way which enables their retrieval in this way (see Chapter 6, Section 6.3.3). However, the granularity of the search using the UDDI inquiry API will be limited if only UDDI core tModels are used, i.e., conditions over only some parts of the WSDL description of a service can be posed (see [Brittenham et al., 2001] for an overview).

In the following, we concentrate on the retrieval of service descriptions from the registry using the mechanisms designed and implemented on top of the UDDI repository.

7.3.1 Pre-processing of goals and submission to the registry

The location coordinator of the SETA client (see Figure 7.1) is the component in charge of coordinating the service location process. This coordinator will submit the consumer's goal to the location manager of the registry in order to retrieve descriptions of candidate services.

However, if the capability filter has been selected, the goal description must be pre-

processed before it is sent to the registry. In particular, if the concept (or concepts) formalizing the capability sought refer to instances not in the set $\mathcal{O}^{DL}$ of domain ontologies, they must be substituted by their definitions in terms of concepts, relations and instances in domain ontologies. Otherwise, the description will not be processable by the registry, as the registry does not have access to consumer knowledge but only to instances in $\mathcal{O}^{DL}$.

Given a goal $\mathcal{G}$ with a concept $Eff_{\mathcal{G}}$ formalizing a sought capability, we will actually have two concepts for this capability, namely: $Eff_{\mathcal{G}}$ and $Eff_{\mathcal{G}}^{\mathcal{O}^{DL}}$. The former is the original concept given by the service consumer, which might refer to instances in KB_c, while the latter is the result of eliminating any instance in KB_c and replacing it by its definition using only instances instances in $\mathcal{O}^{DL}$. If the capability filter is selected, the goal submitted to the registry will contain the description of concept $Eff_{\mathcal{G}}^{\mathcal{O}^{DL}}$, not of $Eff_{\mathcal{G}}$. It will be the input-dependent effects filter, if selected, the one which will operate over concept $Eff_{\mathcal{G}}$ at the consumer side, where the location process can access consumer knowledge.

Example 7.2 Let us consider the goal in Listing 7.2, whose formalization of the sought capability is given by:

$$\begin{aligned} Eff_{BookAFlight1} \equiv FlightBooking \sqcap \\ \exists ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap onFlight.\{flight1234\}) \sqcap forPerson.\{rubenLara\} \sqcap \\ withPaymentMethod.\{myDummyCreditCard\} \end{aligned}$$

This formalization has details of a flight ($flight1234$), of a person ($rubenLara$), and of a credit card ($myDummyCreditCard$) which are not in domain ontologies $\mathcal{O}^{DL}$. Therefore, we will translate this concept into the following concept, following the definitions of instances $flight1234$, $rubenLara$ and $myDummyCreditCard$ given in Listing 7.4:

$$Eff^{\mathcal{O}^{DL}}_{BookAFlight1} \equiv FlightBooking \sqcap$$
$$\exists ofItem \sqcap$$
$$\forall ofItem.(FlightSeat \sqcap \exists onFlight \sqcap$$
$$\forall onFlight.(Flight \sqcap hasNumber.\{1234\} \sqcap hasOrigin.\{Madrid\} \sqcap hasDestination.\{Munich\} \sqcap$$
$$hasDate.\{20071001\} \sqcap operatedBy.\{IntAir\}) \sqcap \exists forPerson \sqcap$$
$$\forall forPerson.(Person \sqcap hasName.\{Ruben\} \sqcap hasSurname.\{Lara\} \sqcap hasBirthDate.\{19790903\}) \sqcap$$
$$\exists withPaymentMethod \sqcap$$
$$\forall withPaymentMethod.DummyCard$$

In the translated concept we have substituted instances $flight1234$, $rubenLara$ and $myDummyCreditCard$, acting as nominals, by restrictions using their definitions so that only datatype values and instances in $\mathcal{O}^{DL}$ are used. In this case, we have not included neither the id of person $rubenLara$ nor the details of $myDummyCreditCard$ as this is considered sensitive information. The identification of sensitive information and the translation shown is currently done manually. We will discuss in Chapter 8 possible extensions to automate this task.

The goal in Listing 7.1 will also require the substitution of instance $MyEntity$, while the goal in Listing 7.3 only refers to instances in $\mathcal{O}^{DL}$.

□

7.3.2 Textual filter

The first filter available for deciding what services published to the registry are relevant for the goal at hand is the textual filter. The application of this filter will be indicated by the *textual* value of the *filter* non-functional property of the WSMO goal submitted to the location manager of the registry. If this filter has been selected, the location manager will perform a simple keyword matching of the textual description of consumer objectives against the textual descriptions of services published to the registry. In particular, we first remove from the textual description of the goal, given by the Dublin Core description non-functional property of the capability (see Listings 7.1, 7.2 and 7.3), words considered *noise* words. For this purpose, we use the *stop-list* given by Oracle Text[1] (descriptions in English are always assumed and, thus, the set of noise words defined for the English language is used). After noise words have been removed, we will simply match services whose textual description contains any of the keywords remaining in the textual description of the

[1] http://www.oracle.com/technology/products/text/pdf/9ir2text_features_overview.pdf

goal. Therefore, those services whose textual description have some keyword in common with the textual description of the goal will pass the filter.

It must be noted that services for which a textual description is not available will not be matched. Therefore, these services will not pass this filter and they will not be retrieved from the registry as services relevant for the goal at hand. However, we expect most services to incorporate a textual description, as it is easy to provide by users and, furthermore, support for providing it is offered by the service description assistant presented in the previous Chapter.

As it can be seen, the textual matching used has been kept very simple. The reason is that having an elaborated textual matching has not been the main concern of our prototype implementation, as specific tools which can facilitate this task already exist. Tools such as Zettair[2], the text matching capabilities provided by major RDBMSs such as Oracle, SQL Server or MySQL, or textual matching mechanisms such as those proposed by [Klusch et al., 2006] could be used. Its incorporation to our registry in order to provide enhanced textual matching, will be part of our future work. Furthermore, we currently do not use the Dublin Core *language* non-functional property in our textual matching mechanism. Explicitly dealing with goal and service descriptions in different languages will be part of our future work.

The application of this filter will provide results in very short times, as only simple queries over the registry database will suffice to find relevant services (see Chapter 8 for further details). Furthermore, this filter only requires a type of description of services and goals which is easy to provide by users. However, the precision of the results provided is limited, and we currently do not distinguish different levels of match (all services that pass the filter are considered a perfect match). We will, therefore, have a single set of filters which pass the filter for a goal $\mathcal{G}$, denoted $\mathcal{S}_{\mathcal{G}}^{textual}$, as this filter currently works as a boolean filter.

Example 7.3 If we consider the goal in Listing 7.2, whose textual description is *"Booking of a seat on flight fight1234 from Madrid to Munich, paid with a DummyCard"*, and the service given in the previous Chapter in Listing 6.4, whose textual description is *"Booking of flights operated by IntAir"*, we will have a match, as the keyword "booking" appears on both descriptions.

However, notice that with our simple keyword matching, we can have problems with singular and plural (e.g. "flight" vs "flights" in the above description), as we only match exact keywords. Furthermore, we equally match a service containing the keyword "booking" than a service containing this keyword and the keyword "flight". The future incorporation of more powerful text matching tools to our platform will yield a more accurate textual matching.

□

[2] http://www.seg.rmit.edu.au/zettair/

7.3.3 Category filter

If the category filter is selected, indicated by the value *category* of the *filter* non-functional property of the WSMO goal capability, the location manager will extract the categories sought services must belong to and will search services which have been categorized under all these categories or parents of these taxonomies.

Given a goal $\mathcal{G}$ for which the set of categories $C_{\mathcal{G}} = \{c_1, \ldots, c_n\}$ have been given as the value of the *category* non-functional property of the WSMO goal capability, we will use the structure of taxonomies loaded in memory (see Chapter 6, Section 6.3.3) to, for each category $c_i \in C_{\mathcal{G}}$, find the set:

$$\mathcal{S}_{c_i}^{\subseteq} = \{(serv \mid serv \in \mathcal{S}_{c_i}) \cup (serv \mid serv \in \mathcal{S}_{c_j}, c_i \subset c_j)\}$$

where $\mathcal{S}_{c_i}$ denotes the set of services categorized under category c_i, and $c_i \subset c_j$ means that c_j is an ancestor category of c_i in the taxonomy $\mathcal{T}$ they both belong to. Intuitively, the set $\mathcal{S}_{c_i}^{\subseteq}$ contains all published services which have been categorized under category c_i or under any category which is an ancestor of such category.

We will also find, for each category $c_i \in C_{\mathcal{G}}$, the set:

$$\mathcal{S}_{c_i}^{\supset} = \{serv \mid serv \in \mathcal{S}_{c_j}, c_i \supset c_j\}$$

Intuitively, the set $\mathcal{S}_{c_i}^{\supset}$ contains those services which are categorized under a category which is a descendant of category c_i.

Once we have find the sets $\mathcal{S}_{c_i}^{\subseteq}$ and $\mathcal{S}_{c_i}^{\supset}$ for each category $c_i \in C_{\mathcal{G}}$ by exploring the taxonomies of categories loaded in memory, which contain for each category c_i the set $\mathcal{S}_{c_i}$ of services associated to such category, we will define the set of *perfect matches* for the goal, denoted $\mathcal{S}_{\mathcal{G}}^{catMatch}$, as follows:

$\mathcal{S}_{\mathcal{G}}^{catMatch} = \bigcap_{c_i \in C_{\mathcal{G}}} \mathcal{S}_{c_i}^{\subseteq}$

Intuitively, the expression above means that we will consider perfect matches under the category filter those services which have been categorized under *all* the categories given by the goal, or under ancestors of such categories. We also consider ancestors of such categories under the assumption that categories which are ancestors of a given category correspond to more general capabilities, and services categorized under a more general category will provide a capability which encompass the capability associated to a more specific category. For example, if we have a category *FlightBooking*, defined as more general than a category *LowCostFlightBooking*, services under the former category are assumed to be able to offer the booking of seats on flights operated by both regular and low-cost airlines.

We now define the set of *partial matches* for the goal $\mathcal{G}$ above, denoted $\mathcal{S}_{\mathcal{G}}^{catPMatch}$, as follows:

$\mathcal{S}_{\mathcal{G}}^{catPMatch} = (\bigcup_{c_i \in C_{\mathcal{G}}} \mathcal{S}_{c_i}^{\subseteq} \cup \mathcal{S}_{c_i}^{\supseteq}) - \mathcal{S}_{\mathcal{G}}^{catMatch}$

where the - symbol denotes set difference. Intuitively, this means that we consider a partial match all services which are categorized under *at least one* of the categories given by the goal, or under ancestors or descendants of such categories, but those services which are a perfect match.

Therefore, two types of services will pass this filter: services to which all the categories given by the goal apply, and services to which only some of these categories apply. Services not categorized will not pass the filter, as we cannot determine whether they are relevant or not for solving the goal.

Response times for the application of this filter will be low, as we only have to explore the taxonomies of categories loaded in memory by our registry, which already contain the set of services categorized under each category. While results will be provided in short times, and based on the use of categories intuitive for end-users, they will be coarse-grained, as the granularity of results will be limited by the granularity of the categories published to the registry and managed by the taxonomy manager.

Example 7.4 Let us consider the goal in Listing 7.3, for which the category *http://www.afi.es/Taxonomy1#FlightSearch* has been given, and service *http://www.afi.es/services/LowCostFlightSearch1* in Listing 6.3, categorized under *http://www.afi.es/Taxonomy1#LowCostFlightSearch*.

This service belongs to the set $\mathcal{S}^{\supseteq}_{http://www.afi.es/Taxonomy1\#FlightSearch}$, as category *http://www.afi.es/Taxonomy1#LowCostFlightSearch* is a descendant of category *http://www.afi.es/Taxonomy1#FlightSearch*. Therefore, this service will be a partial match for the goal.

Let us now consider the goal in Listing 7.1, for which the category *http://www.afi.es/Taxonomy1#InvestmentFundSearch* is given, and the service in Listing 6.2, categorized under the same category. In this case, we have a perfect match, as the service belongs to the set $\mathcal{S}^{\subseteq}_{http://www.afi.es/Taxonomy1\#InvestmentFundSearch}$ for the only category given by the goal. □

7.3.4 Capability filter

If the capability filter is selected, the concept formalizing the set of effects required by the consumer, encoded by the capability postcondition of the WSMO goal, will be used for matching services whose set-based modelling of their capability relates in some way to such concept. In the following, we discuss the types of match identified and how services are actually matched.

Intention of $\mathcal{G}$ / *serv*	$I_{serv} = all$		$I_{serv} = some$	
$I_{\mathcal{G}} = all$	$Eff_{\mathcal{G}} = Eff_{serv}$	Match	$Eff_{\mathcal{G}} = Eff_{serv}$	PMatch
	$Eff_{\mathcal{G}} \subset Eff_{serv}$	Match	$Eff_{\mathcal{G}} \subset Eff_{serv}$	poMatch
	$Eff_{\mathcal{G}} \supset Eff_{serv}$	PMatch	$Eff_{\mathcal{G}} \supset Eff_{serv}$	PMatch
	$Eff_{\mathcal{G}} \cap Eff_{serv} \neq \emptyset$	PMatch	$Eff_{\mathcal{G}} \cap Eff_{serv} \neq \emptyset$	pPMatch
	$Eff_{\mathcal{G}} \cap Eff_{serv} = \emptyset$	Nomatch	$Eff_{\mathcal{G}} \cap Eff_{serv} = \emptyset$	Nomatch
$I_{\mathcal{G}} = some$	$Eff_{\mathcal{G}} = Eff_{serv}$	Match	$Eff_{\mathcal{G}} = Eff_{serv}$	Match
	$Eff_{\mathcal{G}} \subset Eff_{serv}$	Match	$Eff_{\mathcal{G}} \subset Eff_{serv}$	poMatch
	$Eff_{\mathcal{G}} \supset Eff_{serv}$	Match	$Eff_{\mathcal{G}} \supset Eff_{serv}$	Match
	$Eff_{\mathcal{G}} \cap Eff_{serv} \neq \emptyset$	Match	$Eff_{\mathcal{G}} \cap Eff_{serv} \neq \emptyset$	poMatch
	$Eff_{\mathcal{G}} \cap Eff_{serv} = \emptyset$	Nomatch	$Eff_{\mathcal{G}} \cap Eff_{serv} = \emptyset$	Nomatch

Table 7.2: Types of match in terms of the set-theoretic relations and intentions

7.3.4.1 Types of match

Given a service *serv* and a goal $\mathcal{G}$, we consider this service and this goal to match to some extent if there is some relation between the set Eff_{serv} of effects offered by the service and the set $Eff_{\mathcal{G}}$[3] of effects required by the goal. We therefore expect that some set-theoretic relations between the set formalized by Eff_{serv} and the set formalized by $Eff_{\mathcal{G}}$ exists. The most basic set-theoretic relationships that might be considered are the following: $Eff_{serv} = Eff_{\mathcal{G}}$, $Eff_{serv} \subset Eff_{\mathcal{G}}$, $Eff_{serv} \supset Eff_{\mathcal{G}}$, $Eff_{serv} \cap Eff_{\mathcal{G}} \neq \emptyset$, and $Eff_{serv} \cap Eff_{\mathcal{G}} = \emptyset$.

What set-theoretic relation holds between the sets of offered and required effects will condition whether a goal and a service match, and to what extent. However, the intentions of these sets (I_{serv} for service *serv* and $I_{\mathcal{G}}$ for goal $\mathcal{G}$) also play a role in deciding the type of match between the service and the goal [Keller et al., 2006a]. Table 7.2 summarizes what type of match we will consider to hold between a service and a goal depending on the set-theoretic relation between the formalized sets of effects and the respective intentions of these sets. The explanation of each particular type of match and the cases where it occurs is given in the following.

Perfect match. A perfect match (Match) means that *all* the effects required by the consumer can be provided by the service. We explain below the entries in the Table where this match holds:

- ($I_{\mathcal{G}} = all$, $I_{serv} = all$) If all the effects in the set provided by the goal are required, and the service offers all the effects described, we will have a guarantee that all the effects requested

[3] In order to improve readability, in the following we will write $Eff_{\mathcal{G}}$ but we actually refer to $Eff_{\mathcal{G}}^{\mathcal{O}^{DL}}$, i.e., to the pre-processed concept which only refers to instances in the set $\mathcal{O}^{DL}$ of domain ontologies.

by the goal can be achieved only if the set of effects offered by the service is equivalent to ($Eff_{\mathcal{G}} = Eff_{serv}$) or a superset of ($Eff_{\mathcal{G}} \subset Eff_{serv}$) the set of effects required.

- ($I_{\mathcal{G}} = some$, $I_{serv} = all$) In this case, as the requester goal is satisfied if some of the effects in the set given can be provided by the service, and as the service declares it can provide all the effects in the set described, the service will satisfy the goal if it offers some effects in the set described by the goal, i.e., if both sets intersect.

- ($I_{\mathcal{G}} = some$, $I_{serv} = some$) If the consumer requires some effects (at least one) in the set described, and the service only offers some of the effects associated to its capability, we will have a perfect match only if both sets are equivalent or the set of offered effects is a superset of the set required effects. In all other cases, and even though these sets intersect, it can happen that the service actually offers only effects outside this intersection and, thus, we cannot consider them a perfect match.

Partial match. A partial match (PMatch) means that only part of the effects required by the consumer are offered by the service:

- ($I_{\mathcal{G}} = all$, $I_{serv} = all$) In this situation, we will have a partial match if the set of effects offered by the service intersects but is neither a superset not an equivalent set to the set of effects requested. The reason is that, in these cases, the service can provide only part of all the effects required by the consumer.

- ($I_{\mathcal{G}} = all$, $I_{serv} = some$) In this case, if the set of offered effects is equivalent to or a subset of the set of required effects, we will have a partial match as we can guarantee that the service can provide some of the effects required, but not all of them. The case in which the set of offered effects is a superset of the set of required effects is not a partial match, as the effects actually offered by the service might be in the difference between the set of offered effects and the set of requested effects.

Possible match. A possible match (poMatch) means that there might be a perfect match, but we cannot guarantee it:

- ($I_{\mathcal{G}} = all$, $I_{serv} = some$) If the set of required effects is a subset of the set of offered effects, there might happen that the part of the effects actually offered by the service is equivalent to or a superset of the set of required effects, but we cannot guarantee it as we only know that *some* of the effects in Eff_{serv}, i.e., a subset of Eff_{serv}, will be provided, but we do not know what exact subset this will be.

- ($I_{\mathcal{G}} = some$, $I_{serv} = some$) In this situation, we will have a possible match if the set of required and offered effects intersect, but the latter is neither equivalent to nor a superset of the former. If this happens, it might be the case that the actual set of effects the service can provided, a subset of Eff_{serv}, have elements in common with the set $Eff_{\mathcal{G}}$ from which some effects are required, but we have no guarantee of it.

Possible partial match. A possible partial match (pPMatch) refers to the situation where there might be a partial match between the set of effects required by the goal and offered by the service, but we have no guarantee of it and it can turn out to be a non-match:

- ($I_{\mathcal{G}} = all$, $I_{serv} = some$) In this setting, if we only have that the sets of relevant effects for the consumer and the provider intersect, there can be a partial match if the subset of Eff_{serv} actually offered by the service contains elements in $Eff_{\mathcal{G}}$. Otherwise, we will have a non-match, i.e., the service will not be able to provide any effect from the set required by the consumer.

Non-match. A non-match (Nomatch) refers to cases in which the service does not offer any of the effects required by the goal, i.e., in which the service cannot contribute at all to solve the consumer's goal.

We can establish a partial ordering between the different notions of match as follows:
Match $\preceq$ poMatch $\preceq$ PMatch $\preceq$ pPMatch $\preceq$ Nomatch
where a perfect match (Match) is the most preferred notion by users. We rank a possible match (poMatch) better than a partial match (PMatch) as, in the best case, a possible match will correspond to cases in which the goal is completely fulfilled (a perfect match) and, in the worst case, to cases in which only part of the effects required can be provided (a partial match).

7.3.4.2 Matching services

Given the notions of match introduced above and the criteria for deciding when we have each particular type of match between a service and a goal, we will now discuss some aspects of using existential intentions for service capabilities, and we will present how we use DL reasoning in our platform for finding published services which are relevant for a given goal.

Existential intentions for services. If we look at the first and second columns of Table 7.2, which correspond to service descriptions with a universal and existential intention, respectively, we can see how, for the same set-theoretic relation between the goal's set of effects and the service's

set of effects, services with a universal intention are always *an equal or better match* for the goal following the partial ordering of matching notions given above.

Furthermore, remember from Chapter 5 that due to different reasons such as the dynamics of services, the excessive description effort required, and the reticence to disclose sensitive information, we expect service descriptions to be complete but not necessarily correct, i.e., we expect services to declare all the effects they can actually provide, but there might declare extra effects they cannot actually provide at a given point in time.

Given these observations, let us now consider the following example:

Example 7.5 Let us imagine two different services which enable access to the same capability described for the service in Listing 6.4, i.e., they both offer the booking of flights operated by IntAir. However, strictly speaking they will not offer the booking of all flights, but only of those flights for which there are still seats available.

In this setting, if one of the services associates an existential intention to its capability, and the other one a universal intention, for the same goal *the service with the universal intention will always be considered an equal or better match than the service with an existential intention.*

□

We can see from the example above that service providers might tend to associate a universal intention to their descriptions in order to increase the chances of their services of being considered a better match for a goal than other published services. However, and while we admit descriptions of the set of effects offered by a service which declare effects they might not actually provide, we expect providers who cannot guarantee the correctness of such descriptions to declare an existential intention so that this lack of guarantee is made explicit.

In addition, it is important to notice that service providers will not gain anything but failed interactions from declaring on purpose effects they know they can never provide or from associating a false universal intention to their capabilities, as during service contracting the consumer will determine that the service is actually not able to provide the required effects. Furthermore, reputation mechanisms could be used for detecting providers which systematically "inflate" the description of effects their services can provide. However, this is subject of future work.

Matching published services. The set-theoretic relations used in Table 7.2 over the set of effects requested and offered by goals and services, respectively, actually boil down to satisfiability relations between the DL concepts formalizing such sets.

Therefore, given a goal $\mathcal{G}$ which formalizes a capability sought, the location manager of the registry will query the *Services* TBox of the DL reasoner for concepts:

- synonyms of (equivalent to) concept $Eff_{\mathcal{G}}$ formalizing the set of effects required, denoted $\mathcal{S}^{\equiv}$,

- ancestors of (subsuming) concept $Eff_{\mathcal{G}}$, denoted $\mathcal{S}^{\sqsubset}$,
- descendants of (subsumed by) concept $Eff_{\mathcal{G}}$, denoted $\mathcal{S}^{\sqsupset}$,
- synonyms of (equivalent to) concept $not(Eff_{\mathcal{G}})$, denoted $\mathcal{S}^{not(\equiv)}$,
- descendants of (subsumed by) concept $not(Eff_{\mathcal{G}})$, denoted $\mathcal{S}^{not(\sqsupset)}$.

Given these sets of retrieved concepts, we will retrieve the intention which was associated to the services whose capability they formalize. With these data, we define the following sets:

$$\mathcal{S}_{\forall}^{\equiv} = \{serv \mid serv \in \mathcal{S}^{\equiv}, I_{serv} = all\}$$

$$\mathcal{S}_{\exists}^{\equiv} = \{serv \mid serv \in \mathcal{S}^{\equiv}, I_{serv} = exists\}$$

$$\mathcal{S}_{\forall}^{\sqsubset} = \{serv \mid serv \in \mathcal{S}^{\sqsubset}, I_{serv} = all\}$$

$$\mathcal{S}_{\exists}^{\sqsubset} = \{serv \mid serv \in \mathcal{S}^{\sqsubset}, I_{serv} = exists\}$$

$$\mathcal{S}_{\forall}^{\sqsupset} = \{serv \mid serv \in \mathcal{S}^{\sqsupset}, I_{serv} = all\}$$

$$\mathcal{S}_{\exists}^{\sqsupset} = \{serv \mid serv \in \mathcal{S}^{\sqsupset}, I_{serv} = exists\}$$

$$\mathcal{S}_{\forall}^{\sqcap} = \{serv \mid serv \in (\mathcal{S} - (\mathcal{S}^{not(\equiv)} \cup \mathcal{S}^{not(\sqsupset)}), I_{serv} = all\}$$

$$\mathcal{S}_{\exists}^{\sqcap} = \{serv \mid serv \in (\mathcal{S} - (\mathcal{S}^{not(\equiv)} \cup \mathcal{S}^{not(\sqsupset)}), I_{serv} = exists\}$$

where $\mathcal{S}$ denotes the complete set of published services and the - sign denotes set difference. It must be noted that the last two queries retrieve concepts not intersecting $Eff_{\mathcal{G}}$, so that we can

find intersecting concepts ($\mathcal{S}^{\sqcap}_{\forall}$ and $\mathcal{S}^{\sqcap}_{\exists}$) by taking the difference between all published services and the concepts returned by these queries.

With the sets above defined, we will be able to determine what services are: a) a perfect match, b) a possible match, c) a partial match, or d) a possible partial match for the goal. Perfect matches for goal $\mathcal{G}$ are defined by:

$$\mathcal{S}^{capMatch}_{\mathcal{G}} = \mathcal{S}^{\sqcap}_{\forall} \cup \mathcal{S}^{\equiv}_{\exists} \cup \mathcal{S}^{\sqsupset}_{\exists}$$

if $I_{\mathcal{G}} = some$, and

$$\mathcal{S}^{capMatch}_{\mathcal{G}} = \mathcal{S}^{\equiv}_{\forall} \cup \mathcal{S}^{\sqsubset}_{\forall}$$

if $I_{\mathcal{G}} = all$.

Possible matches are defined by:

$$\mathcal{S}^{cappoMatch}_{\mathcal{G}} = \mathcal{S}^{\sqcap}_{\exists} - (\mathcal{S}^{\equiv}_{\exists} \cup \mathcal{S}^{\sqsupset}_{\exists})$$

if $I_{\mathcal{G}} = some$, and

$$\mathcal{S}^{cappoMatch}_{\mathcal{G}} = \mathcal{S}^{\sqsubset}_{\exists}$$

if $I_{\mathcal{G}} = all$.

Partial matches are defined by:

$$\mathcal{S}^{capPMatch}_{\mathcal{G}} = (\mathcal{S}^{\sqcap}_{\forall} - (\mathcal{S}^{\equiv}_{\forall} \cup \mathcal{S}^{\sqsubset}_{\forall})) \cup \mathcal{S}^{\equiv}_{\exists} \cup \mathcal{S}^{\sqsupset}_{\exists}$$

if $I_{\mathcal{G}} = all$.

Possible partial matches are defined by:

$$\mathcal{S}^{cappPMatch}_{\mathcal{G}} = \mathcal{S}^{\sqcap}_{\exists} - (\mathcal{S}^{\equiv}_{\exists} \cup \mathcal{S}^{\sqsubset}_{\exists} \cup \mathcal{S}^{\sqsupset}_{\exists})$$

if $I_{\mathcal{G}} = all$.

Example 7.6 Let us consider the goal in Listing 7.2 of booking a seat on flight *flight1234* from Madrid to Munich, paid with a credit card *myDummyCreditCard*, let us suppose a capability filter

would have been selected, and let us further suppose the translation presented in Example 7.2 to remove instances not in domain ontologies from the capability formalization has been done.

Let us now consider service *http://www.afi.es/services/BookIntAirFlight1*, described in Listing 6.4 have been previously published to the registry.

In this setting, if we use concept $Eff^{\mathcal{O}^{DL}}_{BookAFlight1}$ to query for related concepts, we will obtain that concept $Eff_{BookIntAirFlight1}$ corresponding to service *http://www.afi.es/services/BookIntAirFlight1* will be returned as a concept more general than concept $Eff^{\mathcal{O}}_{BookAFlight1}$. Given that the goal considered has an existential intention, as well as the formalization of the service capability, service *http://www.afi.es/services/BookIntAirFlight1* will be a possible match for it, as we do not have a guarantee that the service will be able to provide the particular flight sought. However, if the service would have associated a universal intention, it would be a perfect match for the goal.

□

Example 7.7 Let us now consider the goal defined in Listing 7.3. If service *http://www.afi.es/services/LowCostFlightSearch1* given in Listing 6.3 has been published to the registry and we query for concepts related to $Eff_{SearchFlights1}$, we will have that concept $Eff_{LowCostFlightSearch1}$ is a concept intersecting, but not a superconcept or a subconcept, of $Eff_{SearchFlights1}$. Therefore, and given that the sought capability has a universal intention and the service an existential intention, this service is only a possible partial match for the goal. If the service would have a universal intention associated to its capability formalization, it would still be only a partial match, as only information about part of the flights required by the consumer can be provided.

□

As the TBox queried for retrieving matching services has been already classified when publishing services, and as the definition of service capabilities and required effects is restricted to the $\mathcal{SHOIN}$ DL, existing DL reasoners can provide responses to the queries over the TBox in relatively low times. More details on the complexity of applying this filter and experimental response times measured can be found in Chapter 8.

In general, the application of the capability filter will be more time-consuming than the other registry-side filters previously presented. Furthermore, it relies on formal descriptions of both services and goals which are more difficult to provide by users, although this difficulty is reduced by the use of the description assistants presented. However, the set-based modelling, with formal semantics, of the capabilities offered by services and sought by consumers, can be much more accurate than the textual description or the categorization of a service or goal, and the results obtained from the application of this filter will be thus more accurate.

7.3.5 Combination of registry-side filters

The three filters presented above operate over different types of descriptions of services and goals, whose provision presents different levels of difficulty for users, and they have different properties in terms of the expected accuracy of goal and service matching and the efficiency of this matching. In principle, any combination of these filters can be selected by consumers to be applied over published service descriptions for determining what services are relevant for achieving the goal given. In general, the location manager will combine the results of applying the different filters.

Given a goal $\mathcal{G}$, perfect matches will be given by:

$$\mathcal{S}_{\mathcal{G}}^{Match} = \mathcal{S}_{\mathcal{G}}^{textual} \cap \mathcal{S}_{\mathcal{G}}^{catMatch} \cap \mathcal{S}_{\mathcal{G}}^{capMatch}$$

Possible matches will be given by:

$$\mathcal{S}_{\mathcal{G}}^{poMatch} = \mathcal{S}_{\mathcal{G}}^{textual} \cap \mathcal{S}_{\mathcal{G}}^{catMatch} \cap \mathcal{S}_{\mathcal{G}}^{cappoMatch}$$

Possible partial matches are given by:

$$\mathcal{S}_{\mathcal{G}}^{pPMatch} = \mathcal{S}_{\mathcal{G}}^{cappPMatch}$$

Partial matches are given by:

$$\mathcal{S}_{\mathcal{G}}^{PMatch} = (\mathcal{S}_{\mathcal{G}}^{textual} \cup \mathcal{S}_{\mathcal{G}}^{catMatch} \cup \mathcal{S}_{\mathcal{G}}^{catPMatch} \cup \mathcal{S}_{\mathcal{G}}^{capMatch} \cup \mathcal{S}_{\mathcal{G}}^{cappoMatch} \cup \mathcal{S}_{\mathcal{G}}^{capPMatch}) - (\mathcal{S}_{\mathcal{G}}^{Match} \cup \mathcal{S}_{\mathcal{G}}^{poMatch})$$

In all the expressions above, if a particular filter has not been selected for application, the set of services being a match under this filter will not be considered, i.e., it will be removed from the expression.

Intuitively, perfect matches will be those services which perfectly match the goal under all filters, and possible matches those services which perfectly match the goal under the textual and category filters but a possible match under the capability filter (if applied). Possible partial matches will only be those services which are a possible partial match under the capability filter and, therefore, this set can contain services only if the capability filter is applied. Finally, partial will be all services matching in some way the goal but those perfectly matching it and those which are a possible perfect match.

While nothing prevents consumers from selecting any combination of the registry-side filters offered, we believe only one combination, besides the stand-alone application of each of the filters, makes sense from a practical perspective, namely: the combination of the textual and the category filter. The reason is that categories are coarse-grained and leave little flexibility to users for further narrowing down the description of their goals. By selecting categories sought services must belong to and, at the same time, specifying relevant keywords, users can express with more detail their needs.

The combination of a category filter and a capability filter is not expected to be used, as if the user is able to formally describe the capability sought, which enables considerable accuracy in reflecting user's needs, it is unlikely he will also want to filter services using coarse-grained categories. We only envision practical reasons for combining these two filters if we would have a distributed registry, and each peer registry would only contain services categorized under a given category, in a similar way [Verma et al., 2005] proposes. In this setting, we would use categories to decide on what registries to query, and we would then query these registries for services related to the goal in terms of their formalized capability.

Finally, the combination of a textual and a capability filter is unlikely, as it means combining an imprecise type of description and matching with a formal and precise type of description and filter. In practice, given that partial matches must be obtained, it does not matter in what order filters are applied when more than one is selected, as we will have to explore the results of the stand-alone application of each filter, i.e., we cannot reduce the number of services to which a given filter is applied by applying before another filter.

After the application of the filters selected, we will have the set of services which are a (perfect, possible perfect, partial, or possible partial) match for the goal given. The location manager of the registry will retrieve from the UDDI repository the complete descriptions of these services and return the service descriptions, grouped according to their degree of match, to the SETA client, which will possibly apply other filters over these descriptions as we describe in the next Section.

7.4 Consumer-side filters

After applying registry-side filters, the description of services that are candidates for (totally or partially) fulfilling a consumer goal are retrieved from the registry and available to the SETA client. In particular, four sets of descriptions are retrieved, corresponding to perfect, possible perfect, partial, and possible partial matches for the goal.

If no consumer-side filter is selected, the location coordinator will directly provide these

service descriptions to the consumer for their selection. Otherwise, selected consumer-side filters will be applied in order to further filter what services are considered a match. In the following, we present the two consumer-side filters available in the current platform, namely: the input availability filter (Section 7.4.1), and the input-dependent effects filter (Section 7.4.2). However, notice that new filters could be introduced and that the consumer can also choose to apply custom filters defined outside the platform, as he already has available the full description of relevant services. Envisioned extensions and modifications of these filters will be outlined in Chapter 8.

7.4.1 Input availability filter

Besides checking whether a service is relevant for solving a consumer's goal in terms of the capability it offers, the consumer might want to know whether the information preconditions of the service can be fulfilled, i.e., whether the consumer has appropriate information available to be submitted to the service as valid input values. For this evaluation, we will make use of information preconditions defined by services retrieved from the registry and of consumer knowledge given by a knowledge base KB_c as defined in Section 7.2.2.

7.4.1.1 Querying for valid input values

Remember from Chapter 6 that information preconditions were described as WSML-Flight queries, i.e., as Datalog queries possibly with inequality and negation (see Chapter 2, Section 2.2.5.3), and that knowledge base KB_c contains instances of WSML-Core ontologies. In this setting, the location coordinator of the SETA client will, for each service retrieved from the registry, use its preconditions as queries to the knowledge base containing consumer knowledge, i.e., we query for valid input values for the service.

If *serv* is a service retrieved from the registry with input variables $i_1, \ldots, i_n$, we will issue the WSML-Flight query $conditions_{serv}(i_1, \ldots, i_n)$ defining the information preconditions of the service to knowledge base KB_c. Answers to this query correspond to valid input bindings for the service given the consumer knowledge contained in KB_c, i.e., answers to the query correspond to assignments of instances in KB_c to the input variables of the service that fulfill the conditions posed by the service.

In particular, issuing query $conditions_{serv}(i_1, \ldots, i_n)$ to KB_c will yield a set of valid input bindings $\Sigma_{serv} = \{\beta^1_{serv}(i_1, \ldots, i_n), \ldots, \beta^m_{serv}(i_1, \ldots, i_n)\}$ for the service. A service will pass the filter if the answer to the query is positive, i.e., if we find at least an answer to the query ($\Sigma \neq \emptyset$) or if the service has no information preconditions, i.e., $conditions_{serv} : -true$.

Remember from Section 7.2.2 that KB_c is implemented as a $\mathcal{FLORA}$-2 knowledge base and, therefore, $\mathcal{FLORA}$-2 provides the reasoning support for answering the query given by infor-

mation preconditions of candidate services. The Java2Flora utility[4] has been used to interact with $\mathcal{FLORA}$-2.

Example 7.8 Let us come back to Example 7.6, in which service *http://www.afi.es/services/BookIntAirFlight1* described in the previous Chapter, was a possible match for the goal described in Listing 7.2 of booking a seat on $flight1234$ with a $DummyCard$ credit card.

Let us now suppose consumer knowledge is defined by the knowledge base given in Listing 7.4, i.e., that consumer knowledge contains an instance $flight1234$ representing the flight to be booked, an instance of person $rubenLara$, a user account $myIntAirUser$, and a credit card $myDummyCreditCard$ instance of $DummyCard$.

Given that information preconditions of service *http://www.afi.es/services/BookIntAirFlight1* are defined by the query:

```
?f memberOf flights#Flight and ?f[flights#operatedBy hasValue flights#IntAir ] and
?cc memberOf fc#CreditCard and ?cc[fc#hasNumber hasValue ?n]
    and ?cc[fc#hasHolder hasValue ?h] and ?cc[fc#hasExpiryDate hasValue ?e] and
?p memberOf g#Person and ?p[g#hasId hasValue ?id] and ?p[g#hasName hasValue ?name].
```

if we issue this query to $\mathcal{FLORA}$-2, which will contain the instances enumerated above plus domain knowledge, there will be a positive answer to the query and the following input binding, defined by variable substitutions which are an answer to the query, will be obtained:

$\beta_{BookIntAirFlight1}(f, cc, p) = \{f = flight1234, cc = myDummyCreditCard, p = rubenLara\}$

In fact, this will the only input binding obtained, which means:

$\Sigma_{BookIntAirFlight1} = \{\beta_{BookIntAirFlight1}(f, cc, p)\}$

Therefore, this service will pass the input availability filter.

□

Example 7.9 Let us now consider goal *http://www.afi.es/goals/FundSearch1* in Listing 7.1 of searching investment funds commercialized in the Spanish market by entity $MyEntity$. Given this goal, and after the application at the registry of a capability filter, service *http://www.afi.es/services/FundSearch1*, described in the previous Chapter in Listing 6.2, will be retrieved as a perfect match.

Information preconditions of the service retrieved are given by:

```
?c memberOf funds#FundsCategory and ?c[funds#definedByEntity hasValue funds#CNMV].
```

This means that the knowledge base KB_c given by the consumer must contain at least an instance of a funds category defined by entity $CNMV$. Otherwise, the consumer will not be able

[4] http://www.ontotext.com/java2flora/

to provide an appropriate input value to the service and, thus, the service will not pass the input availability filter given current consumer knowledge and domain knowledge. □

Input availability will act as a boolean filter, i.e., the information requirements of a service will be either fulfilled or not. However, as we allow providers to publish services with different levels of detail in their descriptions, there might be services retrieved from the registry which do not describe their information preconditions. Therefore, the application of this filter for a goal $\mathcal{G}$ will yield: 1) a set of services which pass the filter, denoted $\mathcal{S}_{\mathcal{G}}^{+input}$, and 2) a set of services to which the filter could not be applied, denoted $\mathcal{S}_{\mathcal{G}}^{-input}$.

Given that we restrict the expressivity of information preconditions to WSML-Flight, query answering for the information preconditions described can be done in polynomial time. Furthermore, we will see in the next Chapter that the experimental evaluation of query answering in $\mathcal{FLORA}$-2 yields low response times for relatively big knowledge bases.

The usage of LP semantics for finding valid input bindings is appropriate as: i) possible input values must only come from consumer and domain knowledge, both of which are deemed closed and, therefore, making the CWA is correct, and ii) we only require checking ground entailment and not general entailment; furthermore, LP reasoners are optimized for query answering. On the contrary, for reasoning with the DL concepts describing the set of effects requested and offered during the application of a capapability filter, the OWA must be made, as the details of effects not specified should not be deemed to be false. That is the reason why we use classical first-order semantics in that case.

7.4.2 Input-dependent effects filter

As discussed in Chapter 3, there is a dependency between what effects of the service capability will be realized and the input values provided by the consumer for the execution of the service. In this setting, we will consider a second consumer-side filter which will obtain a restricted set of abstract effects which can be realized by the service given particular input bindings, and will also evaluate how this subset of the capability effects relates to the capability sought by the consumer. In the following, we present how this filter is applied, if selected, by the location coordinator.

7.4.2.1 Restricting the set of abstract effects

Given a set of possible input bindings $\Sigma_{serv} = \{\beta^1_{serv}(i_1, \ldots, i_n), \ldots, \beta^m_{serv}(i_1, \ldots, i_n)\}$ for a service *serv* obtained by querying the knowledge base KB_c defined by the consumer , we will determine what set of effects, from the abstract set of effects associated to the service capability, can be actually obtained.

An input binding $\beta^i_{serv}(i_1, \ldots, i_n)$ defines an assignment of values to input variables of a service *serv*, and these values will be given to the service at execution time, thereby conditioning the effects which will be realized from the set of abstract effects of the capability the service enables access to. The description of input-dependent effects of a service introduced in the previous Chapter had the aim of defining how the set of effects achievable by using a service was restricted depending on the input values given. With this purpose, *input variables* were introduced in the description of the service effects, yielding a concept $InputEff_{serv}$. These variables, when bound to particular values given by an input binding β, will restrict the global set of effects of the capability to the set of effects achievable for these values. This restricted set is described by the concept $InputEff_{serv,\beta}$ resulting from substituting input variables by their values as defined by β.

Therefore, if we have a service *serv* for which input bindings $\Sigma_{serv} = \{\beta^1_{serv}(i_1, \ldots, i_n), \ldots, \beta^m_{serv}(i_1, \ldots, i_n)\}$ are possible, the set of effects which can be obtained from using the service is defined by:

$$InputEff_{serv,\Sigma_{serv}} \equiv \bigsqcup_{\forall \beta^i_{serv} \in \Sigma_{serv}} InputEff_{serv,\beta^i_{serv}}$$

Intuitively, the set of effects which can be potentially realized by using the service is composed by the union of the sets of effects obtainable from using the service with each available input binding. Given that a service can executed with only one input binding at a time, this means that, as discussed in [Keller et al., 2005], we consider that a goal can be fulfilled by executing the same service multiple times, i.e., we consider multiple possible executions of the same service for obtaining all the effects sought.

Of course, and given that $InputEff_{serv,\beta^i_{serv}} \sqsubseteq Eff_{\mathcal{C}}$ for any service *serv* enabling access to a capability $\mathcal{C}$ and any valid input binding β^i_{serv} for the service, we also have that

$$InputEff_{serv,\Sigma_{serv}} \sqsubseteq Eff_{\mathcal{C}}$$

for any set of valid input bindings Σ_{serv} for the service.

Example 7.10 Let us take example 7.8, in which an input binding $\beta_{BookIntAirFlight1}(f, cc, p) = \{f = flight1234, cc = myDummyCreditCard, p = rubenLara\}$ was obtained, a step further. If the input-dependent effects filter is selected for application, we will now parameterize the set of effects offered by service *http://www.afi.es/services/BookIntAirFlight1*, given (in DL syntax) by:

$$InputEff_{BookIntAirFlight1} \equiv FlightBooking \sqcap$$
$$\exists ofItem \sqcap \forall ofItem.(FlightSeat \sqcap onFlight.\{f\} \sqcap forPerson.\{p\}) \sqcap$$
$$withPaymentMethod.\{cc\}$$

with the input binding obtained. This parametrization will yield the concept:

$$InputEff_{BookIntAirFlight1,\beta_{BookIntAirFlight1}} \equiv FlightBooking \sqcap$$
$$\exists ofItem \forall ofItem.(FlightSeat \sqcap onFlight.\{flight1234\} \sqcap forPerson.\{rubenLara\}) \sqcap$$
$$withPaymentMethod.\{myDummyCreditCard\}$$

The concept above formalizes the set of effects potentially achievable by using the service with the input binding given, which is now restricted to the booking of seats on flight $flight1234$ for passenger $rubenLara$ and paid with the particular credit card $myDummyCreditCard$.

Now, let us imagine the consumer would have another credit card $myDummyCreditCard2$ included in KB_c, and that an additional input binding $\beta^2_{BookIntAirFlight1}(f, cc, p) = \{f = flight1234, cc = myDummyCreditCard2, p = rubenLara\}$ would therefore had been obtained by the application of the input availability filter, i.e., $\Sigma_{BookIntAirFlight1} = \{\beta_{BookIntAirFlight1}, \beta^2_{BookIntAirFlight1}\}$. In this setting, the set of effects obtainable by using the service would be formalized by:

$$\begin{aligned} InputEff_{BookIntAirFlight1,\Sigma_{BookIntAirFlight1}} \equiv \\ (FlightBooking \sqcap \\ \exists ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap onFlight.\{flight1234\} \sqcap forPerson.\{rubenLara\}) \sqcap \\ withPaymentMethod.\{myDummyCreditCard\}) \sqcup \\ (FlightBooking \sqcap \\ \exists ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap onFlight.\{flight1234\} \sqcap forPerson.\{rubenLara\}) \sqcap \\ withPaymentMethod.\{myDummyCreditCard2\}) \end{aligned} \tag{7.1}$$

which means that, if executed with the input bindings given, we would be able to book seats on flight $flight1234$ for passenger $rubenLara$ and paid with either the $myDummyCreditCard$ or the $myDummyCreditCard2$ credit cards.

□

7.4.2.2 Filtering services based on the restricted set of abstract effects

For each service for which valid input bindings were found, i.e., for each service $serv \in \mathcal{S}_{\mathcal{G}}^{+input}$, we will parameterize its input-dependent effects (if described) and define the concepts which describe the restricted set of effects that can be obtained for the input bindings available, as described above. This will yield, for each $serv \in \mathcal{S}_{\mathcal{G}}^{+input}$, a concept $InputEff_{serv,\Sigma_{serv}}$ to which the intention given with the description of the input-dependent capability of the service is associated.

In order to match, based on the restricted set of effects obtained, services to the goal $\mathcal{G}$ at hand, we will publish, for each $serv \in \mathcal{S}_{\mathcal{G}}^{+input}$, concept $InputEff_{serv,\Sigma_{serv}}$ to the TBox of the DL reasoner included in the service location component (see Figure 7.1 in the previous Chapter). The location coordinator will then query this TBox for concepts:

- synonyms of (equivalent to) concept $Eff_{\mathcal{G}}$[5] formalizing the set of effects required, denoted $\mathcal{S}^{\equiv}$,
- ancestors of (subsuming) concept $Eff_{\mathcal{G}}$, denoted $\mathcal{S}'^{\sqsubset}$,
- descendants of (subsumed by) concept $Eff_{\mathcal{G}}$, denoted $\mathcal{S}'^{\sqsupset}$,
- synonyms of (equivalent to) concept $not(Eff_{\mathcal{G}})$, denoted $\mathcal{S}'^{not(\equiv)}$,
- descendants of (subsumed by) concept $not(Eff_{\mathcal{G}})$, denoted $\mathcal{S}'^{not(\sqsupset)}$.

Given these sets of retrieved concepts, corresponding to the formalization of possible service effects for the input bindings available, we will obtain the following sets taking into account the intention associated to the different concepts obtained:

$$\mathcal{S}_{\forall}^{\prime\equiv} = \{serv \mid serv \in \mathcal{S}'^{\equiv}, I_{serv} = all\}$$

$$\mathcal{S}_{\exists}^{\prime\equiv} = \{serv \mid serv \in \mathcal{S}'^{\equiv}, I_{serv} = exists\}$$

$$\mathcal{S}_{\forall}^{\prime\sqsubset} = \{serv \mid serv \in \mathcal{S}'^{\sqsubset}, I_{serv} = all\}$$

$$\mathcal{S}_{\exists}^{\prime\sqsubset} = \{serv \mid serv \in \mathcal{S}'^{\sqsubset}, I_{serv} = exists\}$$

[5] In this Section, we refer to the original concept given, possibly including instances not in $\mathcal{O}^{DL}$.

$$\mathcal{S}'^{\sqsupset}_{\forall} = \{serv \mid serv \in \mathcal{S}'^{\sqsupset}, I_{serv} = all\}$$

$$\mathcal{S}'^{\sqsupset}_{\exists} = \{serv \mid serv \in \mathcal{S}'^{\sqsupset}, I_{serv} = exists\}$$

$$\mathcal{S}'^{\sqcap}_{\forall} = \{serv \mid serv \in (\mathcal{S}^{+input}_{\mathcal{G}} - (\mathcal{S}'^{not(\equiv)} \cup \mathcal{S}'^{not(\sqsupset)})), I_{serv} = all\}$$

$$\mathcal{S}'^{\sqcap}_{\exists} = \{serv \mid serv \in (\mathcal{S}^{+input}_{\mathcal{G}} - (\mathcal{S}'^{not(\equiv)} \cup \mathcal{S}'^{not(\sqsupset)})), I_{serv} = exists\}$$

where $\mathcal{S}^{+input}_{\mathcal{G}}$ is the set of services which passed the input availability filter and the - sign denotes set difference. Given these sets, we will evaluate what services are a perfect, possible perfect, partial, or possible partial match using the criteria given in Table 7.2, as we did for the application of the capability filter.

In particular, perfect matches for goal $\mathcal{G}$ are defined by:

$$\mathcal{S}^{ieMatch}_{\mathcal{G}} = \mathcal{S}'^{\sqcap}_{\forall} \cup \mathcal{S}'^{\equiv}_{\exists} \cup \mathcal{S}'^{\sqsupset}_{\exists}$$

if $I_{\mathcal{G}} = some$, and

$$\mathcal{S}^{ieMatch}_{\mathcal{G}} = \mathcal{S}'^{\equiv}_{\forall} \cup \mathcal{S}'^{\sqsubset}_{\forall}$$

if $I_{\mathcal{G}} = all$.

Possible matches are defined by:

$$\mathcal{S}^{iepoMatch}_{\mathcal{G}} = \mathcal{S}'^{\sqcap}_{\exists} - (\mathcal{S}'^{\equiv}_{\exists} \cup \mathcal{S}'^{\sqsupset}_{\exists})$$

if $I_{\mathcal{G}} = some$, and

$$\mathcal{S}^{iepoMatch}_{\mathcal{G}} = \mathcal{S}'^{\sqsubset}_{\exists}$$

if $I_{\mathcal{G}} = all$.

Partial matches are defined by:

$$\mathcal{S}_{\mathcal{G}}^{iePMatch} = (\mathcal{S}_{\forall}^{\prime\sqcap} - (\mathcal{S}_{\forall}^{\prime\equiv} \cup \mathcal{S}_{\forall}^{\prime\sqsubset})) \cup \mathcal{S}_{\exists}^{\prime\equiv} \cup \mathcal{S}_{\exists}^{\prime\sqsupset}$$

if $I_{\mathcal{G}} = all$.

Possible partial matches are defined by:

$$\mathcal{S}_{\mathcal{G}}^{iepPMatch} = \mathcal{S}_{\exists}^{\prime\sqcap} - (\mathcal{S}_{\exists}^{\prime\equiv} \cup \mathcal{S}_{\exists}^{\prime\sqsubset} \cup \mathcal{S}_{\exists}^{\prime\sqsupset})$$

if $I_{\mathcal{G}} = all$.

In a nutshell, we obtain services that offer, for the input bindings available, the effects expected by the consumer, considering how these effects depend on such input bindings. As some services might not describe input-dependent effects or the input availability filter might not be applicable to them, the service location coordinator will split the results of applying this filter into: a) services to which the filter could be applied and were a perfect ($\mathcal{S}_{\mathcal{G}}^{ieMatch}$), possible perfect ($\mathcal{S}_{\mathcal{G}}^{iepoMatch}$), partial ($\mathcal{S}_{\mathcal{G}}^{PMatch}$), or possible partial match ($\mathcal{S}_{\mathcal{G}}^{pPMatch}$), and b) services to which the filter could not be applied, denoted $\mathcal{S}_{\mathcal{G}}^{-ie}$.

Example 7.11 In Example 7.10 we obtained the following concept formalizing the restricted set of effects obtainable from executing service *http://www.afi.es/services/BookIntAirFlight1* with input binding $\beta_{BookIntAirFlight1}(f, cc, p) = \{f = flight1234, cc = myDummyCreditCard, p = rubenLara\}$:

$$\begin{aligned} InputEff_{BookIntAirFlight1,\beta_{BookIntAirFlight1}} \equiv \\ FlightBooking \sqcap ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap onFlight.\{flight1234\} \sqcap forPerson.\{rubenLara\}) \sqcap \\ withPaymentMethod.\{myDummyCreditCard\} \end{aligned}$$

Given this concept, to which an existential intention was associated, and the goal described in Listing 7.2 of booking a seat on $flight1234$ with a $DummyCard$ credit card, formalized by concept (also with an existential intention):

$$\begin{aligned} Eff_{BookAFlight1} \equiv FlightBooking \sqcap \\ \exists ofItem \sqcap \\ \forall ofItem.(FlightSeat \sqcap onFlight.\{flight1234\}) \sqcap forPerson.\{rubenLara\} \sqcap \\ withPaymentMethod.\{myDummyCreditCard\} \end{aligned}$$

We will have that $InputEff_{BookIntAirFlight1,\beta_{BookIntAirFlight1}} \equiv Eff_{BookAFlight1}$. Therefore, the service is a perfect match for the goal given the respective intentions associated to the service capability description and to the goal capability description, i.e., the set of effects offered by the service for the input binding available completely satisfy the consumer's goal.

□

In a nutshell, the input-dependent effects filter will restrict the set of effects a service offers based on the particular input bindings that can be provided by the consumer, and will evaluate to what extent this restricted set fulfills the consumer's goal. The application of this filter requires the formalization of sought capabilities in the goal description, the description of input-dependent effects of the service, and the availability of a set of input bindings for the service, which is obtained automatically from consumer knowledge.

We will discuss in the next Chapter the complexity of applying this filter and the response times obtained in our experimental evaluation. We anticipate that these times are relatively high, as the concepts describing the restricted set of effects offered by candidate services are obtained at discovery time and, therefore, they cannot be pre-classified in the TBox of the DL reasoner.

7.4.3 Combination of filters

There are two only possible combinations of consumer-side filters: the only application of the input availability filter, or the joint application of both the input availability filter and the input-dependent effects filter. Notice that the stand-alone application of the input-dependent effects filter is not allowed, as we necessarily need to have the input bindings to be tested defined and validated, which will always require the application of the input availability filter.

In this setting, and given the sets of perfect ($\mathcal{S}_{\mathcal{G}}^{Match}$), possible perfect ($\mathcal{S}_{\mathcal{G}}^{poMatch}$), partial ($\mathcal{S}_{\mathcal{G}}^{PMatch}$), and possible partial matches ($\mathcal{S}_{\mathcal{G}}^{pPMatch}$) retrieved from the service registry for a goal $\mathcal{G}$, the final sets of results of the location process will be:

- If no consumer-side filter is selected, the same sets of services $\mathcal{S}_{\mathcal{G}}^{Match}$, $\mathcal{S}_{\mathcal{G}}^{poMatch}$, $\mathcal{S}_{\mathcal{G}}^{PMatch}$, and $\mathcal{S}_{\mathcal{G}}^{pPMatch}$ retrieved from the registry,
- If the input-availability filter is selected, sets:
  1. $\mathcal{S}_{\mathcal{G}}^{Match+input} = \mathcal{S}_{\mathcal{G}}^{Match} \cap \mathcal{S}_{\mathcal{G}}^{+input}$ of services which are a perfect match and which pass the input availability filter,
  2. $\mathcal{S}_{\mathcal{G}}^{poMatch+input} = \mathcal{S}_{\mathcal{G}}^{poMatch} \cap \mathcal{S}_{\mathcal{G}}^{+input}$ of services which are a possible match and which pass the input availability filter,

3. $\mathcal{S}_{\mathcal{G}}^{PMatch+input} = \mathcal{S}_{\mathcal{G}}^{PMatch} \cap \mathcal{S}_{\mathcal{G}}^{+input}$ of services which are a partial match and which pass the input availability filter,
4. $\mathcal{S}_{\mathcal{G}}^{pPMatch+input} = \mathcal{S}_{\mathcal{G}}^{pPMatch} \cap \mathcal{S}_{\mathcal{G}}^{+input}$ of services which are a possible partial match and which pass the input availability filter,
5. $\mathcal{S}_{\mathcal{G}}^{Match-input} = \mathcal{S}_{\mathcal{G}}^{Match} \cap \mathcal{S}_{\mathcal{G}}^{-input}$ of services which are a perfect match and to which the input availability filter cannot be applied as their preconditions are not defined,
6. $\mathcal{S}_{\mathcal{G}}^{poMatch-input} = \mathcal{S}_{\mathcal{G}}^{poMatch} \cap \mathcal{S}_{\mathcal{G}}^{-input}$ of services which are a possible match and to which the input availability filter cannot be applied as their preconditions are not defined,
7. $\mathcal{S}_{\mathcal{G}}^{PMatch-input} = \mathcal{S}_{\mathcal{G}}^{PMatch} \cap \mathcal{S}_{\mathcal{G}}^{-input}$ of services which are a partial match and to which the input availability filter cannot be applied as their preconditions are not defined, and
8. $\mathcal{S}_{\mathcal{G}}^{pPMatch-input} = \mathcal{S}_{\mathcal{G}}^{pPMatch} \cap \mathcal{S}_{\mathcal{G}}^{-input}$ of services which are a possible partial match and to which the input availability filter cannot be applied as their preconditions are not defined.

- If the input-dependent effects filter is selected, services in sets $\mathcal{S}_{\mathcal{G}}^{Match+input}$, $\mathcal{S}_{\mathcal{G}}^{poMatch+input}$, $\mathcal{S}_{\mathcal{G}}^{PMatch+input}$, $\mathcal{S}_{\mathcal{G}}^{pPMatch+input}$ will be further refined into sets:

  1. $\mathcal{S}_{\mathcal{G}}^{ieMatch}$ of services which passed the input-availability and which are a perfect match under the input-dependent effects filter,
  2. $\mathcal{S}_{\mathcal{G}}^{iepoMatch}$ of services which passed the input-availability and which are a possible match under the input-dependent effects filter,
  3. $\mathcal{S}_{\mathcal{G}}^{iePMatch}$ of services which passed the input-availability and which are a partial match under the input-dependent effects filter,
  4. $\mathcal{S}_{\mathcal{G}}^{iepPMatch}$ of services which passed the input-availability and which are a possible partial match under the input-dependent effects filter,
  5. $\mathcal{S}_{\mathcal{G}}^{Match-ie} = \mathcal{S}_{\mathcal{G}}^{Match+input} \cap \mathcal{S}_{\mathcal{G}}^{-ie}$ of services which were a perfect match, which passed the input-availability filter, but to which the input-dependent effects filter could not be applied,
  6. $\mathcal{S}_{\mathcal{G}}^{poMatch-ie} = \mathcal{S}_{\mathcal{G}}^{poMatch+input} \cap \mathcal{S}_{\mathcal{G}}^{-ie}$ of services which were a possible perfect match, which passed the input-availability filter, but to which the input-dependent effects filter could not be applied,
  7. $\mathcal{S}_{\mathcal{G}}^{PMatch-ie} = \mathcal{S}_{\mathcal{G}}^{PMatch+input} \cap \mathcal{S}_{\mathcal{G}}^{-ie}$ of services which were a partial match, which passed the input-availability filter, but to which the input-dependent effects filter could not be applied, and

8. $\mathcal{S}_{\mathcal{G}}^{pPMatch-ie} = \mathcal{S}_{\mathcal{G}}^{pPMatch+input} \cap \mathcal{S}_{\mathcal{G}}^{-ie}$ of services which were a possible partial match, which passed the input-availability filter, but to which the input-dependent effects filter could not be applied.

In principle, any combination of registry-side and consumer-side filters can be selected by a consumer. However, we expect that if the input-dependent effects filter is selected, also the capability filter is selected. The reason is two-fold: a) for the application of the input-dependent effects filter the goal description must incorporate the set-based modelling of sought capabilities, so it is already available for the application of the capability filter, and b) if the accuracy of the input-dependent effects filter is required, it is likely that accuracy for the retrieval of relevant services from the registry is also desired.

Besides using the consumer-side filters provided by our prototype platform, consumers can choose to locally apply custom filters not provided by the platform to the results obtained from applying available filters. This grants flexibility to consumers, who can freely extend the filtering done over the results obtained. However, the implementation of this type of extensions is completely up to consumers.

7.5 Summary

In this Chapter, we have presented a particular instantiation of the second part of the abstract model given in Chapter 5, which deals with the description of goals and the actual discovery of services.

In Section 7.2, we have presented the alternative types of descriptions of consumers' goals considered in our platform, which are very closely related to the types of descriptions of services presented in the previous Chapter. By allowing for the provision of different types of descriptions of goals, we enable the application of alternative filters with different properties and we facilitate the task of describing goals; different consumers will provide the types of descriptions they are comfortable with. Furthermore, consumers can select what particular filters must be applied for the location of relevant services for solving the goal, thereby customizing the accuracy of the results obtained and the response times of the process. The application of different filters will require, though, different types of descriptions of goals to be available. In this setting, support will be provided to consumers for providing the necessary types of descriptions of their goals for the application of the filters selected.

In the next two Sections, we have presented the filters currently included in our model instantiation and prototype implementation, split into filters applied at the registry side and filters applied at the consumer side, where consumer knowledge is accessible. The first group of filters

focuses on the capability offered by published services, disregarding what conditions such services pose for their usage and the particular functionality of the service. Depending on the filters selected, the matching of services to the consumer's goal will be more or less efficient, and the accuracy which can be expected from such matching will also vary. Whatever filter or combination of filters is applied, the result of the application of registry-side filters is the matching of available services and the provision of the complete descriptions of matched services to the service location client at the consumer side.

Consumer-side filters operate over services retrieved from the registry after the application of registry-side filters. These filters have access to consumer knowledge, kept locally, and focus on evaluating whether information preconditions of retrieved services are fulfilled, and what particular effects can be potentially achieved from using these services for the input bindings a consumer can provide.

Overall, our model instantiation enables the flexible combination of filters with the purpose of covering as many usage scenarios as possible while keeping a relative simplicity which guarantees usability and the possibility of supporting consumers for describing their goals. However, we have not addressed the selection of services once they have been located based on their static descriptions, as such selection will require direct communication with services, which is beyond the scope of our work.

In the next Chapter, we will provide an evaluation of our overall model instantiation, a summary of the known limitations of our proposal, work related to the contents of this thesis, and extensions envisioned.

Chapter 8

Evaluation and related work

In the previous Chapters we have presented an abstract model for the location of services which can provide a given value fulfilling a consumer's goal, as well as an instantiation of this model prototypically implemented. The motivation for the proposal of such an abstract model has been the need for an effective location of services in different usage scenarios, which demands flexibility and usability. Furthermore, a particular instantiation of the abstract model has been proposed with the purpose of making concrete certain aspects of the service location process left open by the abstract model so that a proof of concept of the model is available and an evaluation is possible.

In this Chapter, we will present in Section 8.1 an evaluation of the model instantiation and prototype implementation proposed. In particular, we analyze the complexity of the service location process and provide an experimental evaluation of our proposal (Section 8.1.1), discuss to what extent the families of usages scenarios identified in Chapter 5 are covered by the model instantiation proposed in Chapters 6 and 7 (Section 8.1.2), and outline the known limitations of our work (Section 8.1.3). Section 8.2 is devoted to providing an overview of relevant related work.

8.1 Evaluation

8.1.1 Complexity and experimental evaluation

In this Section, we discuss the complexity of the main tasks involved in the management of taxonomies, the publication of service descriptions, and the location of services using different filters. In addition, some experimental results will be given in order to have an estimation of the response times we can obtain when applying the model.

8.1.1.1 Publication of service descriptions

We will start with the publication of service descriptions to the registry as described in Chapter 6.

Parsing. The first thing done by the publication manager of the registry when receiving a WSMO description of a service for publication will be parsing and validating the WSMO description using WSMO4J. The time required for this parsing is marginal as only the syntactic structure of the description is checked and, therefore, we will obviate this cost.

Checking consistency. Once the WSMO description has been parsed, the publication manager will have available the description of the categories the service belongs to, as well as the set-based formalization of the service capability.

If the service capability has been formalized, the publication manager will then request from the taxonomy manager the set-based modelling associated to the categories of the service (if any). As the taxonomy manager has stored each type of description of the capability associated to a category in the relational database of the registry, retrieving this type of description will only imply a simple query, given the complete URI of the category, on the relational database. The cost of this operation is again marginal, and we will obviate it.

Once the set-based modelling of categories given with the service has been obtained, we will build concept Eff^{T}_{serv} as described in Chapter 6 and we will check the satisfiability of such concept with respect to the *Services* TBox of the registry DL reasoner. As discussed in Chapter 2, Section 2.2.2.4, checking the satisfiability of a $\mathcal{SHOIN}$ concept, which is the expressivity allowed for the set-based modelling of capabilities, is NExpTime-complete. While the complexity of this reasoning task, necessary for guaranteeing that the description of the service given is consistent, is high, modern DL reasoners are highly optimized, enabling the obtention of results in relatively low times.

Storing the description. If the satisfiability check succeeds, we will proceed to store the description of the service in the UDDI repository of the registry. The time required for accomplishing this task will be low, and it will not depend on the number of service descriptions already published.

Processing the description for posterior retrieval. Afterwards, if the service has been categorized, it will be associated to the categories in the copy of published taxonomies kept in memory by the registry. This task will have a very low cost and we can obviate it.

Finally, if the set-based modelling of the service capability is available, the concept formalizing such capability will be submitted to the DL reasoner of the registry for its addition to

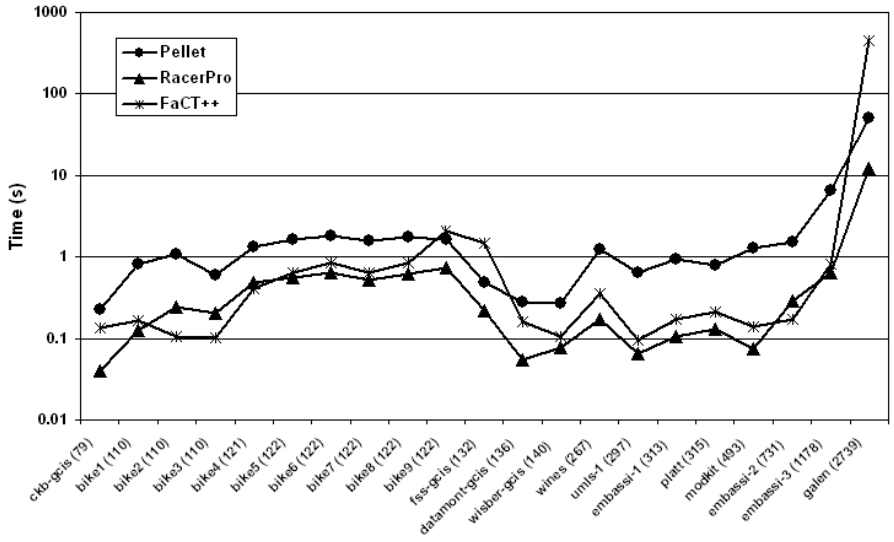

Figure 8.1: Classification times with different DL reasoners [Sirin et al., 2006b]

the *Services* TBox, and this TBox will be classified. The classification of a TBox can be reduced to the reasoning task of checking satisfiability. Therefore, the classification of the TBox will be NExpTime-complete. Still, modern DL reasoners will in most cases offer times we can admit as the publication of service descriptions is not a time-critical task.

Summarizing, the publication of a service description involves two NExpTime-complete tasks, to be performed by the DL reasoner of the registry. However, as discussed in [Gardiner et al., 2006], the hope/claim is that modern highly optimised systems perform well in "realistic" ontology applications. We show in Figure 8.1, the results of evaluating the performance of different DL reasoners (including RacerPro) for classification, obtained by Sirin et al. in [Sirin et al., 2006b]. A logarithmic scale is used in the Figure, and ontologies, taken from the standard DL benchmark test suite [Horrocks and Patel-Schneider, 1998b], are shown in the x axis with the number of concepts they contain between brackets. A Pentium Centrino 1.6GHz computer with 1.5GB memory was used for obtaining these times, the maximum memory amount allowed to Java was set to 256MB for each experiment, and all the timings were computed as the average of 10 independent runs. Pan has also presented in [Pan, 2005] a benchmark of DL reasoners which has shown that (version 1.7 of) Racer was able to classify 135 realistic ontologies in 15 minutes, including not only TBox classification but also ABox consistency checks.

Gardiner et al. argue in [Gardiner et al., 2006] that, to check the validity of the claim that DL reasoners will perform well in realistic applications, it is necessary to test the performance of these systems with (the widest possible range of) ontologies derived from applications, and they

present a system that allows, among other things, to compare the performance of DL reasoners for the classification of ontologies. In particular, in this work real OWL ontologies have been collected and translated into DIG, and the following steps have been followed for each ontology: 1. Load the ontology into the reasoner; 2. Query the reasoner for all the named (atomic) classes in the ontology; 3. Query the reasoner for the consistency of the ontology by checking the satisfiability of the *Top* concept; 4. Query the reasoner for the satisfiability of each of the named classes in the ontology; 5. Query the reasoner for the ontology taxonomy (i.e., the parents and children of all named classes). In this way, the evaluation has forced each reasoner to fully classify the ontology independently of the evaluation strategies they follow. The tests were performed using an Intel Pentium-M processor 1.60 GHz and 1Gb of main memory on Windows XP, and a time-out of 10 minutes was set. In this setting, the tests showed that only very few cases the reasoners could not perform all the tasks within the time limit set; in Figure 8.2, the response times obtained for the 10 ontologies considered to be the most challenging ones are shown, from which ontology number 1 has 27652 concepts and ontology number 2 has 20526 concepts (see [Gardiner et al., 2006] for further details).

In general, existing evaluations show that the times required for TBox classification (and, thus, for checking satisfiability) depend not only on the size of the ontology classified, but also on the characteristics of such ontology. Therefore, we can only regard these times as an estimate of the times which will be obtained; the exact times will vary depending on the size and complexity of the domain ontologies used in each particular use case. Still, we can realistically expect our model to publish services descriptions in acceptable times, as the main cost of publication comes from the reasoning services provided by the DL reasoner of the registry and experimental results show that this cost is kept within acceptable limits.

We have run our own experiments in order to obtain an estimate of response times of our prototype for the publication of services. For this purpose, we have published service descriptions with capabilities obtained as random variations of a set of 10 base services referring to a domain ontology of around 850 concepts. This set of base services includes services offering the booking of flights operated by different airlines, for different itineraries (restricted to certain cities, countries or continents), and with different payment methods, and other services which offer capabilities not related to flight booking such as search of investment funds, contracting of investment funds, or obtention of credit scoring details for public institutions, with which we have worked in the SETA project. We have measured the times necessary for publishing a new service description when a given number of services have been already published and the *Services* TBox has been classified, i.e., we have measured the times for the addition of a single service to the registry as a function of the number of services already published.

In Figure 8.3, we show the total time for the publication of a new service, as well as what portion of this time is required for checking the satisfiability of the intersection of the formalization

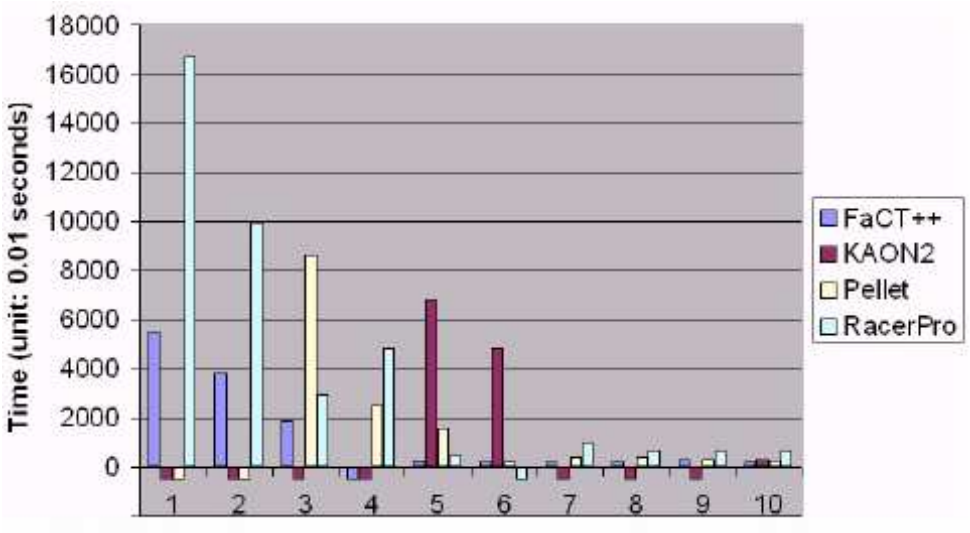

Figure 8.2: Comparison of DL reasoners on 10 challenging ontologies [Gardiner et al., 2006]

of the service capability and of the set-based modelling of capabilities associated to service categories (if any); the difference between total publication time and satisfiability time is almost completely used for the classification of the TBox after the concept formalizing the new service capability has been added. A computer with an AMD Turion 64x2 Mobile 1.61 GHz processor and 1 GB RAM on Windows XP has been used for these and other tests we will present later in this Section.

In a nutshell, we can see that response times for the publication of descriptions can be expected to be kept within limits we find acceptable, considering that the publication of service descriptions is not a time-critical task, as it is done without affecting service location times and no strong efficiency requirements are expected to be posed on this task.

8.1.1.2 Location of services

For the location of services, we allow for the application of the alternative filters introduced in the previous Chapter. In the following, we summarize the complexity of applying such filters and the experimental results obtained.

Textual filter. For the location of services using this filter, we query the relational database of the registry for services which contain any of the keywords in the textual description of the consumer's goal without noise words, as presented in Chapter 7, Section 7.3.2. This is a simple query existing relational database management systems will resolve very efficiently.

In our prototype, we have used Oracle 9i as the relational database of the registry, and

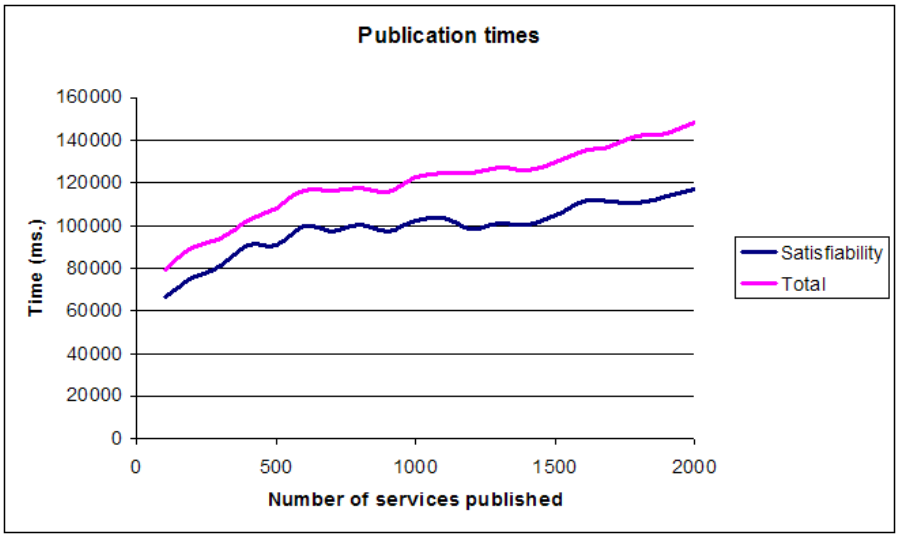

Figure 8.3: Times for the publication of service descriptions

we have issued queries with up to 20 keywords on textual fields of tables with over 400000 rows; our experiments have shown that response times remain almost constant and below 50 milliseconds. The experimental application of this filter over 2000 randomly generated services as discussed above yields times of around 100 milliseconds, including the removal of noise words and the retrieval of descriptions of matching services from the (database of the) UDDI repository.

Category filter. The application of the category filter will imply inspecting the structure of the taxonomies of categories loaded in memory in order to obtain ancestors and descendants of the categories given with the goal. In particular, for each category c_i given with the goal, we will traverse the taxonomy tree from the node corresponding to this category until we reach the root of the tree, and we will also traverse the tree until we reach all the leaves on paths starting from the node corresponding to c_i. In the worst case (c_i is a child of the root of the tree) we will have to traverse $n-1$ tree nodes, where n is the number of categories of the taxonomy.

In this setting, we can see that the number of operations we will have to perform for finding matches for a given category is linear on the size of the taxonomy of categories. If m is the number of categories given with the goal, in the worst case we will have to traverse $(n-1)*m$ categories in order to obtain all matches for the goal, considering that n is the average number of categories in the taxonomies used. However, given that the number m of categories sought services must belong to will be generally low, that the size of taxonomies of categories is also expected to be relatively small in order to keep taxonomies usable, and that we have the structure of taxonomies of categories, as well as a record of what services are associated to each category, loaded in memory, response times

will be low.

Our experimental results show that the response times obtained are kept almost constant, and that for 2000 randomly generated services published, for three test taxonomies with 30 categories each, and for goals with up to 5 categories, we obtain perfect and partial matches from the application of the category filter in around 150 milliseconds. This time includes the time required to retrieve the complete description of matching services from the (database of the) UDDI repository.

Capability filter. For the application of the capability filter, we query for (see Chapter 7, Section 7.3.4): a) synonyms, b) ancestors, and c) descendants of the concept formalizing required effects, and for d) synonyms, and e) descendants of the negation of such concept. As subsumption reasoning can be reduced to satisfiability in the $\mathcal{SHOIN}$ DL (see Chapter 2, Section 2.2.2.3), we have that answering these queries is NExpTime-complete. However, DL reasoners are optimized so that querying for the subsumption relation between concepts over a classified TBox can be done efficiently. We have measured the response times of applying the capability filter for retrieving services which are candidates for achieving randomly generated goals, and we show in Figure 8.4 the results obtained[1].

It must be noted that the times required for obtaining synonyms, ancestors and descendants of the concept given by the goal remain low (less than 20 milliseconds per query in average) [Lara, 2006; Li and Horrocks, 2003], but higher times are necessary for obtaining concepts not intersecting the goal. The reason is that the algorithms used by DL reasoners to classify the TBox are optimized to compute hierarchical relations, but they do not pre-compute disjoint classes. Therefore, the time necessary for applying the capability filter is mainly used for obtaining services which intersect the goal given, as shown in the Figure.

Other tasks required for the application of the capability filter, such as the obtention of complete descriptions of the service given, and the consideration of intentions in order to build the different groups of services we identify according to their degree of match, have a marginal cost compared to querying the DL reasoner for related concepts; this task is actually the main source of complexity of this filter.

Input-availability filter. The application of the input-availability filter will require, for each service retrieved from the registry for which information preconditions are described, to query a $\mathcal{FLORA}$-2 knowledge base kept within the expressivity of WSML-Core.

In Chapter 2, Section 2.2.3.6, we have seen that plain Datalog is data complete for the complexity class P. Furthermore, if we add negation and since the stratification algorithm is a polynomial algorithm, stratified Datalog with negation has the same complexity as plain Datalog. As

[1] These times are lower than the times given in [Lara et al., 2007b] as a different computer has been used and, furthermore, we have reduced some unnecessary costs we had for the retrieval of the complete descriptions of matched services from the UDDI repository of the registry.

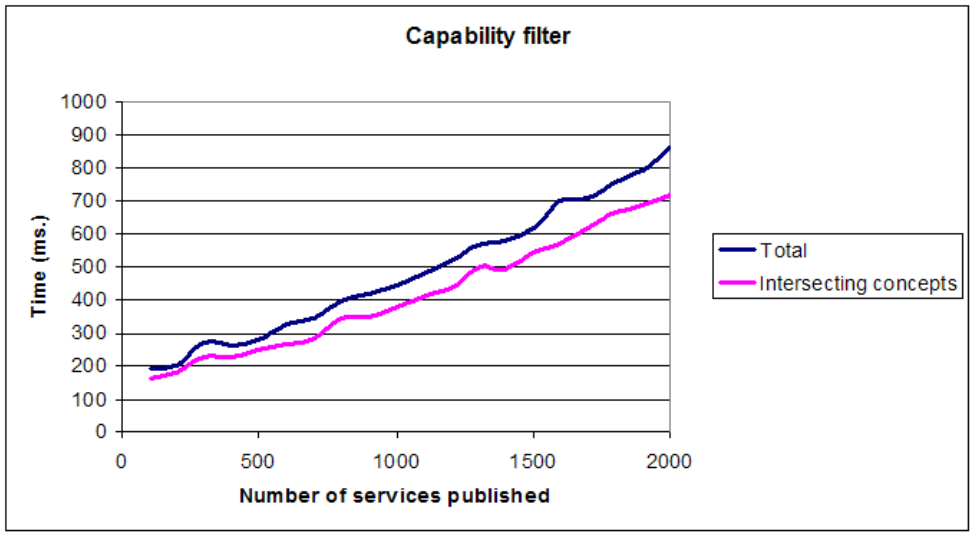

Figure 8.4: Times for the application of the capability filter

WSML-Flight (function-free Datalog) queries are used for the description of information preconditions, which might possibly include negation, and these queries will be issued over WSML-Core (function-free and negation-free) knowledge bases, we will have that query answering can be done in polynomial time.

Our experiments show that querying a plain Datalog knowledge base using $\mathcal{FLORA}$-2 is very efficient. In particular, and once the knowledge base has been loaded to the reasoner, we have seen that answering queries over a knowledge base with 14000 facts can be done in less than 50 milliseconds. The input-availability filter is expected to be generally applied in conjunction with the capability filter and, thus, it will not be applied to a big number of services, as the application of the capability filter will yield accurate results. In this setting, the application of the input-availability filter will be efficient as only a limited number of queries will have to be issued to the $\mathcal{FLORA}$-2 knowledge base. In our prototype, the knowledge base containing consumer knowledge is defined before the goal is issued to the service location component, and it is already compiled and loaded when a goal has to be resolved, which makes querying for valid input values using the definition of information preconditions of the service efficient.

In our tests, and using a smaller knowledge base (of around 1000 facts), the average time measured to answer a single query described by information preconditions of a service has been of around 20 milliseconds. Therefore, applying this filter to e.g. 100 candidate services, and assuming all of them define information preconditions, requires around two seconds.

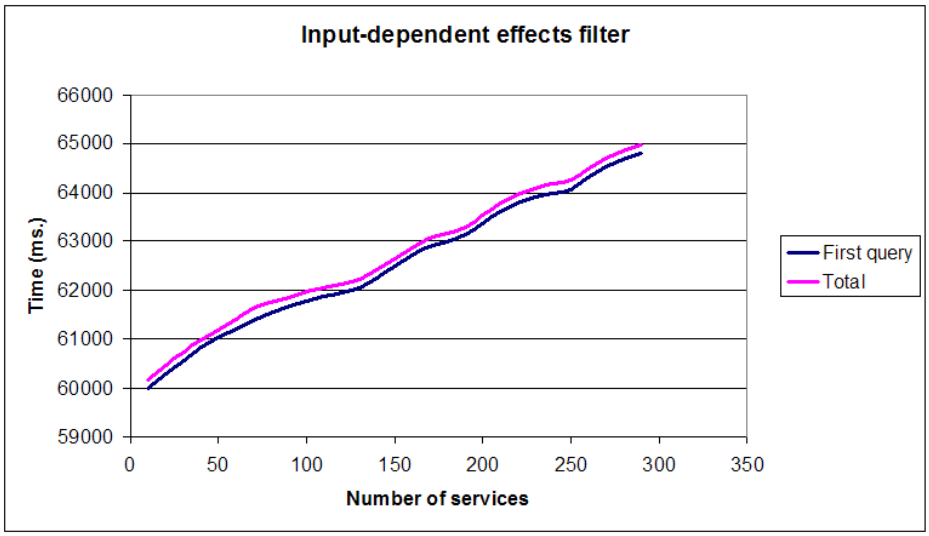

Figure 8.5: Times for the application of the input-dependent effects filter

Input-dependent effects filter. The input-dependent effects filter first requires the parametrization, for each service which has passed the previous filters, of the concept describing achievable effects with the input bindings obtained, although this task has a marginal cost.

Afterwards, the concepts obtained will be sent to the TBox of the DL reasoner of the service location client, to which the domain ontologies used have been already loaded and classified, and we will query for a) synonyms, b) ancestors, and c) descendants of the concept formalizing required effects, and for d) synonyms, and e) descendants of the negation of such concept, as we did for the application of the capability filter. As we have seen before, the complexity of this reasoning service is NExpTime-complete.

In this case, and unlike for the application of the capability filter, we will not have the TBox of the reasoner fully classified, as new concepts formalizing input-dependent effects of candidate services have been added to the TBox. Therefore, the response times obtained will not be as low as those obtained for the application of the capability filter. However, the number of new concepts published is expected to be relatively low and, thus, response times are not expected to be extremely high.

In Figure 8.5, we show the experimental results obtained for the application of this filter, as a function of the number of candidate services considered. We can see that most of the time required for the application of this filter is used to answer the first query (for synonyms of the concept formalizing the capability sought). The reason is that RacerPro classifies the TBox when this first query is issued, and it is already classified for the next queries.

Filter	*Efficiency*	*Accuracy*
Textual	Very low response times	Low precision but flexibility in descriptions
Category	Very low response times	Coarse-grained but precise results
Capability	Low response times	Accurate filtering of services with relevant capabilities
Input availability	Relatively low response times	Accurate filtering of candidate services based on information preconditions
Input-dependent effects	Relatively high response times	Accurate filtering of candidate services based on input-dependent effects

Table 8.1: Properties of filters

We have seen above the complexity and the experimental cost of applying the filters currently offered by our instantiation of the abstract model presented in Chapter 5. The general properties of these filters, which should drive the decision of which filters to apply in each scenario, are summarized in Table 8.1. It must be noted that, if multiple filters are applied, the times required for the location of services will correspond to the sum of the times required for the application of each selected filter.

8.1.1.3 Management of taxonomies

The tasks involved in the management of taxonomies will in general not be time-critical, but searching categories in taxonomies will be often required for the assisted description of goals and services and, thus, their complexity and the estimation of response times which will be obtained in practice is relevant.

Creation of taxonomies. The creation of a taxonomy will require storing in the relational database the taxonomy data and creating a new UDDI tModel, and all these tasks have a marginal cost.

Creation of categories. When a new category is created, the satisfiability of the concept formalizing its associated capability will be checked and, if this check succeeds, we will store the category details in the relational database of the registry, add the concept to the *Categories* TBox, and classify this TBox.

Therefore, the creation of categories is NExpTime-complete, as satisfiability of a concept has to be checked and a TBox classified. In Figure 8.6 we show the times measured for the creation of a new category once a given number of categories (x axis) has been created and the *Services* TBox

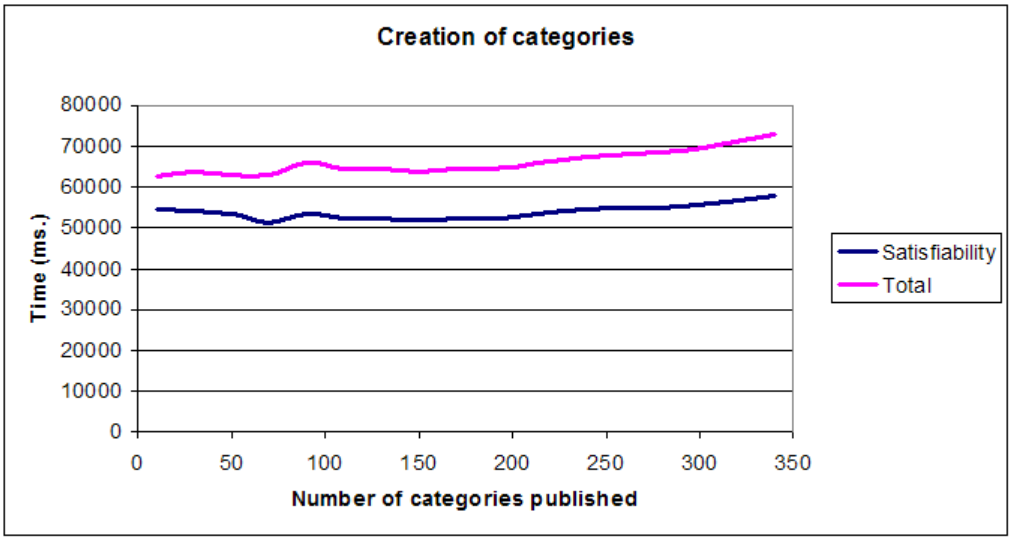

Figure 8.6: Times for the creation of categories

has been classified. It can be seen that most of the time required for creating the category is devoted to checking the satisfiability of the concept formalizing the capability such category represents.

The response times expected are similar to the times necessary for the publication of service descriptions, as the tasks involved are basically the same. However, the times obtained will depend on the number of categories published and not on the number of services. Given that the number of categories available is expected to be generally lower than the number of services published to the registry, response times will also be generally lower. Furthermore, we only have to check the satisfiability of the formalization of the capability associated to the new category, while for the publication of service descriptions we have to check the satisfiability of a more complex concept if the service has been categorized. For this reason, the times necessary for the creation of categories are generally lower than the times for the publication of service descriptions, but their dependency on the number of published categories and services, respectively, will be similar.

Category search In Chapter 6 we have seen three types of category search. From these, direct category search boils down to a simple lookup over the relational database of the registry, which will have a very reduced cost. The textual search of categories has approximately the same cost as the textual filter for the location of services, and it is expected to yield even lower response times as the number of categories which will be available will be generally lower than the number of published services.

Category search based on the set-based modelling of capabilities will be NExpTime-Complete, as it requires subsumption reasoning over published categories. The response times obtained are

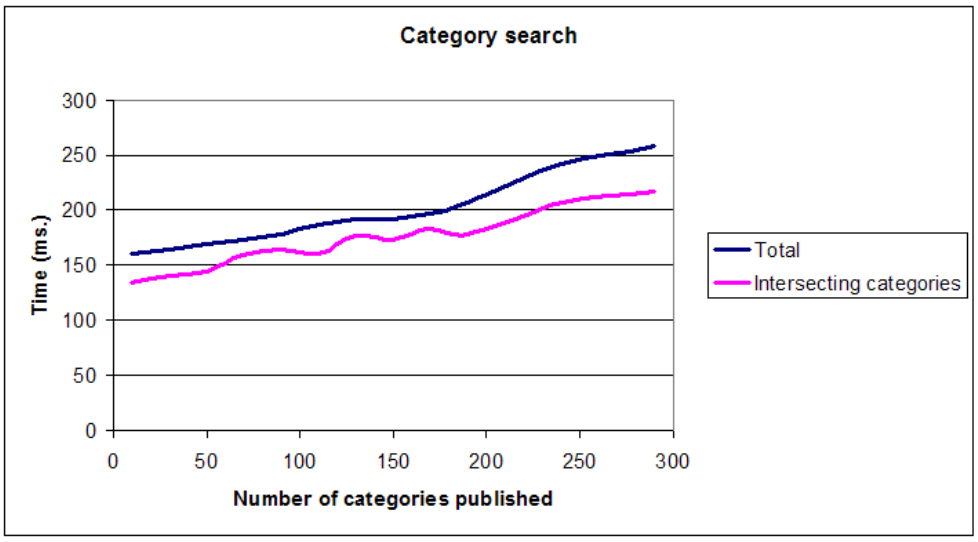

Figure 8.7: Set-based category search

basically equal to the times for the application of the capability filter, as the queries issued will be of the same type. However, they depend on the number of published categories, which is expected to be lower than the number of published services, i.e., the *Categories* TBox is expected to be smaller than the *Services* TBox and, therefore, results will be obtained in lower times. Some experimental results are shown in Figure 8.7.

Category update. Updating a category requires updating the taxonomy structure loaded in memory, whose cost will be marginal. Only if the set-based modelling of the category changes this update be costly, as it requires retracting the old concept from the *Categories* TBox, adding the new concept, and re-classifying the TBox. Therefore, this task will be NExpTime-complete.

The times obtained will correspond to the times required for classifying a TBox, as the addition and retraction of concepts to RacerPro is immediate.

Category deletion. This task will also be NExpTime-complete, as we have to retract the concept corresponding to the category from the *Categories* TBox and then re-classify such TBox. The response times expected are those estimated for the classification of a TBox.

Taxonomy deletion and retrieval. These tasks have a very reduced cost. In particular, taxonomy deletion only requires deleting an entry from our relational database and deleting the tModel defined at the UDDI repository, which will be done almost instantaneously. The retrieval of a taxonomy will only imply returning the structure loaded in memory, which will have a cost close to

zero.

Lower (or higher) response times might be obtained depending on the size and complexity of the domain ontologies these concepts refer to. Therefore, the times measured above must be regarded as an estimate which gives a hint on what time ranges we might obtain, but the real times required will ultimately depend on the performance of the DL reasoner for the particular ontologies used.

8.1.2 Coverage of applications

In this Section, we discuss to what extent the families of applications identified in the design of our abstract model can benefit from the particular model instantiation proposed and prototypically implemented. This discussion will try to determine whether the particular service location model provided can cover the needs of different applications and, if some application requirements are not completely met, what is missing.

We will focus on the location of services already available and whose descriptions have been published, but not on the process of describing services and publishing such descriptions; these tasks will be independent of the particular application scenario, as the same services will be available for different applications and in different usage scenarios. Still, it must be noted that the description of services in alternative ways as presented in Chapter 6 and the appropriate publication of these descriptions is what enables the correct location of services in different scenarios, and that this consideration has been an integral part of our model design.

8.1.2.1 Design-time location of services for their composition or integration into complex systems or processes

In Chapter 5, the location at design time of services that can solve a given goal and their static incorporation into a system or process was identified as a prominent family of applications which could benefit from enhance capabilities for the location of services. Scenarios such as the incorporation at design-time to a pricing system of services which could offer information about different aspects of the customer, about the product catalogue of a bank, or about market interest rates was used to illustrate the benefits of service-oriented system design and of the effective locating services providing a given value for their incorporation into a system or process.

This family of applications was characterized by the following features: a) human users (typically system or process designers) participate in the location of services; no dynamism is required and/or desired in the resolution of goals, but only support for a more efficient location of services which will be statically incorporated into a process or system, b) human users might or might not participate in the execution of the services located, c) the location of services at design-time for their

incorporation into a system or business process will generally be accomplished by IT professionals using the goals defined by domain experts, and d) the majority of application scenarios in this family are expected to require a moderate trade-off between efficiency of the location process and accuracy of results, i.e., moderately precise results will be acceptable if they are given in moderately low times.

Given the defining properties of this type of applications, we believe the instantiation of the abstract service location model proposed will generally cover the needs of such applications.

Description of goals by domain experts. Domain experts will be able to describe their goals in alternative ways according to their skills. Furthermore, they will be supported in this task in the way presented in the previous Chapter. However, domain experts might not have a precise idea of what trade-off between precision and efficiency of the location process should be chosen, as they will leave the actual location of appropriate services fulfilling their goals to other users, typically IT professionals. In this situation, the type of description required from domain experts should be as precise as possible, as this description will have to be used by IT professionals for locating services and, thus, represents the channel for communicating requirements from business experts to IT experts. Therefore, we expect in these cases all registry-side filters to be marked for application so that the business expert is supported in providing all possible types of descriptions of the capability sought.

Selection of filters and location of services. Once a goal has been described by domain experts, it will be usable for guiding the location of services which can provide a given value. IT experts, who will have more precise requirements on the trade-off between accuracy of location results and response times of the location process, will decide what filters will be actually applied for the location of services. This might imply completing some descriptions not provided by the domain expert with the support of the goal description assistant.

As a moderate trade-off between accuracy of results and efficiency is expected in most applications of this type, we expect users to use the capability filter for retrieving services from the registry. The reason is that the set-based modelling of the capability sought enables an accurate filtering of candidate services from a possibly big set of candidates and, as we have seen in the previous Section, the response times obtained experimentally are kept in limits we believe will suffice in most cases.

If lower response times would be required, users can choose to use the textual or capability filters, or both. This might happen in cases in which the set of available services is relatively small and, therefore, this simpler filters will yield a moderate number of results users can manually select. Furthermore, IT professionals can choose to directly use the UDDI inquiry API of the UDDI

repository included in our registry, keeping compatibility with old practices.

Regarding the application of consumer-side filters, we believe input availability and input-dependent effects filters will only be used in a few cases. The reason is that in most cases we do not expect the concrete knowledge the system, process or human user can provide when executing the service to be completely defined, as it will depend on the conditions which hold when the system or process is actually running, i.e., the knowledge of the service consumer (in this case, a system or process possibly with the intervention of a human user at run time) is not completely known at design time and, therefore, the knowledge base containing this knowledge is not defined or incomplete.

In this setting, we believe IT experts will in most cases find more convenient to apply registry-side filters which retrieve relevant services in terms of their capabilities with the precision chosen, and then manually select the service to be used possibly taking into account what type of knowledge is expected to be available at execution time. Notice that the retrieval of services from the registry can have considerable precision if a capability filter is applied, thereby easing the manual selection of an appropriate service from a (usually small) set of candidate services. Furthermore, different notions of match are distinguished with a partial ordering, which can guide the selection of services retrieved.

Still, there can be cases in which IT professionals or even domain experts demand a particular functionality from the service sought, i.e., they demand not only a particular value from a service, but a general relation between the initial conditions expected and what effects will be realized after service execution. If what particular knowledge will be available at run time is not fixed, our model does not enable the resolution of this type of goal.

We will see in Section 8.2 that there exist some works which try to address this issue but, unfortunately, they require an expressivity for which decidability is not guaranteed or they require the resolution of reasoning tasks not supported by state-of-the-art reasoners. We will also see that other works declare the general type of input values that can be provided by the consumer but they do not evaluate the relation between these values and the effects of service execution. Therefore, this type of matching is more similar to an input signature matching than to a functionality matching. Still, extending our model with this type of matching is part of our future work, as it might be useful in certain situations.

We can see that the location of services at design time is generally covered by our proposal, especially the retrieval of relevant services from the service registry, with enough flexibility for choosing a trade-off between accuracy and efficiency appropriate for the particular application. Furthermore, users with different profiles are supported for the provision of goal descriptions, which is expected to ease the communication of requirements between domain experts and IT experts. Limitations can arise, though, if the designer wants to locate services providing a given functionality.

While we expect this need not to be present in the majority of applications, resolving this issue will be subject of future study and extensions of our model.

Another limitation we will discuss in Section 8.1.3, and whose resolution will be part of our future work, is that a more fine-grained ranking of candidate services might be desirable so that filters do not only offer a grouping of relevant services according to their degree of match, but also a ranking of services within each set.

Finally, it must be noted that the separation of the roles played by business experts and IT professionals is not strict, and there might be cases in which the IT professional defines goals to be resolved, locates appropriate services, and wires them into the system or process, and cases in which business experts not only define a goal but also accomplish the location of a service achieving the goal defined. The support provided for describing goals and for locating appropriate services is expected to considerably help both type of users to successfully accomplish these tasks.

8.1.2.2 Location and execution of services by end human users

The second family of applications envisioned in Chapter 5 was the location of services by end human users. These users will formulate their objectives, and services which can be used to fulfill such objectives have to be located.

These applications, which include examples such as the location of eTourism services which can provide e.g. the booking of flights or seats on a train, were characterized by: a) human users will be available during all the service location process, b) human users will participate in the execution of services, c) human users will also describe their goals, and d) in most cases, the on-line locations of services is expected, i.e., results must be returned to the user in relatively short times so that we can consider the location process an on-line process, but only a moderate trade-off between efficiency of the location process and accuracy of results will be generally demanded.

We expect our model to appropriately cover most applications in this group, as we discuss in the following.

Description of goals. Users will be supported so that they can appropriately describe their goals. In particular, they will be able to select the level of precision and efficiency they require and, depending on these requirements, they will be guided in the provision of those types of descriptions of their goals needed for obtaining results with the general properties required.

Certain types of descriptions included in the model are highly usable by average users, such as the textual description of goals and the selection of categories sought services are expected to belong to. Given that we will support the provision of more complex types of descriptions based on descriptions most users feel comfortable with, we achieve an interesting level of usability.

In general, the task of describing goals is considerably eased to end users, and they are granted flexibility in that they can express their particular requirements on the location process and obtain candidate services under these requirements.

Consumer knowledge. Users might select the application of consumer-side filters which require access to consumer knowledge. This will require the definition of a knowledge base which contains consumer knowledge or, at least, that part of consumer knowledge usable for achieving the goal at hand. It must be kept in mind, though, that the user will be involved in the location process and might be able to provide, to some candidate service, extra information not contained in the knowledge base defined. While this knowledge base is expected to be provided by the consumer from scratch in our current prototype, we envision the existence of agents which keep track of consumer knowledge and keep the knowledge base updated so that it reflects at each point in time what knowledge the consumer has available for fulfilling his goals. These agents might also include policies for deciding what knowledge to add to this knowledge base and what knowledge to disclose. Although the usage of these types of agents and policies is not covered by our current prototype, it is envisioned as a possible extension. In addition, special purpose tools for some domains such as eTourism can be defined which keep that consumer knowledge which is commonly needed for resolving eTourism goals e.g. passport details, payment methods, travel preferences, details of planned holiday trips, etc.

Consumers can in any case choose not to apply filters which require access to consumer knowledge. In fact, we believe in some cases users might want to apply only registry-side filters and individually study services retrieved from the registry in order to evaluate their information requirements and decide on their use.

Location of services. Users will be able to select filters with different properties, which offers a certain guarantee that they will be able to locate relevant services in the times and with the accuracy chosen.

The model used for the location of services is centered on the value these services provide compared to the value required by a consumer. This means that we will find services which can provide a certain value and *which can be used by the user to realize such value*. This is in contrast with other proposals which are centered in matching the input-output signature of services (see Section 8.2). We believe end users will have objectives of type "Find a service which can book a seat on a given flight (and make sure that I can use this service for achieving the effects which fulfill my goal)", not objectives like "Find a service which accepts a flight, a passenger and a credit card as inputs, and which returns a flight reservation as output". For this reason, we believe our model is closer to the intuition of end users than most existing proposals.

In our model, the location of services truly driven by the value sought and the value

offered, with different trade-offs between accuracy and efficiency, and using filters which range from a textual or category matching to a complete evaluation of the service capability, of the satisfiability of information preconditions of the service, and of what effects can be obtained for available input bindings, is possible. In addition, different levels of match are identified, which can help users to perform the selection of services from those filtered as candidates for achieving the consumer's goal.

Some limitations exist in our model, though, like the lack of a more elaborated matching of textual descriptions, the lack of a more fine-grained ranking of the services located, or the lack of support to users for revising their goals if no perfect matches for their goals were found. These limitations will be discussed in more detail in Section 8.1.3, and solutions to address them will be an important part of our future work.

8.1.2.3 Run-time location and usage of services

The last group of applications we identified in Chapter 5 was the run-time location and usage of services, i.e., the resolution of goals without human intervention. This type of usage scenarios appears when a system has to dynamically react to changes such as a given service going off-line, or when the service to be used has to be dynamically chosen at run time e.g. for the dynamic configuration of a supply chain.

The main characteristics of this type of usage scenarios are: a) no human user will participate in the location process, b) human users might or might not be involved in the execution of the service located, c) goals will be defined by domain experts or automatically generated or parameterized at run-time, and d) the precision required from the location process will be higher than in the other types of applications we have discussed, as the selection of the service to interact with have to be accomplished automatically and, thus, the set of relevant services must be more accurately determined in order to avoid interactions with inappropriate services.

In the following, we summarize to what extent our model fulfills the needs of this type of scenarios.

Description of goals. Domain experts will be supported in describing their goals, which will be incorporated into a given system or process for their run-time resolution. The type of filters we expect to be selected will all be based on the formal description of services and goals (capability, input-availability and input-dependent effects filter), as a considerable degree of accuracy will be demanded. Therefore, users will have to describe the capability they pursue formally, and they will be supported in this task. However, it will often be the case that IT professionals revise these goals and/or complete their description before it is incorporated to the system or process which will require a particular value to be provided by a service dynamically selected at run time.

It must be noted that the goals defined might not be fixed at design time, but changed

during the system or process execution. In this setting, sometimes only a basic goal will be defined at design time, and it will be changed depending on some dynamic conditions. While support for providing the basic description of the goal is provided by our model, its dynamic parametrization is outside the scope of our proposal and it will be the responsibility of the system or process into which the goal is integrated.

Consumer knowledge. Consumer-side filters are expected to be used in this type of scenarios. Therefore, a knowledge base which contains available knowledge for resolving the goal has to be defined. However, this knowledge base does not have to be necessarily fixed at design time.

We expect cases in which the decision of what knowledge base will be considered for resolving the goal will be defined at design time, but this knowledge will evolve and its contents will not be fixed. Furthermore, cases in which even the complete knowledge base to be used is determined at run time depending on dynamic conditions are also possible. In any case, what particular knowledge is available at the precise moment the goal has to be resolved will be known and, therefore, we will be able to apply input-availability and input-dependent effects filters over the services retrieved from the registry.

As the service to be used has to be determined without user involvement, only knowledge available when the location process starts will be considered for the location of services, no matter whether a human user will be involved in the execution of the selected service or not.

Location of services. Filters based on the formal description of goals and services, and which take into account not only what capability is offered by available services and sought by consumers, but which also evaluate information preconditions of candidate services with respect to consumer knowledge and what effects might be obtained from using a service given available knowledge, can be applied. Therefore, we believe our model offers a considerably high precision level for the location of services, which makes it usable for the run-time location of services. This precision will help to restrict candidate services for resolving a goal to a relatively small set, which will make the direct negotiation with candidate services in order to establish a service contract achievable with a reasonable cost.

Still, what filters will be applied can be chosen and, therefore, users can decide on different levels of precision depending on factors such as how big is the set of available services. In general, users will be able to choose the filters they want to apply for resolving at run-time a goal and, therefore, to adjust the precision and accuracy of the location process to their particular requirements. However, a higher precision comes at a cost and we cannot offer a certain degree of precision if the location process has to offer results in low times.

Once services are located, a selection step will follow. However, this selection must be

fully automatic in this type of scenarios, which is not yet covered by our model. Furthermore, and while different notions of match of services are distinguished, a more fine-grained ranking of services located would be desirable in order to better guide the automatic selection of services.

In general, we believe the run-time location of services is sufficiently covered by our model, but not the usage of such services, as an automated selection step and the automatic execution of selected services is missing in this model. These tasks have been left outside the scope of the model, but they are an interesting path for future work.

8.1.3 Known limitations

Our proposal appropriately covers many aspects of the service location process in different scenarios. However, some limitations have been found and they are summarized in the following.

WSML and reasoners. The usage of the WSML family of languages is one of the key advantages of our proposal for different reasons: a) it provides a basic inter-operability layer (WSML-Core) in which major parts of domain ontologies can be described and used under different semantics, b) it allows for the usage of extensions of WSML-Core both in the direction of Description Logics and Logic Programming, which enables the description of different aspects of services based on the same domain ontologies but with different semantics, appropriate for different purposes, and c) it provides a unified syntax for all the languages used.

However, the efforts to provide an integrated WSML reasoner are still ongoing. The WSML2Reasoner framework[2] provides functionalities for translating WSML descriptions into the syntax used by different reasoners. Wrappers for KAON2, MINS, and Pellet are currently available, but these wrappers do not support tasks such as adding a new concept, described in WSML, to a Pellet TBox, or querying for ancestors of a concept described in WSML.

In this setting, we have not been able to use the WSML2Reasoner framework to interact, using descriptions in WSML syntax, with reasoners that provide the reasoning support our model requires. As a consequence, in the current implementation we require a translation into RacerPro and $\mathcal{FLORA}$-2 syntaxes of descriptions (domain ontologies, service descriptions and goals). This translation is currently obtained manually and encoded using custom non-functional properties of services and goals, and it requires duplicating domain ontologies in different syntaxes.

In addition, RacerPro only approximates nominals, whose use is permitted in the descriptions of sought and offered capabilities. In the current implementation, we have encoded nominals as pair-wise disjoint concepts and used this encoding in the version of descriptions and ontologies given in RacerPro syntax. Some incorrect inferences can be drawn from the resulting translation [Horrocks

[2] http://tools.deri.org/wsml2reasoner/

and Sattler, 2002], but in most cases the subsumption and satisfiability relations computed will be correct.

In general, we expect the WSML2Reasoner Framework to evolve in the near future so that it can be directly used to provide access to existing reasoners so that we can directly work with descriptions given in WSML syntax and, furthermore, if the wrapper to Pellet is extended so that certain functionalities we require from the reasoner can be properly accessed, we will be able to count with a reasoner which provides efficient and correct reasoning for $\mathcal{SHOIN}$. In addition, if a wrapper to $\mathcal{FLORA}$-2 is not provided, we will most likely use MINS as the WSML-Flight reasoner.

Domain ontologies. In our prototype we assume that all formal descriptions refer to a set of domain ontologies which is fixed when the registry and the client are started. However, in real applications new services might refer to new domain ontologies, which have to be incorporated to the registry, i.e., the set of domain ontologies used will evolve over time. Therefore, we expect in the future to extend the current implementation so that new ontologies can be loaded if published services refer to them, and so that consumers can also use them to formally describe their goals.

User support. The user support currently implemented has been conceived as a proof-of-concept of our approach. However, we believe a more elaborated version which includes support to users in working with WSML descriptions, isolating them from the WSML syntax, should be added in the future.

Expressivity. The expressivity allowed for the formalization of capabilities is restricted to the $\mathcal{SHOIN}$ DL. While this guarantees the decidability of the reasoning tasks required, it limits what can be described. In particular, it does not allow for chaining variables over predicates, which might be a limitation in certain cases. For example, a service whose capability is offering information about flights between any two cities as well as information about hotels at the destination city cannot be described in the $\mathcal{SHOIN}$ DL, as it would require chaining a variable over predicates.

In the description of input-dependent effects two predicates can refer to the same shared variable, which can be seen as a work-around this limitation. However, this is done outside the logic and, furthermore, it can only be used if the input-availability and input-dependent effects filters are applied.

Goal pre-processing. We have seen in Chapter 7 that the formal description of sought capabilities given by consumers must be pre-processed so that it does not include instances not in domain ontologies. In particular, these instances must be replaced by restrictions built using their definitions.

Furthermore, including sensitive data in the goal description e.g. a credit card number should be avoided, unless a trust relation with the registry is assumed. In this setting, we must

either require all clients to trust the service registry or, otherwise, include trust policies which describe what information can be disclosed to what parties and under what circumstances (see [Olmedilla et al., 2004], where a preliminary work on adding semantic trust policies to services and considering them for service discovery has been presented).

In our current implementation, the processing of goal descriptions for substituting instances not in domain ontologies has to be done manually if required, and we assume that the service registry is trusted. However, automating this processing and incorporating trust policies for deciding to what extent a registry is trusted and what information can be included in the goal description should be added in the future.

Textual filter. The textual filter currently implemented is very simple. In particular, it only offers search capabilities based on the exact occurrence of keywords, and it does not handle descriptions in (possibly multiple) different languages. The main reason is that performing an accurate textual search of services has not been among the main objectives of our prototype, but to demonstrate the feasibility of other filters and of the general model. Still, we believe move elaborated textual search capabilities and handling of descriptions in different languages should be added to the platform proposed. The support for multilingual descriptions will be possible when the annotation extensions introduced in [Toma and Foxvog, 2006], and which easily allows for incorporating multilingual non-functional properties in WSMO descriptions, become supported by current WSMO parsers and tools.

Real-world preconditions. Our model does not handle real-world preconditions due to the practical difficulties presented in Chapter 5.

Reputation. Our model completely relies on the description of services given by users, and it does not include reputation mechanisms in order to detect users who provide false descriptions of their services. However, we believe reputation mechanisms will be required if such a model is used in relatively open environments.

Ranking of results. While our model instantiation distinguishes different match degrees, a more fine-grained ranking of matches within sets of services all having the same match degree is desirable and it should be added to the model.

Explanations. We do not currently offer explanations of why a service is not a perfect match, which can guide the revision of goals by consumers in order to adjust their requirements to available offers. This is an interesting feature our model lacks and which would be of interest for future developments.

Availability of services and domain ontologies for evaluation. The model instantiation proposed has been empirically evaluated in order to have an estimation of the times required for the application of different filters and for the publication of service descriptions. However, this evaluation has been mostly based on randomly generated services, as there is a lack of an adequate set of real services described in terms of real ontologies which can be used to evaluate with more guarantees proposals for the location of services.

The only test collection we have knowledge of is the OWL-S Service Retrieval Text Collection (OWL-S TC)[3]. However, this test collection is based on OWL-S and, what is more important, it only describes the type of inputs and outputs required by a service, but not the real value it provides. Therefore, we find this test collection inadequate for the evaluation of a service location model based on the value sought by consumers and offered by services.

In this setting, we can have a good estimate of the efficiency of our model as it depends on the efficiency of well-known reasoners for which benchmarks exist, but we do not have a proper benchmark of our model which could improve our level of confidence in the evaluation results obtained.

Automatic selection and wiring of services. Neither the automatic selection of services from the set of services located by our prototype nor the automatic wiring of selected services to existing systems or processes has been addressed. However, these tasks have to be automatically accomplished for the run-time usage of dynamically located services and, thus, they will be part of our future research.

8.2 Related work

8.2.1 Software component retrieval

During the 90's, the computer science research community devoted some efforts to improve the state of the art on reuse of software components. The motivation for this work was to support the reuse of already existing and tested software components as one of the key factors for successful software engineering projects [Schumann and Ficher, 1997].

The efficient reuse of reliable software components providing a given functionality required efficient means to locate such components. A manual approach, in which the software engineer has to browse (possibly a big number of) libraries of components to locate a suitable one, was not found appropriate and, for this reason, research efforts were oriented towards the formal specification of the components functionality and the formal description of the sought component in order to enable

[3] http://projects.semwebcentral.org/projects/owls-tc/

semi-automatic retrieval of appropriate components.

8.2.1.1 Specification matching

Specification matching has been proposed in several works e.g. [Jeng and Cheng, 1992; Jeng and Cheng, 1993; Jeng and Cheng, 1995; Rollings and Wing, 1991; Zaremski and Wing, 1995] to evaluate how software components relate to a given query, i.e., user's need. Specification matching relies on the axiomatization of software components and user queries. A formal (logical) relation is then defined and whether a given query and component satisfy this relation is checked. Such a relation must capture the notion of reusability, i.e., if the relation holds for formally specified components and queries, it means that the component can be reused to solve the problem captured by the query.

At the formalization level, the following questions must be answered: a) How the components and queries are specified, and b) What is the relation to be checked for determining reusability. As related in [Chen and Cheng, 2000], a widely used axiomatization of components and queries is based on [Hoare, 1969]. [Hoare, 1969] provides a logical basis to prove some properties of a program, including determining whether a given program provides a certain functionality. The intended functionality of a program (C) is specified in terms of initial preconditions (C_{pre}), i.e., assertions about certain properties of the values taken by the relevant variables before the program initiation and the relations among them, and postconditions (C_{post}) i.e. the same kind of assertions as for preconditions but about the values after execution. The relation between the preconditions and postconditions of a given program is formulated as follows:

$$C_{pre}\{Q\}C_{post}$$

interpreted as "If the assertion C_{pre} is true before initiation of a program Q, then the assertion C_{post} will be true on its completion." [Hoare, 1969]

Based on this type of axiomatization, most works in specification matching specify a component C as a 2-tuple of predicates (C_{pre}, C_{post}), being C_{pre} the precondition of the component and C_{post} its postcondition. Similarly, a query Q is specified as (Q_{pre}, Q_{post}). Preconditions of the component are logical formulas that must hold prior to the use of the component[4], and postconditions are logical formulas that are guaranteed to be true after the execution of the component. The preconditions and postconditions of the query give a characterization of the desired component in terms of its preconditions and postconditions. Query preconditions can be interpreted as a description of the initial states for which the sought component must guarantee the fulfillment of the query postconditions.

[4]If the preconditions do not hold, the behaviour of the component is undefined.

Notion of match	**Definition**
$M_{exact-pre/post}$	$(Q_{pre} \leftrightarrow C_{pre}) \wedge (C_{post} \leftrightarrow Q_{post})$
$M_{plug-in}$	$(Q_{pre} \rightarrow C_{pre}) \wedge (C_{post} \rightarrow Q_{post})$
$M_{plug-in-post}$	$C_{post} \rightarrow Q_{post}$
$M_{guarded-plug-in}$	$(Q_{pre} \rightarrow C_{pre}) \wedge ((C_{pre} \wedge C_{post}) \rightarrow Q_{post})$
$M_{relaxed-plug-in}$	$(Q_{pre} \rightarrow C_{pre}) \wedge ((Q_{pre} \wedge C_{post}) \rightarrow Q_{post})$
$M_{guarded-post}$	$(C_{pre} \wedge C_{post}) \rightarrow Q_{post}$
$M_{partial-comp}$	$C_{pre} \wedge Q_{pre} \wedge C_{post} \rightarrow Q_{post}$
$M_{exact-pred}$	$(C_{pre} \rightarrow C_{post}) \leftrightarrow (Q_{pre} \rightarrow Q_{post})$
$M_{gen-pred}$	$(C_{pre} \rightarrow C_{post}) \rightarrow (Q_{pre} \rightarrow Q_{post})$
$M_{spe-pred}$	$(Q_{pre} \rightarrow Q_{post}) \rightarrow (C_{pre} \rightarrow C_{post})$
$M_{exact-pred-2}$	$(C_{pre} \wedge C_{post}) \leftrightarrow (Q_{pre} \wedge Q_{post})$
$M_{gen-pred-2}$	$(C_{pre} \wedge C_{post}) \rightarrow (Q_{pre} \wedge Q_{post})$
$M_{guarded-gen-pred}$	$(Q_{pre} \rightarrow C_{pre}) \wedge ((C_{pre} \rightarrow C_{post}) \rightarrow (Q_{pre} \rightarrow Q_{post}))$

Table 8.2: Summary of specification matches

[Zaremski and Wing, 1997] explores different notions of match for retrieving formally-specified software components. A software component C is described in terms of their signature, C_{sig}, and their behaviour specification, C_{spec}. The former describes statically checkable information (component's type information), while the latter describes the component dynamic behaviour (functionality), defined in terms of preconditions and postconditions. Similarly, a query Q is described by its preconditions and postconditions. All the preconditions and postconditions are first-order formulas.

Table 8.2, adapted and extended from [Chen and Cheng, 2000], provides a summary of different notions of match presented in [Jeng and Cheng, 1995; Zaremski and Wing, 1995; Penix and Alexander, 1997; Schumann and Ficher, 1997; America, 1991; Liskov and Wing, 1994; Dhara and Leavens, 1996] (see [Lara et al., 2004a] for an explanation of each notion of match).

8.2.1.2 Relation to the location of services based on their value

Services are described using a similar approach to Hoare's axiomatization of software components, being the major difference that services can also have real-world effects and pose conditions on the real-world for service provision. Queries in software component retrieval are described similarly to goals. Therefore, it is not surprising that some of the notions of match discussed in the context of software component retrieval have been used by works trying to solve the service location problem, as we will see below.

However, most of the notions of match proposed for the retrieval of software components focus on locating a software component that can be used in the place where the software component represented by the query could, while for the location of services we focus on finding services

which can provide a given *value*. For example, the *plug-in* matching notion in Table 8.2, and which corresponds to cases in which the postconditions of the component are more specific than the postconditions of the query and preconditions of the component are more general than preconditions of the query, is interpreted as "the component C can be used can be used for obtaining the behaviour specified in Q, but not the other way around". This means that preconditions of a component are interpreted as the description of a set of initial states acceptable by the component, and the preconditions of the query as the description of the set of states which might hold before using the component. Similarly, component postconditions are interpreted as a description possible final states reached after using the component, and query postconditions are interpreted as a description of acceptable final states.

Furthermore, descriptions of information preconditions and postconditions are usually not restricted in expressivity, and full-first order logic is allowed. While this enables the accurate description of preconditions and postconditions, reasoning in full first-order logic is undecidable.

In general, service-orientation places its focus not only on reusability, but also on a new form of designing systems in which services providing a *meaningful value*, and which can be seamlessly accessed, are available. In this context, consumers will have the objective of finding services which provide a given value and which can be used given current conditions and, furthermore, results will be required in relatively low times. This will imply setting limits on the expressivity of descriptions and adjusting the type of modelling used for describing consumer's goals and service capabilities so that the value required and offered can be expressed while keeping the expressivity of descriptions and the complexity of reasoning under certain limits. Still, some of the notions used for specification matching have been reused by works on service discovery, and they can be used to define additional filters which complement the filters proposed in our model instantiation.

8.2.2 DL-based matching

In the following, we summarize existing works on matching services and consumer's needs based on DL descriptions and DL reasoners.

Castillo et al. present in [Gonzalez-Castillo et al., 2001] one of the first proposals for using DL descriptions of services and goals and exploiting DL reasoning for matchmaking of services. A DAML+OIL [Connolly et al., 2001][5] ($\mathcal{SHOQ}(\mathcal{D})$) concept is used to represent an offered service. Similarly, a request is represented by a DAML+OIL concept, and matches for this request are services represented by concepts equivalent to, sub-concepts of, super-concepts of, or sub-concepts of any direct super-concept of the query whose intersection with the query is satisfiable. Racer has been used for reasoning over DL descriptions.

[5]DAML+OIL is the predecessor of the OWL language.

We can see that the proposal by Castillo et al. is similar to our capability filter. However, we introduce actions in the description of capabilities offered and sought and intentions for these descriptions, thereby enabling a more precise modelling of the value of services; Castillo et al. only use concepts such as $Serv \equiv Computer \sqcap = 1 hasPrinter$ to represent services and requests, which is insufficient for really capturing what value is required or offered.

Paolucci et al. have also been pioneers in using DL reasoning for the matchmaking of services. In [Paolucci et al., 2002], they propose the matching of inputs and outputs in OWL-S profiles[6], which were described by concepts in DL ontologies over which subsumption relations can be computed. In particular, given a request, modelled as an OWL-S profile, and an OWL-S service, each output concept in the request must be matched by an output concept in the service, and each input concept in the service profile must be matched by an input concept in the request.

Different degrees of match between output concepts are distinguished, namely (degrees of match are presented starting with the most preferred one and ending with the least preferred one): a) an exact match, if the concepts are equivalent, b) a plug-in match, if the concept in the advertisement is more general than the concept in the request, c) a subsumes match, if the concept in the advertisement is more specific than the concept in the request, or d) a fail, if no subsumption relation is identified. The same degrees of match apply to inputs, but the direction of the subsumption relations is reversed. The degree of match of a service to a request will be given by the score obtained for each of the outputs and, in case of a tie, by the score obtained for each of the inputs.

We can see that this work focuses on matching inputs and outputs of a service and of a request, requiring the prospective consumer to specify what inputs and outputs are desired. This can be seen, thus, as an interface matching using, instead of input and output datatypes, input and output concepts; the real value provided by the service or required by the consumer is not modelled by only semantically annotating the inputs and the outputs of the service. Regarding the matching notions used, they are derived from previous works in the retrieval of software components introduced above, and they are used to measure the degree of satisfaction of input requirements of the service by the inputs declared by the consumer, as well as to measure to what extent the outputs required and outputs offered are compatible. If we assume a universal intention of both descriptions, an exact match and a plug-in match both correspond to a perfect match in our platform, a subsumes match corresponds to a partial match, and a failed match to a non-match only when both concepts do not intersect; otherwise, we consider this case a partial match.

In a nutshell, this work is interesting for matching the input-output signature of available

[6]Strictly speaking, DAML-S, the predecessor of OWL-S which used DAML+OIL instead of OWL, was used, but the same discussion can be applied to OWL-S descriptions and we will assume in the following OWL and OWL-S were used.

services to a signature required by a consumer, but does not really match services based on the value provided and offered. Still, it can serve as a basis for incorporating a new filter to our platform which takes this type of signature into account.

Li and Horrocks propose in [Li and Horrocks, 2003] the matching of inputs and outputs of services and requests in a similar way [Paolucci et al., 2002] does, but with two major differences: a) OWL-S service profiles and service requests are modelled by a single concept, therefore enabling the classification of the complete profiles of published services in a subsumption hierarchy prior to issuing any request, and b) the degrees of match used are applied to the complete service and request profiles, and intersecting concepts are also considered a match (a new degree of match, *intersection*, is introduced).

This work has inspired our modelling of service capabilities and of sought capabilities of goals by a single DL concept, enabling the pre-classification of services at publication time and the posterior efficient querying of the DL reasoner TBox. However, the work by Li and Horrocks is limited to the matching of inputs and outputs, without taking into account neither the real value offered by the service, and modelled by pre-defined actions in our platform, nor the intention of the modeler of the service profile or request.

In [Noia et al., 2003; Noia et al., 2004], Di Noia et al. present a proposal for matchmaking demands and supplies in an electronic marketplace based on Description Logics. In particular, three notions of match are distinguished: a) an exact match, meaning that the demand can be completely satisfied by the supply, and which corresponds to the equivalence of the DL concepts formalizing the demand and the supply considered, b) a potential match, meaning that some requests in the demand are not specified in the supply, and corresponding to the satisfiability of the intersection of the demand and supply concepts, and c) a partial match, meaning that some requests are in conflict with the supply, and corresponding to the unsatisfiability of the intersection of the supply and demand concepts. In the case of a potential match, this work gives the possibility of asking the supplier whether some features although not advertised are anyway available and, in the case of a partial match, it allows for the revision of the demand by relaxing (retracting) some restriction. Furthermore, potential and partial matches are ranked so that users have an estimation of how far a supply was from being an exact match (potential matches) or of how big the conflict between the demand and the supply is (partial matches).

Compared to our work, the proposal by Di Noia et al. focuses on the matching of concrete supplies and demands e.g. a set of hotel rooms or a particular flight modelled as DL concepts, not on the matching of the capabilities of services and of their abstract effects. The matching mechanism in [Noia et al., 2003; Noia et al., 2004] focuses on whether supplies and demands are in conflict or not and on how different they are, while our capability filter, which would be the filter most

closely related to the algorithm proposed, focuses on whether the capability offered by a service can potentially provide all the effects required by a consumer, *taking into account whether all the effects described are required or only some of them.* In particular, an existential intention seems to be implicitly assumed by Di Noia et al. for demands, and a universal intention for supplies, but this is not completely clear. In general, an exact match corresponds to a perfect match in our framework independently of intentions; a potential match can correspond to a perfect match, a possible match, a partial match, or a possible partial match in our platform depending on the intentions associated to the concepts evaluated and on the particular subsumption relation which may hold; and a partial match corresponds to a non-match in our platform.

In [Colucci et al., 2005], Colucci et al. propose an approach for the matching of offers and requests in an E-Marketplace which includes negotiation. In particular, demands and supplies are modelled as two sets of constraints: non-negotiable (strict) constraints $\mathcal{ST}$ and negotiable constraints $\mathcal{NG}$. If the intersection of offers and supplies is not satisfiable (a partial match in [Noia et al., 2003; Noia et al., 2004]), possibilities for negotiation are detected. Using concept contraction and abduction [Colucci et al., 2003], what constraints are causing the unsatisfiability of the intersection are found and, if such constraints are negotiable, users can consider retracting them; otherwise, the only possibility is that the user turns some strict constraints into negotiable constraints. In this way, negotiable constraints can be introduced and the matching mechanism can find negotiation spaces thus guiding users in such negotiation.

Benatallah et al. present in [Benatallah et al., 2003; Benatallah et al., 2005] an approach to service discovery in which a DL for which the difference operator[7] is semantically unique[8] is used to express the service profile inputs and outputs of an OWL-S (DAML-S) description (preconditions and effects are not considered). Service discovery is not formulated as a subsumption reasoning problem but as a rewriting problem, i.e., as the problem of how to rewrite the request in terms of available services. Given a service request and a service profile, a combination of services that satisfy as much as possible the outputs requested and that require as few as possible inputs not provided in the request is selected. Such combination is the so-called best profile cover of the request using the set of available (advertised) services.

Using the difference operator, the services whose outputs satisfy at least one of the outputs in the request, i.e., the difference between the outputs of the request and the outputs of the service is not the whole set of outputs in the request, are identified. The same operation (using the difference operator in reverse order) is performed for the inputs. The set of services that have the smallest set of outputs in the request and inputs of the service not satisfied is selected. In this way, the best

[7] Given concepts C and D, C-D is the information expressed in C and not in D.
[8] Please refer to [Teege, 1994] for details.

combination of services that provide the higher number of requester outputs and requires less inputs not given in the request is selected. The problem of determining such set of services is reduced to the problem of computing minimal traversals with minimal costs in an hypergraph.

This approach has the advantage of enhancing the discovery process using a simple type of service composition. In addition, incomplete matches can be found and information about what outputs or inputs are missing for a complete match is given, serving as an explanation that the requester can use to refine his request. However, this work is restricted to matching input/output signatures, without truly reflecting the value offered by a service. Furthermore, and as noted in [Colucci et al., 2005], performing a difference operation needs a subsumption relation between descriptions to be found. This condition may make this approach hard to use for matchmaking, as descriptions which overlap might also be considered. Finally, there is no algorithm able to compute an exact concept difference in a DL endowed of the negation constructor.

Baader et al. present in [Baader et al., 2005] an action formalism based on Description Logics and analyze how the choice of the DL influences the complexity of reasoning about services. In particular, projection and executability are analyzed; projection is defined as the problem of checking whether a certain condition always holds after the successful execution of a service, given our knowledge of the current state of the world, and executability is defined as the problem of checking whether, given our current and possibly incomplete knowledge of the world, we can be sure that the service is executable, i.e., all preconditions are satisfied.

This work is of interest as it discusses problems related to the current and future state of the world, of which we will usually have an incomplete knowledge, and it discusses the complexity of reasoning tasks which might be of interest if real-world preconditions and an explicit knowledge of the state of the world are considered. However, this work lacks an analysis of how the state of the world will be known and maintained, which is a pre-requisite for applying the reasoning procedures proposed. In general, the paper focuses on the complexity of the reasoning tasks that would be used for deciding executability and projection of services, but mostly ignores the practical problems for keeping an account of the current state of the world.

In [Grimm et al., 2004; Grimm et al., 2006], Grimm et al. discuss the correspondence between the DL modelling of requests and offers by users and their intuition. Remarkably, in [Grimm et al., 2006] they introduce the autoepistemic knowledge operator **K** [Donini et al., 1998] in the description of services and goals, thereby allowing for local closed-world reasoning [Etzioni et al., 1994]. In this way, the user can choose to use closed-world reasoning in particular parts of descriptions. This is a possibly interesting feature, as it grants users control over when the OWA must be made over descriptions and when the world must be locally closed.

In [Hull et al., 2006], Hull et al. propose a method for matching stateless (information providing) services which takes into account the relation between inputs and outputs of the service (its functionality) and not only its input/output signature. The descriptions used are based on Description Logics, and it performs both an input-output signature matching in the way [Paolucci et al., 2002] does, and a matching of the relation between inputs and outputs; the latter check is reduced to the reasoning task checking containment between two conjunctive queries wrt. a TBox. This reasoning task is proved to be decidable, but neither tight complexity bounds nor an implementation are currently available. Therefore, and while a step beyond input/output signature matching performed by most DL-based proposals, the realizability of the proposal by Hull et al. is still to be demonstrated.

In general, most existing proposals for the location of services based on DL descriptions and reasoning are limited to the matching of input and output signatures, and they concentrate on matching a single type of description without taking into account the diversity of application scenarios to which these proposals might be applied. Some of these works have inspired our capability filter, and to some extent our input-dependent effects filter. However, we have added the usage of pre-defined actions and intentions to descriptions, and we have focused on describing and matching the value offered by services and requested by consumers. Furthermore, we have introduced the parameterization of the set of achievable effects with the input bindings a consumer can provide, and we have proposed a comprehensive model which tries to address the service location problem as a whole.

Some existing proposals introduce features not present in our model, such as the identification of negotiation spaces, the possibility of locally closing the world in descriptions, or ranking of results according to user preferences (see [Lukasiewicz and Schellhase, 2006]). Some of these features have been identified as possible extensions to our model, and their incorporation to our platform will be part of our future work.

8.2.3 Extension of UDDI registries

While the works introduced above focus on how to describe services and goals and on how to match such descriptions, works exist which aim at extending the capabilities of UDDI registries for the location of services. The most relevant works of this type are summarized in the following.

Srinivasan et al. present in [Srinivasan et al., 2004] an approach for adding semantic search capabilities to UDDI. In particular, OWL-S is used to semantically describe services and goals, and the matching method used is an optimized version of the method presented in [Paolucci et al., 2002] and described above. In this work, a UDDI registry is used for storing descriptions. For this purpose

a mapping of the profile of OWL-S services to UDDI is proposed, so that an OWL-S service profile can be stored in the UDDI repository using category bags and appropriate tModels. Furthermore, the UDDI publication and inquiry ports are kept accessible and a new port is defined so that users can choose whether they want to use regular UDDI publish and search capabilities, or whether search capabilities added on top of the UDDI registry will be used. A similar approach has been presented in [Pokraev et al., 2003].

This work has inspired the architecture of our registry, as we pointed out in Chapter 6. However, we do not only store the service capability but the full service description is stored, we use WSMO for encoding our descriptions, we use a different semantic matching algorithm in our capability filter and a different kind of formal description of the service value, and we allow for the usage of other types of descriptions of the capability offered and sought, respectively, and of filters.

In [Luo et al., 2005], an approach which also tries to add semantic matching to UDDI is presented. Semantic markups are stored in the UDDI data model and used for processing queries, but no modification of the UDDI interface is done. This is achieved by resolving and indexing ontology relationships at publishing time, so that queries can be answered by UDDI using syntactic descriptions which have the behaviour of a semantic matchmaking. However, not the full expressivity of OWL is allowed and, actually, the expressivity descriptions can use is very limited (intersection and negation are not permitted).

In [Akkiraju et al., 2003], the UDDI inquiry API is extended in order to incorporate the semantic description of a sought profile, described by its inputs and outputs. A matchmaking method similar to the method proposed by in [Srinivasan et al., 2004] is followed. However, this work also enables the exact matching of the categories given besides the matching of inputs and outputs. In this sense, this work already introduces multiple registry-side filters in a similar way we do. However, the category matching is done by the UDDI registry and, thus, only services categorized exactly under the categories given are matched, and the type of semantic matching used is limited to the input/output signature of the service.

Finally, [Colgrave et al., 2004] proposes an approach in which users can integrate multiple external matching services with a UDDI registry to support multiple external service description languages. In particular, service providers and requesters can publish the location of external descriptions of service capabilities and requirements, respectively, in a UDDI registry, requesters can indicate whether a external description matching has to be performed by the UDDI registry, third-party service providers can publish their matching engines as services in a UDDI registry, and a UDDI registry can select suitable external matching services and dynamically invoke the selected matching service to carry out external description matching of services. This grants flexibility in the

incorporation of new matching methods to be performed for retrieving relevant descriptions from the registry.

8.2.4 Approaches with multiple filters

In the following, we introduce some works which allow for the application of multiple filters for the matchmaking of services (or agents).

Sycara et al. define in [Sycara et al., 2002] the so-called Language for Advertisement and Request for Knowledge Sharing (LARKS), used to describe agent capabilities[9], and which has inspired our model for the location of services based on the application of filters. A capability specification in LARKS defines the context of the specification, the input and output variables, constraints on these variables, the ontological descriptions used, and a textual description. As in OWL-S, a capability specification can be treated as a request or as an advertisement. Given a request and an advertisement, the matchmaking process can apply five different filters, namely [Sycara et al., 2002]: 1) Context matching, 2) Profile comparison, 3) Similarity matching, 4) Signature matching, and 5) Constraint matching. The combination of filters that will be actually applied can be selected by the requester, although some combinations of filters are pre-defined. In this way, the trade-off between accuracy and efficiency of the discovery can be selected.

The context in LARKS is defined as a set of keywords that describe the domain of the agent. Context matching computes the distance [Rosenfield, 1994] between the keywords of the request and the advertisement, and the subsumption relation between the concepts corresponding to the pairs of most similar words. The computed similarity, using a given threshold, will determine if there exists a match.

Profile comparison treats the request and advertisements as documents and determines the degree of similarity between them based on frequency and relevance of words in a document. If the similarity exceeds a given threshold, there is a match.

The profile comparison does not consider the structure of the specification, while similarity matching does. Such similarity is computed combining distance values for the pairs of input and output declarations, including their constraints. Again, if the computed similarity exceeds a threshold, the result will be a match.

Signature matching check if the input and output declarations of the request and the advertisement match. This is done using subtype inference rules and subsumption reasoning.

Constraint matching uses the notion of plug-in match. The logical implications are checked using subsumption reasoning for Horn clauses. Constraint matching uses the signature filter and, therefore, these two filters work together.

[9] Although LARKS is a language for describing agent capabilities, it can equally be applied to services.

The application of different filters for discovery in LARKS has the advantage of customizing the trade-off between accuracy and efficiency by deciding the filters that will be applied. In addition, the input and output constraints can include the relation between the input and the output, capturing more accurately the functionality of the agent. However, and while the work in LARKS has been provided us with the conceptual basis for using different filters in our model and enabling the selection of filters by consumers, the types of filters used by LARKS were not found the most appropriate ones in a service-oriented model. As the focus of consumers when searching services will be placed on the value provided by such services, the types of descriptions used have to concentrate on this aspect of services. Furthermore, LARKS does not differentiate between registry-side and consumer-side filters, and filters which make use of the knowledge a consumer has actually available are not considered. Last, but not least, our model includes support to users in describing their services and goals; while it is stated in [Sycara et al., 2002] that usability is a requirement, no particular strategy is used to ensure it.

Inspired by LARKS, Kawamura et al. present in [Kawamura et al., 2004] a service matchmaker which allows for the application of alternative filters and which enhance the capabilities of UDDI registries. In this work, the API used has the same format as the UDDI API, although a layer is inserted between this API and the UDDI registry which provides new capabilities.

Services and requests can semantically describe their inputs and outputs and constraints on such inputs and outputs. The matching process is organized as a series of the following filters: a) a namespace filter, which only checks whether a request and a candidate filter have some namespace in common, i.e., whether they have some terminology in common, b) a text filter, which applies a Term Frequency Inverse Document Frequency method for deciding relevant services, c) a domain filter, which checks whether each registered service and the requested one belong to an ontology domain, i.e., to a given subtree in an ontology tree, d) an I/O type filter, which checks to see if the definitions of the input and output parameters, defined by ontology concepts, match (where a match is determined similarly to the proposal in [Paolucci et al., 2002]), and e) a constraint filter, which compares the constraints to determine if the registered service is less constrained than the request. From these filters, Kawamura et al. state that the first three are only meant to reduce the computation time of the last two filters.

Semantic descriptions are encoded in WSDL descriptions using a structure similar to an OWL-S profile, and tools for the description of services and requests are built which try to provide an appropriate graphical interface for the creation of descriptions.

This proposal is the most similar one to our work, but the following major differences can be identified: a) [Kawamura et al., 2004] expects all filters to be applied at the registry side, while we organize our location process in two stages where the second one is applied at the consumer-side

where actual consumer knowledge is available and, thus, it can be used for deciding on the usability of candidate services, b) we apply filters which rely on descriptions we believe are more usable for business experts, who will have in many cases the responsibility of defining goals, c) the type of filters we introduce are oriented towards detecting services which are usable for fulfilling a goal in terms of the value they offer, and some of the filters introduced in [Kawamura et al., 2004] are oriented towards input/output signature matching or are too vague as they only determine whether the services refer to some common vocabulary, c) while our prototype is less elaborated in graphically describing services or goals, we help users to provide types of descriptions they are not familiar with by using pre-defined, reusable capabilities, d) an evaluation of the system presented in [Kawamura et al., 2004] is not available, and e) it remains unclear to what extent constraints on inputs and outputs capture the value offered by services and required by consumers in an appropriate way, and how efficient the determination of a plug-in match between offers and requests using these constraints can be performed in acceptable times.

In [Jaeger et al., 2005], the usage of custom-defined matching methods is introduced. In particular, this proposal allows for: a) the matching of service categories, which are assumed to be semantically described and subsumption reasoning is used for their matching, b) the matching of service outputs, c) the matching of service inputs, and d) the application of custom-defined matching methods. Each matching is regarded as a stage to which weights can be assigned, therefore giving more importance to any of the matching methods applied. Furthermore, matching is assumed to be performed at the client-side so that clients can incorporate custom-defined matching methods.

Compared to this proposal, our works enables the application of both registry-side and consumer-side filters. Furthermore, we believe the types of filters we use and their custom combination grants more flexibility to users than the filters proposed by Jaeger et al. in [Jaeger et al., 2005]. However, the application of weights to different filters is an interesting feature of the model proposed by Jaeger et al., and it will be considered for future extensions of our model.

OWLS-MX is a hybrid matchmaker for OWL-S services proposed by Klusch et al. in [Klusch et al., 2006]. The distinguishing feature of this work is that it combines a semantic matching based on the input/output signature described by an OWL-S profile, with a set of alternative filters which use different IR similarity metrics for performing a non logic-based matching of services but, instead, these filters compute the syntactic similarity between requests and offers. These alternative filters and the results obtained from the experimental evaluation presented in [Klusch et al., 2006] are of great interest, and they are firm candidates for future incorporation to our model.

Based on OWLS-MX and on some aspects of the work we have presented in [Keller et al., 2004a], a similar matchmaker, called WSMO-MX [Kaufer and Klusch, 2006], has been developed. The filters applied in this case compare the types of inputs and outputs declared by services and

requests, more detailed constraints on these types, the relation between inputs and outputs, and syntactic similarity.

8.2.5 Other related works

The works presented by Klein et al. in [Klein and König-Ries, 2004; Kuester et al., 2007] recognize the problem of signature matching, as services with different signature might provide the same value, and propose a method for describing offers and requests which tries to capture the actual functionality offered and requested, respectively. In these works, the authors do not use any logical formalism but propose a type of description they call DSD (Diane Service Description) based on its own light-weight ontology language. An advantage of this approach is that it also enables composition of services when a single service cannot solve the request.

In [Valle and Cerizza, 2005; Valle et al., 2005], Della Valle et al. present a discovery engine for WSMO based on the idea of having pre-defined generic goals and services and using mediators to resolve heterogeneity and to explicitly link generic goals and services, of which particular instances will be available. This work is partly based on the work we have presented in Chapter 4 and on the ideas presented in [Keller et al., 2004a; Keller et al., 2005], but while it might work well in closed domains it is a too rigid proposal for solving general service location problems.

In [Zein and Kermarrec, 2004], an F-Logic-based approach for the description of services is presented. The service concept describes the service in terms of non-functional properties such as provider, location, service type, etc., its behaviour (inputs, outputs, and the relation between them), and its operations (including the inputs and outputs for each operation). A requester expresses its goal as a query in terms of the service non-functional properties, behaviour, and operations. However, the service description cannot model world-altering services i.e. services with effects on the world. Furthermore, as it uses simple query answering, the description of the services is limited to ground facts, which considerably reduces the expressivity allowed for describing the service value and does not allow the description of capabilities as sets of abstract effects.

METEOR-S WSDI [Verma et al., 2005; Sivashanmugam et al., 2004] is a P2P infrastructure for matching services. It relies on adding semantic annotation to WSDL operations, inputs and outputs, as well as on adding preconditions and effects to the description of operations in WSDL using the extensibility elements of WSDL. The matching process uses subsumption reasoning to match operations, inputs, outputs, preconditions and effects of the annotated services against the ones of a service template describing the request.

While the matching of WSDL operations, inputs, outputs, preconditions and effects is limited to a simple exact match, an innovative element introduced by this work is the annotation

of registries and its specialization in domains in order to distribute the load for the matching of a potentially big number of available services. In this way, particular registries will be specialized in a given domain and an appropriate annotation describing this domain will be made available so that the service location process can first determine in what registries relevant services might be found.

Other related works include [Dong et al., 2004], in which similarity search for services based on IR techniques but taking into account the structure of WSDL documents is proposed, [Fernandez et al., 2006], in which a service discovery mechanism which takes into account the types of interactions services can be used in and the roles they play in such interactions is presented, [Spanoudakis et al., 2005], in which a framework for the run-time replacement of services based on the syntactic description of the services structure and behaviour is described, [Lynch et al., 2006], in which a proactive, subscription-based discovery mechanism is proposed, [Toma et al., 2005], in which a P2P approach to service discovery using keyword matching is described, and works such as [Mokhtar et al., 2006] and [Constantinescu and Faltings, 2003], where techniques for preprocessing the semantic relation between concepts and for indexing services are proposed in order to speed up the service location process.

Chapter 9

Conclusion

Service-orientation has rapidly gained acceptance in the last years as an architectural paradigm which can enable a true reuse of IT assets and an increase of the cooperation possibilities among systems, processes and businesses by reducing the integration burden. The progressive but relatively fast adoption of this paradigm has meant the adoption of (all or some of) its core design principles (service encapsulation, loose coupling, explicit contracts, abstraction, reusability, composability, autonomy, statelessness and discoverability), facilitated by a set of new languages, technologies and, in some occasions, middleware.

A growing number of business have started to embrace service-orientation, creating initial services, evolving these services to be truly meaningful and reusable pieces of value, composing services, and monitoring and governing their services infrastructure. Progress in the SOA adoption curve (or maturity pyramid), has raised greater awareness of some challenges whose resolution is highly desirable for taking the positive impact of service-orientation a step further; most of these challenges have been summarized in the service-oriented computing research roadmap presented by Papazoglou et al. in [Papazoglou et al., 2006].

Among these challenges, we can find a more effective, efficient and dynamic location of services. As SOA adoption increases and reusable services are made available, the need for mechanisms which can support service-oriented system design, run-time location and replacement of services, and even the location and usage of services by end-users, has naturally arose. Not surprisingly, as research challenges become apparent, and under an incipient industry demand for solutions, proposals have also appeared which try to go beyond what current technologies offer.

In this context, and given the state-of-the-art on the (semi)automatic location of services, we have perceived paths of improvement we have worked in and which have been presented in this document. Our work has focused on the design of an abstract and comprehensive model for the

location of services based on the analysis of the diversity of application scenarios where an effective location of services is required, and where flexibility is demanded, and on the concrete instantiation of this abstract model to offer a tangible proposal for the location of services which includes a prototypical implementation. A brief account of the major contributions of our work is presented next.

9.1 Major contributions

Conceptual model of services and goals. Based on the OASIS reference model for SOA, we have described a conceptual model of those elements relevant for the location of services. We have also discussed the alignment of this model to WSMO (to whose definition the PhD candidate has contributed) and to other frameworks and, furthermore, we have provided a formal characterization of services and goals based on Transaction Logic which constitutes a precise account for different aspects of services and goals.

Identification and general analysis of families of applications. We have provided a general analysis of application scenarios which helps to realize the diversity of situations, with different requirements and involving different user profiles, which can benefit from better service location capabilities. Furthermore, three main groups of applications have been identified.

Design of a comprehensive and flexible abstract model for the location of services. We have presented a model which is comprehensive as it identifies and discusses the various tasks involved in the location of services, which is flexible as it recognizes that requirements might vary and that diverse application scenarios must be covered, which has usability as a central concern, and which pays special attention to practical considerations trying to guarantee the practical applicability of the model.

Particular instantiation of the abstract model. We have described a particular instantiation of the abstract model designed which concretizes aspects left open by the abstract model, and which has the following major features:

- It admits alternative types of descriptions of services and goals, both formal and non-formal. These alternative views of the value offered and required by services and goals, respectively, will improve the usability of the model by users with different profiles and will enable the application of alternative matchmaking methods with different properties which might fit better to different scenarios.

- The types of descriptions used are integrated into an existing framework for the description of services (WSMO) and formal descriptions are expressed using existing languages with formal semantics (the WSML family of languages). Still, portability of descriptions to other frameworks is possible.

- It admits formal descriptions with different semantics. In particular, it enables the combined use of descriptions with first-order and logic programming semantics for matching services and goals, and demonstrates that different semantics are interesting for evaluating different aspects of a service, that all types of descriptions can be integrated into a coherent service location model, and that the usage of certain languages of the WSML family can be an interesting option.

- Descriptions centered in the value offered by services and sought by consumers have been proposed, which is consistent with the principles of SOA design; signature matching, used by most existing works, has been identified as a type of matching relevant for the wiring of services but not for deciding on their relevance for achieving certain effects or on their usability given current conditions.

- Alternative matchmaking mechanisms (filters), with different efficiency and accuracy properties, can be applied in a stand-alone or combined fashion in order to find relevant services for a goal; what filters are to be applied can be freely chosen by the user, who has therefore control over these aspects so that they can be adapted to the particular requirements of his usage scenario.

- Novel matchmaking mechanisms (filters), applied in two phases, are presented, namely: a capability filter which relies on a set-based modelling of offered and sought capabilities with associated user intentions and which enables an effective and value-driven matchmaking, an input-availability filter which operates over actual consumer knowledge to evaluate information preconditions of services, and an input-dependent effects filter which evaluates what effects are potentially offered by a service for available input bindings. These formal filters are complemented by syntactic filters based on taxonomies of categories and on textual descriptions.

- The reasoning support required over formal descriptions for the application of formal filters is provided by existing reasoning infrastructure.

- The alternative types of descriptions proposed and their associated filters keep a balance between simplicity and coverage of application needs for different application types.

- Support to users for describing their services and goals is included in the instantiation proposed. In particular, pre-defined descriptions are associated to service categories so that, starting from

the types of descriptions a user is comfortable with and which he can therefore provide, we can support users for providing other types of descriptions of higher complexity.

- A prototype implementation, as well as an evaluation of the model based on this implementation, has been presented. This prototype demonstrates the feasibility of enhancing current service location practices in a flexible way, of applying formal filters to different types of descriptions with a reasonable efficiency, and of offering support to users for the description of their services and goals.

In general, we have contributed an analysis of the service location problem based on an explicit conceptual model, and a proposal for its resolution which addresses a number of key aspects we believe of great importance for achieving a practical solution to this open challenge in service-oriented computing.

9.2 Future work

Paths for further work have been identified, and they are summarized in the following.

Evolution of prototype implementation. Future work will focus on the evolution of the prototype implementation, with especial emphasis on some of the limitations outlined in the previous Chapter, such as: using reasoners or wrappers which accept WSML syntax without manual translations, dynamically loading domain ontologies, automatically pre-processing goals to eliminate instances not in domain ontologies and sensitive information (if required), improving the user interface of our client, or including strategies for the delayed classification of TBoxes of the DL reasoner of the registry. In general, work will be required in the future for evolving our current implementation from a prototype to a professional service location platform.

Exploring the incorporation of new filters. Adding new filters to our model will be one of the key works we will accomplish in the future. These filter can include: a better filter over textual (syntactic) descriptions such as those incorporated by OWLS-MX, filters over the input-output signature of services, descriptions and filters which enable local-closed world reasoning, or filter over properties not related to the value offered by a service but over other aspects such as QoS or cost. In this sense, we find of particular interest the design of a common API any filter must adhere to so that filters are pluggable and guarantee certain common behaviour.

Ranking, explanations, negotiations and user preferences and constraints. We find of great interest the incorporation of a more fine-grained ranking of location results, the incorporation

of explanations which can better guide users, and the description of users preferences and constraints [Arroyo et al., 2004] which can serve for the identification of negotiation spaces and also contribute to the custom ranking of results (possibly by also weighting the filters to be applied).

Registry architectures. Alternative registry architectures can be investigated. Of special interest can be the specialization of registries in certain domains and the description of this aspect of registries, as proposed by [Verma et al., 2005], and the usage of matching services external to the registry as proposed by [Colgrave et al., 2004].

Automatic selection and automatic wiring. The automatic selection of services, including the definition of contracting or negotiation interfaces, is an interesting path of future work, as well as the automatic wiring and usage of services.

Composition. The integration of composition approaches and our service location model is of considerable interest. In particular, we envision the usage of new filters which can serve to select services also based on criteria appropriate for composition problems.

Reputation. Given that the explicit description of the value of services and of goals is central to our approach, the incorporation of reputation mechanisms to our platform is of interest.

Test collections. The creation of test collections which include real services, and which describe the value of services and not only their input-output signature, will receive our attention in the future. This will of course depend on the availability of complete and truly shared domain ontologies. In this sense, we already actively work on the creation of ontologies in the field of finance, especially on benefitting from the advances in the definition of shared models achieved by the XBRL community (see [Lara et al., 2006a; Lara et al., 2007a] and http://wiki.xbrlontology.com/).

9.3 Concluding remarks

The increasing adoption of the SOA paradigm will require advances in the degree of automation and effectiveness of certain tasks. In particular, we believe the enhanced location of services based on the value they provide and possibly on other aspects (service discovery) will be one of the first tasks for which adopters will require new solutions sooner, as it is, after service creation, the first step for the usage of services in a service-oriented environment.

In this setting, the usage and exploitation of formal descriptions can constitute a more than interesting path for improvement on service discovery. However, not only heavy-weight descriptions and matchmaking mechanisms will be required, but a flexible combination of light-weight (possibly

syntactic), mid-weight and heavy-weight mechanisms will be the choice so that users can cover their own service location requirements in the most appropriate way. Furthermore, a substantial improvement of the usability of new location mechanisms, especially of the descriptions required, can mark an inflexion point for the practical adoption of any solution in this field; users have to be abstracted from underlying complexity, which can be achieved using appropriate tools and support.

Given that the maturity of SOA adoption varies, that shared vocabularies are (or start to be) only available in very few domains, and that the construction of custom domain models for internal use will only be achievable by entities with important resources, we expect the adoption of semantic descriptions of services and of service location mechanisms which rely on such descriptions to be slowly incorporated by early adopters which have already reached a suficient level of maturity in SOA design, and which have considerable resources or focus their business in particular sub-domains. The outreach of this kind of solutions to end users can only be expected in particular domains and based on simple, lightweight solutions which will progressively gain in complexity and capabilities as more domain models are available and tools which guarantee the usability of these complex models are ready for commercial use.

Bibliography

[COR, 1998] (1998). The Common Object Request Broker: Architecture and specification. Technical report, The Object Management Group.

[RAC, 2006] (2006). RacerPro user's guide. version 1.9. Technical report, RACER Systems GmbH & Co. KG.

[WSP, 2006] (2006). Web Services Policy Framework (WS-Policy). Technical report, OASIS.

[Akkiraju et al., 2005] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A., and Verma, K. (2005). Web service semantics - WSDL-S. W3C submission.

[Akkiraju et al., 2003] Akkiraju, R., Goodwin, R., Doshi, P., and Roeder, S. (2003). A method for semantically enhancing the service discovery capabilities of UDDI. In Kambhampati, S. and Knoblock, C. A., editors, *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, pages 87–92.

[Alexiev et al., 2005] Alexiev, V., Breu, M., de Bruijn, J., Fensel, D., Lara, R., and Lausen, H. (2005). *Information Integration With Ontologies. Experiences from an Industrial Showcase*. John Wiley and Sons.

[Alonso et al., 2003] Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2003). *Web Services*. Springer-Verlag.

[America, 1991] America, P. (1991). *Designing an object-oriented programming language with behavioural subtyping*, volume 489 of *Lecture Notes in Computer Science*, pages 60–90. Springer-Verlag.

[Arroyo et al., 2004] Arroyo, S., Bussler, C., Kopecky, J., Lara, R., Polleres, A., and Zaremba, M. (2004). Web service capabilities and constraints in wsmo. In *W3C Workshop on Constraints and Capabilities for Web Services*.

[Baader et al., 2005] Baader, F., Lutz, C., Milicic, M., Sattler, U., and Wolter, F. (2005). A description logic based approach to reasoning about web services. In *Proceedings of the WWW 2005 Workshop on Web Service Semantics (WSS2005)*.

[Baader and Nutt, 2003] Baader, F. and Nutt, W. (2003). Basic description logics. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43–95. Cambridge University Press.

[Baida et al., 2004] Baida, Z., Gordijn, J., Omelayenko, B., and Akkermans, H. (2004). A Shared Service Terminology for Online Service Provisioning. In *ICEC04*, Delft, The Netherlands.

[Battle et al., 2005a] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., and Tabet, S. (2005a). Semantic Web Services Framework (SWSF) overview. W3C submission, W3C.

[Battle et al., 2005b] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., and Tabet, S. (2005b). Semantic web services language (SWSL). W3C submission.

[Battle et al., 2005c] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., and Tabet, S. (2005c). Semantic Web Services Ontology (SWSO). Technical report, SWSI.

[Battle et al., 2005d] Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., and Tabet, S. (2005d). Semantic web services ontology (SWSO). W3C submission.

[Bechhofer, 2003] Bechhofer, S. (2003). The DIG Description Logic Interface: DIG/1.1. URL http://dl-web.man.ac.uk/dig/2003/02/interface.pdf.

[Bechhofer et al., 2005] Bechhofer, S., Horrocks, I., and Turi, D. (2005). The OWL instance store: System description. In *Proceedings CADE-20*, Lecture Notes in Computer Science. Springer-Verlag.

[Bechhofer et al., 2004] Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2004). OWL Web Ontology Language Reference. W3c recommendation, W3C.

[Bellwood et al., 2002] Bellwood, T., Clement, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y., Januszewski, K., Lee, S., McKee, B., Munter, J., and von Riegen, C. (2002). UDDI version 3.0. http://uddi.org/pubs/uddi-v3.00-published-20020719.htm.

[Benatallah et al., 2005] Benatallah, B., Hacid, M.-S., Leger, A., Rey, C., and Toumani, F. (2005). On automating web services discovery. *The VLDB Journal*, 14(1):84–96.

[Benatallah et al., 2003] Benatallah, B., Hacid, M.-S., Rey, C., and Toumani, F. (2003). Request rewriting-based Web service discovery. In *The Semantic Web - ISWC 2003*, pages 242–257.

[Berners-Lee et al., 2001] Berners-Lee, T., Handler, J., and Lassila, O. (2001). The semantic web. *Scientific American*.

[Bonner and Kifer, 1998] Bonner, A. and Kifer, M. (1998). A logic for programming database transactions. In Chomicki, J. and Saake, G., editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers.

[Bonner and Kifer, 1995] Bonner, A. J. and Kifer, M. (1995). Transaction Logic Programming (or, A Logic of Procedural and Declarative Knowledge. Technical report, University of Toronto.

[Brad J. Cox, 1991] Brad J. Cox, A. J. N. (1991). *Object-Oriented Programming: An Evolutionary Approach*. Addison-Wesley, 2nd edition.

[Brickley and Guha, 2004a] Brickley, D. and Guha, R. (2004a). RDF Vocabulary Description Language 1.0: RDF Schema. URL http://www.w3.org/TR/rdf-schema/. Series Editor: Brian McBride.

[Brickley and Guha, 2004b] Brickley, D. and Guha, R. V. (2004b). RDF Schema. Technical report, W3C Recommendation.

[Brittenham et al., 2001] Brittenham, P., Curbera, F., Ehnebuske, D., and Graham, S. (2001). Understanding WSDL in a UDDI registry, part 1: How to publish and find WSDL service descriptions. Technical report, IBM.

[Brodie et al., 2005] Brodie, M. L., Bussler, C., de Bruijn, J., Fahringer, T., Fensel, D., Hepp, M., Lausen, H., Roman, D., Strang, T., Werthner, H., and Zaremba, M. (2005). Semantically enabled service-oriented architectures: A manifesto and a paradigm shift in computer science. Technical report, DERI.

[Calvanese et al., 1998] Calvanese, D., Giacomo, G. D., and Lenzerini, M. (1998). On the decidability of query containment under constraints. In *ACM Symposium on Principles of Database Systems*, pages 149–158.

[Castells et al., 2004] Castells, P., Foncillas, B., Lara, R., Rico, M., and Alonso, J. L. (2004). Semantic web technologies for economic and financial information management. In *European Semantic Web Symposium (ESWS 2004)*.

[Cearley et al., 2005] Cearley, D. W., Fenn, J., and Plummer, D. C. (2005). Gartner's positions on the five hottest IT topics and trends in 2005. Technical report, Gartner Research.

[Chappell, 2004] Chappell, D. (2004). *Enterprise Service Bus: Theory in Practice*. O'Reilly Media.

[Chen et al., 1993] Chen, W., Kifer, M., and Warren, D. S. (1993). HILOG: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230.

[Chen and Warren, 1996] Chen, W. and Warren, D. S. (1996). Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74.

[Chen and Cheng, 2000] Chen, Y. and Cheng, B. (2000). *A Semantic Foundation for Specification Matching*. Cambridge University Press.

[Christensen et al., 2001] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. Note, W3C.

[Christian Halaschek-Wiener and Sirin, 2006] Christian Halaschek-Wiener, B. P. and Sirin, E. (2006). Description Logic Reasoning with Syntactic Updates. In *Proceedings of the 5th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE-2006)*.

[Colgrave et al., 2004] Colgrave, J., Akkiraju, R., and Goodwin, R. (2004). External matching in UDDI. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*.

[Colgrave and Rogers, 2004] Colgrave, J. and Rogers, T. (2004). Using WSDL in a UDDI registry, version 2.0.2. Technical note uddi-spec-tc-tn-wsdl-v2, OASIS.

[Colucci et al., 2003] Colucci, S., Noia, T. D., Sciascio, E. D., Donini, F., and Mongiello, M. (2003). Concept abduction and contraction in description logics. In *Proceedings of the 16th International Workshop on Description Logics (DL'98)*.

[Colucci et al., 2005] Colucci, S., Noia, T. D., Sciascio, E. D., Donini, F. M., and Mongiello, M. (2005). Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in a E-marketplace. *Electronic Commerce Research and Applications*, 4(4):345–361.

[Connolly et al., 2001] Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A. (2001). DAML+OIL (March 2001) Reference Description. Note, W3C.

[Constantinescu and Faltings, 2003] Constantinescu, I. and Faltings, B. (2003). Efficient matchmaking and directory services. In *Proceedings of the IEEE International Conference on Web Intelligence (WI03)*.

[Corella and Castells, 2006a] Corella, M. A. and Castells, P. (2006a). A heuristic approach to semantic web services classification. In *KES-2006*.

[Corella and Castells, 2006b] Corella, M. A. and Castells, P. (2006b). Semi-automatic semantic-based web service classification. In *semantics4ws'06 workshop at BPM 2006*.

[Dahl, 1987] Dahl, O.-J. (1987). *Research directions in object-oriented programming*, chapter Object-Oriented Specifications, pages 561–576. MIT Press.

[Dantsin et al., 2001] Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425.

[de Bruijn, 2007] de Bruijn, J. (2007). *Semantic Web Services: Theory, Tools and Applications*, chapter Logigs for the Semantic Web. IDEA Publishing. In Production.

[de Bruijn et al., 2005a] de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., Kifer, M., Koenig-Ries, B., Kopecky, J., Lara, R., Lausen, H., Oren, E., Polleres, A., Roman, D., Scicluna, J., and Stollberg, M. (2005a). Web Service Modeling Ontology (WSMO). W3C submission.

[de Bruijn et al., 2005b] de Bruijn, J., Feier, C., Keller, U., Lara, R., Polleres, A., and Predoiu, L. (2005b). D16.2v0.2 WSML Reasoner Survey. Working draft D16.2, WSML working group.

[de Bruijn et al., 2005c] de Bruijn, J., Fensel, D., Keller, U., Kifer, M., Lausen, H., Krummenacher, R., Polleres, A., and Predoiu, L. (2005c). Web Service Modeling Language (WSML). W3C submission.

[de Bruijn et al., 2005d] de Bruijn, J., Fensel, D., Keller, U., and Lara, R. (2005d). Using the Web Service Modeling Ontology to enable Semantic eBusiness. *Communications of the ACM (CACM)*, 48(12):43–47.

[de Bruijn et al., 2005e] de Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L., Kifer, M., and Fensel, D. (2005e). The Web Service Modeling Language WSML. WSML Final Draft D16.1v0.2, DERI.

[de Bruijn et al., 2004] de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2004). OWL-. Deliverable d20.1v0.2, WSML.

[de Bruijn et al., 2005f] de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2005f). OWL DL vs. OWL Flight: Conceptual Modeling and Reasoning for the semantic Web. In *Proceedings of the World Wide Web Conference 2005*. Springer-Verlag.

[de Bruijn et al., 2005g] de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2005g). OWL DL vs. OWL Flight: Conceptual modelling and reasoning for the semantic web. In *Proceedings of the World Wide Web Conference 2005 (WWW2005)*.

[Dhara and Leavens, 1996] Dhara, K. and Leavens, G. (1996). Forcing behavioral subtyping through specification inheritance. In *Proceedings of the 18th International Conference on Software Engineering (ICSE'18)*.

[Ding et al., 2004] Ding, Y., Fensel, D., Lara, R., Lausen, H., Stollberg, M., and Han, S.-K., editors (2004). *Application of Semantic Web Technologies to Web Communities 2004. Proc. 1st Intl. Workshop SWWC 2004 at ECAI 2004*. CEUR Workshop Proceedings.

[Ding et al., 2002] Ding, Y., Korotkiy, M., Omelayenko, B., Kartseva, V., Zykov, V., Klein, M., Schulten, E., and Fensel, D. (2002). Goldenbullet: Automated classification of product data in e-commerce. In *Proceedings of Business Information Systems Conference (BIS 2002)*, Poznan, Poland.

[Dong et al., 2004] Dong, X., Halevy, A. Y., Madhavan, J., Nemes, E., and Zhang, J. (2004). Similarity search for web services. In *VLDB 2004*.

[Donini, 2003] Donini, F. M. (2003). Complexity of Reasoning. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 96–136. Cambridge University Press.

[Donini et al., 1998] Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. (1998). An epistemic operator for description logics. *Artificial Intelligence*, 100(1-2):225–274.

[Emerson, 1990] Emerson, E. A. (1990). *Handbook of Theoretical Computer Science*, chapter Temporal and modal logic. MIT Press.

[Erl, 2005] Erl, T. (2005). *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. Prentice Hall PTR.

[Etzioni et al., 1994] Etzioni, O., Golden, K., and Weld, D. (1994). Tractable closed world reasoning with updates. In *Proceedings of the 4th International Conference on Knowledge Representation and Reasoning (KR1994)*, pages 178–189.

[Farrell and (editors), 2007] Farrell, J. and (editors), H. L. (2007). Semantic Annotations for WSDL and XML Schema. Candidate recommendation, W3C.

[Fellbaum, 1998] Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. MIT Press.

[Fensel and Bussler, 2002] Fensel, D. and Bussler, C. (2002). The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2).

[Fensel and Musen, 2001] Fensel, D. and Musen, M. (2001). The semantic web: A brain for humankind. *IEEE Intelligent Systems*, pages 24–25.

[Fernandez et al., 2006] Fernandez, A., Vasirani, M., Caceres, C., and Ossowski, S. (2006). Role-based service description and discovery. In *Service-Oriented Computing and Agent-Based Engineering (SOCABE'06)*.

[Fitting, 1996] Fitting, M. (1996). *First order logic and automated theorem proving*. Springer Verlag, 2nd edition.

[Gardiner et al., 2006] Gardiner, T., Tsarkov, D., and Horrocks, I. (2006). Framework for an automated comparison of description logic reasoners. In *Proc. of the 2006 International Semantic Web Conference (ISWC 2006)*, volume 4273 of *Lecture Notes in Computer Science*, pages 654–667. Springer.

[Gelder et al., 1991] Gelder, A. V., Ross, K., and Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650.

[Gelfond and Lifschitz, 1988] Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In *Fifth international conference on logic programming*, pages 1070–1080.

[Gomez et al., 2004] Gomez, J. M., Rico, M., Bejar, R. M., and Bussler, C. (2004). GODO: Goal Driven Orchestration for semantic web services. In *WSMO Implementation Workshop 2004 (WIW 2004)*.

[Gonzalez-Castillo et al., 2001] Gonzalez-Castillo, J., Trastour, D., and Bartolini, C. (2001). Description logics for matchmaking of services. In *KI-2001 Workshop on Applications of Description Logics*.

[Graedel et al., 1997] Graedel, E., Otto, M., and Rosen, E. (1997). Two-variable logic with counting is decidable. In *2th Annual IEEE Symposium on Logic in Computer Science*.

[Grimm et al., 2004] Grimm, S., Motik, B., and Preist, C. (2004). Variance in e-business service discovery. *Semantic Web Services: Preparing to Meet the World of Business Applications, Workshop at ISWC 2004*.

[Grimm et al., 2006] Grimm, S., Motik, B., and Preist, C. (2006). Matching semantic service descriptions with local closed-world reasoning. In *ESWC*, pages 575–589.

[Grosof et al., 2003] Grosof, B., Horrocks, I., Volz, R., and Decker, S. (2003). Description logic programs: Combining logic programs with description logic. In *Proc. Intl. Conf. on the World Wide Web (WWW-2003)*, Budapest, Hungary.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specification. *Knowledge Acquisition*, 5(2):199–220.

[Haarslev and Möller, 2001] Haarslev, V. and Möller, R. (2001). RACER System Description. volume 2083.

[Haas and Brown, 2004] Haas, H. and Brown, A. (2004). Web Services Glosary. Note, W3C. http://www.w3.org/TR/ws-gloss/.

[Haller, 2004] Haller, A. (2004). D13.6 WSMX use cases. Technical report, WSMX working group.

[Harel, 1979] Harel, D. (1979). First-order dynamic logic. In Springer-Verlag, editor, *Lecture Notes in Computer Science*, volume 68.

[Harel et al., 1982] Harel, D., Kozen, D., and Parikh, R. (1982). Process logic: Expressiveness, decidability, completeness. *Journal of Computer and System Sciences*, 25(2):144–170.

[Harel et al., 2000] Harel, D., Kozen, D., and Tiuryn, J. (2000). *Dynamic Logic*. MIT Press.

[He et al., 2004] He, H., Haas, H., and Orchard, D. (2004). Web services architecture usage scenarios. W3C working group note, W3C.

[Heffner et al., 2006] Heffner, R., Zetie, C., and Hogan, L. (2006). Survey data says: The time for SOA is now. Technical report, Forrester Research.

[Hoare, 1969] Hoare, C. (1969). An axiomatic basis for computer programming. *Communications of the ACM*, 12(10).

[Horrocks, 1997] Horrocks, I. (1997). *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, The University of Manchester. URL http://www.cs.man.ac.uk/ horrocks/Publications/download/1997/phd-2sss.ps.gz.

[Horrocks, 1998] Horrocks, I. (1998). Using an Expressive Description Logic: FaCT or Fiction? In *KR-98*, pages 636–647.

[Horrocks, 2003] Horrocks, I. (2003). Implementation and Optimisation Techniques. In Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 306–346. Cambridge University Press.

[Horrocks et al., 2004] Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A semantic web rule language combining OWL and RuleML. Available at http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[Horrocks and Patel-Schneider, 1998a] Horrocks, I. and Patel-Schneider, P. F. (1998a). Comparing subsumption optimizations. In *Proc. of the Int. Description Logics Workshop (DL98)*.

[Horrocks and Patel-Schneider, 1998b] Horrocks, I. and Patel-Schneider, P. F. (1998b). DL systems comparison. In *Proc. of the Int. Description Logics Workshop (DL98)*, pages 55–57.

[Horrocks et al., 2003] Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26.

[Horrocks and Sattler, 2001] Horrocks, I. and Sattler, U. (2001). Ontology reasoning in the SHOQ(D) description logic. In *IJCAI 2001*, pages 199–204.

[Horrocks and Sattler, 2002] Horrocks, I. and Sattler, U. (2002). Optimised Reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, pages 277–281.

[Hull et al., 2006] Hull, D., Zolin, E., Bovykin, A., Horrocks, I., Sattler, U., and Stevens, R. (2006). Deciding semantic matching of stateless services. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06) and Eighteenth Innovative Applications of Artificial Intelligence (IAAI-06) Conference*, pages 1319–1324.

[Hustadt et al., 2004] Hustadt, U., Motik, B., and Sattler, U. (2004). Reducing SHIQ Description Logic to Disjunctive Datalog Programs. In *Proceedings of the 9th International Conference on Knowledge Representation and Reasoning (KR2004)*, pages 152–162.

[Jaeger et al., 2005] Jaeger, M. C., Rojec-Goldmann, G., Liebetruth, C., Muehl, G., and Geihs, K. (2005). Ranked matching for service descriptions using OWL-S. In *KiVS 2005*, pages 91–102.

[Jeng and Cheng, 1992] Jeng, J. and Cheng, B. (1992). Using automated reasoning techniques to determine software reuse. *International Journal of Software Engineering and Knowledge Engineering*, 2(4).

[Jeng and Cheng, 1993] Jeng, J. and Cheng, B. (1993). *Using Formal Methods to Construct a Software Component Library*, volume 717 of *Lecture Notes in Computer Science*, pages 397–417. Springer Verlag.

[Jeng and Cheng, 1995] Jeng, J. and Cheng, B. (1995). Specification matching for software reuse: A foundation. In *SSR'95. ACM SIGSOFT*. ACM Press.

[Kaufer and Klusch, 2006] Kaufer, F. and Klusch, M. (2006). Wsmo-mx: A logic programming based hybrid service matchmaker. In *ECOWS '06: Proceedings of the European Conference on Web Services*, pages 161–170.

[Kawamura et al., 2004] Kawamura, T., Blasio, J. A. D., Hasegawa, T., Paolucci, M., and Sycara, K. P. (2004). Public Deployment of Semantic Service Matchmaker with UDDI Business Registry. In *International Semantic Web Conference 2004 (ISWC 2004)*, pages 752–766.

[Keller et al., 2004a] Keller, U., Lara, R., and (*editors*), A. P. (2004a). WSMO web service discovery. Technical report, DERI.

[Keller et al., 2006a] Keller, U., Lara, R., Lausen, H., and Fensel, D. (2006a). *Semantic Web: Theory, Tools and Applications*, chapter Semantic Web Service Discovery in the WSMO Framework. IDEA Publishing Group.

[Keller et al., 2005] Keller, U., Lara, R., Lausen, H., Polleres, A., and Fensel, D. (2005). Automatic location of services. In *ESWC 2005*, Heraklion, Greece.

[Keller and Lausen, 2006] Keller, U. and Lausen, H. (2006). Functional description of web services. Deliverable, WSML working group.

[Keller et al., 2006b] Keller, U., Lausen, H., and Stollberg, M. (2006b). On the semantics of functional descriptions of web services. In *3rd European Semantic Web Conference (ESWC2006)*.

[Keller et al., 2004b] Keller, U., Stollberg, M., and Fensel, D. (2004b). WOOGLE meets Semantic Web Fred. In *Proceedings of the Workshop on WSMO Implementations (WIW 2004)*, CEUR-WS.org/Vol-113/.

[Kifer, 2005] Kifer, M. (2005). Service discovery with SWSL-rules. In *W3C Workshop on Frameworks for Semantics in Web Services*.

[Kifer et al., 2004] Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., and Fensel, D. (2004). A logical framework for web service discovery. In *"Semantic Web Services: Preparing to Meet the World of Business Applications" worshop at ISWC 2004*.

[Kifer et al., 1995] Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42:741–843.

[Klein and König-Ries, 2004] Klein, M. and König-Ries, B. (2004). Coupled signature and specification matching for automatic service binding. In *Proc. of the European Conference on Web Services (ECOWS 2004)*, Erfurt, Germany.

[Klusch et al., 2006] Klusch, M., Fries, B., and Sycara, K. (2006). Automated semantic web service discovery with OWLS-MX. In *AAMAS 2006*.

[Kopecky, 2005] Kopecky, J. (2005). WSMO Use Case: Amazon e-commerce service. Technical report, WSMO working group.

[Kopecky et al., 2005] Kopecky, J., Moran, M., Roman, D., and Mocan, A. (2005). D24.2v0.1. WSMO grounding. Working draft 24.2, WSMO working group.

[Kuester et al., 2007] Kuester, U., Koenig-Ries, B., Stern, M., and Klein, M. (2007). Diane: an integrated approach to automated service discovery, matchmaking and composition. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1033–1042, New York, NY, USA. ACM Press.

[Lara, 2006] Lara, R. (2006). Two-phased web service discovery. In *AI-driven Service Oriented Computing workshop at AAAI 2006*.

[Lara et al., 2004a] Lara, R., Binder, W., Constantinescu, I., Fensel, D., Keller, U., Pan, J., Pistore, M., Polleres, A., Toma, I., Traverso, P., and Zaremba, M. (2004a). Semantics for Web Service Discovery and Composition. Technical report, Knowledge Web.

[Lara et al., 2004b] Lara, R., Binder, W., Pistore, M., Traverso, P., Constantinescu, I., Pan, J., Nixon, L., Haller, A., and Vitvar, T. (2004b). Semantic requirements for web service description. Project Deliverable 2.4.1, Knowledge Web.

[Lara et al., 2006a] Lara, R., Cantador, I., and Castells, P. (2006a). XBRL taxonomies and OWL ontologies for investment funds. In *1st International Workshop on Ontologizing Industry Standards (OIS 2006)*.

[Lara et al., 2007a] Lara, R., Cantador, I., and Castells, P. (2007a). Semantic web technologies for the financial domain. In Cardoso, J., editor, *Real-world Applications of Semantic Web Technology and Ontologies*. IDEA Group Publishing.

[Lara et al., 2007b] Lara, R., Corella, M., and Castells, P. (2007b). A flexible model for the location of services. *International Journal of Electronic Commerce, Special Issue on Semantic Matchmaking and Resource Retrieval*. Accepted for publication.

[Lara et al., 2006b] Lara, R., Corella, M. A., and Castells, P. (2006b). A flexible model for web service discovery. In *SMR@VLDB 2006*.

[Lara et al., 2003] Lara, R., Lausen, H., Arroyo, S., de Bruijn, J., and Fensel, D. (2003). Semantic web services: description requirements and current technologies. In *Semantic Web Services for Enterprise Application Integration and e-Commerce (SWSEE03) workshop at ICEC 2003*.

[Lara and Olmedilla, 2005] Lara, R. and Olmedilla, D. (2005). Discovery and contracting of semantic web services. In *W3C Workshop on Frameworks for Semantics in Web Services*.

[Lara et al., 2005] Lara, R., Polleres, A., Lausen, H., Roman, D., de Bruijn, J., and Fensel, D. (2005). A conceptual comparison between WSMO and OWL-S. Final draft, WSMO Working Group.

[Lara et al., 2004c] Lara, R., Roman, D., Polleres, A., and Fensel, D. (2004c). A conceptual comparison of WSMO and OWL-S. In *Proceedings of the European Conference on Web Services (ECOWS 2004)*, Erfurt, Germany.

[Lausen et al., 2005] Lausen, H., Ding, Y., Stollberg, M., Fensel, D., Lara, R., and Han, S.-K. (2005). Semantic web portals: state-of-the-art survey. *Journal of Knowledge Management*, 9(5):40–49.

[Li and Horrocks, 2003] Li, L. and Horrocks, I. (2003). A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web*, Budapest, Hungary.

[Liskov and Wing, 1994] Liskov, B. and Wing, J. (1994). A behavioral notion of subtyping. *ACM Transactions on Programming Languages*, 16(10).

[Lloyd, 1987] Lloyd, J. W. (1987). *Foundations of logic programming (second, extended edition).* Springer series in symbolic computation. Springer-Verlag.

[Lopez et al., 2005] Lopez, A., Lopez, O., Pezuela, C., and Scicluna, J. (2005). D3.6 WSMO use case STREAM flows! system. Technical report, WSMO working group.

[Lukasiewicz and Schellhase, 2006] Lukasiewicz, T. and Schellhase, J. (2006). Variable-strength conditional preferences for ranking objects in ontologies. In *ESWC 2006*, pages 288–302.

[Luo et al., 2005] Luo, J., Montrose, B., and Kang, M. (2005). An Approach for Semantic Query Processing with UDDI. In *Agents, Web Services and Ontologies Merging.*

[Lynch et al., 2006] Lynch, D., Keeney, J., Lewis, D., and O'Sullivan, D. (2006). A proactive approach to semantically oriented service discovery. In *2nd Workshop on Innovations in Web Infrastructure at WWW 2006.*

[MacKenzie et al., 2006] MacKenzie, C. M., Laskey, K., McCabe, Brown, P. F., and Metz, R. (2006). Reference model for service oriented architecture 1.0. Committee Specification 1, OASIS.

[Martin et al., 2004] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., and Sycara, K. (2004). OWL-S: Semantic markup for web services. W3C submission.

[McDermott, 2004] McDermott, D. (2004). DRS: A set of conventions for representing logical languages in RDF. Available at http://www.daml.org/services/owl-s/1.1B/DRSguide.pdf.

[McGuinness and van Harmelen, 2004] McGuinness, D. L. and van Harmelen, F. (2004). OWL Web Ontology Language Overview. Recommendation, W3C.

[Minsky, 1981] Minsky, M. (1981). A Framework for Representing Knoledge. In Haugeland, J., editor, *Mind Design.* The MIT Press.

[Mokhtar et al., 2006] Mokhtar, S. B., Kaul, A., Georgantas, N., and Issarny, V. (2006). Efficient semantic service discovery in pervasive computing environments. In *Proceedings of the ACM/IFIP/USENIX 7th International Middleware Conference (Middleware'06).*

[Monson-Haefel and Chappell, 2000] Monson-Haefel, R. and Chappell, D. (2000). *Java Message Service.* O'Reilly Java Series.

[Nadalin et al., 2004] Nadalin, A., Kaler, C., Hallam-Baker, P., and (editors), R. M. (2004). Web Services Security: SOAP Message Security 1.0 (WS-Security 2004). Technical report, OASIS.

[Nardi et al., 2003] Nardi, D., Brachmanand, R. J., Baaderand, F., Nutt, W., Donini, F. M., Sattler, U., Calvanese, D., Moelitor, R., Giacomo, G. D., Ksters, R., Wolter, F., McGuinness, D. L., Patel-Schneider, P. F., Moeller, R., Haarslev, V., Horrocks, I., Borgida, A., Welty, C., Rector, A., Franconi, E., Lenzerini, M., and Rosati, R. (2003). *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge.

[Newcomer and Lomow, 2004] Newcomer, E. and Lomow, G. (2004). *Understanding SOA with Web Services*. Addison-Wesley Professional.

[Noia et al., 2004] Noia, T. D., Sciascio, E. D., Donini, F. M., and Mogiello, M. (2004). A system for principled matchmaking in an electronic marketplace. *International Journal of Electronic Commerce*, 8(4):9–37.

[Noia et al., 2003] Noia, T. D., Sciascio, E. D., Donini, F. M., and Mongiello, M. (2003). A system for principled matchmaking in an electronic marketplace. In *Proc. Intl. Conf. on the World Wide Web 2003 (WWW2003)*.

[Noy and McGuinness, 2001] Noy, N. F. and McGuinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology. Technical report, Stanford University.

[Olmedilla et al., 2004] Olmedilla, D., lara, R., Polleres, A., and Lausen, H. (2004). Trust negotiation for semantic web services. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004) at ICWS 2004*.

[Ort, 2005] Ort, E. (2005). Service-oriented architecture and web services: Concepts, technologies and tools. Technical report, Sun Developer Network.

[Pan, 2005] Pan, Z. (2005). Benchmarking dl reasoners using realistic ontologies. In *Proc. of the International workshop on OWL: Experience and Directions (OWL-ED2005)*.

[Paolucci et al., 2002] Paolucci, M., Kawamura, T., Payne, T., and Sycara, K. (2002). Semantic matching of web services capabilities. In Horrocks, I. and Handler, J., editors, *1st Int. Semantic Web Conference (ISWC)*, pages 333–347. Springer Verlag.

[Papazoglou et al., 2006] Papazoglou, M. P., Traverso, P., Dustdar, S., and Leymann, F. (2006). Service-oriented computing research roadmap. Technical report.

[Patel-Schneider et al., 2004] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). OWL web ontology language semantics and abstract syntax. Recommendation 10 February 2004, W3C.

[Penix and Alexander, 1997] Penix, J. and Alexander, P. (1997). Towards automated component adaptation. In *Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering*.

[Pokraev et al., 2003] Pokraev, S., Koolwaaij, J., and Wibbels, M. (2003). Extending UDDI with Context-Aware Features Based on Semantic Service Descriptions. In *Proceedings of the 2003 International Conference on Web Services (ICWS'03)*, pages 184–190.

[Preist, 2004] Preist, C. (2004). A conceptual architecture for semantic web services. In *Proceedings of the International Semantic Web Conference 2004 (ISWC 2004)*.

[Prud'hommeaux and Seaborne, 2006] Prud'hommeaux, E. and Seaborne, A. (2006). SPARQL Query Language for RDF. Working draft, W3C.

[Quillian, 1967] Quillian, M. R. (1967). Word Concepts: A Theory and Simulation of Some Basic Capabilities. *Behavioral Science*, 12:410–430.

[Rollings and Wing, 1991] Rollings, E. and Wing, J. (1991). Specifications as search keys for software libraries. In *Proceedings of the Eighth International Conference on Logic Programming*.

[Roman et al., 2005] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Fensel, D., and Bussler, C. (2005). Web Service Modeling Ontology. *Applied Ontology Journal*, 1(1):77–106.

[Rosenfield, 1994] Rosenfield, R. (1994). *Adaptive statistic language model*. PhD thesis, Carnegie Mellon University.

[Schmidt-Schauss and Smolka, 1991] Schmidt-Schauss, M. and Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26.

[Schumann and Ficher, 1997] Schumann, J. and Ficher, B. (1997). Nora/hammr: Making deduction-based software component retrieval practical. In *Proceedings of the 12th IEEE International Automated Software Engineering Conference (ASE97)*.

[Sirin et al., 2006a] Sirin, E., Grau, B. C., and Parsia, B. (2006a). From wine to water: Optimizing description logic reasoning for nominals. In *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KR-2006)*.

[Sirin and Parsia, 2006] Sirin, E. and Parsia, B. (2006). Optimizations for Answering Conjunctive ABox Queries. In *Proceedings of the International Workshop on Description Logic (DL-2006)*.

[Sirin et al., 2006b] Sirin, E., Parsia, B., Cuenca-Grau, B., Kalyanpur, A., and Katz, Y. (2006b). Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*.

[Sivashanmugam et al., 2004] Sivashanmugam, K., Verma, K., and Sheth, A. (2004). Discovery of web services in a federated registry environment. In *Proceedings of IEEE Second International Conference on Web Services*.

[Sleeper, 2001] Sleeper, B. (2001). Defining web services. Technical report, The Stencil Group.

[Smith and Fingar, 2003] Smith, H. and Fingar, P. (2003). *Business Process Management (BPM): The Third Wave*. Meghan-Kiffer Press.

[Spanoudakis et al., 2005] Spanoudakis, G., Zisman, A., and Kozlenkov, A. (2005). A service discovery framework for service centric systems. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 251–259.

[Sprott and Wilkes, 2004] Sprott, D. and Wilkes, L. (2004). Understanding service - oriented arhcitecture. *The Architecture Journal*.

[Srinivasan et al., 2004] Srinivasan, N., Paolucci, M., and Sycara, K. (2004). Adding OWL-S to UDDI, implementation and throughput. In *SWSWPC Workshop at ICWS 2004*.

[Stollber et al., 2004] Stollber, M., Lausen, H., Lara, R., Keller, U., Zaremba, M., Haller, A., Fensel, D., and Kifer, M. (2004). D3.3 WSMO use case virtual travel agency. Technical report, WSMO working group.

[Stollberg et al., 2004] Stollberg, M., Toma, I., Keller, U., Keimel, B., and Zugmann, P. (2004). D3.5 SWF use case - final version 1.2. Technical report, WSMO working group.

[Sycara et al., 2002] Sycara, K., Widoff, S., Klusch, M., and Lu, J. (2002). LARKS: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, pages 173–203.

[Teege, 1994] Teege, G. (1994). Making the difference: A substraction operation for description logics. In *KR'94*.

[Tidwell, 2000] Tidwell, D. (2000). Web services: the web's next revolution. Technical report, IBM developerWorks.

[Toma and Foxvog, 2006] Toma, I. and Foxvog, D. (2006). D28.4 v0.1 non-functional properties in web services. Working draft 28.4, WSMO working group.

[Toma et al., 2005] Toma, I., Sapkota, B., Scicluna, J., Gomez, J. M., Roman, D., and Fensel, D. (2005). A P2P discovery mechanism for web service execution environment. In *Proceedings of the 2nd International WSMO Implementation Workshop (WIW 2005)*.

[Trastour et al., 2002] Trastour, D., Bartolini, C., and Preist, C. (2002). Semantic web support for the business-to-business e-commerce lifecycle. In *World Wide Web Conference (WWW202)*. ACM.

[Tsarkov and Horrocks, 2003] Tsarkov, D. and Horrocks, I. (2003). Reasoner prototype. implementing new reasoner with datatypes support. Technical report, WonderWeb project.

[Ullman, 1988] Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press.

[Valle and Cerizza, 2005] Valle, E. D. and Cerizza, D. (2005). COCOON Glue: a prototype of WSMO Discovery engine for the healthcare field. In *2nd WSMO Implementation Workshop*.

[Valle et al., 2005] Valle, E. D., Cerizza, D., and Celino, I. (2005). The mediators centric approach to automatic web service discovery of glue. In *First International Workshop on Mediation in Semantic Web Services: MEDIATE 2005*.

[Verma et al., 2005] Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., Oundhakar, S., and Miller, J. (2005). METEOR-S WSDI: A scalable infrastructure of registries for semantic publication and discovery of web services. *Journal of Information Technology and Management, Special Issue on Universal Global Integration*, 6(1):17–39.

[Volz, 2004] Volz, R. (2004). *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, Karlsruhe.

[w. Dijkstra, 1972] w. Dijkstra, E. (1972). *Structured programming*, chapter Notes on structured programming. Academic Press.

[W3C, 2003] W3C (2003). SOAP version 1.2 part 0: Primer. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

[Weibel et al., 1998] Weibel, S., Kunze, J., Lagoze, C., and Wolf, M. (1998). Dublin core metadata for resource discovery. RFC 2413, IETF.

[Wielemaker, 2003] Wielemaker, J. (2003). An overview of the SWI-Prolog programming environment. In Mesnard, F. and Serebenik, A., editors, *Proceedings of the 13th International Workshop on Logic Programming Environments*.

[Wikipedia, 2005] Wikipedia (2005). Service-oriented architecture. http://en.wikipedia.org/wiki/Service-oriented_architecture.

[Yang and Kifer, 2003] Yang, G. and Kifer, M. (2003). Reasoning about anonymous resources and meta statements on the semantic web. *Journal of Data Semantics*, 1:69–97.

[Yang et al., 2005] Yang, G., Kifer, M., Zhao, C., and Chowdhary, V. (2005). Flora-2: User's manual (version 0.94). Technical report, SRI International, State University of New York, Microsoft.

[Zaremski and Wing, 1995] Zaremski, A. and Wing, J. (1995). Specification matching of software components. In *3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering*.

[Zaremski and Wing, 1997] Zaremski, A. and Wing, J. (1997). Specification matching of software components. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6:333–369.

[Zein and Kermarrec, 2004] Zein, O. and Kermarrec, Y. (2004). An approach for describing/discovering services and for adapting them to the needs of users in distributed systems. In *Semantic Web Services. Papers from 2004 AAAI Spring Symposium*.

Appendix A

Complete example of discovery based on Transaction Logic

Below is the complete example used in [Kifer et al., 2004], which can be run using $\mathcal{FLORA}$-2.

```
/*
** Title: SERVICE DISCOVERY WITH MEDIATORS
**
** Features:  - WG-mediators
**            - complex goals
**            - rules in effects and goals
**            - service effects can depend on input
*/

//?- debug[#check_undefined(on)]@flora(sys).  // use debug mode


/*
** A taxonomy of cities.
**       europe means European Cities
**       france - French cities, etc.
**       tyrol - Tyrolean cities, etc.
*/
usa::america.
germany::europe.
austria::europe.
france::europe.
tyrol::austria.
nystate::usa.
stonybrook:nystate.
innsbruck:tyrol.
```

```
lienz:tyrol.
vienna:austria.
bonn:germany.
frankfurt:germany.
paris:france.
nancy:france.
// regions - things like europe, germany, tyrol
europe:region.
america:region.
// any subregion of a region is also a region
Reg:region :- Reg1:region and Reg::Reg1.
// A location is any city or town that belongs to any region
Loc:location :- Reg:region and Loc : Reg.

/*
  Services/clients write their descriptions/queries to conform to specific
  ontologies. Most of the intelligence lies in the mediators, which are
  assumed to be written by skilled professionals.
  Client goals are assumed to be written by naive users and thus are the
  simplest.
  Service descriptions are written by knowledge engineers. They can be
  more complex, but shouldn''t require a Ph.D. in knowledge representation.

  Goal ontology:
      Clients' goals use the Goal Ontology, which provides primitives for
      discovering and contracting services.
      The primitives support a range of discovery tasks, from least specific
      to more specific.

      Discovery goals:

          searchTrip(from,to)  - from/to can be cities or regions; if a
                                 region then it means the requested service
                                 must serve every city in that region that
                                 is known to the KB. Assume either both are
                                 cities or both are regions.
          searchCitipass(loc)  - citipass search; loc can be a city or a region

      Contracting goals:
          tripContract(serviceId,fromCity,toCity,date,creditCard)
          citipassContract(serviceId,city,date,creditCard)

      Goals have the form:

          goalId[requestId->someId, query->queryType]

      where
        someId      - the request Id,
        goalId      - goal Id
```

```
       queryType   - a discovery/contracting primitive described above

    Results of a search are stored in the attribute result, e.g.,

        goal1[result ->> {serv1,serv2}]

    The result of a contract execution is stored in the attribute
    confirmation, e.g.,

        goal2[confirmation -> info(service,confNumber,from,to,date)]
       or
        goal2[confirmation -> info(service,confNumber,city,date)]

Service ontologies:
    Services represent their inputs and outputs using ontologies that can
    possibly be different from the ontology used by the users.
    In our examples, services use two different ontologies. The ontology
    mismatch is resolved using the Web service-to-goal mediators
    (wg-mediators). The wg-mediators for the Request ontology and the
    two service ontologies are defined separately below.

    Service Ontology #1
       Inputs have the following form:
            search(requestId,fromLocation,toLocation)
            search(requestId,city)
            contract(requestId,fromLocation,toLocation,date,ccard)
            contract(requestId,city,date,ccard)

         Input is constructed from the client's goal by the mediator and
         passed to the service.

       The output produced by the service has the form

         for searches:
            itinerary(reqNumber)[from->fromCity, to->toCity]
            passinfo(reqNumber)[city->City]

                Note: for searches, services assume that the input provides
                      a specific pair of cities. Services know nothing about
                      searches by regions, so the descriptions of their
                      capabilities are relatively simple. Region-based
                      queries are constructed by mediators. This is what we
                      mean when we say that most of the intelligence is
                      in the mediators.

         for contracting:
            ticket(reqNumber)[confirmation->confNumber,
                              from->fromCity, to->toCity, date->someDate]
            pass(reqNumber)[confirmation->confNumber,
```

```
                              city->City, date->someDate]

      Service Ontology #2
          Provides basically the same information, but uses different
          representation (to illustrate the idea of different mediators).
          Services that use this ontology only sell citipasses.

      Inputs have the following form:
              discover(requestId,city)
              pay(requestId,city,date,ccard)

          The output looks like this:
              reqNumber[location->city)]  for searches
              reqNumber[confirmation->(number,city,date)] for purchases
*******************************************************************************/

/************************ Available services ******************************
  Assume precondition/effects to be mandatory and have uniform representation
  for all services. In general, we could use mediators that the discovery and
  contracting query could invoke to reconcile the different representations.
*******************************************************************************/

// This service uses Ontology #1 and mediator1 to map client ontology
// to Ontology #1
serv1[
   // Input must be a request for ticket from somewhere in Germany to somewhere
   // in Austria  OR  a request for a city pass for a city in Tyrol
   capability->
        _#[
       precondition(Input) ->
                  ${
             (Input = contract(_, From:germany, To:austria, Date, Card)
              or Input = contract(_, City:tyrol, Date, _))
             and validDate(Date) and validCard(Card)
          },
       effects(Input) ->
          ${
              // Note: repeating some preconditions, like From:germany,
              //       because precondition(Input) is not checked
              //       during discovery, but From:germany, To:austria
              //       can be relevant to discovery. However, not all
              //       of the precondition is copied here -- only what
              // is relevant for discovery.
              (itinerary(Req)[from->From,to->To] :-
                Input = search(Req, From:germany, To:austria))
              and
              (passinfo(Req)[city->City] :-
                Input = search(Req,City:tyrol))
              and
```

```
              // Note: precondition is checked at invocation, so
              // no need to repeat those tests here.
              (ticket(Req)[confirmation->Num,
                           from->From, to->To, date->Date] :-
                            Input = contract(Req,From,To,Date,_CCard),
                generateConfNumber(Num))
              and
              (pass(Req)[confirmation->Num, city->City, date->Date] :-
                            Input = contract(Req,City,Date,_CCard),
                generateConfNumber(Num))
            }
     ],

    usedMediators ->> med1
].

// Another Ontology #1 service
serv2[
   capability->
        _#[
       precondition(Input) -> ${
              // Input must be a request for a ticket from a
                      // city in France or Germany to a city in Austria
              Input = contract(_, From:(france or germany),
                                  To:austria, Date, Card)
              and validDate(Date) and validCard(Card)
               },
       effects(Input)-> ${
              (itinerary(Req)[from->From, to->To] :-
              Input = search(Req,
                         From:(france or germany), To:austria))
              and
              (ticket(Req)[confirmation->Num,
                           from->From, to->To, date->Date] :-
                Input = contract(Req,From,To,Date,_CCard) and
                generateConfNumber(Num))
             }
     ],

    usedMediators ->> med1

].

// An Ontology #2 service
serv3[
   capability->
        _#[
       precondition(Input) ->
               ${
```

```
                // request for a pass for a French city
                            Input = pay(_,City:france,Date,Card)
                and validDate(Date) and validCard(Card)
           },
       effects(Input)->
               ${
               (Req[location->City] :-
                             Input = discover(Req,City:france))
                   and
                   (Req[confirmation->(Num,City,Date)] :-
                    Input = pay(Req,City,Date,_Card) and
                    generateConfNumber(Num))
                }
     ],

    usedMediators ->> med2
].

// Another Ontology #2 service
serv4[
   capability->
        _#[
       precondition(Input) ->
                    ${
                // can do passes in any city except Paris
                            Input = pay(_,City:france,Date,Card)
                and City \= paris
                and validDate(Date) and validCard(Card)
             },
       effects(Input)->
                    ${
               (Req[location->City] :-
                Input = discover(Req,City:france)
                    and City \= paris)
               and
               (Req[confirmation->(Num,City,Date)] :-
                Input = pay(Req,City,Date,_Card) and
                generateConfNumber(Num))
             }
     ],

    usedMediators ->> med2
].


/******************************** GOALS ************************************
        Goals are objects that have queries written to the Goal ontology spec
*******************************************************************************/
```

```
goal1[
   requestId -> _#,
   query -> searchTrip(bonn,innsbruck),
   result->>{}
].

goal2[
   requestId -> _#,
   query -> tripContract(serv1,bonn,innsbruck,'1/1/2007',1234567890),
   result->>{}
].

goal2b[
        requestId -> _#,
        query -> tripContract(serv2,stonybrook,innsbruck,'1/1/2007',1234567890),
        result->>{}
     ].

// need services that serve all cities in France and Austria
// WG-mediators will generate the appropriate queries to the services
goal3[
   requestId -> _#,
   query -> searchTrip(france,austria),
   result->>{}
].

goal4[
   requestId -> _#,
   query-> searchCitipass(frankfurt),
   result->>{}
].

goal5[
   requestId -> _#,
   query-> searchCitipass(innsbruck),
   result->>{}
].


goal6[
   requestId -> _#,
   query -> searchCitipass(france),
   result->>{}
].


goal7[
   requestId -> _#,
   query -> citipassContract(serv4,nancy,'22/2/2005',0987654321),
```

```
   result->>{}
].


/************************* Mediators ************************************
  A mediator needs to:
    1. Convert input - <mediatorId>[constructInput(Goal)->Input]
    2. Construct the query to be used for testing the after-state of service
       This is done by <mediatorId>[reportResult(Goal,Result)]

       This method tests that, after the appropriate translation,
       the goal is satisfied in the after-state of the service.
       Result gets bound to a formula that is appropriate for the
       representation of results in the goal ontology. That is,
       it looks like goal[result->>...] or goal[confirmation->...].
       See the header of this file for the explanations of how the goal
       ontology looks like.
*************************************************************************/

//  ******************** MEDIATOR 1 **********************
med1[constructInput(Goal)->Input] :-
    Goal[requestId->ReqId, query->Query] and
        if Query = searchTrip(From,To)
        then (
           generalizeArg(From, From1), generalizeArg(To, To1),
           Input = search(ReqId,From1,To1)
        ) else if Query = searchCitipass(City)
        then (
           generalizeArg(City, City1),
           Input = search(ReqId,City1)
        ) else if Query = tripContract(ServiceId,From,To,Date,CCard)
        then (
           generalizeArg(From, From1), generalizeArg(To, To1),
           Input = contract(ReqId,From1,To1,Date,CCard)
        ) else if Query = citipassContract(ServiceId,City,Date,CCard)
        then (
           generalizeArg(City, City1),
           Input = contract(ReqId,City1,Date,CCard)
        ) else fail.

med1[reportResult(Goal,Serv,Result)] :-
    Goal[query->searchTrip(From:location,To:location)] and
        itinerary(_)[from->From, to->To],
    Result = ${Goal[result->>Serv]}.
med1[reportResult(Goal,Serv,Result)] :-
    Goal[query->searchTrip(From:region,To:region)] and
        refresh{_[doesNotServeCity(_,_)]},
        not med1[doesNotServeCity(From,To)],
    Result = ${Goal[result->>Serv]}.
```

```
med1[reportResult(Goal,Serv,Result)] :-
    Goal[query->searchCitipass(City:location)] and
        passinfo(_)[city->City],
    Result = ${Goal[result->>Serv]}.
med1[reportResult(Goal,Serv,Result)] :-
    Goal[query->searchCitipass(Region:region)] and
        refresh{_[doesNotServeCity(_)]},
    not med1[doesNotServeCity(Region)],
    Result = ${Goal[result->>Serv]}.

// contracting requests
med1[reportResult(Goal,Result)] :-
    Goal[query->tripContract(Serv,From,To,Date,_CCard)] and
    ticket(_)[confirmation->Num, from->From, to->To, date->Date],
    Result = ${Goal[confirmation->info(Serv,Num,From,To,Date)]}.
med1[reportResult(Goal,Result)] :-
    Goal[query->citipassContract(Serv,City,Date,_CCard)] and
    pass(_)[confirmation->Num, city->City, date->Date],
    Result = ${Goal[confirmation->info(Serv,Num,City,Date)]}.

// for region-level queries check if there is a city that is not served
med1[doesNotServeCity(FromReg,ToReg)] :-
    City1:FromReg and City2:ToReg and
        not itinerary(_)[from->City1, to->City2].
med1[doesNotServeCity(Region)] :-
    City:Region and
        not passinfo(_)[city->City].

//  ********************* MEDIATOR 2 **********************
med2[constructInput(Goal)->Input] :-
    Goal[requestId->ReqId, query->Query] and
        if Query = searchCitipass(City)
        then (
           generalizeArg(City, City1),
           Input = discover(ReqId,City1)
        ) else if Query = citipassContract(_ServiceId,City,Date,CCard)
        then (
           generalizeArg(City, City1),
           Input = pay(ReqId,City1,Date,CCard)
        ) else fail.

med2[reportResult(Goal,Serv,Result)] :-
    Goal[query->searchCitipass(City:location)] and
        _[location->City],
    Result = ${Goal[result->>Serv]}.
med2[reportResult(Goal,Serv,Result)] :-
    Goal[query->searchCitipass(Region:region)] and
        refresh{_[doesNotServeCity(_)]},
```

```
    not med2[doesNotServeCity(Region)],
    Result = ${Goal[result->>Serv]}.

// for region-level queries check if there is a city that is not served
med2[doesNotServeCity(Region)] :-
    City:Region and
        not _[location->City].

// contracting request
med2[reportResult(Goal,Result)] :-
    Goal[query->citipassContract(Serv,City,Date,_CCard)] and
    _[confirmation->(Num,City,Date)],
    Result = ${Goal[confirmation->info(Serv,Num,City,Date)]}.


/****************** A generic service discovery query **********************
  Given a goal, find all services that match and print out their Ids.
  Represented as a transaction because it uses hypothetical updates.
  Hypothetical updates are simulated by insert/delete because Flora-2
  doesn''t support the hypotheticals.
*******************************************************************************/
#find_service(Goal) :-
    Serv[usedMediators ->> Mediator[constructInput(Goal) -> Input]],
    Serv.capability[effects(Input) -> Effects],
        insertrule{Effects},      // hypothetically assume the effects

    // Check if the goal is satisfied by the service and report result
    if Mediator[reportResult(Goal,Serv,Result)] then insert{Result},
    // Remove the hypothetically added rules
    deleterule{Effects},
    fail.
#find_service(_Goal) :- true.

/************************ Service contracting ****************************
  Similar to discovery, but also checks precondition
*************************************************************************/
#contract_service(Goal) :-
    // get the service to invoke: contracting queries have 4 or 5 args
    (Goal.query = _(Serv,_,_,_,_) or Goal.query = _(Serv,_,_,_)),
    Serv[usedMediators ->> Mediator[constructInput(Goal) -> Input]],
    Serv.capability.precondition(Input)=Precond,
    Precond,
    Serv.capability[effects(Input) -> Effects],
        insertrule{Effects},      // hypothetically assume effects
    // Check if the goal is satisfied by the service and report result
    if Mediator[reportResult(Goal,Result)] then insert{Result},

    // Remove the hypothetically added facts and rules
    deleterule{Effects},
```

```
    fail.
#contract_service(_Goal) :- true.


// %%%%%%%%%%%%%%% MISC DEFINITIONS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

// use Prolog''s gensym/2 to generate a new conf number
generateConfNumber(Num) :- gensym(conf,Num)@prolog(gensym).

validDate(_). // pretending that we check dates

validCard(_). // pretending that we check credit cards

// if an arg is a region - replace with new variable
generalizeArg(In,_Out) :- nonvar(In), In:region, !.
generalizeArg(_In,_In) :- true.



// %%%%%%%%%%%%%%% Sample service discovery requests %%%%%%%%%%%%%%%%%%%%%%%%
/*
  // serv1, serv2
  ?- #find_service(goal1), goal1[result->>Serv].

  // should succeed for service 1
  ?- #contract_service(goal2), goal2[confirmation->Info].

  // should fail
  ?- #contract_service(goal2b), goal2b[confirmation->Info].

  // serv2
  ?- #find_service(goal3), goal3[result->>Serv].

  // none
  ?- #find_service(goal4), goal4[result->>Serv].

  // serv1
  ?- #find_service(goal5), goal5[result->>Serv].

  // serv3 only. serv4 does not match because it does not serve Paris
  ?- #find_service(goal6), goal6[result->>Serv].

  // should succeed for serv4
  ?- #contract_service(goal7), goal7[confirmation->Info].

*/
```