Automatic Generation of Natural Language Service Descriptions from OWL-S Service Descriptions

Luís Filipe Delgado Alves Ribeiro Gonçalves

Dissertation submitted in partial fulfilment of the requirements for the degree of

Master in Computer Engineering

Adviser:

Doctor Luís Miguel Botelho, Associate Professor of ISCTE

October 2009

I Abstract

As the web grows in both size and diversity, there is an increased need to automate aspects of its use such as service coordination (e.g., discovery, composition and execution). Semantic web services combine semantic web and web service technologies, providing the support for automatic service coordination. Semantic web services are described using semantic languages (e.g., OWL-S) and can be automatically processed by intelligent agents (agent based coordination).

This dissertation aims at enhancing the service coordination process, building upon well-understood and widespread practices on natural language generation. Automated service coordination relies on the existence of formal service descriptions (semantic languages, such as OWL-S or WSML). The use of web services by people is essentially associated with the discovery, composition and execution of services that match their needs. According to the person's will, the discovered or composed service is or is not executed. This decision can only be made if the person understands the description of the service. Therefore, it is necessary that formal descriptions be converted into more natural descriptions, adequate to human comprehension.

This dissertation contributes to empower the users (knowledge engineers and common citizens) of service coordination systems with the capability to better understand and decide about discovered or composed services without the need of understanding the formal language in which the semantic web service is described. We implemented a software program capable of generating natural language service descriptions from OWL-S description. It is a template-based natural language generation system that receives the OWL-S description of a service as input and converts it into an English description.

This system will leverage the use of service coordination technology by people and allow them to have a more active role in the various stages of the service coordination process.

II Acknowledgements

First and foremost, I would like to thank my adviser Luís Botelho for his guidance and help on all aspects of my dissertation. I also thank professor António Lopes and professor Rui Lopes for promptly answering all my questions and doubts about OWL-S, XML Schema and xPath.

Thanks to the experts' panel composed of António Lopes, Fábio Calhau, Francisco Silva, João Silva and Lino Pereira for providing very useful feedback to further improve this dissertation's system, and thanks to all the people who answered the users' inquiry for helping in the evaluation of the system.

Finally, I thank my family for their enormous support and all my friends that helped me to keep motivated and focused in the completion of this thesis. To all a big thank you.

III Index

IV Figure Index

V Glossary

API - Application Program Interface. It is a set of commands, functions, and protocols which programmers can use when building software for a specific operating system

CASCOM - Context-aware Business Application Service Co-ordination in mobile Computing Environments. CASCOM was a project funded by the Framework Program 6.

ESSI - European Semantic Systems Initiative

IOPE - Input, Output, Precondition, and Effects

ISCTE-IUL – ISCTE Instituto Universitário de Lisboa

Java – Object-oriented programming language

KIF - Knowledge Interchange Format KIF is a computer-oriented language for the interchange of knowledge among disparate programs

KB - Knowledge Base. It is database for knowledge management the provides means for computerized collection, organization, and retrieval of knowledge

MINDSWAP - Maryland Information and Network Dynamics Lab Semantic Web Agents Project

NAICS - North American Industry Classification System (NAICS) is the standard used by Federal statistical agencies in classifying business establishments for the purpose of collecting, analyzing, and publishing statistical data related to the U.S. business economy.

NL - Natural Language

NLG - Natural Language Generation

OWL - Web Ontology Language. It is an ontology language for the Semantic Web. OWL is supported by W3C (World Wide Web Consortium), the major standards organization for web technology

OWL-S - Web Ontology Language for Services. OWL-S is a Semantic web service description language based on the OWL language. OWL–S and OWL are both supported by W3C (World Wide Web Consortium), the major standards organization for web technology

OWL-S API - Java API for programmatic access to software that reads, executes and writes OWL-S service descriptions

PDDL - Planning Domain Definition Language. It is the standard language for the representation of planning domains

Prolog - (PROgramming in LOGic). It is a logic programming language associated with artificial intelligence and computational linguistics

RDF - Resource Description Framework. RDF is a general-purpose language for representing information in the Web

SAWSDL - Semantic Annotations for WSDL. It defines how to add semantic annotations to various parts of a WSDL document. It allows to specify ways to describe the abstract functionalities of a service and concretely how and where to invoke it

Semantic Language - The expression Semantic Language is to be interpreted as in the semantic web literature in which it is used to refer to a language providing rich information about the object / entity that is described such as a web service. When the expression applies to service descriptions, it is meant to be understood as a language that provides information enough about the service that it may be used in automatic service coordination processes.

Shallow NLG systems – natural language systems that sacrifice some of the benefits of complete NLG systems in order to reduce their costs, resources and development time

SOAP - Simple Object Access Protocol. SOAP is an XML lightweight protocol for exchange of information in decentralized, distributed environments

SWRL - Semantic Web Rule Language. It is a semantic web rules-language based on a combination of the OWL Web Ontology Language (OWL DL and Lite) and Rule Markup Language

tuProlog - tuProlog is a Java-based Prolog engine developed at the University of Bologna, and maintained by the aliCE Research Group

UDDI - Universal Description, Discovery and Integration. UDDI is a XML-based service descriptions registry that may be used for web service discovery

URI - Uniform Resource Identifier

W3C - World Wide Web Consortium. W3Cis an international consortium that works to develop Web standards

WSDL - Web Services Description Language. WSDL is XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information

WSML - Web Service Modelling Language. It is a language for the specification of ontologies and different aspects of web services

WSMO - Web Service Modelling Ontology. WSMO is an ontology for describing various aspects related to semantic web services.

XML - EXtensible Markup Language. XML is a markup language designed to describe data

1 Introduction

Present technology, namely web services technology, allows services to be available for users or web-related programs. Web services provide some functionality on behalf of their owners, the service providers. As the amount of web services on the web increases, the need to automate web service coordination processes also grows. Semantic web services are web services associated with publicly accessible semantically rich service descriptions. Semantic web services were created with the purpose of allowing automatic service coordination, chiefly, service discovery, composition and execution. Automated service coordination is usually done by intelligent agents able of processing the semantic information contained in semantic web service descriptions. Semantic web services are described using semantic languages such as OWL-S (Web Ontology Language for Services) and WSML (Web Service Modelling Language).

In this first chapter we describe the motivation and goals for creating a natural language system able of generating natural service descriptions from formal service descriptions (i.e., OWL-S). We specify the contributions made by this dissertation to the current state of service coordination, and summarily describe the approach. Also, we briefly present the system's demonstration, as well as a concise description about the evaluation criteria and results of that evaluation. Finally, we show how the remaining of the document is organized.

1.1 Goals and Motivation

Web service coordination usually involves three main processes: service discovery, composition and execution. Service discovery is the process of finding a web service that matches the needs of the service requester. Composition is an aggregation of various services that matches the needs of the service requester that could not satisfied by any of the available services alone. Service execution is the process of executing the discovered or composed service.

The use of web services and service coordination by people is essentially associated with the discovery, composition and execution of services that match their needs. In the general case, the service client asks some service coordination system for a service with desired specified features. Although not mandatory, this process may involve a personal assistant agent that mediates the interaction between the service client and the service coordination system. The service coordination system tries to discover or to create (compose) a service that matches the client needs. Ideally, the service client should be allowed to decide whether or not to use the service. If the client wants the service to be executed, he or she will ask the service coordination system to execute the returned service. Currently, service clients will only be capable of deciding whether or not to use the service returned by the service coordination system if they understand the formal language the service is described in (e.g., OWL-S or WSML). This is possible only for the restricted minority of experts and specialists that understand formal descriptions. By providing the means for the common citizen to understand service descriptions, and hence creating enabling conditions for people to decide if they actually want a given service to be executed, the number of users that can make use of the web service technology will increase. This dissertation provides these means through the generation of natural language descriptions from the formal service description language, OWL-S. We adopted OWL-S because it is becoming a *de facto* standard fully supported by the World Wide Web Consortium and because it is extensively used in national and international research projects, in particular by the research group in which this thesis was done. Moreover, the OWL-S language possesses a large set of tools and applications supporting the adoption of OWL-S for semantic web services. The generated natural language descriptions are written in English, although other languages can be incrementally added later on.

The main goals of this dissertation are:

- Definition and implementation of a software system capable of generating natural language descriptions of semantic web services from their formal service descriptions (i.e., OWL-S); and

- Leveraging the use of semantic web services by common citizens (not experts in formal computational descriptions), endowing them with the possibility to consciously deciding whether or not to execute a given service and making them accountable for the decision of using given services.

In brief, this dissertation's motivation is to solve both the technical issue associated with the user's decision - whether the client wants to execute (accepts) the discovered or composed service - and the need of involving artificial actors as well as people in more stages of the service coordination process.

1.2 Approach Overview and Contributions

This dissertation contributes to advance the current state of the art regarding service coordination. It does not contribute to natural language generation technology. We used well-accepted and widespread natural language generation approaches and good practices to bring the service coordination process closer to the user.

The natural language generation system created for this dissertation solves a problem never solved before. As shown in chapter 2, a system capable of generating natural language descriptions from service descriptions (OWL-S or any other service description language) has never been developed (at least, we could not find any in all the literature we had access to). However, similar work has been done for Description Logics (e.g. OWL DL [Mellish and Sun 2006]). This natural language system is the major contribution of the dissertation. The developed system could be integrated in the service coordination process. More precisely, it could be used by personal agents (PA) in situations where the PA receives formal service descriptions resulting of the service coordination process. The personal agent would use the developed system to generate the natural language descriptions and present them to the user. Figure 1 depicts a hypothetical service coordination system, in which two different users (Person A and Person B) interact with the system through their Personal Agents. Person A's personal agent does not use the natural language system. A receives the OWL-S service description of the service that matched his specifications – OWL-S description of the Currency Converter service. Person B's personal agent uses the developed NLG system

to generate the English description of the Currency Converter service and presents it to Person B.

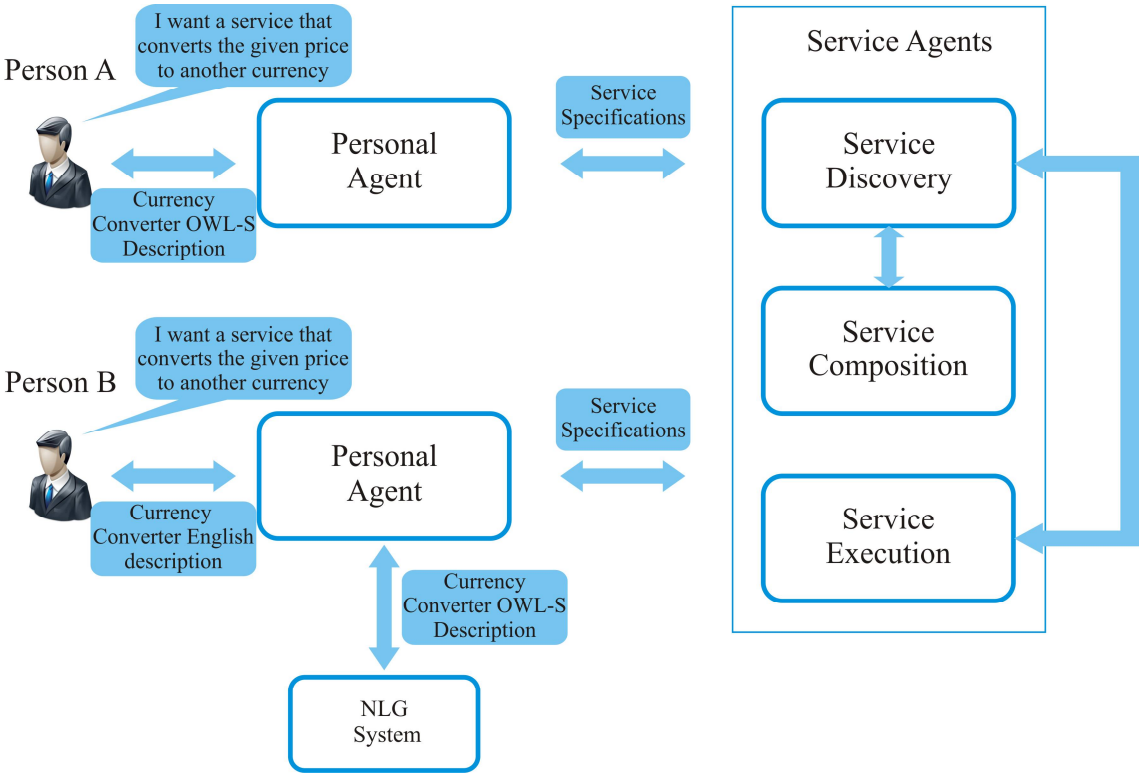

Figure 1 – Example of the NLG system use in the service coordination process

Natural language generation systems (as most software programs) usually use some form of input to produce their outcomes. This dissertation's natural language generation system receives the OWL-S service description as input, producing an English description. Not all information from the OWL-S description is used to generate the NL description; this option is explained in further detail in section 4.2. We mainly use data from the OWL–S service Profile[1] as it contains a significant amount of useful information (e.g., service inputs, outputs, preconditions and effects).

The NLG system presented in this dissertation is a template-based system, where the extracted data from the OWL S service description guides the whole process of

[1] OWL-S service descriptions are composed of three main elements: The Profile, Process Model and Grounding

language generation. That is, the templates are selected according to the information obtained from the OWL-S service description. An alternative could be to seek the required information in the OWL-S description, as guided by the selected template. Although more general, this approach would involve a computationally more complex process and would not lead to better results. We chose the less demanding data driven process in which templates selection is guided by the information extracted from the description.

The natural description generation process carried out by the system comprises four main steps. The first step extracts necessary information from the OWL-S service description. This information consists of selected service variables (e.g. service name, inputs and outputs). Unfortunately, service designers may freely label their variables (e.g., *add010* or *address*). This situation poses a problem if our system uses those labels to generate natural language text. The second step solves this problem: it maps the original service variable names into their linguistic equivalents, adequate for human comprehension. In fact, this is one of this dissertation's main contributions. Later in this document, we propose to extend the present OWL-S structure with the possibility to represent linguistic mappings. Now we are ready to advance to step number three. In template-based systems, sentences are represented as fixed text structures normally containing gaps that need to be filled. We use the information resultant from the second step to fill those gaps in the selected text templates. The outcome of this step is a natural language description close to its final form. The fourth and last step structures the description generated in step number three by adding paragraphs, producing the final description.

This dissertation contributes to the current state of service coordination with the following:

- Allows users to make an informed decision;
- Makes users accountable for their decisions (to use the service); and
- Brings the service coordination process closer to common citizens, making the whole process more natural and human while still keeping

the service coordination process automate (automatic discovery, composition and execution);

Next, we present a brief description of the system's demonstration implemented for this dissertation.

1.3 System's Demonstration

Demonstrations can help readers understand the way the developed system works and especially the natural language descriptions it generates. To a certain extent, the examples used in the demonstration will help readers develop a preliminary idea regarding the quality of the system's results.

The greater the diversity in the examples the better can we attest the system's quality and comprehensiveness. Even though we generated descriptions from several OWL-S described services, in chapter 3, we only display two demonstration examples: one for an atomic service with preconditions, and another one for a compound service. The purpose of the designed demonstration is not to provide a thorough demonstration of the system's abilities, but merely to illustrate the way the system works and the generated results.

The examples used in the demonstration were taken from the OWL-S service descriptions publically available on the mindswap's[2] web page. Mindswap, Maryland Information and Network Dynamics Lab Semantic Web Agents Project, is being carried out by a research group of the MIND LAB at University of Maryland Institute for Advanced Computer Studies. This group is accountable for web service tools such as OWL-S API, Pellet and Web Service Composer.

For each service the corresponding natural language service description was generated. The details and results of these demonstration examples, presented in chapter 3, show that the developed NLG system is easy to use and generates useful and understandable English service descriptions.

[2] http://www.mindswap.org

1.4 System's Evaluation and Results

Using a system capable of converting OWL-S descriptions into descriptions that can easily be understood by people, users will be able of deciding whether they accept the (compound or atomic) service returned to them by the service coordination system. Since the purpose of the system developed in this dissertation is to help web service users, the best way of evaluating its success is by enquiring potential users. Besides evaluating the usefulness of the generated descriptions, an enquiry to OWL-S experts can also help evaluating the technical correctness of the generated descriptions.

Two different enquiries were created: one inquiry addressed OWL-S experts, and another one designed for potential system users, users that do not have any knowledge about the used formal description language (i.e., OWL-S).

First, we applied the experts' inquiry and used the produced outcome to overcome some deficiencies of the originally generated English descriptions, both regarding their usefulness and their correctness. In this context, "correctness" means the degree to which the generated natural description correctly matches the formal service description. For the Experts Inquiry, we have created an evaluation panel composed of five OWL-S experts.

The enquiry for non expert system users mainly focuses on their comprehension of the services from the generated natural language descriptions.

On one hand, these enquiries allowed to show how well people could understand the service from the generated descriptions. On the other hand, they gave an expert's perspective on how well the natural language descriptions matched the OWL-S descriptions.

The experts' inquiry helped us improve the system's output before the non expert systems users' inquiry was applied. More precisely, information about the type of service (atomic or compound) was originally presented in technical terms not appropriate to common users (without knowledge about formal service descriptions). Being so, we changed the output description before the non-experts' inquiry was done.

The results obtained from the experts' inquiry show that the descriptions generated by the system correctly match the formal service descriptions. Experts evaluated most of the generated natural language service descriptions with the highest score regarding the system's "correctness". In the case of service comprehension from the generated natural language description, results were also satisfactory, although not as good as the ones related to the system's "correctness". This inquiry also showed that experts value detail in the description. Most suggest more depth in the description of inputs and outputs. For example, adding the type of the variables (e.g., integer, string or double) to the description, or provide more detail to variables described in other ontologies.

In the users' inquiry we asked twenty two people to rate the quality and usefulness of the generated natural language description. Results indicated that the majority of users were able to easily understand each of the described services and use the generated description to decide whether or not to use the service. Further details about the results of this evaluation are presented in chapter 5.

1.5 Document Structure

The remaining chapters of this dissertation are organized as follows. Chapter 2 presents the state of the art on the topics of semantic web service coordination, OWL-S and natural language generation. In chapter 3, we describe in detail the examples used in the system's demonstration. Chapter 4 presents a detailed description of all the aspects of the designed and implemented natural language generation system. It describes the features of the OWL-S language, as well as the taken approach and system's implementation. Chapter 5 presents the system's evaluation and obtained results. Finally, chapter 6 concludes about the work done in this dissertation and presents directions for future developments.

2 State of the Art

The main goal of this dissertation is to enhance the service coordination process, by enabling users without computer science background to understand service descriptions returned to them by the service coordination system, and hence allowing them to make informed decisions of whether or not to execute the discovered or the composed service. We start this chapter by describing the topic of agent based service coordination (section 2.1). To achieve this goal, we defined and implemented a software system capable of generating natural language descriptions of semantic web services from their formal service descriptions (i.e., OWL-S). In section 2.2 we describe some of the most used semantic service description languages. We also point out the reasons for choosing OWL-S instead of any other language, and briefly present the OWL-S structure. Finally, in section 2.3 we present some common good practices in natural language generation as this dissertation's main contribution is not to natural language generation but service coordination.

2.1 Agent Based Service Coordination

Agent based service coordination makes use of two different technologies: intelligent agents and semantic web services.

According to a widespread definition, *Agent* is a computer system situated in some environment and capable of autonomous action in this environment in order to meet its design objectives [Wooldridge 2002]. An agent has to fulfil some properties to be considered an intelligent agent: reactivity, pro-activeness, social ability (being able to interact with other agents) [Wooldridge 2002], and other properties such as temporal continuity, reasoning, rationality, veracity, mobility and learning ability [García-Sanchez et al 2009]. Intelligent agents can decide for themselves what they need to do in order to satisfy their design objectives. Agents can also be stand-alone entities or be in an environment with other agents (MAS - Multi-Agent Systems) that can potentially interact with each other and collaborate to achieve a common goal [Shoham and Leyton-Brown 2008].

Semantic web services use a technology resultant of the combination of two other technologies - semantic web and web services - in which, web services are described using semantic languages, bringing them to their full potential, and by so providing support to automatic service discovery, composition and execution [Fensel and Bussler 2002]. These can be done, for example, by intelligent agents able of processing semantic service descriptions [McIlraith, Son and Zeng 2001]. Agent-based service coordination can lead to the development of new, more powerful applications by using these different technologies cooperatively.

In subsection 2.1.1, the web service architecture, its actors, their roles and relationships are described. Subsequently the topic of semantic web service coordination (2.1.2) - processes of service discovery, composition and execution - is addressed. In the end of the semantic web service coordination subsection we identify specific shortages of current service coordination technology and elect them as the major areas for the contributions of this dissertation.

2.1.1 Web Service Architecture

The internet was created as a distributed source of information. The appearance of the web service technology enabled to extend it to a distributed source of functionality [García-Sanchez et al 2009]. A web service can be defined simply as a service with the capability of performing tasks located at some point in the Internet that can be accessed through a standard protocol [Booth et al 2004]. Web services are also described as software systems designed to support interoperable machine-to-machine interaction over a network [Haas and Brown 2004].

The web service architecture relies on the interaction of three entities: the service requester, the service broker (or service registry) and the service provider [Gottschalk et al 2002] [Booth et al 2004].

The service provider creates a web service and publishes the service by registering to the service broker's registry. Maybe the widest disseminated service registry is the UDDI (Universal Description, Discovery and Integration). UDDI is a XML-based registry which contains information about existing web services and their

functionalities [Clement et al 2004]. UDDI enables businesses to publish service listings and discover each other. A service on the UDDI is usually described using WSDL (Web Services Description Language), a XML-based description language providing information regarding the web service's invocation, location, available operations and signatures [Christensen et al 2001].

The service requester is an entity that uses a web service, provided by a service provider, to perform a task. The service requester locates the entries in the service broker's registry and binds to the service provider using SOAP (Simple Object Access Protocol), a W3C standard communication protocol [Gudgin et al 2007]. A view of the mentioned entities and their relationships is depicted in Figure 2.

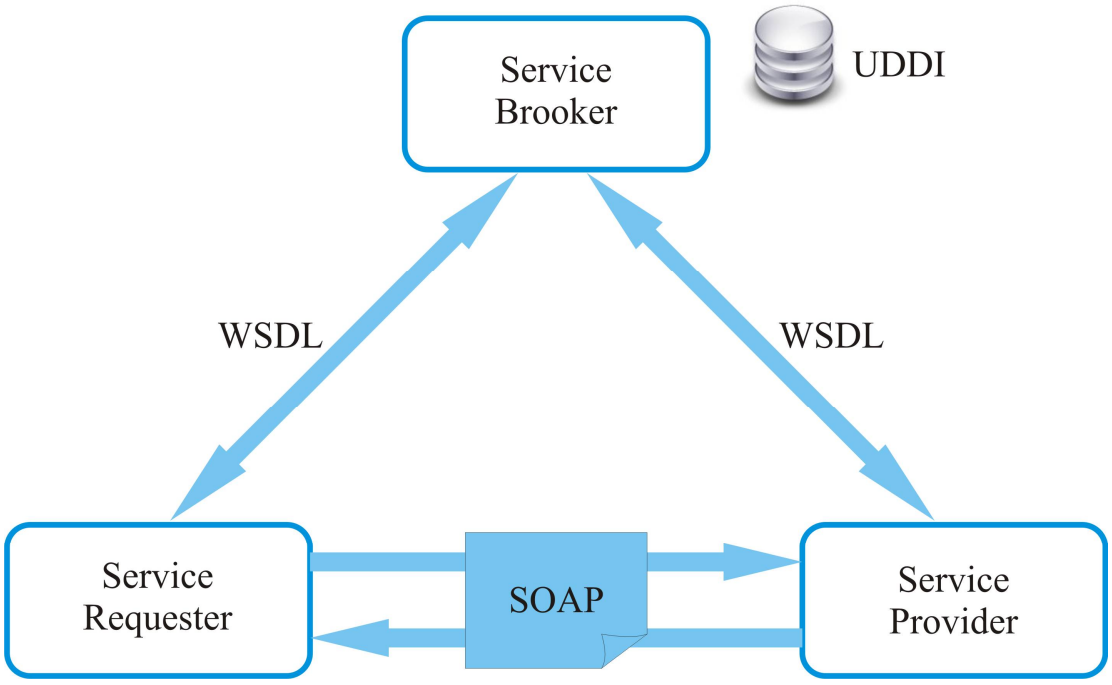

Figure 2 – Web service architecture

With this technology, any available web service can be found and used without taking into account the programming language in which the service was originally implemented. However there is an increased need to automate aspects of web services such as automatic web service discovery, automatic web service execution, automatic web service selection and composition. In fact, one of the key advantages of web

services is that they enable dynamic service composition using independent, reusable software components.

Intelligent agents are suited for automating aspects of web services. [Huhns 2002] [Chi and Song 2007] suggest that agents have characteristics that can complement web services in various aspects. For instance web services have a detailed definition of the describing, publishing and searching mechanism, but are static, lack autonomy and work in a passive way. On the other hand, intelligent agents are communicative, active and proactive. One of the main properties usually assigned to agents is their autonomy [Russell and Norvig 1995]. Together with their adaptability, intelligent agents have a set of characteristics suited to help with the automation of web services.

2.1.2 Semantic Web Service Coordination

As mentioned in the previous subsection, agents are active autonomous entities that interact with one another to achieve their objectives [Barker 2005]. Agent's autonomy and intelligence have been used to automate and improve the process of semantic web service coordination.

Semantic web service coordination aims at the coherent and efficient discovery, composition, negotiation, and execution of web services in a given context [Klush 2008]. A semantic web service is a web service whose functionalities are described using a logic-based semantic annotation over a well-defined ontology.

Service discovery is the process of locating existing web services based on the description of their functional and non-functional semantics, for example "Finding a service that sells airline tickets between Lisbon and London and that accepts payment with MasterCard credit card". Three main components are involved in service discovery: the service description language (e.g., OWL-S) that represents the functional and non-functional semantics of web services; service matching means, which can be non-logic based, logic based (using for instance OWL-S) and hybrid (combination of both); and the discovery architecture that refers to the environment in which the

discovery is performed (including physical or semantic overlay network, a kind of service information storage, and location mechanisms, among others).

If the service requester needs a service that cannot be provided by any of the known service providers alone, then service composition is needed. We can then describe service composition as the process of taking a set of component services and bundling them together, using the adequate structure, to meet the whole set of the criteria put forth by the service requester [Schumacher, Helin and Schuldt 2008]. The new compound service combines and links existing web services and other components to create new processes. Standards for service composition cover three different (overlapping) viewpoints [Barros, Dumas and Oaks 2005]:

- *Choreography* – This viewpoint captures collaborative processes involving multiple services. The interactions between these services are seen from a global perspective.

- *Behavioural interface* - This viewpoint captures the behavioural dependencies between the interactions.

- *Orchestration* - This viewpoint deals with the description of the interactions in which a given service can engage with other services, as well as the internal steps between these.

Although service composition can be done automatically by service composition planning systems, often service composition requires the service requester to specify an abstract workflow of the required compound service. Service composition can be also static or dynamic depending on whether services have to be composed at design time or at run time, respectively.

The last process comprised in service coordination is service execution, which executes the service description including the possible selection and ordered invocation of web service providers [Schumacher, Helin and Schuldt 2008]. On successful execution, a set of output results may be returned to the service requester. Sometimes, service execution also causes some effects to take place in the world. For instance, after a table reservation service in a restaurant has made the desired reservation, one or more

restaurant tables become reserved during a certain period of time. In the interest of both the service requester and the service provider certain properties/conditions need to be met, such as preserving a consistent state before and after the execution even in the presence of failures.

In both processes, service discovery and service composition, the service requester must have the possibility of evaluating the discovered or the composed service, and decide if it should be executed. In case the service requester is a person, a description of the service is required so he or she can decide whether or not to execute the service, when and under which circumstances. This description has to be understood by the person using the service even though automatic service coordination requires formal description languages. Therefore, formal service descriptions must be translated into some form of human-readable descriptions that can be understood by human service clients. Since there are no computational tools that perform such a translation, this dissertation aims to offer a solution to that problem, where formal service descriptions are translated into natural language.

Even though agents can completely automate the process of service coordination, there might be some issues worth considering. Take the case of service discovery, where the agent finds the service by itself. If the service requester is a person, he or she should have the opportunity to decide whether or not to use the received service, so that he or she may be deemed responsible for the consequences of using it. Although technically agents can automate the whole service coordination process, it is not ethically acceptable to ever leave humans out of the loop when it comes to the decision of whether or not to use discovered or composed services. Providing human readable service descriptions will allow people analyzing and evaluating received services and to make their decisions, while avoiding the need of understanding formal service description languages. The subject of this dissertation goes along with the growing amount of work on Natural Language Generation from semantic web material, due to practical importance of presenting such material to people [Mellish and Sun 2006].

In our view, service coordination may take advantage of context information by using the context-aware computing paradigm. In this paradigm, applications can use context information to better adapt their behaviour to the current situation. Context information can be defined as any information that characterizes a situation related to the interaction between users, applications, and the surrounding environment [Dey, Abowd and Salber 2001]. In context-aware computing, context information on an entity (person, places or objects relevant to the interaction between a user and application, including themselves) is collected by some component/process of the application or by some third party system (often termed context management system), next this information is analyzed and context-dependent actions are executed accordingly. This process aims at tailoring the system's behaviour to its or its user's context [Schumacher, Helin and Schuldt 2008].

The generation of natural language descriptions from formal service descriptions may also take advantage of the context computing paradigm, so that the generated natural description is more adapted to the current circumstances including the user profile. For instance, if the natural language generation system has several target languages, context information may be used to help it choose a language that may be understood by the user; or if a word has two possible translations, context information may be used to help choose the translation that makes more sense in that context. Even though this paradigm is gaining attention of the research community [Gonçalves, Costa and Botelho 2008] and may present some benefits to use in software systems, the context computing paradigm was not used in present work because its use is not the main purpose of this dissertation and because it would not be possible to implement it within the dissertation time constraints.

2.1.3 Summary

Summarizing, the research on agent-based service coordination has been focusing mainly on service discovery [Sivashanmugam, Verma and Sheth 2004] [Oundhakar et al 2005], service matchmaking [Fenza, Loia and Senatore 2008], service composition processes [Guidi, Lucchi and Mazzara 2007][Chi and Song 2007], service execution [Lopes and Botelho 2005], and their automation by using agents in the

service coordination process [Barker 2005] [Schumacher, Helin and Schuldt 2008]. Also, a significant amount of research has been made regarding the acquisition, management and use of context information for improving several aspects of agent based service coordination [Gonçalves, Costa and Botelho 2008]. Although there has been some work aiming the natural language presentation of formal semantic web material, such as OWL DL [Mellish and Sun 2006] or other web material [Wilcock 2003] [Bontcheva and Wilks 2004], no work has been done regarding the translation of formal service descriptions in general, and OWL-S descriptions in particular, to human understandable descriptions. This was selected as the main contribution of this dissertation.

Semantic service description languages provide means for automatic web service discovery, composition and execution. In section 2.2 we present three of the most widespread semantic description languages - SAWSDL, WSML and OWL-S and then further elaborate about OWL-S.

2.2 Semantic Languages

In the semantic web, services are described using semantic languages, such as OWL-S. For a web service to be discovered and used, service coordination software agents need computer interpretable service descriptions that help them understand what the service does and how, as well as the way it can be accessed.

Until recently, web services have been described using the Web Service Description Language (WSDL), an XML format for describing network services [Christensen et al 2001]. However the WSDL language does not provide enough support for the purpose of automatic service discovery and composition. In order to solve this problem a mechanism was developed in which semantic annotations could be added to WSDL components, the Semantic Annotations for WSDL (SAWSDL).

Other prominent semantic web service description language is the OWL-S language. Contrary to SAWSDL, OWL-S is not a mere extension to other language, but a new language for web service description based on OWL.

Another language recently proposed for the description of web services is the Web Service Modelling Language (WSML), created by the European Semantic Systems Initiative (ESSI) WSML working group. WSML is a rule-based language for describing semantic web services, but is not a W3C recommendation as SAWSDL or OWL-S.

For the purpose of this dissertation, a semantic service description language had to be chosen. In order to decide which language to pick, a comparison between the most used service description languages is made in the subsections 2.2.1, 2.2.2 and 2.2.3. We briefly describe the SAWSDL, WSML e OWL-S languages and present the reasons why OWL-S was selected for this dissertation, its advantages and disadvantages.

2.2.1 SAWSDL

Since WSDL lacked the required information for automated service coordination, W3C proposed to extend it with so-called semantic annotations. The result is actually named SAWSDL (Semantic Annotations for WSDL). SAWSDL only provides mechanisms to help define how to add semantic annotations to various parts of a WSDL document such as input and output message structures, interfaces and operations [Farrell and Lausen 2007]. However SAWSDL does not specify a language for representing such semantic models. Therefore SAWSDL is simply a syntactic extension of WSDL, unlike OWL-S or WSML that are new languages for semantic service description. Moreover, although being a W3C standard, SAWSDL has limited software support and no defined formal grounding, which constitutes two serious disadvantages [Schumacher, Helin and Schuldt 2008].

2.2.2 WSML

Web Service Modelling Ontology (WSMO) is an ontology for describing various aspects associated to semantic web services, while WSML provides the syntax and the semantics for this ontology [Bruijn 2008]. WSML can describe semantic web services in terms of their functionality (service capability), imported ontologies, and the interface through which they can be accessed for orchestration and choreography [Schumacher, Helin and Schuldt 2008]. WSML is based on logical formalisms (e.g., F Logic). As opposed to SAWSDL and OWL-S, WSML is not a W3C standard. Due to

the fact that in WSMO's submission to the W3C in 2005, WSMO was considered to be developed apart from the W3C standards, specifically, it did not make use of the RDFS concept of Class and Property; and it did not connect to WSDL's definition of services. In addition WSML still lacks formal semantics for the service interface part (orchestration and choreography) [Schumacher, Helin and Schuldt 2008].

2.2.3 OWL-S

OWL-S is a language, replacing the former language known as DAML-S, used to describe the semantics of services. This language is also based on the Ontology Web Language (OWL), a recommendation produced by the Web-Ontology Working Group at the World Wide Web Consortium (W3C) and is grounded in WSDL (Web Services Description Language), an XML-based language that provides a model for describing web services [Schumacher, Helin and Schuldt 2008].

One of the main problems of OWL-S is its limited expressiveness. It does not take into account the notion of time, change or uncertainty, only describing static aspects of the world. Also OWL-S conditional effects can only be conditioned by their results, not their inputs [Schumacher, Helin and Schuldt 2008]. Finally, OWL-S documentation does not provide a specification regarding the way conditions should be written and interpreted. Thus, preconditions, conditional effects and control constructs involving conditions cannot be written in OWL-S, they must be written in another language. Nevertheless, OWL-S is supported by W3C and is based on its standards, such as OWL or WSDL, and there is a large set of applications and research using and about this language [Schumacher, Helin and Schuldt 2008].

After considering various available web services descriptions languages, OWL-S was the language adopted, as it is fully supported by W3C and widely used in international research projects such as CASCOM, and in the industry. In particular, the ISCTE-IUL Agents and Artificial Intelligence research group, in which this dissertation was done, is using OWL-S in their projects. The OWL-S language also possesses a large set of tools and applications supporting the adoption of OWL-S for semantic web services.

2.2.3.1 OWL-S Structure

The OWL-S language is composed of three main elements [Martin et al 2004a]: Profile, Process Model, and Grounding. The Profile describes what the service does. It is used mainly for service discovery and service composition using artificial intelligence planners. The Process Model provides a detailed perspective of how to interact with a service, and describes the way the service is composed of simpler elements. The Grounding specifies the details of how to access the service, as provided by specific service providers. Each of these elements is explained in further detail in subsections 2.2.3.2, 2.2.3.3 and 2.2.3.4.

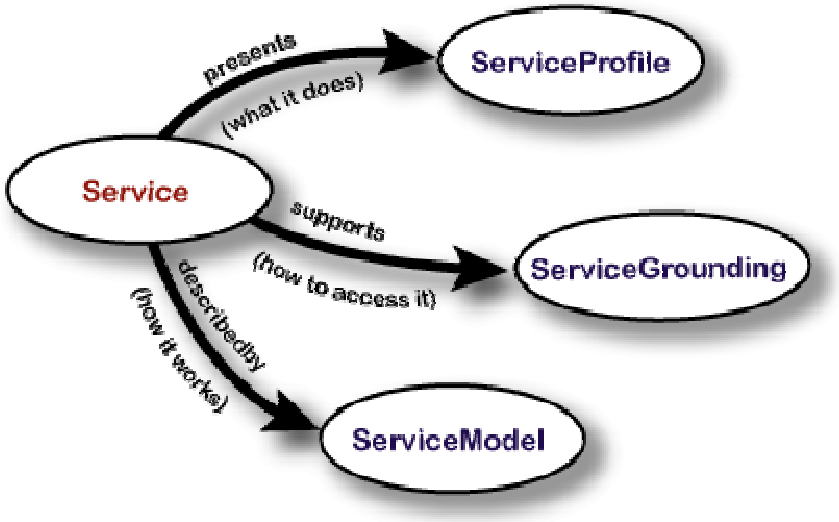

Figure 3 - Top level of the service ontology [Martin et al 2004a]

2.2.3.2 Service Profile

The Service Profile provides a high-level description of a service and its provider, it is also used to request or advertise services. OWL-S Profile describes a service as a function of three basic types of information: the organization that provides the service, the function computed by the service and a host of features that specify characteristics of the service [Martin et al 2004a]. Information regarding the owner of the service includes contact information of the organization responsible for it, such as references to entities responsible for the service (or some aspect of the service). For the second type of information, the functional description, OWL-S describes the service,

inputs and generated outputs, its pre-conditions (if any), and the effects produced by its execution. Finally, the last type of information is an unbounded list of service parameters that can contain any type of information, for example characteristics of the service such as an estimate of the average response time.

2.2.3.3 Process Model

The Process Model provides a detailed perspective of the way the service works and specifies the ways to interact with it. It describes the service control flow and data flow and the possible steps performed during service execution. In OWL-S, processes can be distinguished between atomic, simple and compound.

Atomic processes have no sub-processes, and are executed on a single step, from the requester's point of view. They are also required to have the Grounding. Simple processes, however, are not associated with the Grounding, thus not having a specific binding to a physical service. They too have single-step executions as atomic processes, and can be used to be realized by an atomic process or to provide a simplified representation of some compound process [Martin et al 2004a].

On the other hand, compound services are workflows, consisting of other non-compound or compound processes. For such workflows to be constructed, different control flow operators are used, such as *Sequence*, *Split* or *If-Then-Else*. While atomic processes expect one message and returns one message in response, compound processes maintain some state that is changed following the reception of messages [Martin et al 2004a].

Processes may have two types of purpose. First, they can create and return some new information based on the information given and the state of the world. The generated information is described by the inputs and outputs of the process. Secondly, they can change the world. This transformation is described by the preconditions and effects of the process.

2.2.3.4 Grounding

The Service Profile and the Process Model are considered as abstract representations, while the Grounding deals with the concrete level of specification.

The grounding specifies the details of how to access the service, including communication protocol information, serialization information, etc. The Grounding maps from an abstract to a concrete specification of the service description elements required for interacting with the service, the mapping is made between OWL-S and WSDL [Martin et al 2004a].

2.2.4 Summary

In this section we made a comparison between the most used service description languages. SAWSDL is only a syntactic extension of WSDL, unlike OWL-S or WSML that are new languages for semantic service description. Although being a W3C standard, SAWSDL has limited software support and no defined formal grounding. The WSML language can describe semantic web services in terms of their functionality (service capability), imported ontologies, and the interface through which they can be accessed for orchestration and choreography. But contrarily to SAWSDL and OWL-S, WSML is not a W3C standard. It is also known that this language still lacks formal semantics for the service interface part (orchestration and choreography). Finally, OWL-S is based on the W3C standard OWL. One of the main problems of OWL-S is its limited expressiveness, also OWL-S documentation does not provide a specification regarding the way conditions should be written and interpreted. Thus, preconditions, conditional effects and control constructs involving conditions cannot be written in OWL-S, they must be written in another language. However, it is supported by W3C and based on other W3C standards (OWL and WSDL). Moreover, OWL-S is widely used in international projects and there is a large set of applications and research using and about this language. All this reasons lead us to choose OWL-S instead of the other service description languages.

None of the semantic languages described in subsections 2.2.1, 2.2.2 and 2.2.3 provide support to allow their translation to some human understandable form. However,

this dissertation's NLG system generates human readable descriptions using information directly extracted from the formal description. Most of the expressions used to label the variables in the formal descriptions (e.g., *inputxpto* or *output2009*) are completely idiosyncratic of the service designers and cannot be directly used in human readable descriptions. Due to their idiosyncratic nature, it is not feasible to provide linguistic equivalents to such expressions for all possibly existing services. These linguistic equivalents must be provided for each service description. OWL-S and the other service description languages do not provide ways to describe the mappings between the idiosyncratic expressions used in the service description and their linguistic equivalents. This dissertation proposes an approach to advance the state of the art regarding this problem. More precisely, we propose an extension to the current OWL-S structure that will allow the addition of such linguistic mappings.

2.3 Natural Language Generation

The amount of work on natural language generation (NLG) from semantic web contents is increasing [Mellish and Sun 2006] [Bontcheva and Wilks 2004].The reason behind this uprising interest is the importance of offering such contents (semantic web material) in an easy and understandable way to both knowledge engineers and common citizens. Section 2.3 aims at describing, understanding, and selecting common good practices in natural language generation systems. However, as the main contribution of this dissertation is not to natural language generation, this section does not provide a detailed critical comparative analysis of concurrent approaches. Nevertheless, the last subsection 2.3.3 presents a brief comparative analysis of three of the most used approaches to natural language generation.

Natural language generation is the process of generating natural language outputs (e.g., texts and sentences) from computer based representations or building computer software systems that can produce meaningful texts in English or other human languages from some underlying non-linguistic representation of information [Reiter and Dale 2000]. Daniel Jurafsky and James Martin define Natural Language

Generation as the process of constructing natural language outputs from non linguistic inputs [Jurasfsky and Martin 1999].

The Corpus Based Approach [Reiter and Dale 2000] is a widely used approach to analyse and design natural language generation systems. It is explained in subsection 2.3.1. Subsequently, we will make a brief overview of a generic architecture of many natural language generation systems. Section 2.3 ends with a summary of the different types of natural language generation systems described in the literature.

2.3.1 Corpus Based Approach

The determination of the inputs the NLG system will be provided with, the outputs it is expected to produce, and whether the output only communicates the data and information available in the system's input is of key importance for the requirements analysis of specific NLG system [Reiter and Dale 2000].

The Corpus, in the Corpus Based approach, is a collection of natural language outputs and associated inputs. The Corpus Based approach goes through a series of steps. The first step is to create the initial Corpus. The outputs contained in the initial Corpus are human-authored texts - natural language outputs expected to be produced by the NLG system. As in some natural language generation systems, human-authored texts may not exist. In such cases, no output texts will be used in the analysis.

Subsequently to the creation of the Initial Corpus, the NLG system analysis per se can begin. The main goal is to identify and classify the information content of the texts in the Initial Corpus, particularly if any of the human-authored Corpus texts contains information not accessible to the system. This analysis requires classifying all the grammatical constituents of the Corpus text (sentences, clauses, others) into categories. In [Reiter and Dale 2000], four categories of constituents are proposed: Unchanging text (always present in the output texts); Directly-available data (accessible directly in the input data or an associated knowledge or database); Computable data (information derived from the input data through some computation or reasoning involving other data sources) and Unavailable data (information which is not present in or derivable from the input data).

There are a number of solutions to the problem of unavailable data:

- Make more information available to the NLG system;
- Change the output text to eliminate constituents that express unavailable data; and
- Expect a human to write and incorporate textual constituents which communicate unavailable data.

After the natural language generation system's analysis the people responsible for the system (developers and users) must agree on a final text Corpus. This takes into account the required changes to the initial Corpus, such as improving the target texts readability, dealing with the unavailable data or removing textual constituents (computable data elements) that provide insufficient benefit to justify the cost of generating them. The outcome of all these changes is a set of texts that represents the output to be generated by the NLG system, the Target Text Corpus.

2.3.2 Architecture

There is no definite architecture when it comes to language generation systems, as it depends on the requirements of the system itself. In spite of that, the architecture presented in [Reiter and Dale 2000] is used, to a certain extent, by many NLG systems and allows us to get a brief overview of all the decisions that need to be addressed when building such systems.

A language generation system can be decomposed into three modules: the Document Planner; the Microplanner and the Surface Realiser. These three modules can be seen as a linear pipeline, starting in the Document Planner and ending in the Surface Realiser.

The Document Planner is responsible for deciding what information should be presented in the output text (Content Determination) and how the parts of the content should be grouped in a single document (structural aspect of the Document Planner, also known as Document Structuring).

The Microplanner's task is to take the output of the Document Planner and refine it to produce a more detailed text specification. That is achieved by making the following decisions:

- Lexicalisation - choosing the particular words, syntactic constructs or other linguistic resources to express the content selected by the Content Determination of the Document Planner;
- Referring Expression Generation - determining the expressions that should be used to refer to entities; and
- Aggregation - making decisions of how the structures generated by the Document Planner are transformed into linguistic structures as sentences and paragraphs.

Finally the Surface Realiser is accountable for converting the Microplanner's abstract representations of sentences (Linguistic Realisation) and structures (Structure Realisation) into real text.

2.3.3 Types of Language Generation Systems

We will now present some of the possible types of language generation systems as described in the literature. [Hovy 1992] divides generation systems in four types: canned-text based, template based, phrase-based and feature-based. Also Reiter and Dale [Reiter and Dale 1997] split between systems that use template-based approaches for what they call sentence plans, and systems that use Abstract Sentential Representations. [Jurasfsky and Martin 1999] have similar thoughts to those of Reiter and Dale [Reiter and Dale 1997] and Hovy [Hovy 1992] since they distinguish three types of Natural Language Systems similar to the ones of the mentioned authors: canned-text, template filling and a third more complex type with a similar architecture to the one of the subsection 1.3.2, what van Deemter, Krahmery & Theunez [van Deemter, Krahmery and Theunez 2003] call "real" Language Generation System.

Now we illustrate the known pros and cons of three types of NLG systems mentioned above: Canned-Text, Template-Based and "real" Language Generation Systems.

In a **Canned-Text** system [Hovy 1992] [Jurasfsky and Martin 1999] the character sequence to be used has already been determined by the author of the program. It is used to display unchangeable information as error messages, warnings, titles, and so on.

- **Pros**: it is trivial to produce.
- **Cons**: it is unable to adapt to new situations without the intervention of a programmer which makes it completely inflexible.

Template-Based systems represent sentences as "boilerplate" text and parameters that need to be inserted into the boilerplate [Reiter and Dale 1997]. This "boilerplate" represents fixed linguistic structures that usually contain gaps, where these gaps are filled by the NLG system's inputs [Hovy 1992]. These inputs usually don't receive any further processing in older systems. However some new systems do perform linguistic processing (intermediate representations). Template-Based approaches are normally used in systems that produce messages or text with slight alterations, since they are not flexible enough to handle applications with any realistic variation in the content to be expressed or the context of the expression [Jurasfsky and Martin 1999].

- **Pros**: a little more complex than Canned-Text, but still easy to implement.
- **Cons**: this systems weakness is also their inflexibility, even though more flexible than Canned-Text; as Canned-Text systems, they are difficult to maintain.

Finally, **'real' language generation** systems [van Deemter, Krahmery and Theunez 2003] are much more complex than the types of systems previously presented. These systems are based on linguistic principles, and use grammars that apply rules so that syntactically, morphologically, and orthographically correct texts are produced

[Reiter and Dale 1997]. Systems of this kind specify phrasal units (nouns, verbs, adjectives, adverbs) and their relations with abstract representation languages.

- **Pros**: better maintainability, better output quality and variation, and linguistically well-founded.
- **Cons**: complex and more costly to implement.

There has been some discussion on the advantages of choosing one of the NLG system approaches over the other. Busemann and Horacek [Busemann and Horacek 1998] go as far as claiming that shallow approaches combining different granularity in a flexible way are better suited for small applications when compared to "real" NLG systems. Shallow approaches or "intermediate" techniques [Reiter and Mellish 1993] are approaches that sacrifice some of the benefits of "real" NLG systems in order to reduce their costs, resources and development time. Reiter & Mellish [Reiter and Mellish 1993] also defend that shallow approaches are more suitable as long as the corresponding "real" NLG approaches are poorly understood, less efficient, or more costly to develop. In their turn, Deemter, Krahmery & Theunez [van Deemter, Krahmery and Theunez 2003] challenge the thoughts that template-based approaches are necessarily inferior to other approaches and report some recent systems that are able of achieving generative capabilities beyond what is usually expected from template-based systems.

After taking into account all the pros and cons of the different types of natural language systems, a decision was made to use a Template-Based system in this thesis. This decision was based on the fact that the general form of sentences or other constructions present on the Natural Language descriptions generated from the OWL-S description are sufficiently invariant so that they can be predetermined and stored as templates. Also the fact that this kind of system is easier to implement (no need to specify phrase representations) is a plus since there is a limited time schedule to the completion of this NLG system.

3 System's demonstration

The system developed for this dissertation was tested with nine OWL-S service descriptions, all of them taken from the public website of the Mindswap project. Although the system has generated natural language descriptions for all the nine OWL-S descriptions, this chapter illustrates the way the system works for only two service descriptions. However, the nine services were all used for the system's evaluation, as explained in chapter 5.

The examples presented in this chapter allow us to understand the way the implemented natural language generation system works. With this demonstration we want to explain, through examples, the generation of natural language descriptions from OWL-S service descriptions with special emphasis on the interaction between the system and the user. Figure 4 depicts the user's first interaction with the natural language system's interface.

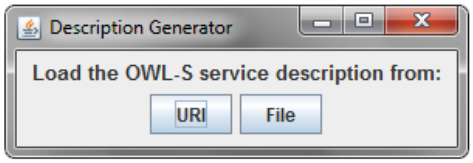

Figure 4 – NLG system's first menu

The user must choose between loading the OWL-S service description from an URI or a file stored in her or his computer disk. According to the user's choice, the system either opens an input frame to allow the insertion of the URI of the input service description, or opens a file chooser in which the user can specify the OWL-S description's pathname. From this stage on the user intervention is no longer required. The NLG system will produce a natural language description of the specified service and present it back to the user. Figure 5 shows the user's interaction with the service.

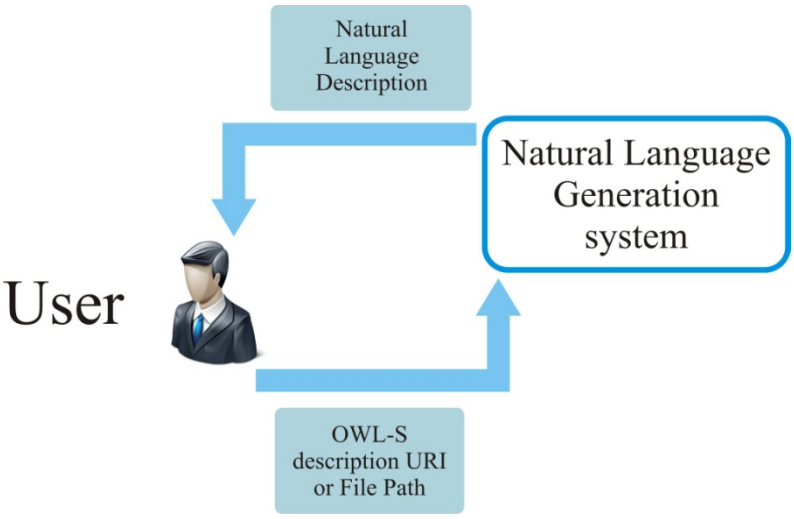

Figure 5 – Natural Language Generation System

This demonstration chapter describes two examples, one for an atomic service with specified preconditions, and another for a compound service. The used OWL-S service descriptions are both taken from public website of the Mindswap project.

- The first example is the atomic service *Babel Fish Translator*.
- The second example is the compound service *Book Price Finder*.

Babel Fish Translator Example

As previously illustrated in Figure 4, the users choose between loading the OWL-S service description from a disk file or from a specified URI. Assuming the description is loaded from an URI, a frame as the one in Figure 6 appears. The user now inserts the service description's web address - http://www.mindswap.org/2004/owl-s/1.1/BabelFishTranslator.owl for the *Babel Fish Translator's* OWL-S description.

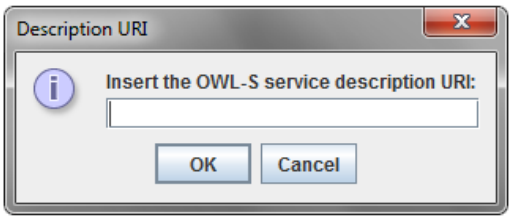

Figure 6 – Load OWL-S description from an URI

The system will retrieve the entire service description from the hosting server and extracts the relevant information. Figure 7 shows *Babel Fish Translator's* Profile since the NLG system mainly uses that part of the service description to generate its output.

```
<!-- Profile description -->
<mind:LanguageService rdf:ID="BabelFishTranslatorProfile">
        <service:presentedBy rdf:resource="#BabelFishTranslatorService"/>
        <profile:serviceName xml:lang="en">BabelFish Translator</profile:serviceName>
        <profile:textDescription xml:lang="en">
        Convert text from one language to another language. Supported languages are Dutch,
        English, French, German, Italian, Japanese, Korean, Portuguese, Spanish, and Russian.
        The valid input output pairs is given by the property canBeTranslatedTo.
        </profile:textDescription>
        <profile:hasInput rdf:resource="#InputString"/>
        <profile:hasInput rdf:resource="#InputLanguage"/>
        <profile:hasInput rdf:resource="#OutputLanguage"/>
        <profile:hasOutput rdf:resource="#OutputString"/>
        <profile:hasPrecondition rdf:resource="#SupportedLanguagePair"/>
</mind:LanguageService>
```

Figure 7 – *Babel Fish Translator's* Profile

From this Profile, and using the OWL-S API, the system extracts the following variables:

- Service name: *BabelFish Translator*.
- Service inputs: *InputString*, *InputLanguage* and *OutputLanguage*.
- Service output: *OutputString*.
- Service precondition: *SupportedLanguagePair*.

The system also extracts the informal service description explicitly provided by the service owner. In addition to the Profile, we use the Process Model to determine if the service is atomic or compound. If the process is an instance of an atomic process it means that the service is atomic, else it is a compound service.

All the extracted information, with the exception of the informal description provided by its owner, as a comment, is converted into a more understandable form.

This is accomplished by mapping the extracted variables into their linguistic equivalents, using a linguistic mapping table. Currently, OWL-S descriptions do not include linguistic mapping tables. They have been built-in within the system. One of the main contributions of this dissertation is to propose an extension to OWL-S that represents the linguistic mapping tables, one for each desired language. This contribution is described in more detail in section 4.2.2. Figure 8 shows an example of the linguistic mapping applied to the current service variables. The variables originally extracted from the OWL-S description appear on the left of the → symbol, and their linguistic equivalents appear on the right.

```
BabelFish Translator -> Babel Fish Translator
InputLanguage -> input language
OutputLanguage -> output language
InputString -> the text to be translated
OutputString -> translated text
SupportedLanguagePair -> the input and output language pair
is supported
```

Figure 8 – *Babel Fish Translator's* mapping table

The system uses the output from the linguistic processing to guide the selection and filling of the text templates used to generate the natural language description. That is, according to the value of the variables the system selects and fills the appropriate templates. For instance, if the service had no inputs, meaning the value of the inputs variable would be *none*, that would lead to the selection of the text template representing the no inputs situation. On the other hand if the input's value were *input language*, the system would select the template representing the case with only one input and fill the template with the value - *input language*.

Figure 9 depicts the text templates selected by the NLG system used to generate the first sentence of *Babel Fish Translator's* English description. It also shows the result of filling those templates with the linguistically processed information extracted from the OWL-S description.

Used templates

1. "The service _____"
2. " needs to receive the _____"
3. ", _____"
4. " and _____ as parameters."

Filled templates

1. "The service Babel Fish Translator"
2. " needs to receive the input language"
3. ", output language"
4. " and the text to be translated as parameters."

Figure 9 – Examples of text templates used in *Babel Fish Translator's* first sentence

After all the remaining selected templates are filled, the system aggregates them into a filled single final text template representing an almost ready natural language description. Figure 10 shows the result of aggregating all the filled templates of the *Babel Fish Translator* example.

"The service Babel Fish Translator needs to receive the input language, output language and the text to be translated as parameters. It has only the translated text as output./ln This service also has the following precondition: the input and output language pair is supported./ln The owner of the service describes it as: "Convert text from one language to another language. Supported languages are Dutch, English, French, German, Italian, Japanese, Korean, Portuguese, Spanish, and Russian. The valid input output pairs is given by the property canBeTranslatedTo."./ln Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service."

Figure 10 – Aggregation of the text templates used in *Babel Fish Translator's* description

At this stage the English description requires only structural processing - addition of paragraphs (represented in Figure 10 by the unique special symbol "*/nl*"). This is done when the description is presented to the user. In Figure 11 we show the *Babel Fish Translator's* final English description, the NLG system's output.

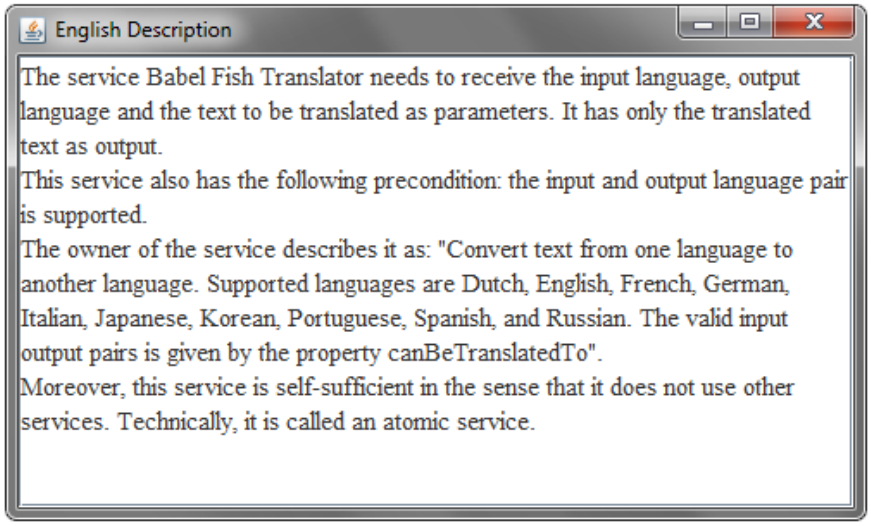

Figure 11 – Output English description – *Babel Fish Translator*

In the next subsection we describe the example for the compound service *Book Price Finder*.

Book Price Finder Example

In the first example the OWL-S description was loaded from an URI. In the *Book Price Finder's* example we chose to load the service description from a file. The system provides an interface as the one shown in Figure 12, allowing users to navigate their file system and select the service.

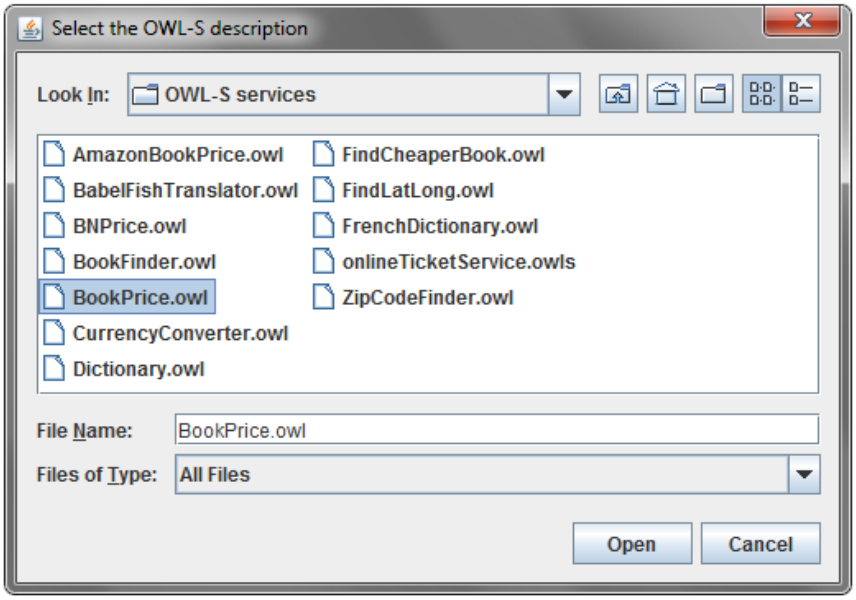

Figure 12 – Load the OWL-S description from a file

The steps leading to the generation of the natural language description for the *Book Price Finder* service, and compound services, are very similar to ones for atomic services. In fact, the only changes in the generation of English description for each type of service are the information about the type of service extracted from the OWL-S description and the selection of the correct template. That is, instead of selecting the first text template (1) in Figure 13, which corresponds to atomic services, the system chooses the second template (2) for compound services.

1. `Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service.`
2. `Technically, this is a compound service because it combines other services in a new aggregate service.`

Figure 13 – Text templates for atomic and compound services

The natural language generation system's outcome for the service *Book Price Finder* is depicted in Figure 14. In this example, the text template for compound services is used.

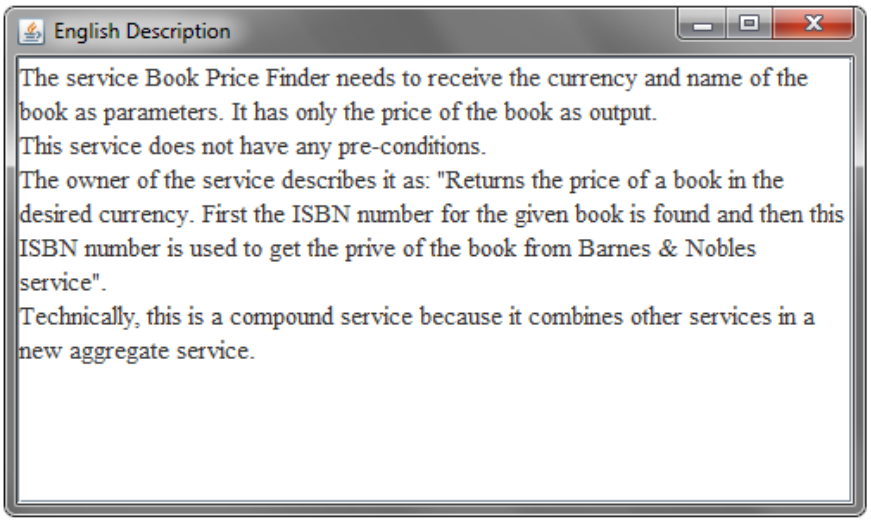

Figure 14 – Output English description – *Book Price Finder*

In this chapter we explained how the system interacts with the user. Through this demonstration's examples we were able to describe the steps taken by the system in order to generate the English descriptions and demonstrate how the implemented natural language generation system works. Moreover, we identified the main differences in generating descriptions for atomic and compound services: the information about the type of service extracted from the OWL-S description and the selection of the corresponding template. We also described how OWL-S variables (inputs, outputs, preconditions) are presented in the English description. In the next chapter we describe the natural language system in detail.

4 Natural Language Generation System

In this chapter we describe the natural language generation system created in the scope of this dissertation. This NLG system follows a template-based approach, where text templates are defined prior to the time of generation. It receives OWL-S service descriptions and produces the corresponding natural language (English) descriptions.

We chose to generate the natural language descriptions from OWL-S descriptions because the OWL-S language is vastly used in industry projects and has a large set of tools and applications supporting the adoption of OWL-S for semantic web services. Additionally, OWL-S is a W3C standard, building upon the also standard OWL Semantic Web language, and can be used in the context of other web service standards such as WSDL or UDDI [Martin et al 2004b]. We selected English to be the NL description's language because it is the most spoken language worldwide, and is the dominant international language in communications, science, business and other areas. However, our choosing the English language does not preclude other languages to be incrementally added later.

In section 4.1 we present the OWL-S language. Section 4.2 describes the approach used in this system. More specifically, we reveal and justify the choices we made and describe the parts composing the system. Finally, in 4.3 we briefly present the system's implementation.

4.1 OWL-S Language

The OWL-S description language was created with the purpose of providing the support for automatic service coordination. As described in subsection 2.2.3 an OWL-S description is composed of three main elements: Profile, Process Model, and Grounding. These elements provide three essential types of knowledge about a service: what it does (Profile), how it works (Process Model) and how to access it (Grounding).

OWL-S descriptions are organized in an object-oriented fashion. The description top class, that is, the root of an OWL-S description, is the *Service* class, which provides an organizational point of reference for a declared web service. The *Service* class has

the properties *presents*, *describedBy*, and *supports*. Each instance of *Service* will *present* a ServiceProfile, will be described by (*describedBy*) a ServiceModel, and will *support* a ServiceGrounding [Martin et al 2004a]. This is depicted in Figure 15.

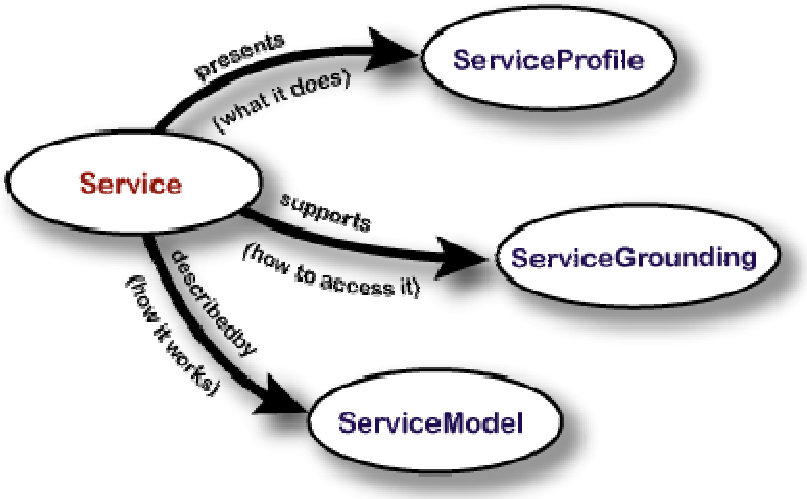

Figure 15 - Top level of the service ontology [Martin et al 2004a]

The Profile, Process Model and Grounding are specified in more detail in subsections 4.1.1, 4.1.2 and 4.1.3.

4.1.1 Profile

The service Profile presents information suitable for service discovery and service selection. The service coordination client specifies the desired services through the specification of a possibly incomplete Profile. The service discovery process seeks for services whose Profile matches the specification. If more than one service is found, one is selected using its Profile. If no service matches the specification, the service composition process tries to create a compound service that matches the specified Profile. The composition process may also use service Profiles for discovering and selecting services to use in the new compound service, in a similar fashion as the discovery process.

The Profile holds different types of information. It provides human-readable information such as the service name, text description or contact information, represented in OWL-S Profiles through the properties: *serviceName*, *textDescription*

and *contactInformation*. The *serviceName* property refers to the name of the service and can also be used as service identifier. The *textDescription* presents a brief description of the service, produced by the service owner. Ideally it summarizes what the service offers, it describes what the service requires to work, and any additional useful information. Only one service name and text description is allowed per service Profile. The property *contactInformation* offers a mechanism of referring to humans or individuals responsible for the service. However the range of this property is unspecified within OWL-S, but can be restricted to some ontology [Martin et al 2004a]. Both the service name and text description are used to generate the natural language description. The former is used to identify the service in the NL description; the latter may add information not present in the generated part of the description. Figure 16 shows the properties and classes of the Profile.

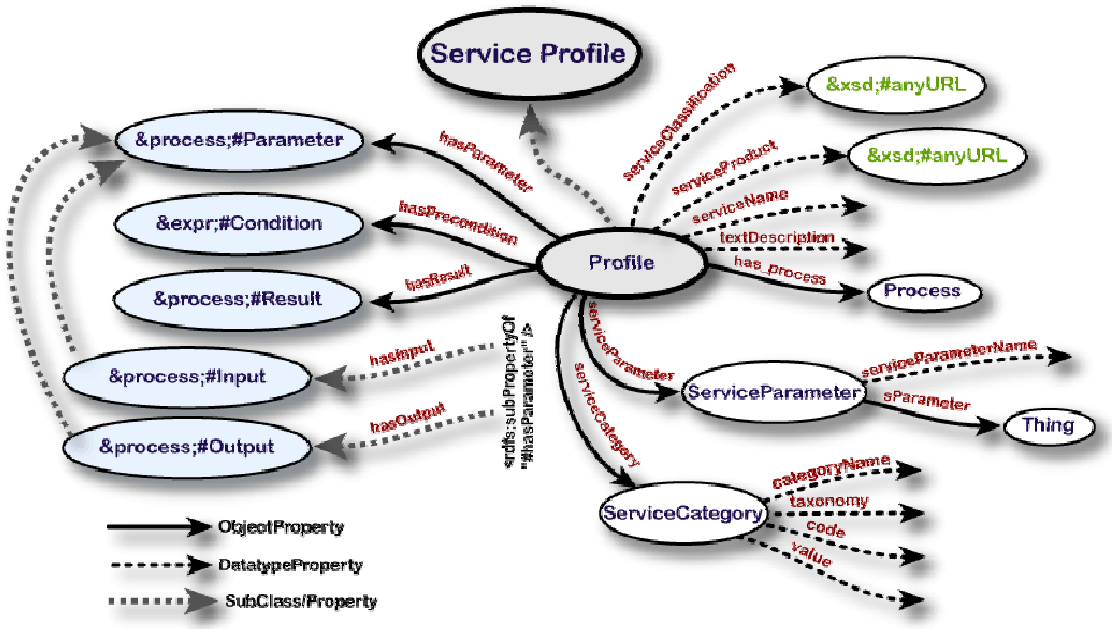

Figure 16 - Profile [Martin et al 2004a]

The Profile also specifies information about the transformations the service is capable of (represented by inputs and outputs) and information about the state changes produced by the execution of the service (represented by preconditions and effects). This last set of information is also known as IOPE (Inputs, Outputs, Preconditions and Effects). The OWL-S Profile does not provide a schema to describe IOPE instances.

That information is provided in the Process Model. Profile's IOPEs are expected to be a subset of those published by the Process, where the Process Model part of a description creates all the IOPE instances while the Profile instance simply points to these instances [Martin et al 2004a]. As shown in Figure 16, the Profile uses the *hasParameter*, *hasInput*, *hasOutput*, *hasPrecondition* and *hasResult* as properties for pointing to the IOPEs. The role of the *hasParameter* property is only making domain knowledge explicit (inputs and outputs are kinds of parameters) therefore, it is not expected to be instantiated. The properties *hasInput* and *hasOutput* range over instances of Inputs and Outputs as defined in the Process Model. The *hasPrecondition* property specifies one of the preconditions of the service and also ranges over a Precondition instance defined in the Process Model. At last, the property *hasResult* specifies one of the results of the service as defined by the Result class of the Process Model. It also describes the conditions under which the outputs are generated and the domain changes that are produced during the execution of the service [Martin et al 2004a]. The Profile information regarding IOPEs is used to generate the output description of the NLG system as it presents key information about web services: information needed to produce the service input (input), the information it produces (output), the conditions that need to be satisfied before executing the service (preconditions) and state changes produced by the execution of the service (effects).

Finally, the Profile may also contain additional information about quality guarantees provided by the service, possible classification of the service, and additional parameters that the service owner may want to specify. This is specified by the properties: *serviceParameter*, *serviceCategory*, *serviceClassification* and *serviceProduct*. This information can help the user decide whether or not to execute the service, mainly the information about the classification and quality of service. However, we do not present this information in the natural language description because it is not essential to the comprehension of the service and chiefly because this information is described in most cases outside OWL-S: categories of services are described on the bases of some classification that may be outside OWL-S and possibly outside OWL (this may require some specialized reasoner if any inference has to be done); and service classifications are instances of classes specified in OWL ontologies of services and products (OWL specification of NAICS - North American Industry Classification

System). Moreover, due to its complexity and time constraints, this information could only be added to the natural language description in future work.

```
<mind:LanguageServicerdf:ID="BabelFishTranslatorProfile">
    <service:presentedByrdf:resource="#BabelFishTranslatorService"/>
    <profile:serviceNamexml:lang="en">BabelFish Translator</profile:serviceName>
    <profile:textDescriptionxml:lang="en">
            Convert text from one language to another language. Supported languages are Dutch,
            English, French, German, Italian, Japanese, Korean, Portuguese, Spanish, and Russian.
            The valid input output pairs is given by the property canBeTranslatedTo.
    </profile:textDescription>
    <profile:hasInputrdf:resource="#InputString"/>
    <profile:hasInputrdf:resource="#InputLanguage"/>
    <profile:hasInputrdf:resource="#OutputLanguage"/>
    <profile:hasOutputrdf:resource="#OutputString"/>
    <profile:hasPreconditionrdf:resource="#SupportedLanguagePair"/>
</mind:LanguageService>
```

Figure 17 - Example of the Profile for the service *BabelFish Translator*

Figure 17 shows the Profile for the service "*BabelFish Translator*". From this example, we are able to know the name of the service (*BabelFish Translator*) and to have access to the text description created by the service owner. We can also learn that the service has two inputs (*InputString* and *InputLanguage*) and only one output, the *OutputString*. Moreover, the service can only properly execute if the precondition *SupportedLanguagePair* is satisfied. In the next subsection we take a closer look at the Process Model.

4.1.2 Process Model

The Process Model gives a detailed perspective on how to interact with a service. It details the possible steps required to execute a service. For each type of service (i.e., atomic or compound) there is a different kind of process. An atomic process expects one (possibly complex) message and returns one (possibly complex) message in response. A compound process maintains some state and each message the client sends advances it through the process [Martin et al 2004a]. We use the information about the type of process present in the Process Model to indentify the type of service in the output NLG description. That is, if the process described by the Process Model is an instance of an AtomicProcess the service is atomic; else if the described process is an instance of a

CompositeProcess it is a compound service. Atomic and compound services are specified in OWL-S as depicted in Figure 18.

Atomic Process:

<process:AtomicProcess rdf:ID="BookFinderProcess">

Compound Process:

<process:CompositeProcess rdf:about="FindCheaperBookProcess ">

Figure 18 – Atomic and Compound Processes

Processes can either produce a change in the world or create and return new information based on the received information and the world state. The latter is described by the inputs and outputs of the Process Model, while the former is described by the preconditions and effects. Classes and properties of the Process Model are presented in Figure 19.

In OWL-S, both preconditions and effects are represented as logical formulas. These expressions are treated as literals (string or XML). Since the Profile already presents information about the service's IOPEs, the Process Model's IOPE description is not used to generate the natural language service description. However, such type of information could be used to create more complex descriptions in future research. For example, thoroughly describing service variables specified in domain ontologies would allow presenting a more complete perspective on the service.

As described before there are two types of services: atomic and compound. However, OWL-S distinguishes between atomic, simple and compound processes. Simple and atomic processes have no sub processes and are executed in a single step, at least from the service requester's point of view. For each atomic process, a grounding must be provided. Simple processes differ from atomic ones in that they are not evocable and are not associated with grounding.

Contrary to atomic and simple processes, compound processes can be decomposed into other (non-compound or compound) processes. Control constructs help specify the decomposition and organization of compound processes. Example of control constructs include:

- Sequence - list of control constructs to be executed in a sequential order.
- Split - elements of a bag of processes to be executed concurrently. Completes as soon as all the processes have been scheduled for execution.
- If-Then-Else - if the specified condition is true, executes the *Then* part, else executes the *Else* part.

The Process Model provides information about both the control structure and data flow structure of the service. Although this is undoubtedly useful information to understand how the service works and interacts with the world we chose not to use this information in the natural language description. In fact, we only use Process Model's information to determine the type of service. These choices are explained in 4.2.

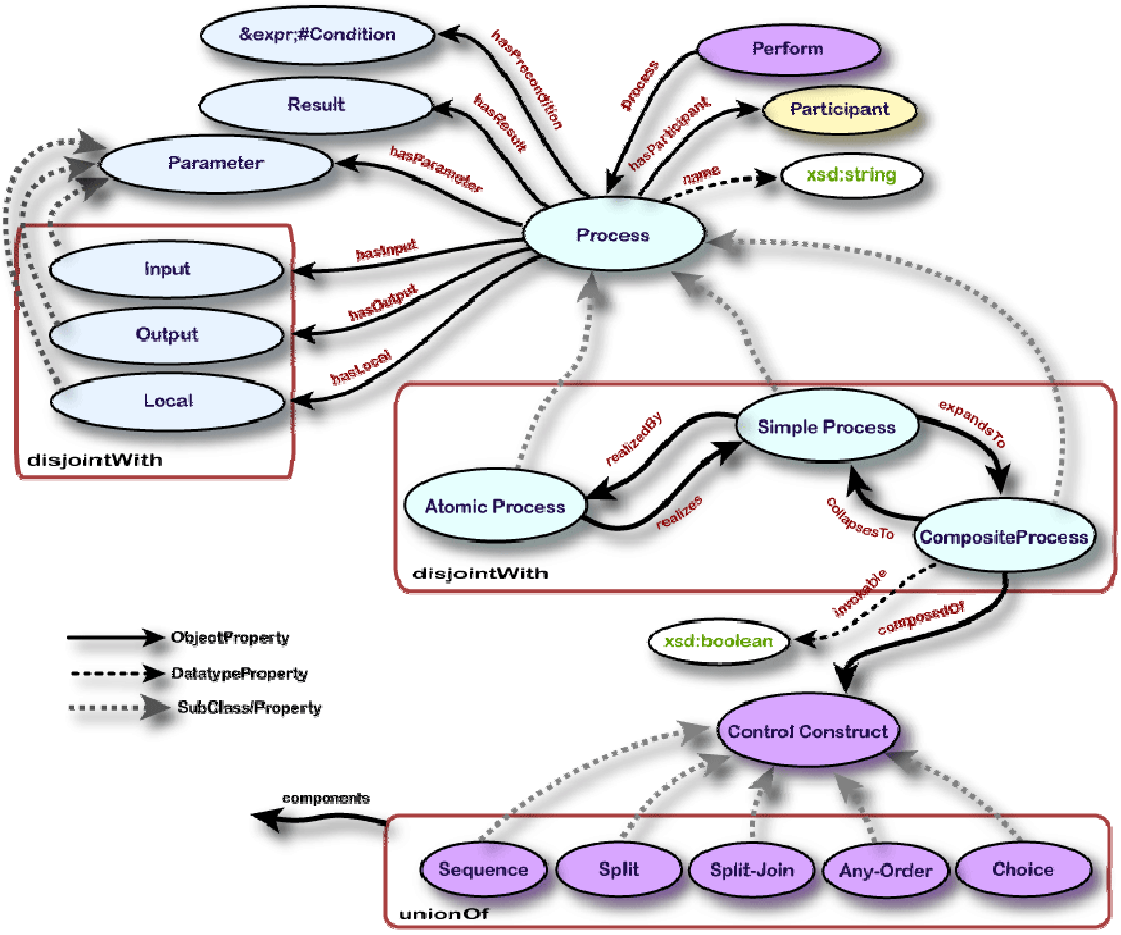

Figure 19 - Process Model class [Martin et al 2004a]

The service description is not complete if we do not have a way of evoking the service. In the next subsection we present the last element of OWL-S, which is responsible for specifying the way the service is accessed and evoked - the Grounding.

4.1.3 Grounding

The Grounding maps the description elements necessary for the interaction with the service from an abstract specification (Process Model) to a concrete specification. Specifically, the Grounding fundamental function is to show how the (abstract) inputs and outputs of an atomic process are to be realized as concrete messages [Martin et al 2004a]. However, the OWL-S language does not specify the details of particular message formats, protocols, and network addresses by which a web service is instantiated. For that reason, OWL-S uses WSDL (Web Service Description Language) to help ground OWL-S services. Hence, for specifying the Grounding, both languages are required as they do not cover the same conceptual space. This is depicted in Figure 20.

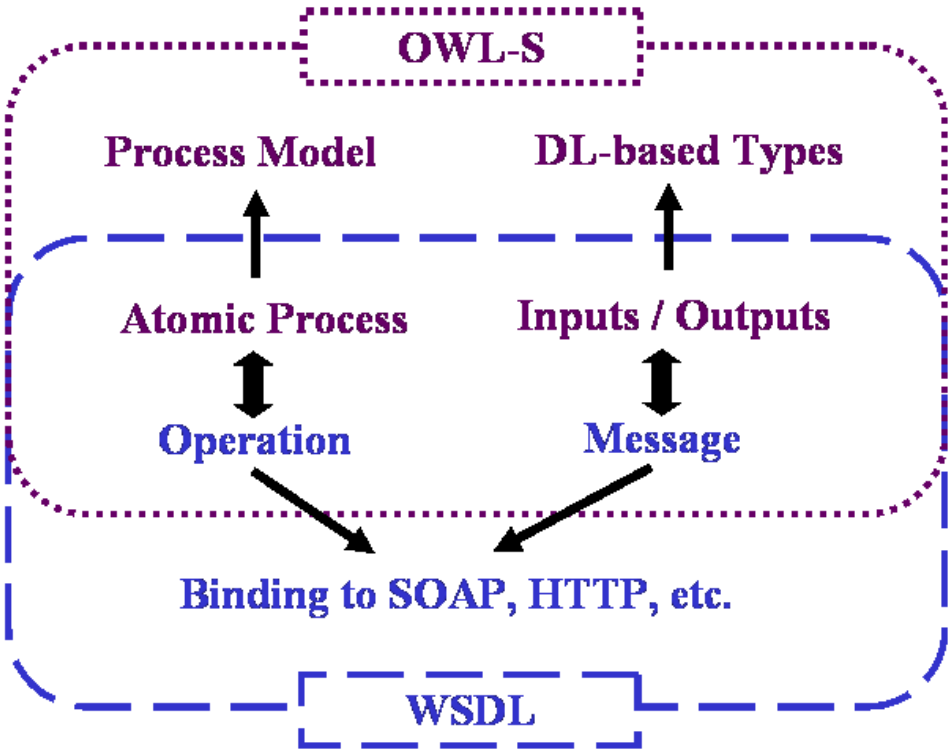

Figure 20 - OWL-S and WSDL [Martin et al 2004a]

An OWL-S/WSDL grounding uses OWL classes as the abstract types of message parts declared in WSDL, and then relies on WSDL binding constructs to specify the formatting of the messages [Martin et al 2004a]. In conclusion, the Grounding is based upon the correspondences between OWL-S and WSDL. We not use information from the Grounding to produce outcome description. The Grounding's main goal is to provide ways of realizing the inputs and outputs of an atomic process as concrete messages, but the Profile already possesses information about inputs and outputs, making it a useless source of information. Section 4.2 further elaborates on this issue.

4.2 Natural Language Generation Approach

In this section, we describe the template-based Natural Language Generation (NLG) system used in this dissertation. This system takes relevant information from the OWL-S service description and generates a natural language description of the specified service. Before describing the system itself, we first reason about the information to be present in the natural language description.

OWL-S descriptions are composed of Profile, Process Model and Grounding. The main source of information for the output description is the Profile, as it contains key information about the service (name, inputs and outputs, pre conditions and effects, among others). Although the Process Model and the Grounding parts of the OWL-S service description provide information about the service, they are not thoroughly used (except the Process Model to determine the type of service) in the natural language description, for different reasons. Some of the relevant information contained in the Process Model (service name, inputs, outputs, pre conditions and effects) is also present in the Profile. The part of the Process Model describing the ways to interact with the service, which is not contained in the Service Profile, could in fact be useful information for the user, since, to make an informed decision regarding the service, it might be convenient for him or her to know how the service interacts with the world. However, mainly due to time constraints, we have decided to focus exclusively on

declarative information regarding the service. Procedural information, as contained in the Process Model section of the description, will be handled in future research.

As a result of this choice, generated natural language descriptions do not explain how the service works. Instead, they provide a brief overview of what the service is and what it does. Although arguably incomplete, such declarative view of the service can help users deciding whether or not to use (execute) the service.

As for the Grounding, it specifies the details of how to access the service. Information about providers of a service would be useful and interesting to have in the natural language description, as it would allow people to know who the service providers are, and choose between them. However, we are not able to extract that type information from the Grounding, which makes it a not exploitable source of information.

In this approach, the process of generating natural language descriptions from OWL-S descriptions is a data driven algorithm. That is, it is the information extracted from the OWL-S service description that guides the subsequent processing of the generation. After having extracted a relevant piece of information from the OWL–S description, the system selects the appropriate templates guiding the content and the structure of the output natural language description.

We first present a brief summary of the various elements composing the NLG system, and further elaborate about each of these elements in the following subsections. The NLG system presented in this dissertation is a template-based system similar to a pipeline, where the output of one element is used as input for the next. The system is composed of four main elements as presented in Figure 21.

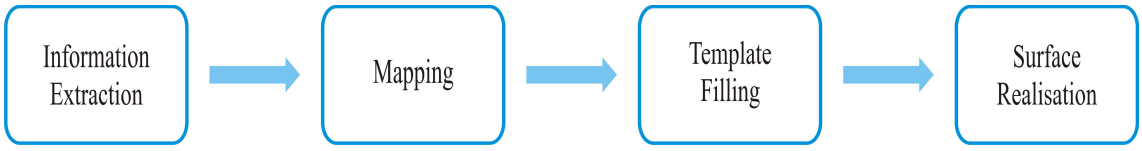

Figure 21 - NLG System Overview

The first element, Information Extraction, is responsible for dealing with the system's inputs. It extracts the parts of the OWL-S service description that will be present in the natural language description. Since it is possible that information extracted directly from the OWL-S description is not linguist information, the Mapping element maps the extracted information into meaningful text so that it can be used in the natural language description later on. At this point the system has all the information extracted from the OWL-S service description in a human readable form. Now it needs to take this information and use it to fill the text templates. This process is called Template Filling. In the fourth and last element of the system, Surface Realisation, the templates are processed so that the description is easier to read (presented as structured text with paragraphs and other phrase structures if necessary).

4.2.1 Information Extraction

The first element, Information Extraction, is responsible for extracting the information of the OWL-S service description to be present in the natural language description. As stated previously, the main source of information for the natural language description is the Profile, namely information about the service name, inputs, outputs, pre-conditions, results and the service description made by the programmer (if there is any). The Process Model is used only to determine whether the service is compound or atomic. Human-written OWL-S descriptions are not just pure non-linguistic inputs. They contain a large amount of complex linguistic material. Service creators label their variables freely – they may use meaningful names people will understand (*Fictitious Price Finder*, for example) but they may also label their variables in an arbitrary manner (e.g., service *FPF0709*, or *parameter x*). This freedom causes a problem when the system tries to use such possibly human readable labels to generate natural language outputs. The systems needs to make sure that the used labels are in fact human-comprehendible names. We try to achieve this by mapping "raw" information from the OWL-S description into information that can be understood by people. The mapping process is explained in further detail in subsection 4.2.2.

4.2.2 Mapping

The mapping process, in this context, can be seen as taking "raw" information the system extracts from the OWL-S service description and mapping it to the corresponding linguistic equivalent expressions. For instance, if the creator of the service named it *FPF0709* the linguistic equivalent could be *Fictitious Price Finder*. The mapping process is represented by the predicate *mapping/3* such that *mapping(Service,OwlsLabel,LinguisticExpression)* means that the specified OWL-S label (*OwlsLabel*) of the specified service (*Service*) is mapped into the corresponding specified linguistic expression (*LinguisticExpression*). The mapping of the previous example would be represented as *mapping(FPF0709,FPF0709,'Fictitious Price Finder')*. This avoids the problem of possible non-meaningful labels in the OWL-S description. However this requires a mapping to be made.

In this dissertation, we argue that such mappings should be provided by the service owner, attached to the OWL-S service description because, given the idiosyncratic nature of used labels, it is impossible or at least very difficult to create a mapping that could be used to all and every existing service description. However as it is of now, OWL-S descriptions do not provide support to integrate these mapping tables We propose adding specifications (in XML Schema) about linguistic mapping tables to the OWL-S descriptions. More precisely, integrate the mapping table in the OWL-S Profile since this type of information could also be used to describe the service. Each mapping table could be integrated in the service Profile in a similar way as the following example:

```
<mappingtable target_language="English">
   <mapping_pair>
      <original_label>//mind:BookInformationServiceBook[@rdf:ID='BookFinde
      rProfile']/profile:serviceName[@xml:lang='en']
      </original_label>
      <linguistic_equivalent>Book Finder</linguistic_equivalent>
   </mapping_pair>
   <mapping_pair>
      <original_label>//mind:BookInformationServiceBook[@rdf:ID='BookFinde
      rProfile']/profile:hasInput[@rdf:resource='#BookName']
      </original_label>
      <linguistic_equivalent>name of the book</linguistic_equivalent>
   </mapping_pair>
   <mapping_pair>
      <original_label>//mind:BookInformationServiceBook[@rdf:ID='BookFinde
      rProfile']/profile:hasOutput[@rdf:resource='#BookInfo]
      </original_label>
      <linguistic_equivalent>book information</linguistic_equivalent>
   </mapping_pair>
</mapping_table>
```

Figure 22 – Example of the mapping table for the *Book Finder Service*

Mapping tables as the one in Figure 22 would be written according to the following XML Schema:

```
<xs:element name="mapping_table">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="mapping_pair" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="original_lable" type="xpath"/>
            <xs:element name="linguistic_equivalent" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    <xs:attribute name="target_language" type="xs:string" use="required"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 23 – XML Schema for the mapping tables

If the natural language description were to be generated in multiple languages, multiple mappings would also have to be created. This however does not seem to be a problem. As a matter of fact most users would understand English descriptions. In any case, a single mapping is better than no mapping at all and the fact that, in a given moment a given service description is associated to only a single mapping does not prevent new mappings, for new languages, to be incrementally added.

Multiple mapping tables could be included in the OWL-S Profile using the *hasLinguistMappings* property as presented in Figure 24.

```
<owl:ObjectProperty  rdf:ID="hasLinguisticMapping"/>
        <rdfs:domain rdf:resource="#Profile"/>
        <rdfs:range rdf:resource="#LinguisticMappings"/>
</owl:ObjectProperty>
```

Figure 24 – The property *hasLinguisticMappings'* specification in the OWL-S Profile

The value of the property *hasLingisticMappings* is defined by the range *LinguisticMappings*. *LinguisticMappings* needs to have one or more mapping tables as specified in the XML Schema of Figure 25.

```
<xs:element name="LinguisticMappings">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="mapping_table" minOccurs="1" maxOccurs="unbounded">
    </xs:sequence>
  </xs:complexType>
 </xs:element>
```

Figure 25 – XML Schema for the *LinguisticMappings*

Due to the OWL-S' restrictions described above, we have built-in the mapping tables within this system. This allowed us to properly test the NLG system. Since all the information is now in a human understandable form, the system needs to fill the text templates with this information. This process is called Template Filling and will be explained in further detail in subsection 4.2.3.

4.2.3 Template Filling

Template-based systems have fixed linguistic structures typically containing gaps that need to be filled by the system with information extracted from or adapted out of its inputs. In the system described in this dissertation, the templates are filled with the result of the Mapping process and are represented by the predicate *template/3*. The first parameter of this predicate is the name of the OWL-S variable, to which the template is assigned (e.g. service name, input). The second parameter specifies the cases of that variable – there are three possible cases for OWL-S variables: they have no value, have only one value or have several values (e.g. the service may have no inputs, only one input or various inputs). The third parameter is the actual text template that is appropriate for the situation specified by the first two parameters. For the moment, we will focus on the third parameter, the text templates. Text templates are fixed linguistic structures that require no further orthographic or grammatical treatment other than some

structural processing (paragraphs, for example). In Figure 26 we present some examples of the text templates that can be used in the system.

1. `[' needs no parameters.']`
2. `[' needs to receive the ', Input, ' as parameter.']`

Figure 26 - Text Templates

The first text template (1) is applied when there are no inputs to the OWL-S service description, so there is no need to fill the template with any information. The second text template (2) represents the case where there is only one input to the service. This template must be filled with the name of the single input parameter, which will be inserted where specified by the variable *Input* (underlined in the figure just for improving readability). Text parts of the template and variables are separated with commas.

In Figure 27 we present two examples of the *template/3* predicate for the case of the service name variable. Templates, for which the first argument is *service name* can only be used for the service name variable of the OWL-S service description. The use of each template is determined, in the first place, by its first argument. For instance, the first parameter of the templates representing the service inputs or pre-conditions would be *service inputs* and *preconditions* respectively.

1. `template (service name, [], ['This service']).`
2. `template (service name, [Service name], [ 'The service ', Service name]).`

Figure 27 - Templates for the service name

Supposing the service has no inputs, the system cannot use the template for when there is one or more inputs to the service. This issue is common to all the variables. Being so, the second parameter of the predicate *template/3* specifies the particular situation of the variable represented in that template. Figure 27 shows two templates used to represent two possible situations regarding the service name part of

the service description: (1) the case where the name of the service is not specified; and (2) the case where the name of the service is specified. In the first situation, the second parameter is an empty list meaning that there is no specified service name, whereas in the second situation the second parameter is a list with one element, the *Service name*, meaning that a service name is specified. In this representation, *Service name* is the name of the variable, not its value. The third parameter of the predicate *template/3* represents the actual text template, as seen previously and in Figure 26.

In 4.2.2, we acknowledged that if we needed to generate the description in multiple languages, different mappings would have to be provided by the service owner within the service description. Since it is possible to create a set of text templates covering the whole OWL-S description, used templates do not have to be provided by service owners. Instead, they are represented in the natural language generation system's knowledge base. Because text templates are in a final static form, no further grammatical or orthographic processing is required, for every new language a new set of text templates has to be created and loaded into the system's knowledge base. Moreover there is other important difference between mappings for formal description labels and text templates. The text templates are specific of a given service description language, in this case, the OWL-S. This means that the same set of templates, developed for OWL-S descriptions, may be used for every service described in OWL-S. Mappings, contrarily to text templates, are specific of each service. Therefore text templates may be created, at once, for all OWL-S service descriptions; whereas mappings must be provided for each description.

Subsection 4.2.4 describes the last element of the Natural Language System, the Surface Realiser.

4.2.4 Surface Realiser

The outcome of the Template Filling system element is a close approximation to the final natural language description, with the filled text templates producing a meaningful description. In this last element of the NLG system, the filled templates are subject to the process of Surface Realisation, where they are mapped from an abstract text representation to surface text, made up of sequences of words, punctuation symbols

and text structures. Because the text present in the templates already encompasses words and punctuation symbols (Linguistic Realisation), the task of the Surface Realiser is restricted to adding paragraphs to the description. In this NLG system, paragraphs are represented in the templates by a unique special symbol (i.e., "*/nl*"). If we wanted to add a paragraph at the end of the sentence "*This service needs no parameters.*", the resulting template would be [*' This service needs no parameters./nl'*] . In this example, the special symbol "*/nl*" indicates that a new line needs to be added by the Surface Realiser.

In subsection 4.2.5 we present a general schema of the whole process of creating a natural language description using this kind of NLG System.

4.2.5 General Schema

In Figure 28, the system begins by selecting and extracting information from the OWL-S service description *FPF0709*, which is to be present in the natural language description. This information is then mapped to meaningful linguistic equivalents. In this particular example, the name of the service *FPF0709* is mapped to *Fictitious Price Finder*. This same mapping process is applied for the rest of the information extracted from the OWL-S description. Subsequently, the already mapped information is used to fill the templates and, according to its properties, the right templates are chosen. For example, this service has no parameters, hence the chosen text template is [*' This service needs no parameters. '*]. After filling the chosen templates with all the information, the system is now ready for the Surface Realisation. After being mapped by the Surface Realiser from an abstract text representation to a surface text format, the final Natural Language Description is created.

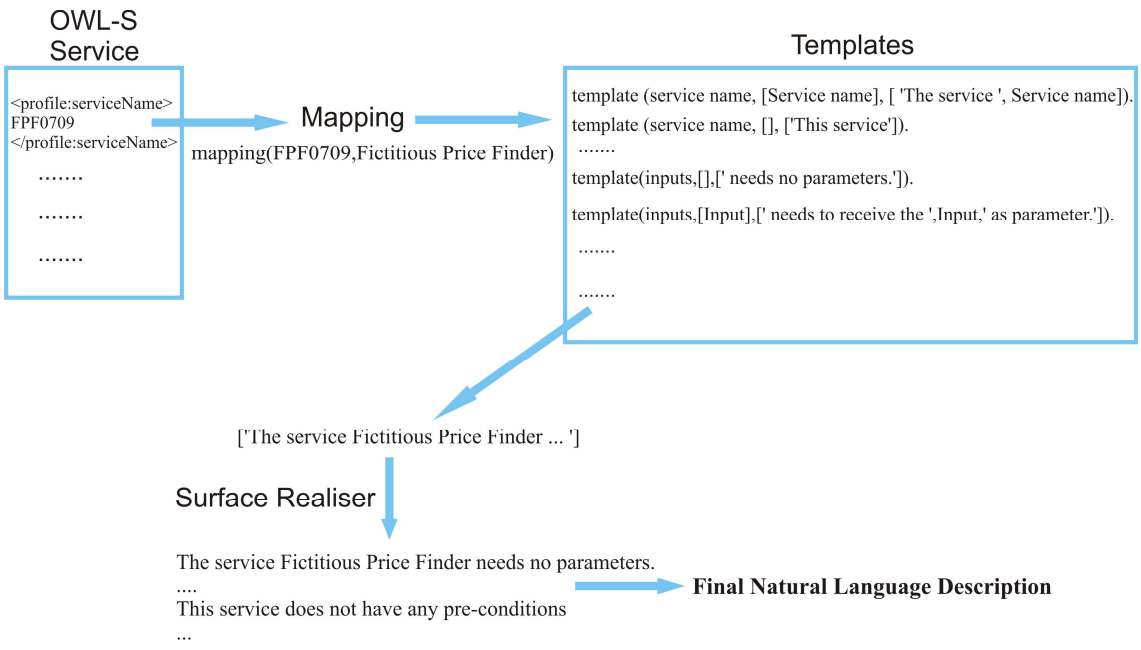

Figure 28 - NLG System Schema

In the next section we detail the implementation of the natural language system. More precisely, we describe the choices taken in the implementation of the service such as the software and programming languages we used as well as some the details intrinsic to the implementation.

4.3 Natural Language Generation System Implementation

In this section, we describe the implementation of the natural language generation system and all their various modules.

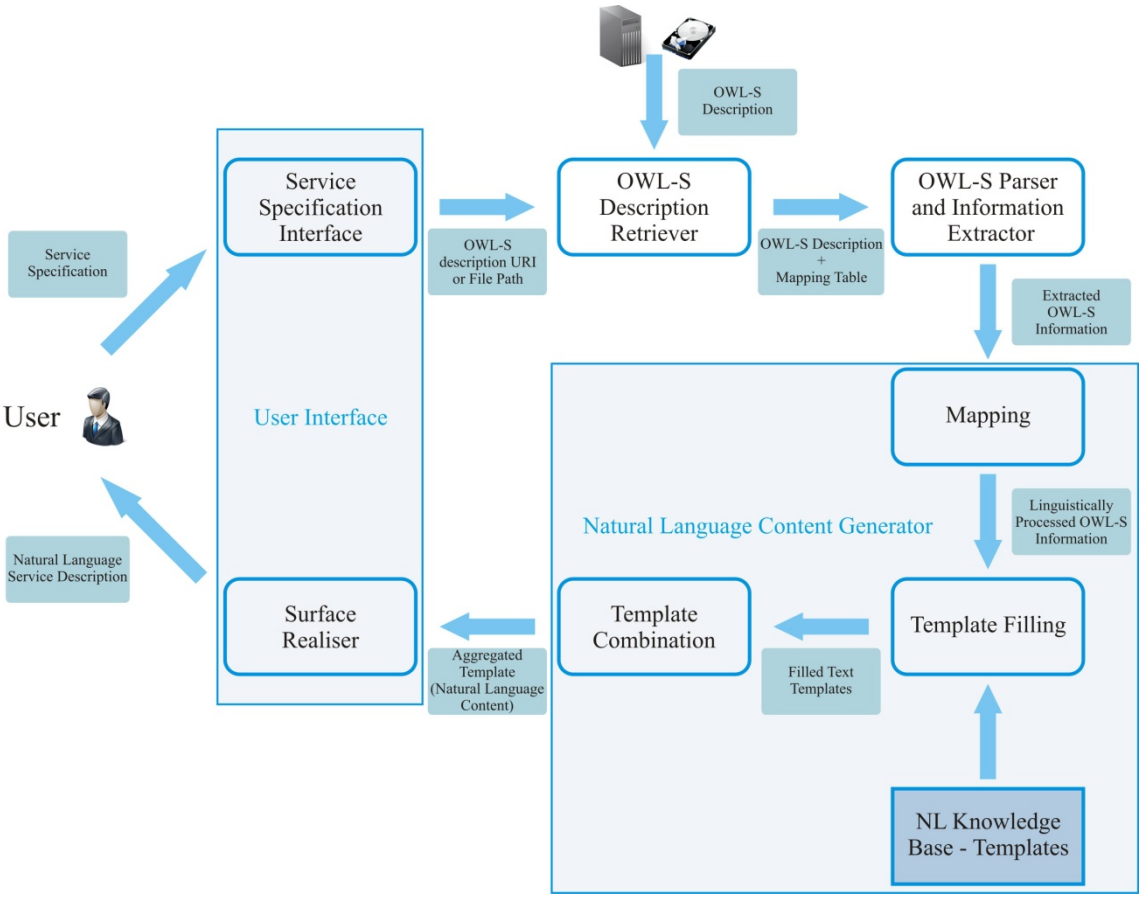

Figure 29 – NLG system's implementation schema

Users interact with the system through a User Interface, as depicted in Figure 29. The User Interface is composed of two modules: Service Specification Interface and Surface Realiser. Users specify the services whose formal descriptions are to be translated to natural language in the Service Specification Interface. The specification may be either the URI of the formal service description or its pathname in case the description is stored on the computer disk. The NLG system uses this information to retrieve the service description (OWL-S Description Retriever) and extract the information that it will later use to generate the natural description (OWL-S Parser and Information Extractor). The Natural Content Generator part of the implementation - Mapping, Template Filling and Template Combination - uses the extracted information to create the natural language content. The Surface Realiser module, part of the User Interface, structures the generated NL content and presents it back to the user.

In the following sections we describe in detail each of the modules composing the natural language generation system.

4.3.1 Service Specification Interface

The Service Specification Interface module, which is part of the User Interface, allows users to select the service they want to be translated. It was implemented in Java because this language possesses various tools that help building user interfaces, such as Swing.

Users have two options: specify the service via URI, or select a service saved in their file system. This module's output is either the service's URI address or the file path of the disk file containing the service formal description, that is, the OWL-S description.

4.3.2 OWL-S Description Retriever

While users specify the service's URI address or file path in the Service Specification Interface module, the OWL-S Description Retriever uses that information to retrieve the service's OWL-S description, either from the specified server (in the case of the URI), or from the system's disk (in the case of the file pathname). From this point on user intervention is no longer required.

If the user specifies the service's URI, the system retrieves the OWL-S description from the web server where the description is hosted. If the user chooses a service from her or his own file system, the system converts the *file path* to an URI and proceeds the same way as if a URI had been provided in the first place. We picked this approach since the alternative would be to install a web server in the user's computer, which would also require that all the information associated with the service (e.g. ontologies that describe inputs or outputs) would be accessible to the local web server. With the selected approach it is required to have an active Internet connection but the whole process becomes much simpler to the user.

We use the OWL-S API to retrieve the OWL-S description. OWL-S API is a Java API for programmatic access to read, execute and write OWL-S service

descriptions, developed by the Mindswap Project of the Maryland University (Maryland Information and Network Dynamics Lab Semantic Web Agents Project)[3]. OWL-S API's *OWLKnowledgeBase.readService (URI)* method retrieves the OWL-S description receiving its *URI* as parameter. The outcome of this module would ideally be the retrieved OWL-S description and the attached linguistic mapping table as depicted in Figure 29. But as described in 4.2.2, OWL-S service descriptions do not provide means to integrate a linguistic mapping table. If these tables were to be specified within OWL-S descriptions, the OWL-S API, used in this dissertation to parse the description into Java objects, would have to be modified. In the present system the mapping tables are produced by ourselves and coded into the program in order to test the system. The OWL-S Description Retriever is also implemented in Java.

4.3.3 OWL-S Parser and Information Extractor

Once the OWL-S description is retrieved the OWL-S Parser and Information Extractor module parses the OWL-S service description also with the help of the OWL-S API. With this API, we are able to read OWL-S service descriptions and convert them into java objects. This is the reason this module is also implemented in Java. Using the OWL-S API we can create an instance of *Service* class, from which we can access other java classes such as the *Profile*. The relevant information for the natural language description to be generated is taken from instances of the Profile class, calling methods such as *Profile.getServiceName()*, *Profile.getInputs()* or *Profile.getOutputs()*. The former returns a string containing the service name, while the other two methods return the input and output lists respectively. The result of this module is the relevant information extracted from the service description, which is composed of several OWL-S variable values.

4.3.4 Mapping

This module uses the linguistic mapping table to convert the encoded names of the information extracted by the OWL-S Parser and Information Extractor module into

[3] http://www.mindswap.org/2004/owl-s/api/

their linguistic equivalents, which we hope, will be easier to understand. Contrary to the previous modules, the Mapping module is implemented in Prolog. Prolog ability to represent relationships makes it a good choice for representing and processing the required linguistic mapping table. Java could as well be used but, since the template filling process is also implemented in Prolog, we chose to use Prolog for the mapping process. Specifically, we used the tuProlog as our Prolog engine Java implementation because, since it is written in Java, it is easily integrated with the system's Java blocks. The linguistically processed information will be the Mapping module output.

4.3.5 Template Filling

The Template Filling module is responsible for filling the adequate templates with the information generated by the Mapping module. The templates used in this task are stored in the system's natural language knowledge base (KB), which is represented in Prolog. Similarly to the previous module and the knowledge base, the Template Filling part of the system is also implemented in Prolog. Since Prolog is a declarative language with powerful pattern matching capabilities, we decided to use it to actively find and fill the correct templates, which is the goal of this stage of the generation process. The outcome of Template Filling module is a sequence of filled text templates.

4.3.6 Template Combination

Receiving the filled text templates as argument, the Template Combination's task is to aggregate all these text templates into a single one. Although lacking some processing, it already contains the entire information to be present in the NL description. Template Combination is the last module where Prolog language is used because it would not be easy to pass the several templates, represented as Prolog lists, to the Java modules of the Program. The output of this module is a single text template resulting of appending all the filled templates, one after the other (aggregate template, in the figure).

4.3.7 Surface Realiser

The last module of this system, the Surface Realiser, receives the natural language content from the Template Combination module and applies structural processing in order to present the NL description in a suitable way to the user. Structural processing comprises the addition of paragraphs. This module then presents the final natural language to the user requesting the translation. The Surface Realiser is implemented in Java for the same reasons described in 4.3.1.

In summary, the natural language generation system developed in this dissertation was implemented in two different programming languages: Java and Prolog. Java was used for interacting with the user, retrieving the specified OWL-S descriptions and extracting the relevant information from them; for the structural processing; and for presenting the NL description text. Prolog was used for mapping the OWL-S variables to meaningful linguistic text; and for representing and processing the used natural language templates, filling them with the information.

5 Evaluation Criteria and Results

The natural language generation system described in this dissertation was created with the goal of providing means by which people can understand service descriptions without knowing the formal language they are described in. As presented in section 1.4, one of the ways we can measure this system's success is by questioning potential users of the NLG system. Hence, we have designed an inquiry for users without computer science background, addressing their comprehension of the services from the generated natural language descriptions. Since this inquiry does not allow us to evaluate the correctness of the generated natural language description, we also created an inquiry to OWL-S experts, asking them to rate the degree to which the generated natural description correctly matches the formal service description. In the remaining of this chapter, we describe both inquiries, analyse the results, and show how the responses from the inquired OWL-S experts contributed to improve the system.

5.1 Experts' Inquiry

The experts' inquiry was created to evaluate the correctness of the generated natural language description, that is, the degree to which it matches the OWL-S description, and also to use the expert's feedback to improve the system so that better (more correct) descriptions can be generated. The experts' inquiry was applied to a panel of five OWL-S experts. For this inquiry we used seven atomic and two compound service OWL-S descriptions from the OWL-S service examples publicly available on the site of the Maryland Information and Network Dynamics Lab Semantic Web Agents Project (Mindswap). The same OWL-S descriptions are used in the user inquiry. Figure 30 depicts an example of the expert inquiry for the atomic service *Zip Code Finder*.

Profile:

```
<!-- Profile description -->
<mind:MapService rdf:ID="ZipCodeFinderProfile">
        <service:presentedBy rdf:resource="#ZipCodeFinderService"/>
        <profile:serviceName xml:lang="en">Find ZipCode</profile:serviceName>
        <profile:textDescription xml:lang="en">
        Returns the zip code for the given city/state. If there are more than one zip  codes
        associated with that city only the first one is returned.
        </profile:textDescription>
        <profile:hasInput rdf:resource="#City"/>
        <profile:hasInput rdf:resource="#State"/>
        <profile:hasOutput rdf:resource="#ZipCode"/>
</mind:MapService>
```

Natural Language Description:

```
The service Zip Code Finder is not a compound service.
This service needs to receive the city and state as
parameters. It has only the zip code as output.
This service does not have any pre-conditions.
Also, the owner of the service describes it as: "Returns
the zip code for the given city/state. If there are more
than one zip codes associated with that city only the first
one is returned".
```

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?
2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?
3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?
4. Commentary (facultative)

Figure 30 - Expert inquiry's part for the service *Zip Code Finder*

To allow the evaluation of the system's correctness, the expert inquiry presents the OWL-S service Profile along with the generated natural language description for each service. We included only the Profile part of the OWL-S description because the system does not make use of either the Process Model, apart from determining the type of service, or the Grounding to generate the NL description. The inquiry also possesses non-mandatory fields to comment each service description. In its last page, the inquiry

contains a non-mandatory field prompting experts for general comments and suggestions about the global results.

The results obtained from this inquiry provide helpful and expert feedback about the generated descriptions. The first question tests the correctness of the description generated by the system. Positive results ensure that the NL description matches the corresponding OWL-S description. Questions two and three address the degree to which the descriptions can be understood and their usefulness. These questions are similar to those in the user inquiry. Finally, comment and suggestion fields may offer a valuable source of information, contributing with ideas and ways to improve the English description and possibly the whole system. In the following we present the results for the experts' inquiry.

5.2 Experts Inquiry Outcome

Since the evaluation panel for the experts inquiry was composed of only five OWL-S experts, a qualitative analysis of the inquiry outcome is more adequate then a quantitative one. Nonetheless, and only for the sake of completeness, we also show the quantitative results. The first question had the best results from all the questions with the average of 3.97 (in an integer scale from 1 to 4). This implies that OWL-S experts consider the natural language description to be very accurate, having a high degree of correctness. That is, the experts found the generated descriptions to accurately match the Profiles section of the OWL-S descriptions. The second question, regarding the comprehension of the English generated description, also attained good results with the average of 3.75 from the nine service descriptions. In this case, experts consider that the generated description allows the user to understand the basics and essentials of the service. Finally, the third question has the lowest average of the three - 3.68. It is still a very good result and shows that the natural language description can be very useful to help users decide whether or not to execute the service.

One of the OWL-S experts' main concerns was about the level of detail in the description. They suggested more depth in the description of inputs and outputs, more specifically detail about the variables described in the service domain ontologies. Other issue appointed by experts was the relevance and technical terms in which we presented

the information about the type of service. The natural language description began by presenting the name of the service and whether it was atomic or compound. This choice (presenting the service type – atomic or compound – upfront) exaggerated the relevance of this information in the description's context. However, this is not a very important piece of information. For example, information about inputs and outputs plays a more important role in the understanding of the service description. Another concern of the OWL-S specialists was the fact that the service being atomic or compound is not understandable to users without any background in this area. Some of these suggestions led to system modifications described next.

5.3 System Improvements

We applied the experts' inquiry before the users inquiry, which allowed improving the system's output before the users inquiry was applied. Instead of presenting the type of service in the first paragraph we provide this information only in the last paragraph, decreasing its importance in the description's context. We chose to do so because this information is mainly informative about the service, not providing key elements to service understanding as the ones in the previous paragraphs.

1. `Technically, this is a compound service because it combines other services in a new aggregate service`

2. `Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service.`

Figure 31 - Text used to describe atomic and compound services

Other problem about the natural language system output was the use of technical terms not appropriate to common users, namely in the type of service (atomic or compound) related information. To solve this issue we added a little description together with that information. Figure 31 depicts the new text description used to describe atomic and compound services. These changes were applied in the user inquiry presented in the next section.

Some of the experts also suggested more depth in the description of inputs and outputs described in the service domain ontologies. Even though this was not part of this dissertation's goal, we tried to extract information about inputs and outputs from the OWL ontology in which they were described in. This proved to be an arduous and not straightforward. In fact, similar work is already being done [Mellish and Sun 2006], and as described by the authors of this project, this task can be very complex as information about a concept, in our case inputs and outputs, usually cannot be presented in a single sentence but requires an extended text with multiple sentences where the overall structure has to be planned so as to be coherent as a discourse. Moreover, the same concept may be described in many different ways. To produce elegant and easily understandable natural language descriptions of these concepts, a solution similar to the one proposed as extension to OWL-S (use of mappings) would have to be created. Additionally, we would also have to develop a set of text templates, this time for OWL ontologies.

Nonetheless we were able to obtain some results, although some of the OWL constructs were not yet implemented (e.g., intersectionOf, unionOf, complementOf constructs).

This problem appears in some of the OWL-S service descriptions with *book information* as input or output. Book information is described in the domain ontology in which the OWL-S description is based. This ontology is written in OWL. We have developed a computational procedure that was able to generate the description depicted in Figure 32.

1. Book information has at least 1 Publisher, at least 1 Title, at least 1 Year, at least 1 humanCreator and is subclass of Entry.

Figure 32 – Description of *book information*

From this description we can understand what composes the *book information*, but it also presents unnecessary information that can, in fact, confuse the user. For example stating that the *book information* is a subclass of *Entry* (*Entry* is a base class for all entries in that ontology) is not natural or comprehendible to users. We would

have to be able to filter some of the information contained in the OWL description. However, in some other cases, the class / super-class relationship might indeed be very useful. The decision of whether or not to discard some of the information contained in the OWL ontology would have to involve thorough context aware computation which falls off this dissertation objectives.

Since the information we were able of extracting from the OWL ontology should not be presented to users as is, and since the generation of adequate English descriptions from OWL class descriptions would involve at least another MSc dissertation (templates for OWL ontologies, linguistic mappings for OWL ontologies, use of context to filter some useless information present in the OWL ontology), we chose not to present this information in the final description. However, information generated about inputs and outputs from OWL can be used to complement the OWL-S description presented in this dissertation in future work: either by creating a new more complete system (e.g., capable of generating English descriptions from OWL ontologies) or by integrating the current NLG system with work being done in this area [Mellish and Sun 2006].

5.4 Users' Inquiry

The inquiry for users without computer science background focuses on their understanding of the information provided by the generated natural language service descriptions. It comprises various natural language descriptions generated from OWL-S service descriptions. The users' inquiry was answered by twenty two non-expert users.

For each service, we asked users to rate the NL description, in a scale from one (bad) to four (good) regarding the degree to which they understand the description and the degree to which the description would be useful for them to decide whether or not to use it:

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet,

would the information be useful to decide whether or not to use the described service?

From this inquiry we can attest the quality and usefulness of the generated natural language description. Good (bad) results in the first question show that the generated description and, by extension the NLG system, present good (bad) quality. The second question, on the other hand, addresses the usefulness of both the system and the descriptions.

If the results of this inquiry are positive it means the system achieved its goal, that is, it empowers users with the capability to better understand and decide about discovered or composed services independently from the formal service description language used to describe them. These questions also offer useful feedback about the descriptions. For instance, if users cannot understand the service NL description, it probably indicates that there is either not enough information about the service in the description or the English used in the NL description is not adequate for non-specialized users. In summary, this information can be used to further improve the system.

5.5 Users' Inquiry Outcome

To evaluate the system from the users' point of view, we applied this inquiry to twenty two people. This number of people, despite not being a large-scale sample, can already provide us with some practical feedback. The user inquiry was composed of two questions. Results to the first question, average of 3.51 from nine OWL-S descriptions, indicate that the majority is able to easily understand each of the described services. The second question prompted users about the usefulness of the generated description in order to decide whether or not to use it. This question obtained slightly worst results compared to the first question with an average of 3.44. These results provide evidence that the English description and NLG system are undoubtedly useful to help people decide about using the service.

We must, however, take into account that the users sample is not extensive or people may not be entirely satisfied with the service descriptions although they understand them. People we inquired, although not many, evaluated some generated

descriptions poorly. This situation can be caused by either the lack of information in OWL-S descriptions, or by system deficiencies in the generation of NL descriptions. We only have control of the latter as OWL-S descriptions are the responsibility of the service owner. Unfortunately to us, it is impossible to know at this stage the exact reasons behind this slight discontent. This is mainly because the inquiry did not comprise questions requiring users to justify their choices. Also, the time span necessary to realize a new inquiry would be far too great in order to incorporate the results in this dissertation.

Nevertheless, if the NLG system was really the cause for this minor discontent it means that there is still room for improvement. Users (all of them are Portuguese) might be a little puzzled to find out that the descriptions were written in English. They may also consider them too technical. Comprehension could be made easier if we added different output description languages and allowed the user to choose the one they were most comfortable with. Other reason for the slight discomfort may be the generation of descriptions that lack some information that could be useful for better comprehension. But these are all hypotheses. Users may already be very happy with the descriptions (proved by the average rating of both questions). And there might be even different reasons from the ones advanced in this subsection to why some NL service descriptions were classified below average.

In conclusion, we evaluated the NLG system through its outcome, the natural language descriptions. With the two inquiries, one for potential users and one for OWL-S experts, we had a more complete and accurate evaluation of the system. On one side experts with vast knowledge of OWL-S provided exact and pertinent criticism that allowed us to further improve the system and its outcome. On the other side users, the system target-public, stated their ratings of the description. The results from both inquiries were very satisfactory, attesting the system's usefulness in understanding the service and deciding about its execution.

6 Conclusions

The main objective of this dissertation was enhancing the service coordination process. We set out to develop a natural language generation system that empowers users of service coordination systems with the capability of better understanding and deciding whether or not to use discovered or composed services without the need of understanding the formal language (i.e., OWL-S) in which the semantic web service is described.

We opted to create a template-based NLG system because the general form of sentences in the generated natural language descriptions is invariant enough to be represented as templates. Moreover, if we used a more complex natural language generation approach, besides requiring a lot more time and effort, it would not bring significant benefits compared to the template-based approach. This system receives an OWL-S service description as input and produces the corresponding English description. The natural language generation process uses mainly the Profile of the OWL-S service description since it contains key information about the service - service name, inputs, outputs, preconditions, effects and description provided by the service owner. It also uses the Process Model to determine the type of service. The generated natural language descriptions are written in the English because it is the most spoken and dominant international language in areas such as communications, science and business. By providing descriptions in English common citizens (without computer science background) are able to better understand the service and be made responsible for their decision of whether or not to use the service.

Besides templates, the used technical approach requires linguistic mapping tables that are used to provide linguist expressions for each OWL-S variable used to generate the English description. This mapping allows the generated descriptions to be written in more human form. Once we have chosen to generate the natural descriptions in English, the text templates can be applied to all OWL-S descriptions. However, each OWL-S description uses its own variables with specific names. Therefore, there must be a linguistic mapping table for each service. One of the main contributions of this dissertation was to propose an extension of the OWL-S formalism with the

representation of linguistic mapping tables, one for each desired language. At this time OWL-S descriptions do not include such type of tables, for that reason they have been built-in within this system. We think the addition of these linguistic mapping tables within OWL-S descriptions could prove to be an advantage for future work in this area, that is, the generation of natural language from semantic web material.

Other of this dissertation's contribution is a proposal regarding the possible integration of the NLG system in the service coordination process. More precisely, personal agents (PA) would use this system to generate natural language descriptions from the formal descriptions resultant from the service coordination process, and present them to their users.

In order to evaluate the NLG system we created two inquiries: one addressed OWL-S experts and other for potential users of the system. The experts' inquiry addressed the degree to which the OWL-S experts understood the descriptions, the degree to which the generated description is useful to inform the decision of whether or not to use the service, as well as the degree to which the generated natural description correctly matches the formal service description - correctness. The experts' inquiry provided evidence that the generated natural language descriptions not only accurately matched the OWL-S descriptions, but also allowed to understand the described services and helped users decide whether or not to use the service. It also provided very useful feedback that allowed us to further improve the system. More precisely, the information about the type of service (atomic or compound) was presented in technical terms not appropriate to common users and was given too much relevance. We opted to add a little description together with that information to facilitate users' comprehension as well as provide this information only in the last paragraph, decreasing its importance in the description's context. These modifications were applied before the users' inquiry.

The users' inquiry addressed the users understanding of the service from the generated description and whether it would be useful to help them decide to use or not to use the service. Results from this inquiry were positive. In a scale from one (bad) to four (good), the first question scored an average of 3.51 showing that people can easily understand each of the described services. The second question with an average score of

3.44 proved that the generated description can be used in order to decide whether or not to use the service. The outcome of both inquiries provided evidence that the English description, and by extension the NLG system, fulfilled the goals of this dissertation: potentiate the use of service coordination technology by common citizens (not expert in formal computational descriptions), endowing them with the possibility to consciously decide whether or not to execute a given service and making them accountable for the decision of using given services.

Despite the good results achieved in the system's evaluation, there are various ways to further improve it and to carry out future work. Present generated descriptions only describe the inputs and outputs of the OWL-S service but do not specify their types. For example, some of the OWL-S service descriptions used to test the system received the variable *book information* as input. From this label we cannot understand what the *book information* is. It might be the book name, the ISBN, both or something else. We created a computational procedure that was able to generate descriptions from OWL concepts such as the *book information*, but as described in section 5.3 the outcome of this procedure presented some problems and would involve at least another MSc dissertation (templates for OWL ontologies, linguistic mappings for OWL ontologies, use of context to filter some useless information present in the OWL ontology), so we chose not to present this information in the final description. Providing more detail about inputs and outputs could be one additional improvement to the present system in future work. This could be achieved either by creating a new more complete system (e.g., capable of generating English descriptions from OWL ontologies) or by integrating the current NLG system with work being done in this area [Mellish and Sun 2006]. Information about service quality guarantees and classification could also be present in a future description. Natural language descriptions were generated from information mainly extracted from the OWL-S service Profile, except the service type (atomic or compound) that was determined from the OWL-S Process Model. Future work may focus on the procedural part of OWL-S' Process Model, providing information about how the service interacts with the world as well as presenting a view of the service's workflow. The latter is useful mainly to compound services which make use of other services. Finally, the NLG system could also generate descriptions in

different languages and even make use of context information to choose the language of the output description.

7 References

[Wooldridge 2002] M. Wooldridge, “An Introduction to Multi Agent Systems”, John Wiley & Sons, ISBN 9780471496915, 2002

[Shoham and Leyton-Brown 2008] Y. Shoham and K. Leyton-Brown, “Multiagent Systems”, Cambridge University Press, ISBN 9780521899437, 2008

[Fensel and Bussler 2002] D. Fensel and C. Bussler, “The Web Service Modeling Framework WSMF”, Electronic Commerce Research and Applications, 2002

[McIlraith, Son and Zeng 2001] S. A. McIlraith, T. C. Son, and H. Zeng, “Semantic Web Services”, IEEE Intelligent Systems Volume 16 Issue 2, Stanford University, 2001

[Booth et al 2004] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris and D. Orchard, “Web services architecture”, W3C working group note, 2004

[Haas and Brown 2004] H. Haas and A. Brown, “Web Services Glossary”, W3C Working Group Note, 2004

[Gottschalk et al 2002] K. Gottschalk, S. Graham, H. Kreger and J. Snell, “Introduction to Web services architecture”, IBM Systems Journal Vol 41, 2002

[Clement et al 2004] L. Clement, A. Hately, C. von Riegen, T. Rogers, “UDDI version 3.0.2.”, UDDI Spec Technical Committee Draft, 2004

[Gudgin et al 2007] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, H. F. Nielsen, A. Karmarkar, Y. Lafon, “SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)”, W3C Recommendation, 2007

[Klush 2008] M. Klush, “CASCOM: Intelligent Service Coordination in the Semantic Web”, Chapter 4 “Semantic Web Service Coordination” Birkhäuser Basel, ISBN 9783764385743, 2008

[Gonçalves, Costa and Botelho 2008] B. Gonçalves, P. Costa and L. Botelho, “CASCOM: Intelligent Service Coordination in the Semantic Web”, Chapter 5 “Context-Awareness” Birkhäuser Basel, ISBN 9783764385743, 2008

[Oundhakar et al 2005] S. Oundhakar, K. Verma, K. Sivashanmugam, A. Sheth, J.Miller, “Discovery of Web Services in a Multi-Ontology and Federated Registry Environment”, International Journal of Web Services Research, 2005

[Sivashanmugam, Verma and Sheth 2004] K. Sivashanmugam, K. Verma, A. Sheth, “Discovery of Web Services in a Federated Registry Environment”, Proceedings of the IEEE International Conference on Web Services, 2004

[Guidi, Lucchi and Mazzara 2007]C. Guidi, R. Lucchi and M. Mazzara, “A Formal Framework for Web Services Coordination”, Electronic Notes in Theoretical Computer Science, 2007

[Fenza, Loia and Senatore 2008] G. Fenza, V. Loia and S. Senatore, “A hybrid approach to semantic web services matchmaking”, International Journal of Approximate Reasoning, 2008

[Wilcock 2003] G. Wilcock, “Talking owls: Towards an Ontology Verbalizer”, Human Language Technology for the Semantic Web and Web Services, 2003

[Bontcheva and Wilks 2004] K. Bontcheva and Y. Wilks, “Automatic report generation from ontologies:the MIAKT approach”, Ninth International Conference onApplications of Natural Language to Information Systems (NLDB’2004), 2004.

[Barker 2005] A. Barker, “Agent-Based Service Coordination for the Grid”, Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT’05), 2005

[Barros, Dumas and Oaks 2005] A. Barros, M. Dumas and P. Oaks, “A Critical Overview of the Web Services Choreography Description Language (WS-CDL)”, BPTrends Newsletter 3, 2005

[Christensen et al 2001] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, “Web Services Description Language (WSDL) 1.1”, W3C Note, 2001

[Farrell and Lausen 2007] J. Farrel and H. Lausen, “Semantic Annotations for WSDL and XML Schema”, W3C Recommendation, 2007

[Bruijn 2008] J. de Bruijn, “The Web Service Modeling Language WSML”, http://www.wsmo.org/wsml/wsml-syntax, 2008

[Huhns 2002] M. N. Huhns, “Agents as Web Services”, IEEE Internet Computing 6 (4), 93 – 95, 2002

[Chi and Song 2007] J. Chi and J. Song, “Intelligent-Agent and Web-Service Based Service Composition for E-Business”, Electrical and Computer Engineering CCECE Canadian Conference on, 2007

[Lopes and Botelho 2005] A. Lopes and L. Botelho, "Agent Technology for Context-aware Execution of Semantic Web Services", Proceedings of the First UK Young Researchers Workshop on Service Oriented Computing, 2005

[García-Sanchez et al 2009] F. García-Sanchez, R. Valencia-García, R. Martínez-Béjar and J. T. Fernández-Breis, “An ontology, intelligent agent-based framework for the provision of semantic web services”, Expert Systems with Applications Issue 2 Part 2 (3167–3187), Elsevier Ltd, 2009

[Russell and Norvig 1995] S. Russell and P. Norvig, “Artificial Intelligence: A Modern Approach”, Chapter 2 “Intelligent Agents”, Prentice-Hall, Inc, ISBN 9780131038059, 1995

[Schumacher, Helin and Schuldt 2008] M. Schumacher, H. Helin, H. Schuldt, "CASCOM: Intelligent Service Coordination in the Semantic Web", Birkhäuser Basel, ISBN 9783764385743, 2008

[Dey, Abowd and Salber 2001] A. Dey, G. Abowd, D. Salber, "A conceptual framework and toolkit for supporting the rapid prototyping of context-aware applications in special issue on context-aware computing", Human-Computer Interaction Vol. 16, 2001

[Martin et al 2004a] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, K. Sycara, "OWL-S: Semantic Markup for Web Services", W3C Member Submission, 2004

[Martin et al 2004b] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara, "Bringing Semantics to Web Services:The OWL-S Approach", First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC), 2004

[Berglund et al 2007] A. Berglund, S. Boag, D. Chamberlin, M. Fernández, M. Kay, J. Robie, J. Siméon, "XML Path Language (XPath) 2.0", W3C Recommendation, 2007

[Reiter and Dale 2000] E. Reiter and R. Dale, "Building Natural Language Generation Systems, Cambridge University Press", ISBN 0521620368

[Jurasfsky and Martin 1999] D. Jurafsky and J. H. Martin, "Natural Language Generation", chapter to be included in "Speech and Language Processing: An introduction to speech recognition computational linguistics and natural language processing ", 1999

[Hovy 1992] E. H. Hovy, "A New Level of Language Generation Technology: Capabilities and Possibilities", IEEE Expert: Intelligent Systems and Their Applications, 1992

[Reiter and Dale 1997] E. Reiter and R. Dale, "Building Applied Natural Language Generation Systems", Natural Language Engineering Volume 3 Issue 1, Cambridge University Press, 1997

[van Deemter, Krahmery and Theunez 2003] K. van Deemter, E. Krahmery and M. Theunez, "Real vs. template-based natural language generation: a false opposition?", Computational Linguistics Volume 31 Issue 1, 2003

[Busemann and Horacek 1998] S. Busemann and H. Horacek, "A Flexible Shallow Approach to Text Generation", Workshop On Natural Language Generation EWNLG, 1998

[Reiter and Mellish 1993] Ehud Reiter and Chris Mellish , "Optimizing the Costs and Benefits of Natural Language Generation ", Proceedings of the 13th International Joint Conference on Artificial Intelligence, 1993

[Mellish and Sun 2006] Chris Mellish and Xiantang Sun, “The semantic web as a Linguistic resource: Opportunities for natural language generation”, Knowledge-Based Systems 19, 2006

[Bontcheva and Wilks 2004] K. Bontcheva, Y. Wilks, “Automatic report generation from ontologies: the MIAKT approach”, 9th International Conference on Applications of Natural Language to Information Systems, 2004.

8 Appendix

8.1 Appendix A Text Templates

In this section we present all the text templates used to generate the natural language description.

Service name templates:

1. 'This service'
2. 'The service '

Inputs templates:

1. ' needs no parameters.'
2. ' needs to receive the ',Input,' as parameter.'
3. ' needs to receive the ',First input
4. ' and ',Input,' as parameters.'
5. ', ',Input

Outputs templates:

1. ' This service has no output./ln'
2. ' It has only the ',Output,' as output./ln'
3. 'It has ',First_output
4. ' and',Output,' as output./ln'
5. ', ',Output

Preconditions templates:

1. ' The system was not able to process the precondition(s).'
2. ' This service does not have any pre-conditions.'
3. ' This service also has the following precondition: ',Precondition,'.'

4. 'This service also has the following preconditions: ',First precondition

5. ' and ',Precondition,'.'

6. ', ',Precondition

Results templates:

1. ' The system was not able to process the result(s).'

2. ' This service only has ',Result,' as result.'

3. 'It has the following results: result',First result

4. ' and ',Result,'.'

5. ', ',Result

Description templates:

1. '/ln The owner of the service describes it as: "',Description,'".'

Type of service templates:

2. '/ln Technically, this is a compound service because it combines other services in a new aggregate service.'

3. '/ln Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service.'

8.2 Expert's Inquiry

Service: Book Finder

Profile:

```
<!-- Profile description -->
<mind:BookInformationService rdf:ID="BookFinderProfile">
        <service:presentedBy rdf:resource="#BookFinderService"/>
        <profile:serviceName xml:lang="en">Book Finder</profile:serviceName>
        <profile:textDescription xml:lang="en">
        This service returns the information of a book whose title best matches the given string.
        </profile:textDescription>
        <profile:hasInput rdf:resource="#BookName"/>
        <profile:hasOutput rdf:resource="#BookInfo"/>
</mind:BookInformationService>
```

Natural Language Description:

```
The service Book Finder is not a compound service.
This service needs to receive the name of the book as
parameter. It has only the book information as output.
This service does not have any pre-conditions.
Also, the owner of the service describes it as: "This
service returns the information of a book whose title best
matches the given string".
```

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?
2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?
3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?
4. Commentary (facultative)

Service: Zip Code Finder

Profile:

```
<!-- Profile description -->
<mind:MapService rdf:ID="ZipCodeFinderProfile">
        <service:presentedBy rdf:resource="#ZipCodeFinderService"/>
        <profile:serviceName xml:lang="en">Find ZipCode</profile:serviceName>
        <profile:textDescription xml:lang="en">
        Returns the zip code for the given city/state. If there are more than one zip  codes
        associated with that city only the first one is returned.
        </profile:textDescription>
        <profile:hasInput rdf:resource="#City"/>
        <profile:hasInput rdf:resource="#State"/>
        <profile:hasOutput rdf:resource="#ZipCode"/>
</mind:MapService>
```

Natural Language Description:

```
The service Zip Code Finder is not a compound service.
This service needs to receive the city and state as
parameters. It has only the zip code as output.
This service does not have any pre-conditions.
Also, the owner of the service describes it as: "Returns
the zip code for the given city/state. If there are more
than one zip codes associated with that city only the first
one is returned".
```

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?

2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?

3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?

4. Commentary (facultative)

Service: Find Latitude & Longitude

Profile:

```
<!-- Profile description -->
<mind:MapService rdf:ID="FindLatLongProfile">
        <service:isPresentedBy rdf:resource="#FindLatLongService"/>
        <profile:serviceName xml:lang="en">
        Find Latitude & Longitude
        </profile:serviceName>
        <profile:textDescription xml:lang="en">
        Find the latitude and longitude of the given US zip code.
        </profile:textDescription>
        <profile:hasInput rdf:resource="#ZipCode"/>
        <profile:hasOutput rdf:resource="#LatLong"/>
</mind:MapService>
```

Natural Language Description:

```
The service Find Latitude & Longitude is not a compound
service.
This service needs to receive the zip code as parameter. It
has only the latitude and longitude as output.
This service does not have any pre-conditions.
Also, the owner of the service describes it as: "Find the
latitude and longitude of the given US zip code".
```

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?

2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?

3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?

4. Commentary (facultative)

Service: Barnes and Nobles Price Check

Profile:

```
<!-- Profile description -->
<mind:BookInformationService rdf:ID="BNPriceProfile">
        <service:presentedBy rdf:resource="#BNPriceService"/>
        <profile:serviceName xml:lang="en">BN Price Check</profile:serviceName>
        <profile:textDescription xml:lang="en">
        This service returns the price of a book as advertised in Barnes and Nobles web site given the ISBN Number.
        </profile:textDescription>
        <profile:hasInput rdf:resource="#BookInfo"/>
        <profile:hasOutput rdf:resource="#BookPrice"/>
</mind:BookInformationService>
```

Natural Language Description:

The service Barnes and Nobles Price Check is not a compound service.
This service needs to receive the book information as parameter. It has only the price of the book as output.
This service does not have any pre-conditions.
Also, the owner of the service describes it as: "This service returns the price of a book as advertised in Barnes and Nobles web site given the ISBN Number".

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?

2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?

3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?

4. Commentary (facultative)

Service: Amazon Book Price

Profile:

```
<mind:BookInformationService rdf:ID="AmazonPriceProfile">
        <service:isPresentedBy rdf:resource="#AmazonPriceService"/>
        <profile:serviceName xml:lang="en">Amazon Book Price</profile:serviceName>
        <profile:hasInput rdf:resource="#BookInfo"/>
        <profile:hasOutput rdf:resource="#BookPrice"/>
</mind:BookInformationService>
```

Natural Language Description:

The service Amazon Book Price is not a compound service. This service needs to receive the book information as parameter. It has only the price of the book as output. This service does not have any pre-conditions.

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?
2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?
3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?
4. Commentary (facultative)

Service: Babel Fish Translator

Profile:

```
<!-- Profile description -->
<mind:LanguageService rdf:ID="BabelFishTranslatorProfile">
        <service:presentedBy rdf:resource="#BabelFishTranslatorService"/>
        <profile:serviceName xml:lang="en">BabelFish Translator</profile:serviceName>
        <profile:textDescription xml:lang="en">
        Convert text from one language to another language. Supported languages are Dutch, English, French, German, Italian, Japanese, Korean, Portuguese, Spanish, and Russian. The valid input output pairs is given by the property canBeTranslatedTo.
        </profile:textDescription>
        <profile:hasInput rdf:resource="#InputString"/>
        <profile:hasInput rdf:resource="#InputLanguage"/>
        <profile:hasInput rdf:resource="#OutputLanguage"/>
        <profile:hasOutput rdf:resource="#OutputString"/>
        <profile:hasPrecondition rdf:resource="#SupportedLanguagePair"/>
</mind:LanguageService>
```

Natural Language Description:

```
The service Babel Fish Translator is not a compound service.
This service needs to receive the language the text is written in, the language the text will be translated to and the text to be translated as parameters. It has only the translated text as output.
This service also has the following precondition: the input and output language pair is supported.
Also, the owner of the service describes it as: "Convert text from one language to another language. Supported languages are Dutch, English, French, German, Italian, Japanese, Korean, Portuguese, Spanish, and Russian. The valid input output pairs is given by the property canBeTranslatedTo".
```

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?

2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?

3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?

4. Commentary (facultative)

Service: Currency Converter

Profile:

```
<!-- Profile description -->
<mind:CurrencyService rdf:ID="CurrencyConverterProfile">
        <service:isPresentedBy rdf:resource="#CurrencyConverterService"/>
        <profile:serviceName xml:lang="en">Price Converter</profile:serviceName>
        <profile:textDescription xml:lang="en">Converts the given price to another currency.</profile:textDescription>
        <profile:hasInput rdf:resource="#InputPrice"/>
        <profile:hasInput rdf:resource="#OutputCurrency"/>
        <profile:hasOutput rdf:resource="#OutputPrice"/>
</mind:CurrencyService>
```

Natural Language Description:

```
The service Price Converter is not a compound service.
This service needs to receive the input price and output
currency as parameters. It has only the output price as
output.
This service does not have any pre-conditions.
Also, the owner of the service describes it as: "Converts
the given price to another currency".
```

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?

2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?

3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?

4. Commentary (facultative)

Service: Book Price Finder

Profile:

```
<mind:BookInformationService rdf:about="#BookPriceProfile">
        <service:presentedBy rdf:resource="#BookPriceService"/>
        <profile:serviceName xml:lang="en">Book Price Finder</profile:serviceName>
        <profile:textDescription xml:lang="en">
        Returns the price of a book in the desired currency. First the ISBN number for the given
        book is found and then this ISBN number is used to get the prive of the book from
        Barnes & Nobles service.
        </profile:textDescription>
        <profile:hasInput>
            <process:Input rdf:ID="BookName">
                <rdfs:label>Book Name</rdfs:label>
                <process:parameterType
                rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                http://www.w3.org/2001/XMLSchema#string
                </process:parameterType>
            </process:Input>
        </profile:hasInput>
        <profile:hasInput>
            <process:Input rdf:ID="Currency">
                <rdfs:label>Output Currency</rdfs:label>
                <process:parameterType
                rdf:datatype=http://www.w3.org/2001/XMLSchema#anyURI">
                http://www.daml.ecs.soton.ac.uk/ont/currency.owl#Currency
                </process:parameterType>
            </process:Input>
        </profile:hasInput>
        <profile:hasOutput>
            <process:Output rdf:ID="BookPrice">
                <rdfs:label>Output Price</rdfs:label>
                <process:parameterType
                rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
                http://www.mindswap.org/2004/owl-s/concepts.owl#Price
                </process:parameterType>
            </process:Output>
        </profile:hasOutput>
</mind:BookInformationService>
```

Natural Language Description:

```
The service Book Price Finder is a compound service.
This service needs to receive the currency and name of the
book as parameters. It has only the price of the book as
output.
This service does not have any pre-conditions.
Also, the owner of the service describes it as: "Returns
the price of a book in the desired currency. First the ISBN
number for the given book is found and then this ISBN
number is used to get the prive of the book from Barnes &
Nobles service".
```

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?

2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?

3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?

4. Commentary (facultative)

Service: Cheaper Book Finder

Profile:

```
<!-- Profile description -->
<profile:Profile rdf:ID="FindCheaperBookProfile">
        <service:isPresentedBy rdf:resource="#FindCheaperBookService"/>
        <profile:serviceName xml:lang="en">Cheaper Book Finder</profile:serviceName>
        <profile:hasInput rdf:resource="#BookName"/>
        <profile:hasOutput rdf:resource="#BookInfo"/>
</profile:Profile>
```

Natural Language Description:

```
The service Cheaper Book Finder is a compound service.
This service needs to receive the name of the book as
parameter. It has only the book information as output.
This service does not have any pre-conditions.
```

1. In a scale from 1 (bad) to 4 (good), do you think the English description matches the Profile of the OWL-S description?

2. In a scale from 1 (bad) to 4 (good), do you think the English generated description allows understanding the service's essential?

3. In a scale from 1 (bad) to 4 (good), do you think the service's English description is useful to the potential user, assuming it is a person?

4. Commentary (facultative)

About all the previous services:

- Final comments on the generality of the obtained results (facultative)

- Suggestions, for example what extra information could be added to the English descriptions from other parts of the OWL-S description (facultative)

8.3 User's Inquiry

Service: Book Finder

Natural Language Description:

```
The service Book Finder needs to receive the name of the
book as parameter. It has only the book information as
output.
This service does not have any pre-conditions.
The owner of the service describes it as: "This service
returns the information of a book whose title best matches
the given string".
Moreover, this service is self-sufficient in the sense that
it does not use other services. Technically, it is called
an atomic service.
```

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?

Service: Zip Code Finder

Natural Language Description:

```
The service Zip Code Finder needs to receive the city and
state as parameters. It has only the zip code as output.
This service does not have any pre-conditions.
The owner of the service describes it as: "Returns the zip
code for the given citystate. If there are more than one
zip codes associated with that city only the first one is
returned".
Moreover, this service is self-sufficient in the sense that
it does not use other services. Technically, it is called
an atomic service.
```

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?

Service: Find Latitude & Longitude

Natural Language Description:

The service Find Latitude & Longitude needs to receive the zip code as parameter. It has only the latitude and longitude as output.
This service does not have any pre-conditions.
The owner of the service describes it as: "Find the latitude and longitude of the given US zip code".
Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service.

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?

Service: Barnes and Nobles Price Check

Natural Language Description:

The service Barnes and Nobles Price Check needs to receive the book information as parameter. It has only the price of the book as output.
This service does not have any pre-conditions.
The owner of the service describes it as: "This service returns the price of a book as advertised in Barnes and Nobles web site given the ISBN Number".
Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service.

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?

Service: Amazon Book Price

Natural Language Description:

The service Amazon Book Price needs to receive the book information as parameter. It has only the price of the book as output.
This service does not have any pre-conditions. This service only has description as result.
Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service.

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?

Service: Babel Fish Translator

Natural Language Description:

The service Babel Fish Translator needs to receive the language the text is written in, the language the text will be translated to and the text to be translated as parameters. It has only the translated text as output.
This service also has the following precondition: the input and output language pair is supported.
The owner of the service describes it as: "Convert text from one language to another language. Supported languages are Dutch, English, French, German, Italian, Japanese, Korean, Portuguese, Spanish, and Russian. The valid input output pairs is given by the property canBeTranslatedTo".
Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service.

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?

Service: Currency Converter

Natural Language Description:

The service Price Converter needs to receive the input price and output currency as parameters. It has only the output price as output.
This service does not have any pre-conditions.
The owner of the service describes it as: "Converts the given price to another currency".
Moreover, this service is self-sufficient in the sense that it does not use other services. Technically, it is called an atomic service.

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?

Service: Book Price Finder

Natural Language Description:

The service Book Price Finder needs to receive the currency and name of the book as parameters. It has only the price of the book as output.
This service does not have any pre-conditions.
The owner of the service describes it as: "Returns the price of a book in the desired currency. First the ISBN number for the given book is found and then this ISBN number is used to get the prive of the book from Barnes & Nobles service".
Technically, this is a compound service because it combines other services in a new aggregate service.

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?

Service: Cheaper Book Finder

Natural Language Description:

```
The service Cheaper Book Finder needs to receive the name
of the book as parameter. It has only the book information
as output.
This service does not have any pre-conditions. This service
only has description as result.
Technically, this is a compound service because it combines
other services in a new aggregate service.
```

1. In a scale from 1 (bad) to 4 (good), do you think you can understand the described service?

2. In a scale from 1 (bad) to 4 (good), if the natural language description were the result of a service discovery made by the user in the Internet, would the information be useful to decide whether or not to use the described service?