Semantic Web Services Provisioning

José María García, 31724461-D

josemgarcia@us.es

Supervised by Prof. Dr. David Ruiz Cortés
and Prof. Dr. Antonio Ruiz Cortés

Departamento de
Lenguajes y Sistemas Informáticos
Universidad de Sevilla

Research Report submitted to the Department of Computer Languages
and Systems of the University of Sevilla in partial fulfillment
of the requirements for the degree of Ph.D. in Computer Engineering.

(Research Period)

Support: This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472) and by the Andalusian Government under project ISABEL (TIC-2533).

Contents

List of Figures

List of Tables

Acknowledgements

Now that this research report finally comes to an end, it is time to express my gratitude to the people who have supported me along this path that I has just started. First of all, I would like to thank my research advisors, David and Antonio, who have taught me their vision about the amazing world of research, each one his own way. Their help and support become one of the pillars of this work. Thanks for believing in me, and for that phone call that brought me back to University.

To all my colleagues here at the School, who have made so easy to get used to this interesting change in my career: Manolo, Pablo F., Bena, Octavio, Joaquín, Irene, and all the rest... but especially, to Carlos, Pablo T. and Sergio, who make those after-lunch conversations one of the best moments of the day. *Ain't no mountain high enough*.

Finally, I would like to thank my family and friends for their kindly support and love, encouraging me to accomplish my dreams. And to that light that appeared in my life when it was so necessary, supporting me in all those difficult decisions, that has finally became the right choice. Thanks for all, Puri.

Abstract

Semantic Web Services constitute an important research area, where various underlying frameworks, such as WSMO and OWL-S, define Semantic Web ontologies to describe Web services, so they can be automatically discovered, composed, and invoked. Service discovery has been traditionally interpreted as a functional filter in current Semantic Web Services frameworks, frequently performed by Description Logics reasoners. However, semantic provisioning has to be performed taking Quality-of-Service (QoS) into account, defining user preferences that enable QoS-aware Semantic Web Service selection.

Nowadays, the research focus is actually on QoS-aware processes, so current proposals are developing the field by providing QoS support to semantic provisioning, especially in selection processes. These processes lead to optimization problems, where the best service among a set of services has to be selected, so Description Logics cannot be used in this context. Furthermore, user preferences has to be semantically defined so they can be used within selection processes.

There are several proposals that extend Semantic Web Services frameworks allowing QoS-aware semantic provisioning. However, proposed selection techniques are very coupled with their proposed extensions, most of them being implemented *ad hoc*. Thus, there is a semantic gap between functional descriptions (usually using WSMO or OWL-S) and user preferences, which are specific for each proposal, using different ontologies or even non-semantic descriptions, and depending on its corresponding *ad hoc* selection technique.

In this report, we give an overview of most important Semantic Web Services frameworks, showing a comparison between them. Then, a thorough analysis of state-of-the art proposals on QoS-aware semantic provisioning and user preferences descriptions is presented, discussing about their applicability, advantages, and defects. Results from this analysis motivate our research work, which has been already materialized in two early contributions.

Resumen

Los servicios web semánticos constituyen un importante campo de investigación, en el cual distintos frameworks, como por ejemplo WSMO y OWL-S, definen ontologías de la web semántica para describir servicios web, de forma que estos puedan ser descubiertos, compuestos e invocados de manera automática. El descubrimiento de servicios ha sido interpretado tradicionalmente como un filtro funcional en los frameworks actuales de servicios web semánticos, usando para ello razonadores de lógica descriptiva. Sin embargo, las tareas de aprovisionamiento semántico deberían tener en cuenta la calidad del servicio, definiendo para ello preferencias de usuario de manera que sea posible realizar una selección de servicios web semánticos sensible a la calidad.

Actualmente, el foco de la investigación está en procesos sensibles a la calidad, por lo que las propuestas actuales están trabajando en este campo introduciendo el soporte adecuado a la calidad del servicio dentro del aprovisionamiento semántico, y principalmente en las tareas de selección. Estas tareas desembocan en problemas de optimización, donde el mejor servicio de entre un concjunto debe ser seleccionado, por lo que las lógicas descriptivas no pueden ser usadas en este contexto. Además, las preferencias de usuario deben ser definidas semánticamente, de forma que puedan ser usadas en las tareas de selección.

Existen bastantes propuestas que extienden los frameworks de servicios web semánticos para habilitar el aprovisionamiento sensible a la calidad. Sin embargo, las técnicas de selección propuestas están altamente acopladas con dichas extensiones, donde la mayoría de ellas implementan algoritmos *ad hoc*. Por tanto, existe un salto semántico entre las descripciones funcionales (normalmente usando WSMO o OWL-S) y las preferencias de usuario, las cuales son definidas específicamente por cada propuesta, usando ontologías distintas o incluso descripciones no semánticas que dependen de la correspondiente técnica de selección *ad hoc*.

En este informe, ofrecemos una introducción a los frameworks de servicios web semánticos más importantes, mostrando una comparación entre ellos. Seguidamente, presentamos un análisis exhaustivo del estado del arte en aprovisionamiento semántico sensible a la calidad y de las formas de describir preferencias de usuario, discutiendo sobre su aplicabilidad, ventajas e inconvenientes. Los resultados de este análisis motivan nuestro trabajo de investigación, el cual ya se ha materializado en algunas publicaciones iniciales.

Part I

Preface

Chapter 1

Introduction

Semantic Web Services (SWS) are becoming an important research area, presenting an extension to current Web Service technology aimed at performing processes automatically. The present report is focused on discovery and selection of SWS, discussing the state of the art on these processes, specially analyzing QoS*-aware proposals. Thus, different alternatives are analyzed within the present report, comparing their approaches to* QoS*-aware provisioning and user preferences descriptions.*

1.1 Research context

In order to introduce the context of the research done, this section briefly overview the basics of the treated topic. Firstly, Web Services technologies are described, showing how current Web Services can be discovered and invoked. Secondly, the Semantic Web is presented as an alternative to give semantics to current Web contents. The union between the Semantic Web and Web Services technologies conforms the Semantic Web Services, which are also introduced. Finally, the concept of QoS-aware provisioning is detailed, because it is the key point of this research work.

1.1.1 Web Services

The term *Web Services* (WS) is widely used nowadays, but its meaning is not always the same. There are several definitions of this term, some of them being too abstract, and some being too coupled with certain protocols. One of this definitions stated that a Web Service can be seen as an application accessible to other applications over the Web. This definition is too open, because it can include any URL, CGI scripts, or programs with an interface published on the Web.

Another definition comes from the UDDI consortium, and it defines a WS as a *"a self-contained, modular business application that have open, Internet-oriented, standards-based interfaces"*. One important point of this more detailed definition is that it focuses on the *open* nature of WS, which means that a WS has a published interface that can be invoked across the Internet. However, this definition is not clear enough, not explaining what *a self-contained, modular application* means.

An early definition published by the W3C Web Services Architecture Working Group define a WS as *"a software application identified by a* URI*, whose interfaces and binding are capable of being defined, described and discovered by* XML *artifacts and supports direct interactions with other software applications using* XML *based messages via Internet-based protocols"*. This definition is quite accurate, stressing that WS should be accessible (*defined, described and discovered*), and support interactions with clients over XML-based protocols. However, the current definition published by the W3C, which we are going to adopt in this report, is the following:

> *A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described*

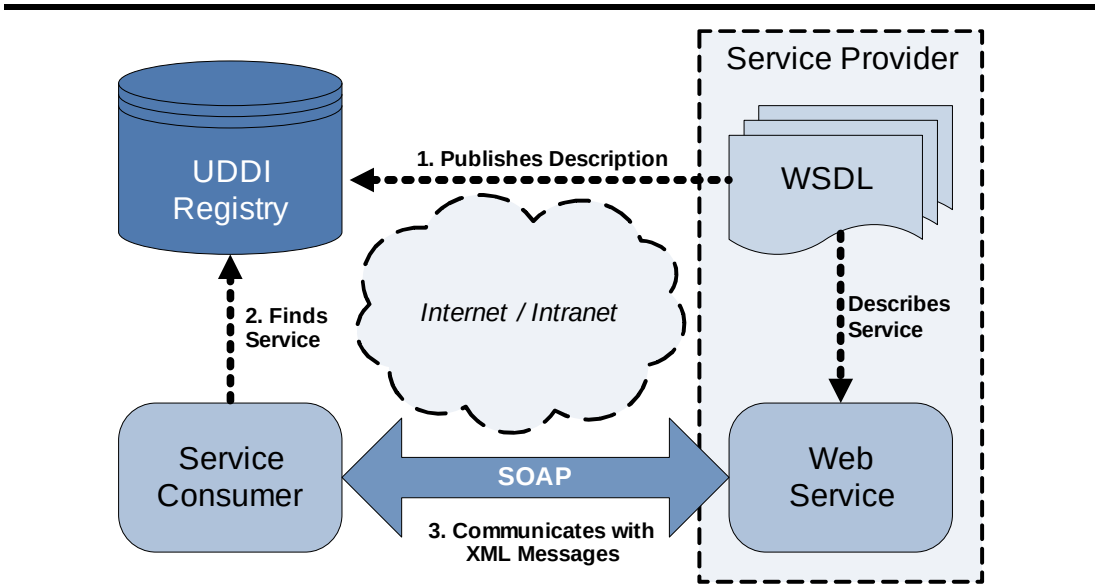


Figure 1.1: *Web Services architecture.*

> *in a machine-processable format (specifically* WSDL*). Other systems interact with the Web service in a manner prescribed by its description using* SOAP *messages, typically conveyed using* HTTP *with an* XML *serialization in conjunction with other Web-related standards [15].*

This definition, though it is coupled with some technologies that could be changed without losing its meaning, presents some key points. They are software systems aimed at supporting *interoperable machine-to-machine interaction*, *i.e.* it discards Web pages as services. Additionally, being described in a *machine-processable format*, WS can be managed autonomously by machines, enabling the developing of Service Oriented Architectures.

Service Oriented Architecture (SOA) constitutes a new computing paradigm that has been taking great interest from both the research community and business people. Within this paradigm, Web Services are the key to a web-based SOA. Essentially, WS are reusable, loosely coupled components that provides a concrete functionality to a potential user. In order to access to this functionality, they run over standard Internet protocols and languages, such as WSDL, SOAP or UDDI, that allow a distributed operation, as seen in Fig. §1.1 .

Within this SOA vision, standards specifications are very important. The Web Services Description Language (WSDL [19]) is one of these core specifications, providing a model and a XML-based language for describing WS. This

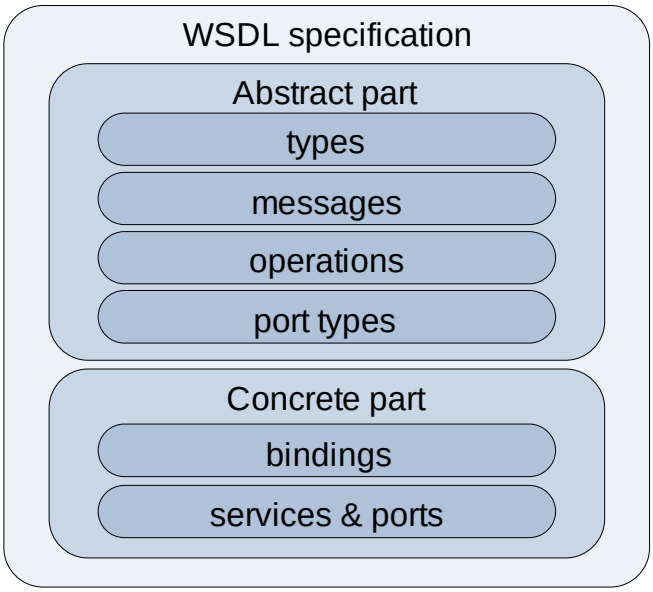


Figure 1.2: WSDL *specification components.*

standard allows to define the interaction mechanism with certain service in detail. Thus, the definition of a WS in WSDL contains its interfaces, its bindings, and its endpoints. Nevertheless, each WSDL file may contain more than one WS definition. The main components of a WSDL definition are depicted in Fig. §1.2 and explained in the following:

Interface It describes the different *operations* that can be performed by the WS. It corresponds to an abstract definition of those operations by means of their inputs and outputs, depicted as the *abstract* portion of Fig. §1.2 . In fact, these inputs and outputs are referenced as *messages*, which have to be defined separately using built-in XML Schema *types* or user-defined ones.

Binding This component describes the access protocol to use in order to interact with the WS defined. Basically it allows to define the concrete grounding between interfaces (operations) and the chosen messaging protocols. However, it does not point to the specific location of a WS, so this remains independent from the network protocol used.

Endpoint This points to a specific physical address for the WS defined, in opposition to the binding. By this property, the location of the service is defined, so its invocation becomes possible.

By separating the abstract definition or interface of a service from its concrete bindings and endpoints, WSDL allows to be applied to not only Web services, but services in general. Moreover, using both the abstract and concrete parts of a WSDL definition, compiler tools, such as Apache AXIS [†1], automatically generate proxies so that applications can use WS without bothering about WS protocols and languages.

Usually, in the binding section of a WSDL, SOAP is used to establish the link between the endpoint and the interface of a WS. SOAP[†2] is a lightweight protocol that can be used to exchange structured information between two services [39]. The messages exchanged are built upon XML, and the SOAP specification covers the following:

- How to specify the information of the messages in XML, i.e. the message layout and its content.
- Communication aspects, including the transport protocol to send messages, such as HTTP, SMTP, etc.
- Information about the sender and receiver entities, such as which parts of the message they have to handle with.
- How to express SOAP messages in order to perform remote procedure calls.
- Extension of the protocol to allow using other features upon SOAP, such as routing, security, reliability, etc.

There are some alternatives and extensions to SOAP that can be also used for the binding of WS, such as SOAP-MTOM, DCOM, REST or RMI. However, SOAP is more widely used and simpler, because of the existing tool support for integrated solutions with WSDL and SOAP.

Using WSDL to define the interaction between services and clients and SOAP to actually interact with them, these clients may access to any WS they know. However, in the SOA scenario presented in Fig. §1.1 , the client or service consumer does not necessarily know which WS brings the functionality required. Thus, a public repository of services is needed, where a service consumer may search for some required functionality provided by one or more different companies, i.e. service providers.

[†1] http://ws.apache.org/axis/

[†2] It stands for Simple Access Object Protocol, though in last version of the recommendation is not explicitly defined so.

Universal Description, Discovery and Integration standard (UDDI) constitutes an effort from different companies, within OASIS group, to define a public or private services repository so the ideal SOA scenario from Fig. §1.1 can be materialized [20]. This standard defines how a provider can describe the services he offers, how a consumer can search for a service in an UDDI repository by means of certain information fields, and finally how to technically access to the discovered service. Thus, the main characteristics of an UDDI registry have to comprise are the following:

Description Service descriptions have to be extensible, and layered in different abstraction levels, so the description system can be changed easily.

Discovery The UDDI system have to provide a flexible search mechanism, so it is possible to search services and entities by means of any of the information stored it the registry.

Integration Once discovered, the UDDI registry have to allow the seamless integration between the service and the consumer system, so it is necessary an infrastructure that provides the required information to invoke the discovered service.

Using these three standards (WSDL, SOAP, and UDDI) a complete SOA scenario can be implemented and used. However, current WS technology has an important flaw: usage and integration of WS needs to be performed manually. Discovery, composition, and invocation are supported by syntactical information descriptions, so these processes can not take benefit of the Semantic Web. What is more, current version of UDDI can not be directly used to discover services using semantic information, because of its key-based search engine. Thus, to perform discovery, composition and invocation processes, among others, automatically, it is needed to define new standards that extend current technology with semantics.

1.1.2 The Semantic Web

The current Web is aimed at providing information and services that are directly consumed by human beings. Although most of the content is stored in databases, the information is presented without the structural information found in databases. From a machine point of view, it is very difficult to process all this information, so the Web can not be automatically manipulated by computers [7].

The Semantic Web [13] constitutes an extension to current Web, where information has associated semantics that can be processed and *understood* by machines. Thus, using Semantic Web technologies, all this information can be automatically processed, extracting knowledge to feed algorithms, to allow autonomous interaction between computers, and to improve the cooperation between people and computers.

Sometimes the Semantic Web is confused with the so-called *Web 2.0*. This Web 2.0 is interpreted as an evolution of the old, data-driven Web to a *social-driven* Web, which provides great enhancements on the way data and information is offered to users, so they can truly interact with the data and with other users. However, it still is a *syntactic* Web, because it does not deliver inter-application integration and interoperation. Thus, the Semantic Web is another different, but compatible trend, where a semantically-enabled Web allows the interaction between computers.

Ontologies are the building block of the Semantic Web. An ontology is *a formal, explicit specification of a shared conceptualization* [37]. Each part of this definition, which is widely adopted in the bibliography, should be further explained:

- Being a *formal specification*, an ontology can be read and processed by a machine that can handle with computational semantics.
- By means of a *explicit specification*, the vocabulary used within a certain domain is explicitly defined, leaving no room for ambiguity.
- Its *shared* nature allows to homogenize the knowledge in a domain, so the ontology constitutes the commonly accepted understanding.
- Finally, an ontology is a *conceptualization* of certain domain, providing a conceptual model to define this domain, along with the rest of the benefits of ontologies.

In general, an ontology describes certain domain formally. Specifically, it is composed of a finite list of terms and relationships between them. These terms denote *concepts* or *classes* of objects of certain importance from a concrete domain. Each term may have a set of *properties* that denote attributes that describe a concept. Moreover, there can be *relations* between concepts or properties that explicitly define their relationship. Additionally, an ontology may contain a list of *axioms* that constraint and check for consistency and coherency the different concepts, properties and relations, by means of logical expressions. For example, Fig. §1.3 shows an ontology from educational domain where all the previously cited components appear.

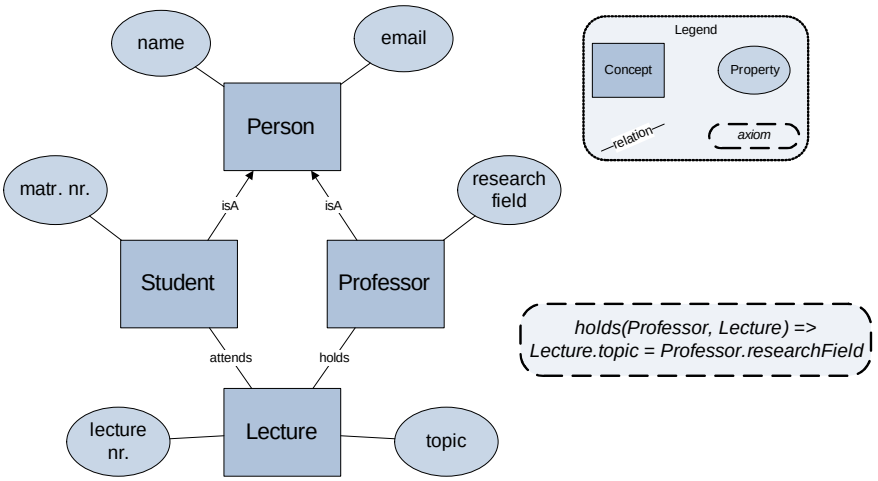


Figure 1.3: *An ontology from educational domain.*

A noteworthy relation type that appears in Fig. §1.3 is the *isA* relation. This relation states hierarchical relationships between terms. A hierarchy defines which classes are subclasses of others. Thus, a class *C* is a subclass of another class *C'* if every object in *C* is also a member of *C'*. In Fig. §1.3 , both *Student* and *Professor* classes are subclasses of *Person*, i.e. an object of *Student* (and *Professor*) class is also an object of *Person* class.

The development of the Semantic Web has been being done using a layered approach, where each layer may be developed independently. The *"layer cake"*, proposed by Tim Berners-Lee, is shown in Fig. §1.4 , where the main layers of the Semantic Web architecture are described. In these layers, different Web standards, which are described below, are proposed in order to homogenize and standardize the Semantic Web development.

The lower layer only indicates how the information is coded and how each object is identified. Unicode codification is the chosen solution to represent information, while URIs are the identifiers to use. Thus, there are no differences with the *syntactic* Web. The next layer is also taken from current Web technology, using XML as the language to express semantic information. Furthermore, the Semantic Web is also based on XML Schema and namespaces, that allow to define the user vocabulary and structure of each document, in order to provide interoperability, validation and reuse of different XML documents.

The next layer (and following ones) are specific to the Semantic Web. RDF (*Resource Description Framework* [57]) is a XML-based language for representing semantic information about resources in the Web. Basically, it is a basic data

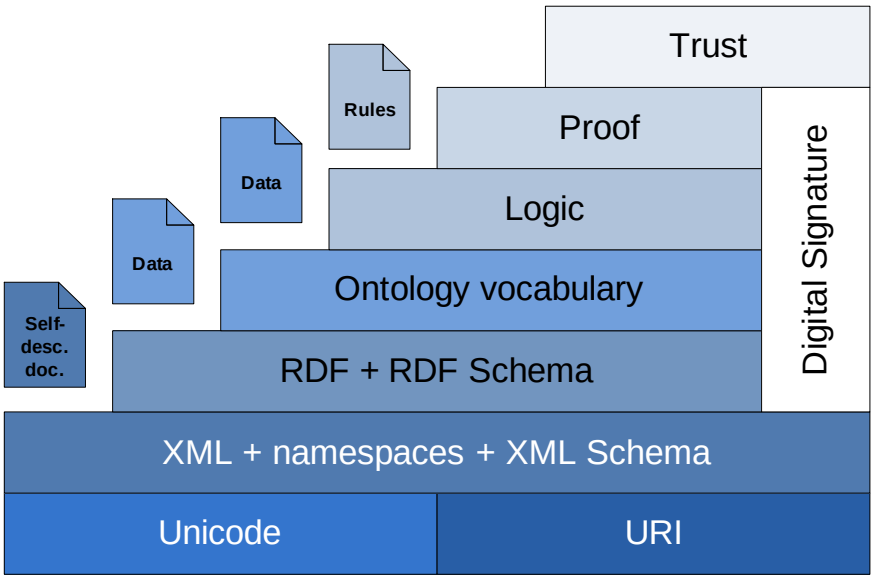


Figure 1.4: *A layered approach to the Semantic Web.*

model, where information are defined using triples associating a subject, a property, and an object. Thus, a RDF triple states that certain subject has a property which value is the object referenced. Moreover, subjects, properties and objects can be restricted to certain classes or values using RDF Schema, similarly to XML Schema. This includes constraints about which individuals in some class can have certain properties associated.

Although RDF Schema allows to define basic ontologies and hierarchies, there is a need for more powerful ontology languages that should be able to represent more complex relationships between objects, and that can be used to share knowledge between machines so they can process RDF semantic annotations. This language have to be defined in the *ontology* layer, and the most common language used is OWL [61], which is a W3C Recommendation that extends RDF Schema to fulfill the requirements of the *ontology* layer.

The *logic* layer includes the ability to process the semantic information from previous layers in order to find logical deductions about that information. The *proof* layer further add more powerful deductive process, so proofs can be represented and validated, possibly using rule languages. Finally, the *trust* layer, through the use of *digital signatures* and other knowledge sources, have to assure that the information from the lower layers do not have contradictions and can be trusted.

The last three layers, although less developed than the rest, are more fo-

cused on processing the semantic information than on representing it. However, they need a logic formalism to perform that processing. The most widely used logic formalisms in the Semantic Web field are Description Logics (DLs) [8]. DLs are a family of knowledge representation languages, that make use of reasonable subsets of first order and predicate logics. This logics family are mainly used to represent terminological knowledge in a structured way. DLs are behind some subsets of OWL ontology language. Actually, OWL specifies three variants, depending on the level of expressiveness and the underlying logic formalism: OWL-LITE, OWL-DL and OWL-FULL, where the last one sacrifices decidability to provide more expressiveness.

The complete support for these last three layers enables inferring and answering mechanisms over the semantic information. Thus, reasoners such as RACER [40] and FaCT [45] are able to classify and perform consistency checks in OWL ontologies, and other languages such as SWRL [44] or Prolog [16] allow to infer and validate proofs. Additionally, there are query languages available to answer questions and evidences extracted from ontologies, such as SPARQL [72], which defines both a language for querying an RDF store and a protocol to issue queries to a server. All these technologies are supported within tools and platforms that ease their usage and integrate them, such as Protégé [†3].

The Semantic Web are emerging as a powerful tool so that machines are able to understand and process the great amount of information available on the current Web. Thus, it is straightforward to apply these technologies to WS so they can be processed automatically by computers.

1.1.3 Semantic Web Services

As stated before, current WS technologies do not allow the automation of common processes such as discovery, execution and composition, which are necessary to develop the service-oriented computing. Furthermore, Semantic Web standards can be applied to markup information on the Web, adding semantics so machines are able to process and understand that information. In this scenario, a semantic markup of WS definitions naturally comes up as the solution to perform automatic service discovery, execution, composition and interoperation [62]. Thus, a Semantic Web Service (SWS) can be simply defined as a Web Service whose description is in a language that has well-defined semantics [83].

This vision of joining together both Web Services and Semantic Web technologies to develop SWS is usually shown as in Fig. §1.5 . We start from a

[†3] http://protege.stanford.edu/

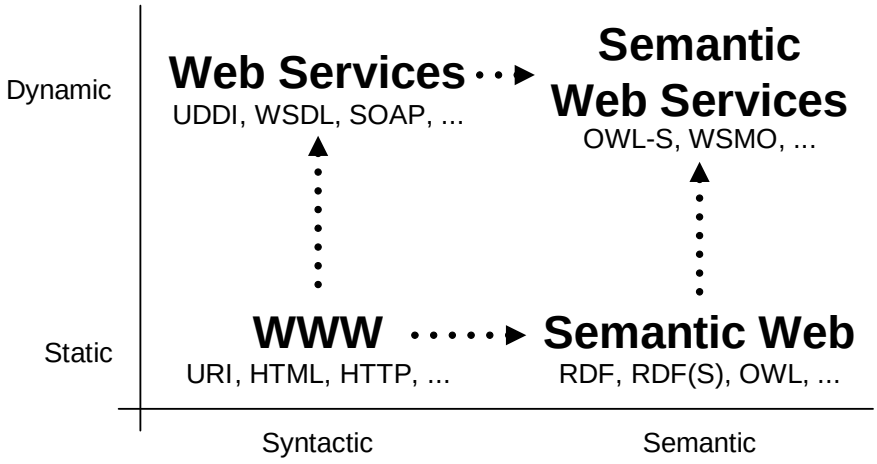


Figure 1.5: *The Semantic Web Services vision.*

static and syntactic World Wide Web, mainly focused on providing billions of information pages, where users have to interact directly with the Web to gather the required information. Because of the inherent characteristics of the current Web, this information is difficult to find, extract and interpret by computers, so the Semantic Web appears as a powerful solution, giving semantics to this static information. On the other hand, to improve new forms of interaction and to develop processes between computers connected to the Web, WS technologies are used within this dynamic environment. SWS transform the current Web from a static collection of information into a distributed device of computation, using the Semantic Web as its foundation, so this information becomes processable and interpretable by a computer [17].

A SWS framework should concentrate on three key aspects, in order to enable such SWS vision. Firstly, it has to define exhaustive description frameworks for semantically describing WS and related aspects. Secondly, it has to support ontologies to describe WS, using them as its underlying data model, so machines are able to interpret all that information. Finally, a complete SWS framework has to define semantically driven technologies that support the automation of WS usage processes.

The most important proposals concerning SWS frameworks are WSMO [30] and OWL-S [58]. They support the previously presented aspects of a SWS framework, and provide tools to actually put in practice the SWS vision. Furthermore, there are other framework proposals that take other approaches, such as METEOR-S project that aims at extending current Web standards [71], and SWSF [9], wich is another W3C member submission to standardize SWS.

All these proposals are further discussed in Chapter §2 .

The usage scenarios of a SWS framework involves many different processes, depending on the concrete needs of each scenario. Generally, the processes that SWS frameworks automatize are the following:

Publication Make the description of a Web service available on the Web.

Discovery Detect suitable services for solving a given task.

Selection Choose the most appropriate services among the usable ones.

Composition Combine services to achieve a complex goal.

Meditation Solve data, protocol, and process mismatches among the elements that shall interoperate.

Execution Invoke services according to consumption interface and programmatic conventions.

Additionally, to properly support execution, there are some extra processes that have to be aware of, such as *monitoring* the execution, *compensation* in case an unwanted effect appears, *replacement* of services that have to be substituted by equivalent ones, and *auditing* that the service execution is occurring as expected.

1.1.4 QoS-Aware Provisioning

Once a service has been published, it is available from a repository, where potential users fetch for desired services. These fetching involves two separate processes (sketched in Fig. §1.6 , that are referenced together as *service provisioning*, as well as service procurement [76]:

i. **Discovery**, where candidate services, which fulfills the user requirements, are obtained from a repository.

ii. **Selection**, where the most appropriate service is chosen from the previous set of candidate services, with regards to user preferences.

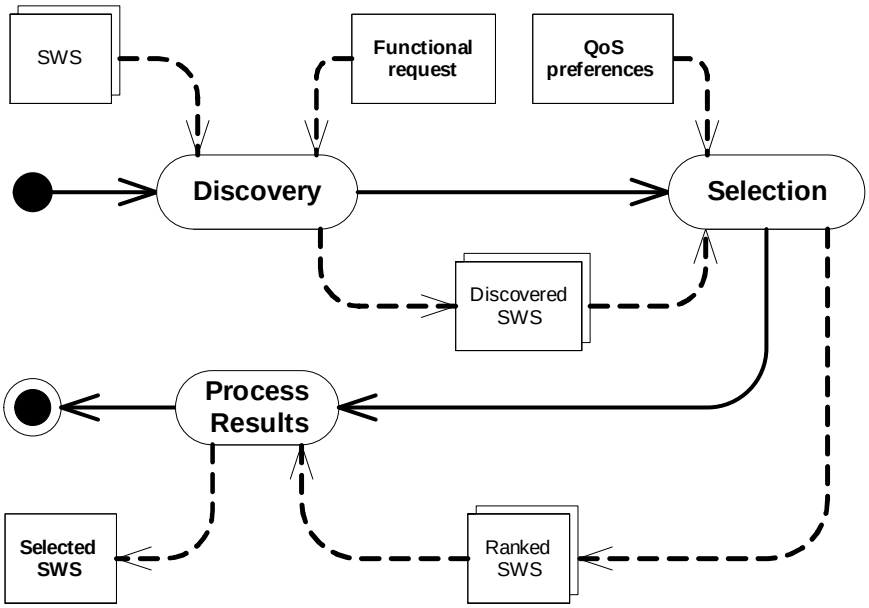


Figure 1.6: QOS-*aware SWS provisioning activities.*

With respect to SWS discovery, McIlraith *et al.* define it as the process of automatically locating WS that provide a particular service and that adhere to requested properties [62]. This definition is very agnostic about the type of properties that the requester can use. Additionally, it does not consider the common scenario where discovery processes return not only one but a set of candidate services. The most common scenario in discovery process results in a search within the published services using functionality requirements to obtain a set of compatible services. SWS discovery is usually performed by DLs reasoners, because semantic definitions are commonly based on this logic formalism.

The next step on SWS provisioning is to actually select the best service that fulfills the user requirements. In opposition to discovery, selection processes focus on quality of service (QOS) requirements, i.e. non-functional ones. Using this type of requirements, the set of discovered services are ranked so the best service can be chosen [77]. Because QOS requirements serve to state an order of preference, they are usually referenced as *user preferences*. Besides, user preferences transform the selection process into an optimization problem, so discovery techniques cannot be applied in this case.

In conclusion, SWS provisioning is performed using both functional and QOS preferences, so it conforms a QOS-aware process. Specifically, the use of

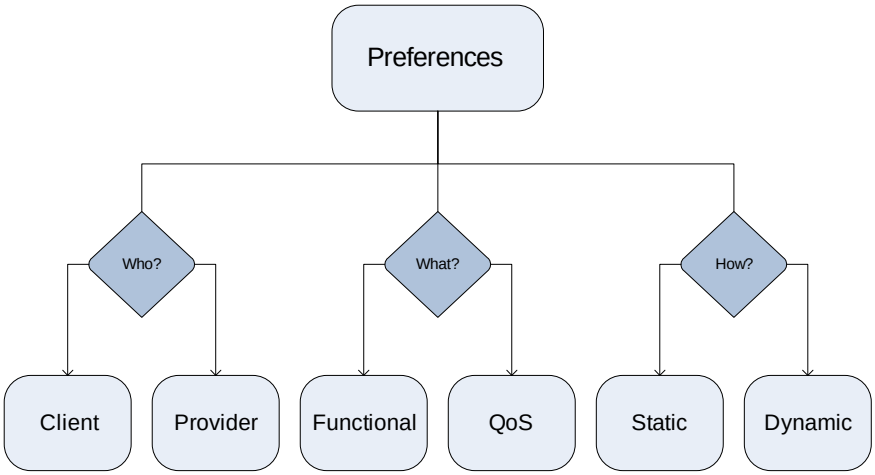


Figure 1.7: *User preferences categories.*

QoS user preferences allows to perform a QoS-aware selection of SWS, that constitutes the focus of our research work.

1.1.5 User Preferences

User preferences define the optimality criterion that is applied when selecting a service among a set of candidates. These preferences usually refer to QoS requirements from the client [10], but can be also defined by a service provider, for instance. Preferences are transformed into optimization problems that are able to rank SWS.

It is possible to define several types of user preferences attending different categories, presented in Fig. §1.7 . A selection algorithm have not to handle every type of preference, but at least QoS preferences have to be supported in order to perform a QoS-aware selection. Moreover, preferences may fall in more than one of the categories, depending on *who* define the preferences, *what* area of the description is about, and *how* their values can be checked. Each category is defined in the following:

Client preferences refer to the requirements of the client of the desired service. Because this kind is the most commonly supported, it is frequently referred just as user preferences.

Provider preferences state requirements from the service provider part, in case it has constraints or preferences about its clients.

Functional preferences restricts the functionality of the desired service. Usually these preferences are only used at the discovery stage.

QoS preferences define the preferred values of QoS parameters for a user.

Static preferences refer to parameters whose values can be predicted before performing the QoS-aware provisioning process.

Dynamic preferences are defined on parameters whose values have to be checked while performing the service provisioning.

Finally, there are different formats to define user preferences, depending on the desired level of expressiveness, and usually coupled with the concrete technique that performs the selection process. Thus, user preferences can be described using one of the following representations [10]:

Enumeration All the possible parameter combinations are enumerated in the preferred order.

Preference Relations A directed graph where each node represent a state or a value and the edges indicate which states are preferred to a given node.

Worth Functions It operates on the representation of a QoS parameter, and delivers as a result value the user preference regarding that parameter.

Mapping to Vectorspaces QoS parameter values is related with a point in a vector space, and by means of user-defined metrics, the preference can be stated as the distance between points in the space.

Preference Language It maps QoS parameter values or ranges to a qualitative specification.

Additionally, when preferences are defined using more than one parameter, it is necessary to establish relative weights between the preferences of each parameter, to obtain the global preference. Thus, the concrete format to define weights have to be defined too.

1.2 Hypothesis

Our research work takes current approaches on service provisioning as its starting point. After a thoroughly review of the state-of-the-art on this topic,

focused on SWS, we found out that there are some extensions to current SWS frameworks that enable the description of QoS preferences. Furthermore, these extensions are being used in provisioning processes, specially in selection.

However, user preferences are strongly coupled with the selection technique used in each proposal, so there are not a common preferences ontology. This causes functional preferences (already defined by SWS frameworks) and QoS preferences are not connected, even the latter being described without semantics.

Our hypothesis is precisely the previous statement: there is a semantic gap between functional and QoS preferences that have to be solved, extending current SWS ontologies with a QoS ontology that allows the user to define both functional and QoS preferences at the same level. Furthermore, the actual selection technique has to be independent of the representation of QoS preferences, so there is a need for a novel approach to SWS selection, and in extension, to QoS-aware SWS provisioning. Our research work presented in this report consists on an analysis of current approaches in order to motivate a solution that prove our hypothesis.

1.3 Report Structure

The rest of this report is structured as follows. In Chapter §2 current SWS frameworks are introduced, in addition to a brief comparison between them. Then, in Chapter §3 a state-of-the-art study of QoS-aware provisioning of SWS is presented, describing and comparing related work about discovery, selection and user preferences. At the end of this Chapter, a global analysis of the state-of-the-art is discussed, focusing on how different proposals approach the QoS provisioning. Chapter §4 gives the conclusions of the research currently done, enumerating hints about future work. Finally, in Appendix §A relevant contributions already published or submitted are enclosed, and in Appendix §B the curriculum vitae of the author of this report is included.

Part II

Related Work

Chapter 2

Semantic Web Services

The collusion of the Semantic Web technologies into the Web Services world results in the Semantic Web Service technology. Essentially, what we get is the ability to extend current Web services with semantic information about its operations. In this section, the three main alternatives that exist nowadays in the Semantic Web Services world are introduced. WSMO, OWL-S *and* METEOR-S *are described in detail, along with some other interesting frameworks that are also briefly introduced.*

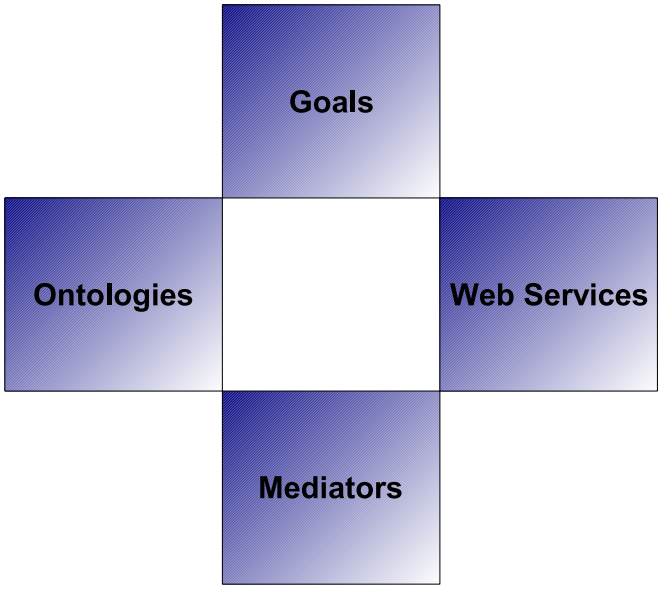


Figure 2.1: WSMO *core elements.*

2.1 WSMO

WSMO stands for *Web Service Modeling Ontology*, and it is an European initiative that try to develop SWS [75], as the American proposal OWL-S does (*cf.* Sec. §2.2). The aim is put in providing a successful standard to describe SWS. The starting point of the authors is the Web Service Modeling Framework (WSMF) [31], that has been refined and extended, developing a formal ontology (WSMO itself) and a specification language (Web Service Modeling Language or WSML) [23]. In addition, there is a working subgroup developing a reference implementation, known as WSMX, which stands for *Web Service Execution Environment*, providing an appropriate playground to test and use Semantic Web Services.

The main elements for describing WS as it is enumerated in WSMF are: (1) ontologies that provide the terminology used by the rest of the elements, (2) goals that describe the requirements of the service client, (3) Web Services descriptions that define various aspects related to a WS, and (4) mediators whose aim is to achieve interoperability problems. WSMO provides means to specify these core elements of SWS (Figure §2.1), extending and refining the definitions given in [31]. These elements are more precisely defined below:

Ontologies Represent the most important element in WSMO, because they

provide terminology for describing all other elements, from a specific domain. An ontology is a set of formal definitions of concepts from certain domain (*cf.* Sec. §1.1.2). A WSMO ontology is defined by its non-functional properties, mediators which are used to solve interoperability problems between ontologies which are interconnected, and obviously ontology items definitions, *i.e.* concepts, relations, axioms, functions and instances.

Web services Connect applications with each other using standard Web-based protocols to exchange data and/or generate new data. Web services are platform independent and can be combined in order to provide a complex functionality composed of other services that provide a specific and atomic piece of functionality. In WSMO, WS are described from three different points of view: functionality, behavior and non-functional properties, in order to allow automatic accomplishment of tasks like discovery, invocation, composition and execution, among others. More precisely, a WSMO Web service is defined by the ontologies supporting its terminology, the mediators it is using, its capability and its interfaces, in addition to its non-functional properties. The *capability* of a WS refers to its functionality defined in terms of pre and postconditions, assumptions about the world state prior to the execution and the effects that execution causes in the world. On the other hand, the interface of a WS provides information about the behavior of the service, meaning its choreography and orchestration. While a WS could have more than one interface, it has one and only one capability.

Goals Specify aspects related to the requested functionality of a WS, from the user's point of view. In WSMO, goals is characterized by a set of non-functional properties, the ontologies and mediators used, the requested capability and the requested interface, if needed.

Mediators They are one of the key elements of WSMO. They try to overcome structural, semantic or conceptual mismatches that appear between WSMO components, i.e. interoperability problems. Currently, there are four existing types of mediators in the specification: *ooMediators*, which solve representation mismatches between a source ontology and another target ontology; *ggMediators*, which connect related goals possibly resolving mismatches; *wgMediators*, that link Web services to goals and resolve mismatches; and *wwMediators*, which allow collaboration between several WS. A mediator, whatever type it is, is defined by source and target components, imported ontologies, non-functional properties, and the mediation service doing the mapping, in form of a goal, a Web service, or another *wwMediator*.

There is a common element used to describe the core elements in WSMO, that is non-functional properties. In its last version, WSMO differentiates two kinds of these properties. On the first hand there are annotations, that declare general information about each element, like its creator, description, version, etc. On the other hand there is proper non-functional, QoS properties, like cost-related properties, performance, reliability or security, among others. The latter can be applied to mediators, goals, Web services, capabilities and interfaces, while the former can also be applied to ontologies, apart from the said elements.

The design of WSMO is organized around several principles taken from Web applications, the Semantic Web and design principles for distributed, service-oriented computing on the Web [30]. It could be said that the framework is centered in two complementary design principles [31]:

Strict decoupling of the various resources, meaning that each resource is specified independently from the others, without regard of possible interactions with other resources.

Strong mediation enabling anybody to speak with anybody, regardless of different terminologies or interaction styles. So the heterogeneities that naturally arise in open environments are addressed by mediators, which are a core element of WSMO.

The four core WSMO elements are described more detailed in the following sections.

2.1.1 Ontologies

In WSMO, ontologies provide machine-readable semantics for the information used by all actors implied in the Web service scenario, either providers or requesters, allowing interoperability and information interchange among components. These ontologies, as well as the rest of the core elements, are defined in separate files using WSML, modularizing the definitions.

Firstly, a namespace declaration can appear at the beginning of each WSML file. These declarations are declared similar to XML namespaces, so they comprise the default namespace and abbreviations for the rest of the namespaces used. If there are shared namespaces between different WSML files, it is possible (and desirable) to use the same abbreviations among the files.

Ontologies define a common vocabulary from a domain, so it can be used by both providers and requesters to define all the necessary WSMO elements. This vocabulary are composed of concepts and relationships between them, among other elements to capture all the semantic properties of them, like axioms. In addition, WSMO allows the importing of other ontologies either directly or using mediators. In the first case it is supposed that the ontologies do not have conflicts between them, but in the second case, the mediation process resolves any conflicts through aligning, merging or transforming imported ontologies. These mediators are discussed in Section §2.1.4 .

The building blocks of an ontology are concepts, relations, functions, instances and axioms.

2.1.1.1 Concepts

Concepts are the basic elements of the agreed terminology for some problem domain. They are defined by their hierarchy and their attributes. Concept hierarchy explicits *is-a* relations between a concept and its super-concepts (possibly none), enabling concept subsumption.

Attributes are defined by their names and ranges. This range can be a simple datatype or another concept. In fact, ranges define constraints on the values that attributes can have in certain concept instance. Actually, these are typing constraints on the values of the attribute.

The extension of a concept, i.e. the set of possible instances, can be defined or restricted using one or more logical expression. These expressions may specify necessary and/or sufficient conditions for instance membership in the extension of the concept.

2.1.1.2 Relations

Relations describe interdepencies between several concepts and, consequently, between instances of these concepts. Like concepts, it is possible to define a set of relations being super-relations of a given relation. In this case, the sub-relation inherits the constraints defined in its parents. Furthermore, the set of tuples belonging to a relation (the extension of the relation) is a subset of each of the extensions of the corresponding super-relations.

The extension of a relation is defined or constrained by logical expressions using the corresponding concepts, specifically their attributes. In addition, parameters can be declared in an attribute fashion to be used in the definition of

the extension of a relation. The domain of these parameters can be a datatype or a concept.

2.1.1.3 Functions

A function is a special type of relation that have an unary range and a n-ary domain in the form of the set of parameters inherited from the relation. So a function defines a special parameter *range* that is used in the logical expresion, which states the functional dependency, as the return value.

2.1.1.4 Instances

Instances are concrete materializations of concepts and relations. They can be defined either explicitly or by a link to an instance store. In the first case, concrete values are specified for attributes or parameters. These values have to be compatible with the corresponding type declared in the concept or relation definition. Generally, instances defined explicitly are those that are shared together with the ontology.

However, most instances exist outside the ontology, in instance stores. That is the case of instances defined by the actual provider of a Web service, dependent to its policy.

2.1.1.5 Axioms

An axiom is a logical expression that have to be comprised. Axioms help to formalize domain specific knowledge. These logical expressions, which are used in almost every element of the WSMO model, are defined formally in the WSML specification [23], and their expressiveness depends of the concrete WSML variant used to model the service.

2.1.2 Web Service Descriptions

A Web service is a computational entity which is able (by invocation) to achieve users goal [75]. A service, in contrast, are the actual value provided by this invocation. Thereby a Web service might provide several services.

In WSMO, Web services are described by (1) their imported ontologies, (2) their used mediators to achieve interoperability with conflicting ontologies and/or to deal with process and protocol mediation, (3) a listing of non-functional properties defined by logical expressions, (4) a capability and (5) interfaces. Apart from these, as every WSMO element, it has the possibility to include meta-information via annotations.

2.1.2.1 Capability

A capability defines the Web service funtionality in terms of pre and post-conditions, assumptions and effects. For each Web service described in WSMO there is one and only one capability describing its functionality.

As other WSMO elements, capabilities import ontologies to be used in their definition, either directly or via used mediators. These mediators can also be used in this case to link a capability, therefore a Web service, with a goal, using a *wgMediator* (*cf.* Sec. §2.1.4). Moreover, annotations and non-functional properties can also be defined in capabilities.

The basic blocks which define a Web service functionality are expressed through a set of axioms possibly using a set of shared variables declared within the capability. These axioms can be defined in one of the following elements:

Precondition specifying conditions about the information space before the execution of the Web service.

Assumption which describes the state of the world before the execution of the Web service.

Postcondition describing the information space of the Web service after its execution.

Effect that describes the state of the world after the execution.

The shared variables can be used through all these elements, and they are implicitly all-quantified. Thus, informally, the logical interpretation of a Web service capability is: for any given values of the shared variables, the conjunction of the precondition and the assumption implies the conjunction of the postcondition and the effect.

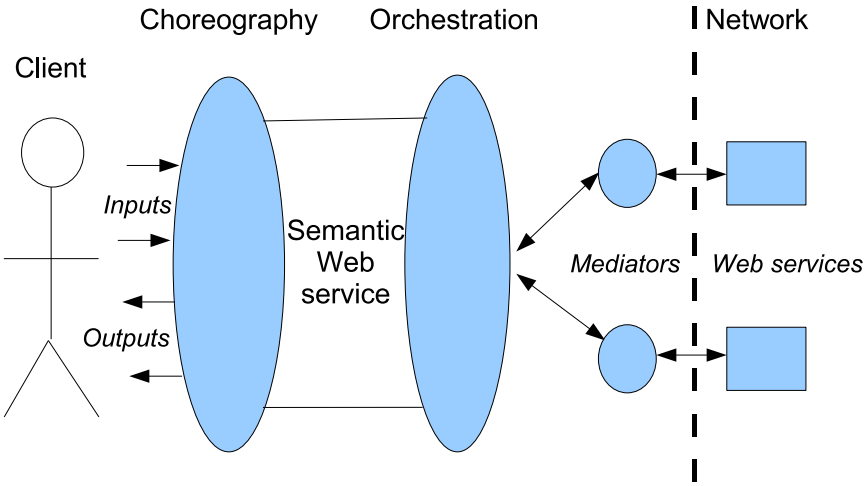


Figure 2.2: *Components of a* WSMO *interface.*

2.1.2.2 Interface

An interface describes how the functionality of the Web service is achieved, that is, it describes behavioral aspects of the Web service. It contains two views on the behavior of a Web service:

Choreography describes the communication pattern that allows to a client to interact with the Web service.

Orchestration decomposes a capability in terms of functionality required form other Web service providers, so the overall functionality is achieved cooperating with them.

This two interface components are depicted in Figure §2.2 , and detailed in the following.

Choreography The choreography specifies the behavior interface that a Web service consumer has to support to actually consume the Web service. That is, it states the information that a client needs to communicate with the Web service. A choreography description consists on two elements: the state and the guarded transitions. The former are represented by an ontology, while the latter are if-then rules that specify transitions between states.

The ontology modeling the state provides the terminology to express the transition rules, and contains the set on instances that change their values from one state to another. Thus, it is a changing ontology. In this scenario, a state is a dynamic set of instances at a certain point in time.

On the other hand, guarded transitions are rules that are triggered when the current state fulfills certain conditions. Thus, when the values of the attributes of certain instances in the state ontology meet come criteria or there are certain instances in the ontology in the current state, a transition to a new state is performed, possibly changing the values of some attributes of the instances, or even creating new instances in the state ontology.

Orchestration The orchestration defines how a WSMO Web service makes use of other WSMO Web services or goals in order to achieve its capability. As in choreography, an orchestration description has a state ontology and a series of guarded transitions specifying transition between states. In extension of the choreography, in an orchestration can also appear transition rules that have as their postconditions the invocation of certain mediator, linking the orchestration with the choreography of the desired Web service. This linking may be done directly using a *wwMediator*, if the wanted Web service is known, or via a goal when the Web service is not known (using a *wgMediator*).

2.1.3 Goals

A goal specifies objectives that a client want to be fulfilled by the execution of a Web service, i.e. goals are descriptions of the functionality a user need from a Web service. The existence of goals ensures the decoupling between requests and Web services. So, the requester defines a goal to be resolved and the Web Service Discovery search for suitable Web services that solve the goal automatically.

In order to describe a goal in WSMO, we can define its non-functional properties, annotations, imported ontologies and used mediators, as usual, besides of requested capability and requested interface. In this case, the used mediators allow reusing of ontologies: merging, aligning and transforming imported ontologies if necessary, but also it is possible to reuse one or several already defined goals to define other goals using goal-to-goal mediators (*ggMediator*). In the latter case, the relation between a goal and another goal imported via such a mediator is one of refinement.

The main element of a goal is the requested capability, which specifies the functionality required from a Web service. This is declared in the same way as

a Web service capability (*cf.* Sec. §2.1.2.1). In addition, the user may specify the desired way of interacting with the Web service by declaring the requested interface, in the same way.

2.1.4 Mediators

Heterogeneity is an inherent characteristic of open and distributed environments like the Internet. This fact hampers interoperability and make difficult an actual automatization of Web service tasks. Semantic Web services provides unambiguous semantics to information and resources, but interoperability problems still arise. Mediation is concerned with handling these problems resolving mismatches and providing interoperability, using a component named mediator [63].

Mediators are one of the key elements of the WSMO ontology, addressing interoperability. They link heterogeneous components, resolving incompatibilities at different levels:

- *Data level*, mainly addressing the problem of ontology integration.
- *Protocol level*, mediating between heterogeneous communication protocols between different Web services.
- *Process level*, linking heterogeneous business processes, that is, aligning the different WSMO interfaces descriptions for information interchange and cooperation between Web services.

WSMO mediators are defined by means of their non-functional properties, annotations, imported ontologies, source component, target component and mediation service, where source and target components can be a mediator, a Web service, an ontology or a goal, and the mediation service points to a goal that declarative describes the mapping or to a Web service that actually implements the mapping. The current WSMO specification defines four types of mediator, depending the type of top-level element it links and denotated by its initials: *ooMediators*, *ggMediators*, *wgMediators* and *wwMediators*.

2.1.4.1 OOMediators

OOMediators allow any WSMO element to import an ontology by solving all the terminology mismatches that can appear during the process. Ontology mediation may be done in various steps, so a *ooMediator* could mediate between an ontology and another *ooMediator*.

These mediators allows to reuse ontologies from different domains, even described in different ontological languages. They are concerned with data level mediation, and may be used inside the rest of the types of mediators to do said data mediation.

2.1.4.2 GGMediators

GGMediators can link goals between them, possibly in more than one step linking *ggMediators*, like in the *ooMediators* case. This is useful to explicit refinement relations between goals, so it is possible to define sub-goal hierarchies.

The concrete mediation technique is not well defined in WSMO specification yet, so it is only used to link a goal with its sub-goals. Also, in the data level, it is possible to use *ooMediators* within a *ggMediator*.

2.1.4.3 WGMediators

In order to link Web services to goals, *wgMediators* can be used. They resolve possibly occurring mismatches between goals and Web services, supporting Web service discovery. These mediators are concerned with data mediation, using *ooMediators* if necessary, and process mediation for communication, resolving mismatches between the Choreography interface definitions of source and target components.

WGMediators can be defined to address two different problems: (1) to link a goal to a Web service via its choreography interface, so the Web service should fulfill the linked goal, or (2) to link a Web service to a goal via its orchestration interface, so the Web service needs the corresponding goal to be resolved in order to fulfill its functionality.

2.1.4.4 WWMediators

WWMediators link two Web services in order to enable interoperability between heterogeneous Web services. These mediators can be used by an orchestrator in order to aggregate different Web services defined in an orchestration, resolving mismatches between them. As in the others, data mediation are achieved by using *ooMediators* within these mediators, which are concerned with process mediation for communication and for coordination.

2.1.5 Provisioning Approaches

In this section, some studied approaches on discovering and selection within WSMO framework are discussed. In Sec. §3 , these semantic provisioning approaches are further discussed and analyzed, among others based on the presented frameworks.

First of all, Keller *et al.* discuss the different approaches on WSMO discovery in [48]. They show three methods to perform discovery tasks in WSMO, based on terminologies, controlled vocabularies, or ontology-based rich descriptions. However, selection tasks are not properly defined within the framework, mainly because they have to be performed using other methodologies.

Wang *et al.* provide an extension to WSMO ontology to handle QoS parameters [87]. They define a QoS selection model and an algorithm based on a quality matrix that contains values of QoS parameters. The selection model are defined both from the perspectives of users and service providers. The discovery process is left to the underlying WSMO implementation, basically filtering services in terms of functionality (capabilities).

An approach that merges discovery and selection algorithms execution is presented by Vu *et al.* [86]. They show a QoS-aware discovery framework that takes QoS values of WS based on user feedback and perform the discovery process, ranking the services in terms of QoS compliance. Additionally, they sketch a scalable architecture, similar to WSMX, that can be deployed in a peer-to-peer network.

Finally, WSMO-MX is a hybrid matchmaker that uses several matching filters to perform the discovery process [47]. It extends WSML with Logic Programming features and recursively filter the set of services using different approaches. Selection is performed by the recursive computing of matching degree while the discovery process is being executed.

2.1.6 Existing Tools

There are plenty of tools available that work with WSMO. In this section we introduce some of them, giving their main characteristics and if they are currently actively developed. This way we pretend to known the actual support of this technology in the community.

2.1.6.1 WSMO Studio

WSMO *Studio* [26] is a Semantic Web Services editor compliant with WSMO. It is implemented as a set of *Eclipse* plug-ins that can be further extended by third parties. It is a successor of *SWWS Studio*, another WSMO compliant Semantic Web Service editor currently discontinued.

Indeed it is an application full of features, integrating other tools like *wsmo4j*, WSML *reasoner* and more from third parties, via plug-ins. It is available under LGPL license, so it can be used in commercial projects without license incompatibilities. Also it is a very active project, partly funded by several European research projects, being its last version released on 20th March 2007.

WSMO Studio main features include the following:

i. A visual editor for WSMO ontologies, web services, goals and mediators.

ii. A WSMO centric choreographies designer.

iii. An editor for adding semantic annotations to WSDL documents, SAWSDL compliant (*cf.* Sec. §2.3.2 .

iv. Import and export from WSML, a subset of OWL-DL, RDF and a XML representation of WSML.

v. An integrated WSML validator and a reasoner in order to check consistency of ontologies.

vi. A front-end to repositories of ontologies, services and goals, including a repository implementation and adapters to third party ones.

vii. A front-end to service discovery components, including an implementation of a QoS based discovery component.

viii. A WSML editor with syntax coloring.

ix. A graphical axiom editor by a third party.

Figure §2.3 shows a screenshot of one of the components of WSMO Studio: the visual concept editor. WSMO *Studio* is available from http://www.wsmostudio.org/.

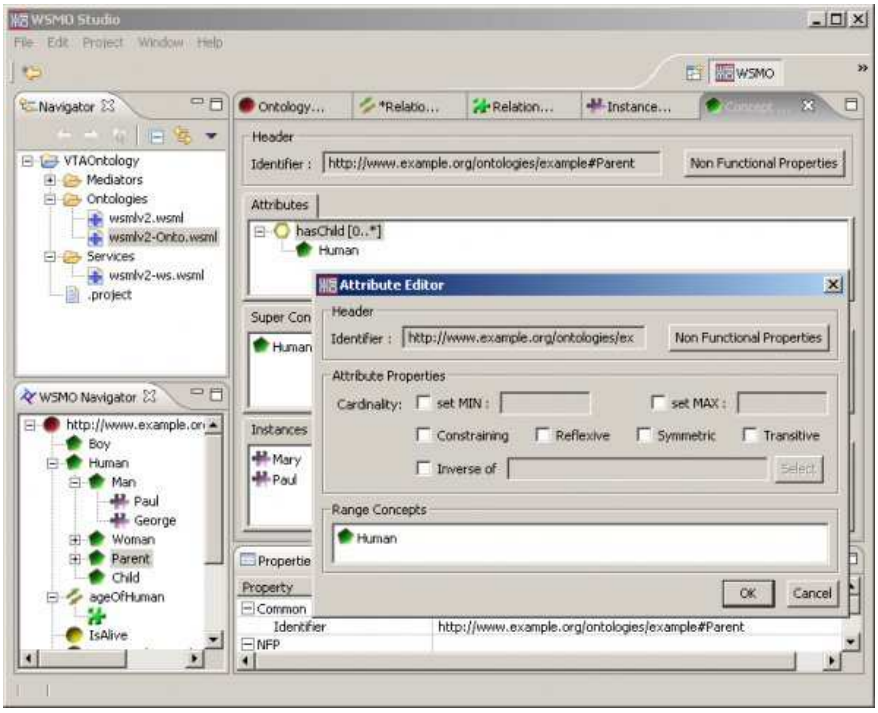


Figure 2.3: *WSMO Studio concept editor.*

2.1.6.2 wsmo4j

wsmo4j is an API and a reference implementation for building Semantic Web Services based on WSMO [25], written in Java programming language. It consists on a set classes representing all the concepts defined in WSMO, with methods to use them, in addition to grounding classes and a choreography API. *wsmo4j* is used by most of the rest of the tools that use WSMO and are written in Java. One of them is a simple WSML *Validator* that checks if an ontology is compliant with certain variant of WSML.

wsmo4j Web site is located at http://wsmo4j.sourceforge.net/, while WSML Validator is published as both Web service and Web site in http://tools.deri.org/wsml/validator/v1.2/.

2.1.6.3 WSML Reasoners

The *WSML2Reasoner* framework is a modular architecture that includes functionalities to validate, normalize and transform WSML ontologies to the

appropriate syntax of a set of reasoning engines. It is written in Java, and depends on *wsmo4j*.

At http://tools.deri.org/wsml2reasoner/index.html there are different releases available, depending on which reasoning engine is needed. The translations currently implemented translate from WSML-Flight to *KAON2* engine, WSML-Rule to *MINS* and WSML-DL to *Pellet*.

2.1.6.4 Web Service Execution Environment (WSMX)

WSMX (Web Service Modeling eXecution environment) [42] is a reference implementation of WSMO, and is a subproject of the WSMO initiative itself. It is an execution environment for business application integration using enhanced Web services. WSMX internally uses WSML.

The main objective behind WSMX project is to allow dynamic discovery, composition and invocation of Semantic Web Services. In addition, WSMX supports interaction with classical, non-WSMO Web services, so it ensures a seamless interaction with existing Web services. Furthermore, it can be used by both service providers and requesters, so it provides a complex architecture to support all these operations.

That architecture allows to register Semantic Web Services and to execute registered ones. To do so, it consists of several components that work together depending on the concrete task to be performed. WSMX has a compiler component to validate WSML documents, a repository of services, mathmakers and selectors to discover Semantic Web Services, a choreographer to execute them, etc.

WSMX is freely available as an open source project and can be downloaded completely or in a component basis from http://www.wsmx.org/ under GNU Public License.

2.1.6.5 The Web Service Modeling Toolkit (WSMT)

Formerly an editor component of WSMX, the Web Service Modeling Toolkit (WSMT) is a collection of tools for Semantic Web Services intended for use with WSMO, WSML and WSMX [49]. It is now available from *SourceForge* at http://sourceforge.net/projects/wsmt. Its aim is at ease the creation and deployment of tools to handle Semantic Web Services

This set of tools includes WSML editors with syntax highlighting and reasoners, a WSMX manager and a data mediation mapping tool, in addition to

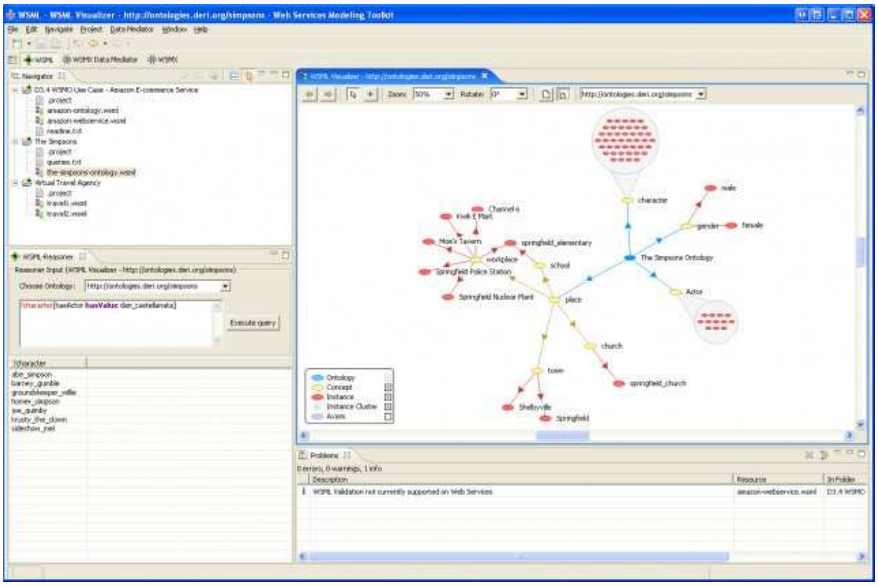


Figure 2.4: *The* WSML *Visualizer and* WSML *Reasoner.*

a graphical visualizer of WSML documents, shown in Figure §2.4 . All this tools are implemented using the Eclipse plug-in architecture, so it is easier to develop an Eclipse application like WSMO Studio.

2.1.6.6 Ontology Management Working Group Tools

Finally, a set of tools for manipulating WSMO ontologies have been developed by the Ontology Management Working Group from DERI. Its main result is the DOME tool, that consists in an ontology editor.

The DERI Ontology Management Environment (DOME) is also done using the Eclipse plug-in architecture, and consists in tools supporting editing, browsing, versioning, evolution, mapping and merging ontologies. All this tasks are done using a graphical interface, so the user does not have to write ontologies manually.

2.2 OWL-S

The second main alternative in SWS is supported by the DARPA Agent Markup Language program of the American Army. OWL-S is a joint initia-

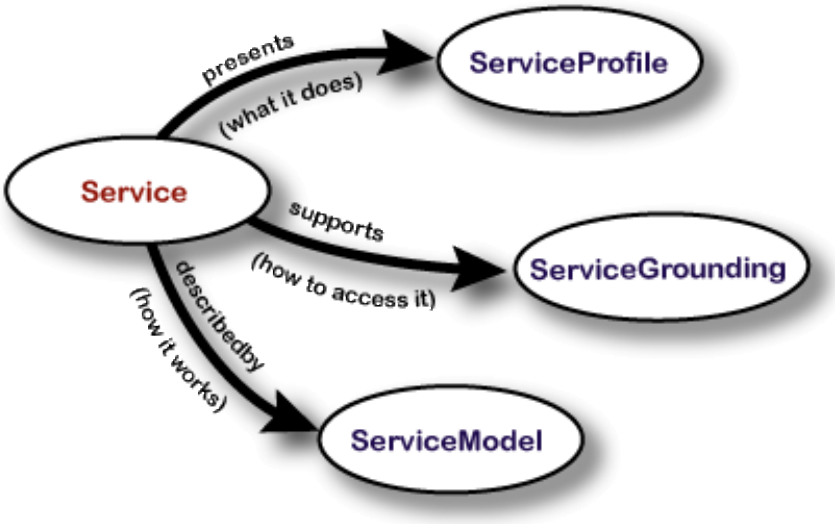


Figure 2.5: OWL-S *upper ontology for services.*

tive of companies and institutions like BBN Technologies, Carnegie Mellon University, Nokia, Stanford University, SRI International and Yale University, among others, and its aim is to develop an ontology to semantically annotate WS. The project objective is to model the WS concept, its operations and its process model defining concepts and establishing relations between them in order to allow automatic reasoning using an inference engine, enabling automatic discovery, invocation, composition and monitoring of WS.

As opposed to WSMO, OWL-S uses a standard language to define ontologies, *i.e.* OWL, so it can benefit from the wide range of tools developed for this language, e.g. editors, reasoners and verifiers. Current published version of the proposal is the 1.1, although it is constantly improved and version 1.2 is been developed at the moment. Previously, OWL-S were released with the name of DAML-S, and were built upon DAML+OIL (a predecessor of OWL).

The OWL-S upper ontology is shown in Figure §2.5 , taken from [58]. As it can be seen, the concept where we start from is the *service*. Services in OWL-S are "Web sites that do not merely provide static information but allow one to effect some action or change in the world, such as the sale of a product or the control of a physical device" [58]. This is a wide definition, so a simple Web site to book flights could be a service in that context. However, the main target of OWL-S is to enable automatic discovery, invocation, composition and monitoring of WS, as we have said before.

In order to completely describe a service, three different ontologies are used: *ServiceProfile*, *ServiceModel* and *ServiceGrounding*. Each one answers a question about the service (Figure §2.5).

Firstly, *ServiceProfile* answers to the question "what the service does?". Here we define information about the provider entity of the service, the required information for the execution of the service, and the output it provides. Furthermore, we can define QoS characteristics about the service provider.

Secondly, *ServiceModel* answers to the question "how the service works?". In this ontology we define the business protocol so the client could interact with the WS correctly. Moreover, processes taking part in the execution of the service, their preconditions and effects are described in this ontology. With the service model, the client may be able to understand the provider operations, so it can (1) conclude that the service fits their needs, (2) allow proper interaction with the service, (3) enable the coordination of several processes, and (4) allow the monitoring of the service execution. Finally, *ServiceGrounding* tells the answer of the question "how to access the service?". In this ontology the abstract entities defined in the *ServiceModel* were mapped to specific communication mechanisms. Particularly, OWL-S provides a grounding to WSDL, but there is no limitation in defining others.

To sum up, the *ServiceProfile* is used at a discovery and selection stage. The *ServiceModel* tells the client the steps it has to follow in order to execute the service correctly, and the information it has to provide. Finally, the *ServiceGrounding* is used to invoke the service, generally using SOAP and WSDL.

Using the *Service* class, the three different descriptions enumerated before are brought together. For each published service by a provider we get an instance of the *Service* class consisting in three properties: (1) *presents*, whose domain is the *ServiceProfile* class; (2) *describedBy*, whose domain is the *ServiceModel* class; and (3) *supports*, whose domain is the *ServiceGrounding* class. In this way, a service is characterized by instances of these three classes. There are two cardinality constraints for the former relationships: a service can be described by at most one service model, and a grounding must be associated with one and only one service. There is no specification about the cardinality for the *presents* and *describedBy* properties, so it is possible to define services without a corresponding model or profile. Unfortunately, in that case, services cannot be invoked or discovered by the client, respectively.

In the next subsections the three enumerated ontologies to describe SWS in OWL-S are analyzed in detail, in addition to the usage of each one.

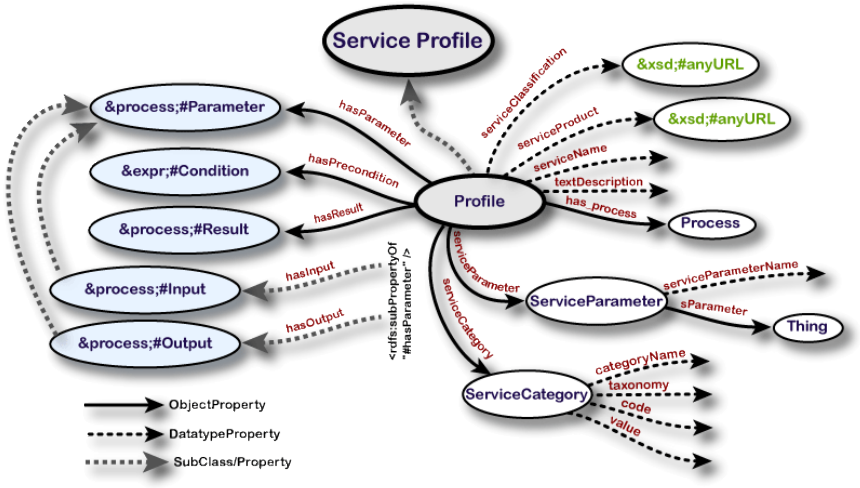


Figure 2.6: *Selected classes and properties of the service profile.*

2.2.1 Service Profiles

The service profile serves the purpose of giving a high level description of functionality and non-functional characteristics of a given service. It is aimed to enable discovery and selection of Web services, normally querying a service directory like UDDI. Although the integration of a service profile in UDDI is not explicitly defined in the OWL-S specification, there are some proposals in that way [59].

The *ServiceProfile* base class (Figure §2.6 , taken from [58]) allows to define three different aspects related to the service: (1) information to a human reader, like contact data or service description; (2) service functionality; and (3) categorization and additional parameters about the service.

The name of the service being offered is specified via *serviceName* property. In addition, it is possible to include a short description within the *textDescription* property. Finally, the *contactInformation* property can be used to provide information about the supporters of the service. The range of the last property is unspecified in OWL-S, so it is allowed to use some other ontology to specify this contact information, such as *FOAF* or *VCard*. This three properties are more suitable for a human reader than for an automatic processing.

Functionality description is presented through transformations that the service execution apply to input parameters and to the world. On the first hand, *hasInput* and *hasOutput* properties are used to specify the necessary input for the service execution, and the output obtained from that execution.

These two properties are ranged as subclasses of the Parameter class from the Process ontology (*cf.* Sec. §2.2.2). On the other hand, *hasPrecondition* and *hasResult* properties set the service preconditions for a successful execution, and how this affects the world. There is no way to specify inputs, outputs, preconditions and results in the *ServiceProfile* ontology, but these properties range over corresponding classes from the *ServiceModel* one. This is done because, in general, service inputs, outputs, preconditions and results are a subset of the model ones, so the profile should only point to these instances. However, if they are disjoint with the model ones, it is possible to create new ones using the *ServiceModel* classes and refer to them.

Finally, the *Profile* class also provides properties related to non-functional and QoS characteristics. There are two different mechanisms to do the former. On the first hand, we can specify the service category inside a given taxonomy of categories. In this case, we use the *serviceCategory* property, whose range is the *ServiceCategory* class. On the other hand, we can set any additional parameter, like QoS or geographical constraints, using the *serviceParameter* property, which ranges over the *ServiceParameter* class. Parameters are defined as instances of this class, which consists of two properties: (1) *serviceParameterName*, to identify the name of the actual parameter; and (2) *sParameter*, which points to the value of the parameter within some OWL ontology.

To sum up, the service profile contains characteristics about the service in order to allow automatic discovery and selection. The service model specifies the necessary interaction between client and provider in order to execute the service. Obviously, there is a relation between them, so inputs, outputs, preconditions and results described in the profile have to be indicated in the model, too. However, OWL-S specification does not explicitly dictate that both descriptions must be consistent. Because of this, it is possible to have a profile describing a service to book flights connected to a model describing an interaction of a book selling agent, for example. In this case, the interaction between client and provider will break at some point. So it is convenient that both descriptions would be consistent in order to successfully execute the service. That does not imply that the profile has to describe the whole model. That could be the case of a provider not wanting to publicly publish all the functionality of a service.

2.2.2 Modeling Services as Processes

The service model description serves three purposes. Firstly, it gives information to a client about how to correctly interact with a service. Secondly, it gives a better knowledge about a service to distinguish if it accomplish a

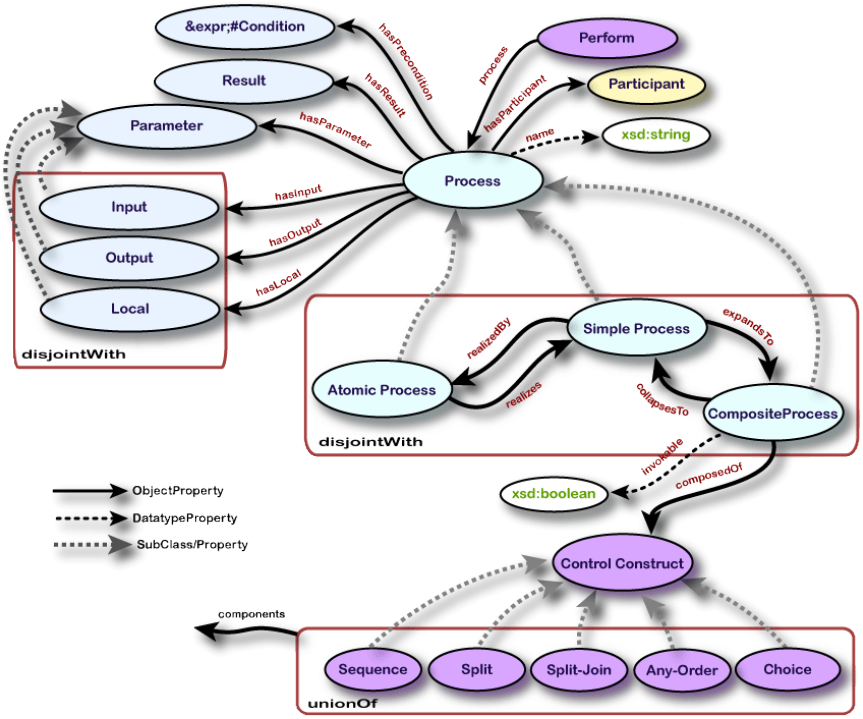


Figure 2.7: *Simplified Process ontology.*

client's needs. Finally, it allows coordination tasks between other services, because the client knows the results of each step in the interaction.

Figure §2.7 , taken from [58], shows the basic scheme of how to define the OWL-S service model. To do so, OWL-S define the Process class, subclass of ServiceModel, which is the used to model the process.

2.2.2.1 Processes

The main class in this ontology is *Process*. A process has inputs, preconditions, outputs and results, among other properties. Additionally, the last two can have conditions associated to them, i.e. certain circumstances may appear in a process execution, generating an output or some results, and other circumstances may generate different outputs and results. To define them, we use the *hasInput*, *hasOutput*, *hasPrecondition* and *hasResult* properties, whose domains are instances of *Input*, *Output*, *Condition* (an expression, in fact) and *Result* classes, respectively.

Inputs and Outputs Process inputs and outputs are equivalent to inputs and outputs in a function defined in any programming language. For each process, we indicate its inputs and outputs as instances of *Input* and *Output* classes. These are subclasses of *Parameter*, so each instance has a *parameterType* property to indicate the parameter type.

Preconditions and Results Process execution may involve changes in the current state of the world. An example could be to do a charge in a client's credit card. These state changes are described using preconditions and results. On the one hand, Preconditions are circumstances that have to be met in order to correctly execute the process. On the other hand, results are a coupled output and effect, that may occur depending on the fulfillment of certain conditions. The difference between an output and an effect, belonging to a result, is that the latter express conditions that become true if the result conditions are true, while the former describe information to be returned by the process if the said conditions are met.

Expressing Conditions and Effects We need to express conditions defining the process model in several places: results, effects, loops or conditional sentences, and process preconditions. These conditions are logic formulas evaluating true or false. The problem is that there is no standard way to describe these conditions in OWL. Although there are some proposals in this aspect, like KIF or SWRL, none of them have been included in OWL standard yet. So, the solution adopted in OWL-S is to give freedom about the language used to specify conditions, transferring the problem to the specific implementations of OWL-S reasoners.

2.2.2.2 Types of Processes

There are three different kinds of processes in OWL-S, as seen in Figure §2.7 : (1) atomic processes, (2) simple processes and (3) composite processes. Atomic processes are supposed to run in a single step and not to have any subprocesses. Furthermore, they can be invoked directly and for each atomic process there is a grounding defined in order to allow a successful invocation. In general, an atomic process corresponds to a operation defined in WSDL of a web service.

Simple processes cannot be invoked, so they do not have any associated grounding. Conceptually, they have also single-step executions, like atomic

ones. Therefore, the actual objective of simple processes is to provide an abstract mechanism to simply represent a composite process. They are linked through the *expandsTo* property. It is also related with an atomic process via *realizedBy* property.

Finally, composite processes are those which can be separated in severals processes. These processes may also be composite processes. Each composite process is compound (*composedBy* property) of an instance from *ControlConstruct* class. OWL-S specification defines a minimal set of control constructs which can be extended in order to describe a wide range of Web services. Noteworthy that these constructs do not compound a behavior a service will do, but a behavior the client can perform by sending and receiving messages. Furthermore, each control construct is compound (*components* property) of other control constructs or processes (named process components). Following are the list of control constructs defined by OWL-S:

Sequence represents a list of control constructs (or processes) that have to be done in order.

Split is compound of a bag of process that are executed concurrently. There is no established synchronization.

Split+Join is like the later but it is possible to define partial synchronizations between processes.

Any-Order allows the processes defined in a bag to execute unordered, and not concurrently.

Choice calls for the execution of a single control construct from a given bag of control constructs.

If-Then-Else has the semantics of "test *ifCondition*; if it is true do *then*, if it is false do *else*". *ifCondition*, *then* and *else* are properties of the construct.

Repeat-While is the classical loop presents in programming languages. This construct test if *whileCondition* property is evaluated to true. In this case, it executes the process pointed by *whileProcess* property. In other case, it exits.

Repeat-Until is like the previous but with semantics of *until*. Due to the similarity, this and the previous one are defined as subclasses of *Iterate* abstract class.

2.2.2.3 Parameter Bindings

Finally, it is possible to link parameters of the processes from one step to another, binding their values through the execution of the process. Thus, the input of one process component may be obtained as one of the outputs of a preceding step. This is done using *Binding* elements, which can explicit relationships between parameters, among some other classes and attributes, as the *ValueOf* class which have two parameters, *theVar* and *fromProcess*, that indicate the value that an instance of the said class take from a concrete process.

2.2.3 Accessing Services

In the last subsection we have defined the service model abstractly. We have said that atomic processes are executed in a single step and can be invoked. However, the client cannot invoke that processes because he or she does not know how to. That is, the client need to know how to access the service. That information is given by the *ServiceGrounding*, mapping the abstract semantic representation of the messages from the *ServiceModel* to the syntactic form of these messages during the actual information interchange.

At the moment there is only one grounding defined between the service model and the WSDL specification. However there is no limitation in the grounding being linked to Web service invocations. It is possible to develop a *ServiceGrounding* ontology to support invocations through DCOM or CORBA, for example.

Figure §2.8 , taken from [58], shows the relationship between a OWL-S atomic process and a WSDL operation, and between their parameters. In fact, these relationships are the ones that have to be modeled in the *ServiceGrounding*. Specifically, it is necessary to map an atomic process with one or more WSDL operations, the input and output of an atomic process with a WSDL message, and input and output types with WSDL abstract types. In the last two cases, we can use XSLT transformations to show how to derive each WSDL input from one or more OWL-S inputs, and how to get OWL-S outputs from some parts of a WSDL output message.

The way to do the mapping is as follows. We have to add the next extensions to the WSDL document:

i. We have to add the *owl-s-parameter* attribute in some places in the WSDL message definition in order to specify the corresponding instance of *Input* or *Output* OWL-S classes.

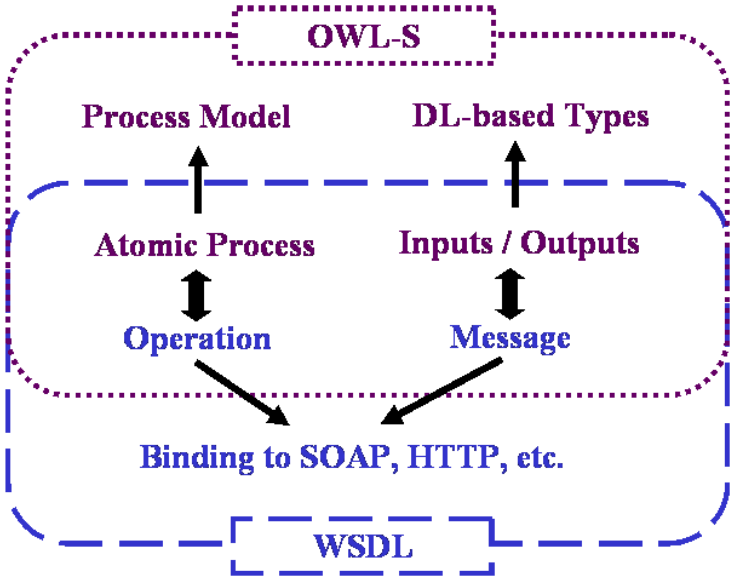


Figure 2.8: *Relationship between* OWL-S *and* WSDL.

ii. When the message part uses an OWL type, we have to set the *encodingStyle* attribute inside the *binding* WSDL element to a value like http://www.w3.org/2002/07/owl to indicate that message parts will be serialized as OWL instances.

iii. For each WSDL *operation* element, we have to add the *owl-s-process* attribute indicating the atomic process name which corresponds to the operation.

Finally, we have to describe the grounding as an instance from the *WsdlGrounding* class, which extends *ServiceGrounding*, and have a parameter named *hasAtomicProcessGrounding* which ranges over instances from the *WsdlAtomicProcessGrounding* class. Each of these instances presents the mapping between an atomic process and its corresponding WSDL operation and its parameters. The actual form of the mapping is out of this report scope, and there are tools that do it automatically from a WSDL file.

2.2.4 Provisioning Approaches

Several provisioning proposals are based on OWL-S or its precursor DAML-S. In this section, some of them are introduced to show different alternatives to perform discovery and selection within this framework. Further discussion of these and more proposals can be found in Sec. §3 .

In the context of DAML-S, Sycara *et al.* show how semantic information allows automatic discovery, invocation and composition of Web Services [83]. They provide an early integration of semantic information in a UDDI registry, and propose a matchmaking architecture. The matching engine matches the user requirements with "sufficiently similar" provided services. Selection can be performed by scoring the matching degree between the requirements and the services discovered.

An extension to DAML-S to include QoS profiles is proposed in [89] by Zhou *et al.* This proposal only allows order conditions between QoS parameters, so it performs discovery and selection using DLs. The QoS ontology is simple and can be easily linked to the DAML-S service profile. Additionally, its selection algorithm uses matching degrees to rank the resulting set of services.

Dobson *et al.* presents QoSOnt in [27], which is another ontology that extends OWL-S to describe QoS attributes and metrics. Although they do not explicitly explain how to perform discovery and selection and their proposal suffers from OWL limitations, their proposal adds QoS support to OWL-S so that QoS-aware provisioning mechanisms can be defined upon QoSOnt.

Another DAML-based proposal is also presented in [82], where Bilgin and Singh provide a DAML-based query language, instead of just extending OWL-S. Using this Semantic Web Services Query and Manipulation Language, they advertise QoS attributes and perform the selection. They propose the use of UDDI registries, which have to be extended to support this discovery process, though selection process is not flexible.

2.2.5 Existing Tools

The OWL-S community has also developed several tools that bring to practice the specification. That amount of available tools is a good signal about the impact that OWL-S has in the research and industrial community. In the following we introduce some of them that we have found interesting.

2.2.5.1 OWL-S Protégé-based Editor

The OWL-S Protégé-based Editor is an open-source project that is implemented as a plug-in for the knowledge base editor Protégé Ontology Editor. It allows the user to create and maintain OWL-S service descriptions with a user-friendly interaction organized around the conceptual structure of OWL-S.

Its main features includes graphical editing of control constructs, visualization of documents as a graph, consistency and links checks between elements of the OWL-S description, and WSDL support in order to generate a basic OWL-S description based on a WSDL document.

The OWL-S Editor is available at http://owlseditor.semwebcentral.org.

2.2.5.2 OWL-S Matcher (OWLSM)

OWLSM is a matcher for elements of OWL-S descriptions [46]. The algorithm implemented in this matcher considers elements of the service profile. Ranking a criterion, OWLSM can select a service among a large set of results, so after the matchmaking, the matching services are ordered after a criterion supporting an automatic decision of the best service possible.

OWL-S Matcher is implemented in Java, and it allows to select two OWL-S descriptions for requester and provider, executing the algorithm on them. It is licensed under LGPL and it is available at http://owlsm.projects.semwebcentral.org/.

2.2.5.3 Semantic Web Author

Semantic Web Author is a validating parser, editor and Web development environment of multiple markup languages (XML, RDF and OWL). It is a Windows application programed in C#, so it requires MS .NET Framework. It is available at http://www.web-iq.com/SemanticWebAuthor/SWA.zip.

2.2.5.4 OWL-S Editor

OWL-S Editor [78] is an application intended for users without experience that want to create OWL-S descriptions in a short time. The tool features consists in: (1) a creator of OWL-S descriptions, using a wizard that

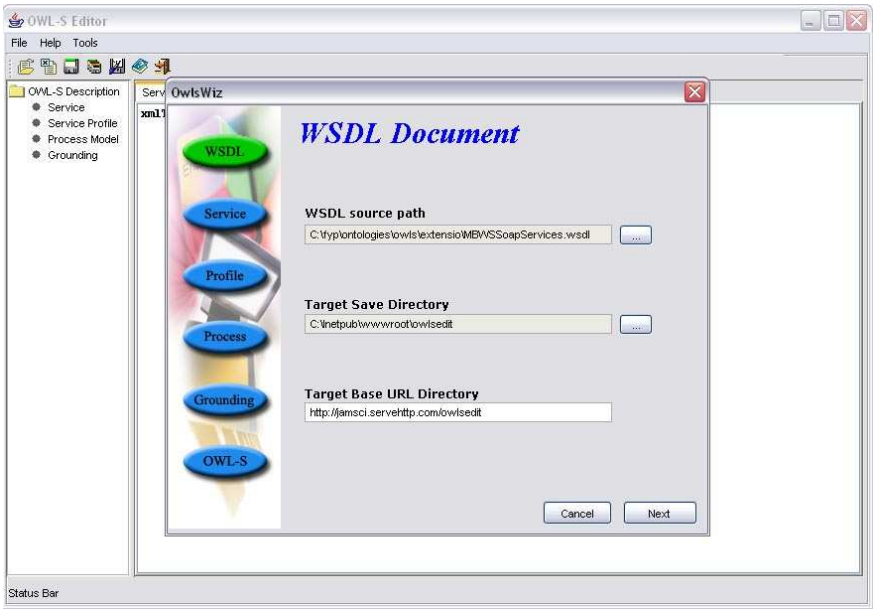


Figure 2.9: OWL-S *Editor wizard.*

accepts an input WSDL file and extracts information to create a basic OWL-S description (*cf.* Fig. §2.9), or from a template; (2) a validator that checks the validity of the ontologies; and (3) a visualizer that shows the descriptions and service compositions in a graphical, UML-like manner. More information can be found at http://staff.um.edu.mt/cabe2/supervising/undergraduate/owlseditFYP/OwlSEdit.html.

2.2.5.5 Semantic Web Service Composer

Semantic Web Service Composer is a semantic composition and matching engine that helps service requesters to find and compose suitable Web services. If a single service can not perform the requested functionality, the tool attempt to find a composition of services, using AI planning algorithms, that meet the requirements.

This tool is developed by IBM and can be found as a part of IBM's Emerging Technologies Toolkit at http://alphaworks.ibm.com/tech/ettk along with other tools grouped in *Semantic Tools for Web Services*, that is a set of Eclipse plug-ins.

2.2.5.6 ASSAM Web Service Annotator

ASSAM stands for *Automated Semantic Service Annotation with Machine learning*, and it is an application that helps the user to effectively annotate Web services described with WSDL [43]. These annotations can be exported in OWL-S.

This tool provides a graphical interface to annotate the WSDL file, but the actually important feature of it is the machine learning assisted annotation. Thus, after a training period, ASSAM can make recommendations on how to annotate types in WSDL. It is available at http://moguntia.ucd.ie/projects/annotator.

2.3 METEOR-S

The third framework in which we concentrate our report is the proposed by METEOR-S. METEOR-S is a project lead by the group LSDIS from the University of Georgia. The main target of this project is to extend current standards in WS adding semantic concepts [80]. Their work includes annotation and semi-automatic publication of WS using a proposed extension of WSDL [71], semantic discovery of WS extending UDDI [85], and orchestration and composition of WS described with a semantically-annotated version of BPEL4WS [81].

As a matter of fact, their proposals of extending WSDL with semantic concepts have been chosen as a starting point to add semantics in the next version of WSDL. These proposals have materialized in two different frameworks or languages: SAWSDL [29] and WSDL-S [5]. In fact, WSDL-S is the base proposal for the work of the Semantic Annotation for WSDL Working Group of the W3C, which has recently published as a W3C Recommendation its Semantic Annotations for WSDL and XML Schema specification (SAWSDL).

Finally, they have done some work related to QoS in a WS composition scenario [79]. In this work, they model QoS properties of WS, specially time, cost and reliability, in order to evaluate the real QoS of a Web process composed of such WS.

In the following two sections we detail firstly the proposed extension of WSDL, developed by METEOR-S, adding semantics to it (the so called WSDL-S), and secondly we describe the ongoing proposal SAWSDL to semantically annotate WSDL and XML Schema, which is the current proposed W3C standard.

2.3.1 Web Service Semantics

One of the first results of the METEOR-S project, that serves as a foundation to the more evolved SAWSDL proposal, is WSDL-S [5]. The approach introduced in [80] presents an upgrade to current Web service standards, specially to Web service descriptions. WSDL-S extends WSDL with semantics, employing concepts analogous to those in OWL-S, but it is not limited to one specific semantics representation language. The advantages of this approach to add semantics to WSDL are: (1) the use of a wide used, well-known language to describe both semantic and operational level details of Web services, and (2) the ability to reuse external semantic domain models described in any ontology representation language, so Web service developers can annotate their WSDL files with their choice of modeling language.

When designing WSDL-S, its authors follow a set of design principles that they encourage to be taken in consideration when designing any framework for Semantic Web services.

- *Build on existing Web Services standards:* Companies are already using current Web service standards, so any approach to adding semantics to Web services should use those standards.
- *The mechanism for annotating Web services with semantics should support user's choice of the semantic representation language:* It should be possible to use any ontology modeling language to annotate Web services, such as OWL-S or WSML. This gives flexibility to Web service developers to select a language that fits their actual needs.
- *The mechanism for annotating Web services with semantics should allow the association of multiple annotations written in different semantic representation languages:* Related with the previous one, elements may be simultaneously annotated with multiple semantic representation language, so Web services could be discovered using multiple discovering engines using different languages.
- *Support semantic annotation of Web Services whose data types are described in XML schema:* Due to the wide-spread use of XML Schema to define business documents, semantic annotation of inputs and outputs defined using XML schemas has to be supported.
- *Provide support for rich mapping mechanisms between WS schema types and ontologies:* The possibility to annotate schemas is as important as providing a way to map their complex types to ontological concepts. This mapping can be described using any schema mapping languages, such as OWL, RDF, XSLT, etc.

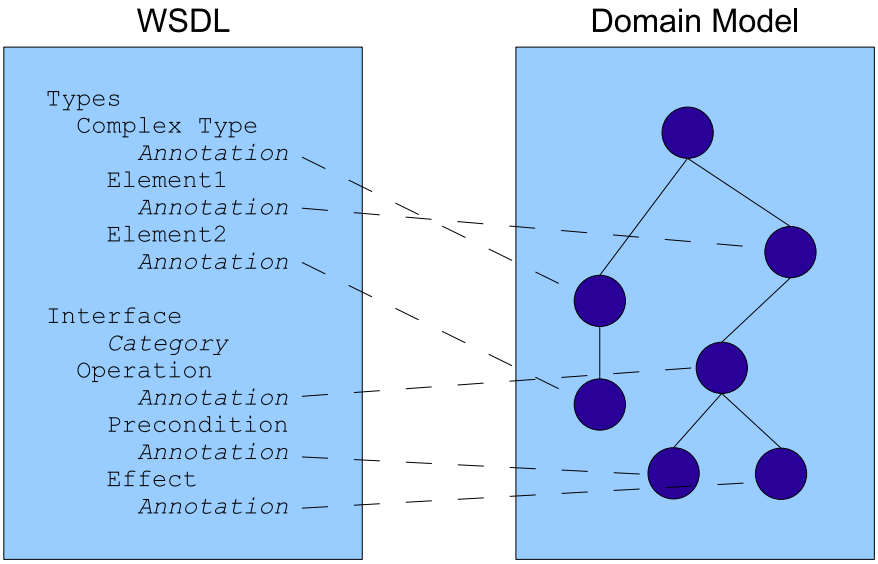


Figure 2.10: *Associating semantics to* WSDL *elements.*

In the following extending elements are defined and its usage is described in detail.

2.3.1.1 Extending WSDL

Figure §2.10 shows how semantic annotations from an external domain model are associated with various elements of a WSDL document, including inputs, outputs, operations and additions from WSDL-S approach. The domain model may be composed by one or more ontologies. These semantic annotations are added to WSDL documents extending the standard.

This proposal has been developed using WSDL 2.0. Conceptually, WSDL 2.0 represents service descriptions using the following constructs: interface, operation, message, binding, service and endpoint. The first three constructs deal with the abstract definition of a service, while the remaining deal with service implementation. WSDL-S focus on semantically annotating the abstract definition of a service, but it could be useful to semantically annotate service implementations.

WSDL-S provide URI reference mechanisms via extensibility elements introduced in interface, operation and message constructs, so they point to semantic representations defined in an external domain model. These extensibility elements are described below:

modelReference is an extension attribute to specify the association between a WSDL entity and a concept of the semantic model used. It can be applied to complex type, element, operation and the extension elements (*precondition* and *effect*).

schemaMapping is also an extension attribute which is added to XML Schema elements and complex types in order to handle structural differences between the semantic model concepts and the schema elements of a Web service that the former are related to.

precondition is one of the new elements added by WSDL-S, which are a child element of the *operation* element. It defines a set of assertions that must be met before a Web service operation can be invoked. The detailed representation of preconditions depends on the semantic domain representation model.

effect is also an added child element of the *operation* element, and it defines the result of invoking an operation. As in preconditions, its representation depends on the semantic domain model used.

category is an extension attribute of the interface element, consisting of service categorization information, that could be used when publishing the service in a registry such as UDDI [85].

2.3.1.2 Annotating Elements

WSDL-S proposes that operations should be annotated by a concept in a semantic model providing a high level description of them. Thus, a *modelReference* attribute is added to operation element in WSDL, pointing at the said semantic concept. Although inputs, outputs, preconditions and effects are often used to capture the semantics of a given operation, using a simple semantic annotation to describe the behavior of that operation could be useful during discovery, providing a first cut indication whether the service matches a given request.

In addition, WSDL-S provides means to annotate XML Schema types using the *modelReference* attribute also. That is applied to semantically annotate inputs and outputs of operations described in WSDL. Specifically, to annotate inputs and outputs defined with simple types, the said attribute is used to point to the corresponding semantic concept. That is possible by the use of the extensibility of an element tag of XML Schema, so the said attribute is added to the definition of the element from the simple type.

2.3.1.3 Annotating Complex Types

Complex types can also be annotated in multiple ways. WSDL-S proposes two alternatives: (1) bottom level annotation, which consists in annotating leaf elements a complex type; or (2) top level annotation, that consists int annotating at complex type level. These two alternatives are complementary and the user could use any of them when semantically describing Web services in WSDL-S.

Bottom level annotation When elements of a complex type correspond in a one-to-one way with concepts in a domain model, a simple method is provided to annotate that type. As described above, this is supported by adding a *modelReference* attribute to the relevant schema element definition, that points to the said concept via URI.

Top level annotation When the associations between the schema elements and the concepts in the domain model are one-to-many or many-to-one, bottom level annotation is impossible. In this case, it is necessary to explicit the mapping between a complex type and the corresponding concept in the domain model. A complex type can have a semantic annotation using a *modelReference* attribute that points to a high level concept in an ontology. However, to annotate any elements contained within a complex type, a *schemaMapping* attribute in the complex type could be used. That attribute points to a definition of the mapping in a given language, such as XSLT, which WSDL-S does not restrict.

2.3.1.4 Added Elements

Before in this Section we introduced that WSDL-S adds two new elements to WSDL specification, in order to complete the semantic annotation of an operation. Both preconditions and effects points to concepts defined in semantic domain model that provides assertions that must be met prior to Web service execution, or define the results of the invocation , respectively.

Both preconditions and effects are specified as child elements of the operation element of WSDL. Each operation can have at most one precondition and one effect, in order to keep the specification simple. The attributes of these elements are the same:

- *name*, specifying a unique identifier within the WSDL document.

- *modelReference*, which specifies the URI of the part of a semantic model that describes the precondition or the effect.
- *expression* corresponds to a logical expression defining the precondition or the effect. The format of the expression is defined by the semantic representation language used to express the semantic model.

The *modelReference* and *expression* attributes are mutually exclusive.

2.3.1.5 Service Categorization

WSDL-S main purpose is to enable automatic discovery of Web services. This is possible when there are means to publish, catalog and annotate services with semantics. The annotation mechanisms proposed have been described above. But it is possible to add categorization information to services, so they can be published using UDDI for example, in order to aid the discovery process.

The service category is modeled using the extensibility elements on a WSDL interface. WSDL-S provides a *category* element which contains the following attributes:

- *categoryName* specifies the name of the category within a given taxonomy of categories.
- *taxonomyURI*, which points to the taxonomy definition, generally where it can be obtained.
- *taxonomyValue*, that is the value associated with a category in the taxonomy.
- *taxonomyCode*, which defines the code associated with a category in the taxonomy.

2.3.2 Semantic Annotations for WSDL and XML Schema

Semantic Annotations for WSDL and XML Schema (SAWSDL) defines how to semantically annotate various parts of a WSDL document, such as input and output message structures, operations and interfaces. These annotations of WSDL structures and XML Schema types allow automatic discovery, composition and invocation of annotated services. In order to accomplish semantic

annotation, SAWSDL defines extension attributes that can be applied to WSDL and XML Schema elements.

As in WSDL-S, semantic annotations are references from an element within a WSDL or XML Schema document to a concept in an externally defined ontology or to a mapping into an ontological schema. SAWSDL specifies some annotation mechanisms, that are independent of the actual ontology expression language, so do the mapping language used. Thus, there are no restrictions on the choose of a semantic domain specification language.

SAWSDL has been recently published as a Recommendation of the W3C, so it is foreseeable that it may be widely used to semantically annotate WSDL documents, with the possibility of being an integrated part of future WSDL specifications. Meanwhile, SAWSDL supports WSDL 2.0, in addition to WSDL 1.1 and RDF.

In the following we present the different mechanisms provided by SAWSDL to annotate WSDL and XML Schema documents.

2.3.2.1 Annotation Mechanism

SAWSDL specification defines two semantic annotation constructs that are applied to WSDL and XML Schema as explained in the sections below. This specification concentrates on annotating the abstract definition of a service, that is Type Definition, Interface, Interface Operation and Interface Fault components of WSDL 2.0. Annotating these elements enable dynamic discovery, composition and invocation.

The semantic constructs specified in SAWSDL are the following extension attributes:

- The association between a WSDL or XML schema element and a concept in some semantic domain model defined externally is specified by the *modelReference* attribute. It can be used to annotate WSDL interfaces, operations and faults, as well as XML Schema complex and simple type definitions, and element and attribute declarations.

- When an explicit mapping is needed, two extension attributes, named *liftingSchemaMapping* and *loweringSchemaMapping*, can be used to annotate XML Schema element declarations, complex type definitions and simple type definitions. These mappings can be used during service invocation.

WSDL elements can be annotated with multiple model references and multiple schema mappings. If an element has more than one model reference, all of them apply, although no logical relations can be explicitly defined between them. But an element referencing multiple schema mappings indicate mapping alternatives.

Model Reference Although a model reference may be associated to any WSDL element, SAWSDL only defines its meaning for *wsdl:interface*, *wsdl:operation*, *wsdl:fault*, *xs:element*, *xs:complexType*, *xs:simpleType* and *xs:attribute*. The *modelReference* attribute can take a list of URIs that references concepts in a semantic domain model, providing semantic information associated to a WSDL or XML Schema element. As previously said, SAWSDL specification remains agnostic to the concrete semantic representation language of those concepts, and it does not specifies how the document processor resolve a model reference to a document containing the concept definition.

Schema Mapping SAWSDL uses schema mapping to resolve mismatches in the data structure of inputs and outputs. That feature is used after discovering (where model references are used), relating data instances defined in some XML Schema document with some semantic data defined in a semantic model. Such mappings are useful when the structure of the instance data can not be trivially related to the corresponding semantic model. Again, the mapping language to use is not constrained. The usage of *loweringSchemaMapping* and *liftingSchemaMapping* attributes is described in the following.

2.3.2.2 Annotating WSDL Documents

The main WSDL 2.0 components that can be annotated using a model reference property (set by *modelReference* attribute) are interfaces, interface operations and faults. The value of that property is a set of URIs taken from the attribute value. Using a *modelReference* attribute on a WSDL interface provides a semantic description or a classification of that interface, while on an operation or a fault provides semantic information about that operation.

Annotating interfaces with *modelReference* attribute may be useful to specify behavioral aspects, categorize the interface or associate other semantic definitions, all defined in an external semantic model. As usual, SAWSDL does not constraint the form of those semantic definitions, even in the categorization case. Furthermore, if an interface element extends one or more interface elements, the model reference of the extended interfaces are also applied to

the new interface, so an interface can be categorized using the categorization of the interfaces it extends.

The annotation of operation elements is done by a reference to a concept in a semantic domain model providing a high level description of the corresponding operation, specifying its behavioral aspects or including other semantic definitions. That high level description is useful during service discovery, because of it provides a simple annotation that reduce the number of services that match with a given request, without checking inputs and outputs of those services. That check may be done later, using the concrete annotations on input and output elements. Moreover, operations can also be annotated with category references, but they are separate from other categorization done in other elements (e.g. interfaces).

In addition, it is possible to annotate fault elements, with the same meaning that in operations. That is, the annotation provides a high level description of the corresponding fault. The fault message is not described by that annotation, being annotated as in an output message.

2.3.2.3 Annotating XML Schema Documents

In this case, both available constructs can be applied when annotating XML Schema documents, i.e. model references and schema mappings. On the one hand, a model reference allows to link any XML structure defined to its corresponding semantic model, so this information can be used in discovery processes. On the other hand, schema mappings can be applied after discovery stages, so structural mismatches between the semantic model and service inputs and outputs can be overcame.

The *modelReference* attribute can be included on simple type definitions, so any element or attribute whose type has this attribute is described by the referenced concept from the corresponding semantic model. Additionally, elements and attribute declarations can be also annotated by using a model reference. Particularly, model references on an element declaration are also described by its type definition model reference, if present, such as an annotated complex type.

Concerning complex types, they can be annotated using two approaches: (1) bottom level annotation, where the model references are included at member element or attribute level; or (2) top level annotation, where the complex type container is annotated with corresponding model references. As in the previous cases, *modelReference* attribute is the placeholder to semantically annotate complex types at any level. These two approaches can coexist in the same complex types, each one being treated independently.

Another form of annotating XML Schema documents are performed by *liftingSchemaMapping* and *loweringSchemaMapping* attributes. These attributes allow to associate types (complex and simple) and elements with a mapping to an ontology. Lifting and lowering information can be both specified in a single type or element, but it is usually specified only one. Also, within each attribute multiple schema mappings URI can be specified, each one being interpreted as alternatives.

Differences between both mappings are the following. On the one hand, a mapping referenced by *liftingSchemaMapping* attribute defines how an XML instance document conforming to the element or type defined in a schema is transformed to data that conforms to some semantic model. Thus, the input of the transformation process is that XML element on whose declaration the mapping is located, while its output is the corresponding semantic data. On the other hand, a mapping referenced by *loweringSchemaMapping* attribute defines how data in a semantic model is transformed to XML instance data, as opposite of a lifting mapping. In this case, the input is some semantic data and the output of the process is the XML element on whose declaration the mapping is located.

2.3.3 Provisioning Approaches

The METEOR-S project seeks to address the entire life-cycle of SWS, including not only semantic specifications as the analyzed before, but discovery, selection and composition. Concerning the provisioning, there are some noteworthy results within that project, which are introduced here and further discussed in Sec. §3 . Furthermore, proposals based on WSMO and OWL-S frameworks can be also applied to METEOR-S annotation mechanisms, because these annotations can be defined using WSMO or OWL-S.

Sivashanmugam *et al.* present extensions to current Web standards, such as WSDL and UDDI, that are aimed at improving discovery and composition [80]. They propose a three phase discovery and selection algorithm that uses templates of ontological concepts to state service requirements. Firstly, WS are matched based on the functionality provided. Then, the result set is ranked using semantic similarity between inputs and outputs of the selected operations and the one from the template. Finally, the ranking is performed using preconditions and effects.

Concerning UDDI extensions and discovery, a peer-to-peer approach to them are presented by Verma *et al.* [85]. They propose a scalable infrastructure of UDDI registries where the discovery process are also performed using

templates. However, this approach only performs a simple categorization and it do not explicitly rank the services, though they can be ranked by semantic similarity as in [80].

Finally, Oldham *et al.* show a selection mechanism applied to Service Level Agreements [67]. Although agreement making is not contemplated in our definition of service provisioning, this work presents a matching algorithm that can be applied to discovery processes using SAWSDL. Moreover, their selection algorithm compute a scoring for each possible agreement, where some QoS properties are preferred among others.

2.3.4 Existing Tools

METEOR-S project has released some tools supporting its work in Semantic Web Services with actual implementations of its results. Although there are fewer tools than in the other discussed frameworks (*cf.* Sec. §2.1.6 and §2.2.5), it is worth to study their approach, due to the usage of proposed W3C standards.

The first three tools described below can be downloaded from the METEOR-S Web site, sited in http://lsdis.cs.uga.edu/projects/meteor-s/.

2.3.4.1 MWSAF

MWSAF stands for METEOR-S Web Service Annotation Framework [71]. It constitutes one of the very first proposals to semantically annotate Web services developed in METEOR-S. MWSAF is a framework that semi-automatically annotate WSDL documents with relevant ontologies.

As it is shown in Figure §2.11 , the main feature of the tool is to do mappings between a Web service described in WSDL with DAML-S or RDF ontologies. To do so, MWSAF architecture is composed of the following components:

i. An *ontology-store* that stores the ontologies, as its name suggests. The ontologies are categorized in domains, and the system allows to add new ontologies.

ii. The *translator library* that contains algorithms to translate ontologies and WSDL descriptions to *SchemaGraph* representations [71], that are used to feed the matching algorithm.

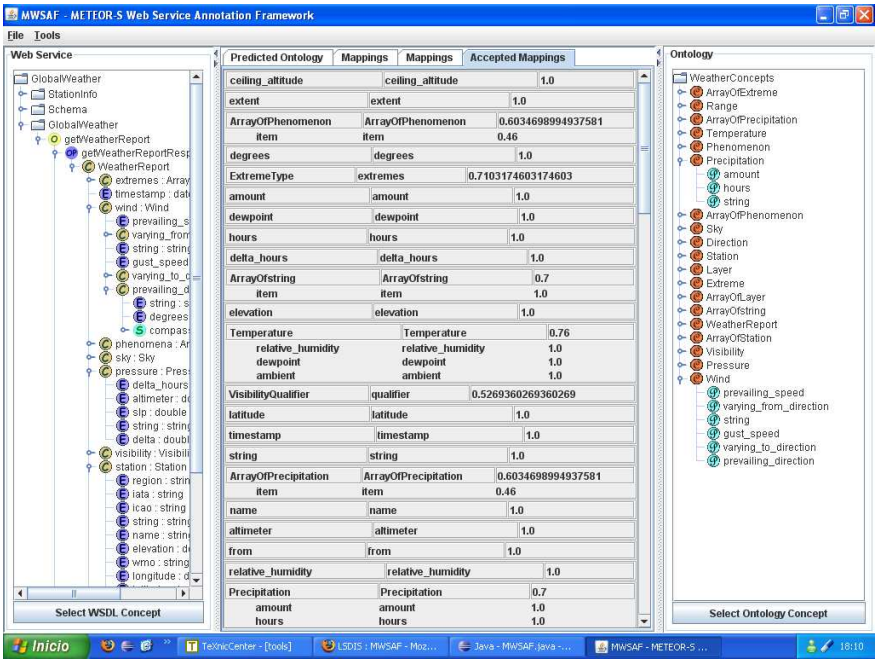


Figure 2.11: *MWSAF tool main screen.*

iii. The *matcher library*, that provides matching algorithms to find the most suitable domain ontology from the ontology-store to annotate the Web service.

The tool suggests mappings from WSDL descriptions to ontologies stored in the ontology-store, and additionally allows the user to add extra mappings manually.

2.3.4.2 Radiant

Radiant [35] is an Eclipse plug-in that provides a user interface to annotate existing WSDL documents into WSDL-S or SAWSDL via an OWL ontology. Its features also include an ontology viewer and a method to publish Web services in UDDI registries. A screenshot is shown in Figure §2.12 .

Radiant is frequently used together with Lumina, described in the following. That way, the user annotate Web services using the former, that can publish to an UDDI register doing a mapping between the description and the registry fields, and then that user can discover those services with the latter.

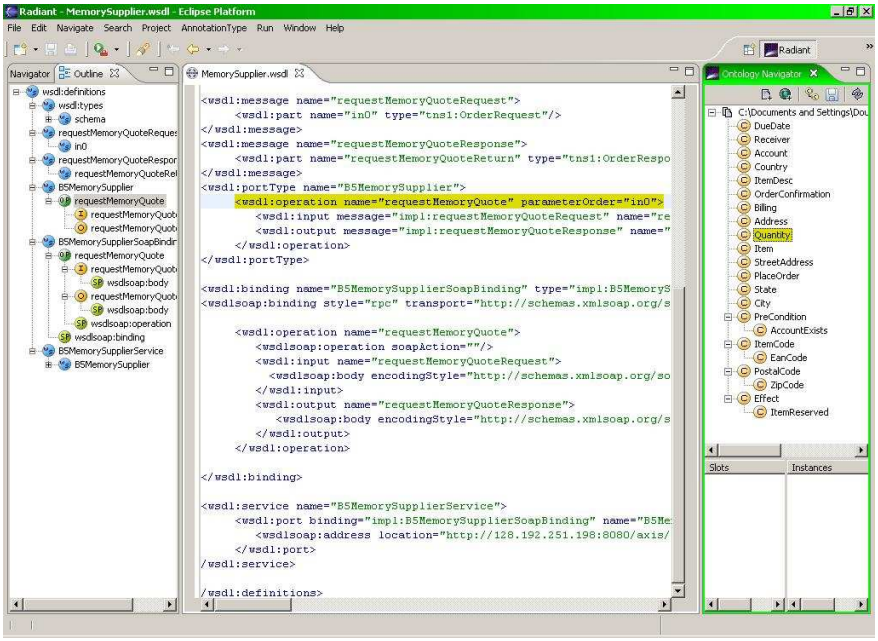


Figure 2.12: *Radiant Eclipse plug-in.*

2.3.4.3 Lumina

Lumina is a component of the METEOR-S project that deals with all the aspects of Semantic Web Services lifecycle, specially semantic discovery [54]. This project uses the WSDL-S/SAWSDL approach to provide discovering Web services with ontological concepts. It supplies a consistent mapping structure between WSDL-S/SAWSDL and UDDI, so the user can discover specific operations by annotating ontological concepts to the operation functional concepts, inputs and outputs. It is also possible to discover Web services using the more traditional UDDI discovery based on Business Entity, Business Service and TModel.

Basically Lumina acts as an extended UDDI registry so it becomes a facade to several UDDI implementations, providing three different ways to discover Web services: (1) general UDDI discovery, which do a traditional registry search but unifying the different implementations of UDDI; (2) WSDL-S discovery, that uses semantically annotated documents that Lumina maps to UDDI so the user can do searches based upon ontological concepts; and (3) WSDL discovery, similar to the former but the user has to describe his search with concrete operations, inputs and outputs.

2.3.4.4 SAWSDL4J

SAWSDL4J is an extension of the WSDL4J API that implements the SAWSDL specification. This library, written in Java, provides an object model for SAWSDL documents, so they can be easily supported in third parties applications. Essentially, SAWSDL4J modifies WSDL4J refactoring some interfaces by the inclusion of methods to handle semantic annotations (model references and lifting and lowering mappings).

More information on http://knoesis.wright.edu/opensource/sawsdl4j.

2.3.4.5 WSMO Studio and Semantic Tools for Web Services

Although both tools are discussed in Section §2.1.6.1 and Section §2.2.5.5 , respectively, they provide support to semantic annotation of WSDL documents in a SAWSDL-compliant way, so it is worth to name them here. Those tools have plenty of features and are a good choice to compare the different frameworks discussed in this report.

2.4 Other Frameworks

Although the former are the most developed proposals in the field of SWS, having been submitted to the W3C Semantic Web Services Interest Group, there are some other frameworks that are worth to point. We do not get into details of these proposals, but explain briefly their contribution and main objectives. Additionally, proposals on provisioning and developed tools for these frameworks are grouped together in Sec. §2.4.4 and Sec. §2.4.5 , respectively.

2.4.1 Semantic Web Services Framework

The Semantic Web Services Framework (SWSF) [9] is a SWS approach proposed within the Semantic Web Services Initiative[†1]. This proposal is divided in two complementary components: (1) the Semantic Web Services Ontology (SWSO), which is an ontology and conceptual model by which SWS can be

[†1] http://www.swsi.org/

described; and (2)the Semantic Web Services Language (SWSL), by which concepts and descriptions related directly with SWS can be specified, though its features are not service-specific.

Initially, SWSF research effort was aimed at overcoming the faults and further needs of other frameworks, such as OWL-S and WSMO. Particularly, SWSF took a parallel but complementary approach of WSMO, focusing on a first-order process ontology that enables automated simulation, verification and composition. Although SWSF approach is currently discontinued, it is worth to discuss because it constitutes an alternative to further extend WSMO.

In the following, the two enumerated components are described, along with its two different SWSO variants, FLOWS and ROWS.

2.4.1.1 Semantic Web Services Ontology

SWSO provides a conceptual model that allows to semantically describe WS. This conceptualization is formalized in two variants of SWSL, that can be used depending on the expressiveness required. On the one hand, the First-Order Ontology for Web Services (FLOWS) is an ontology for SWS based on First-Order Logic, so it is described using the variant SWSL-FOL. On the other hand, the Rule Ontology for Web Services (ROWS) is based on Logic Programming, and is represented with the SWSL-RULES variant of SWSL. ROWS ontology can be easily derived from FLOWS constructs, because both represents the same conceptual model. Thus, the rest of this section presents FLOWS in more detail.

FLOWS is divided in three major components: *Service Descriptors*, *Process Model*, and *Grounding*, similar to OWL-S. In fact, OWL-S was a big influence in FLOWS development [22], extending and improving its flaws, using, for instance, a richer behavioral process model based on Process Specification Language (PSL) [38]. Thus, FLOWS provides more interoperability and is based on Web standards by which are given semantics. Additionally, FLOWS has more expressiveness than OWL-S, because the former relies on First-Order logic, while the latter is based on OWL-DL. In the following FLOWS major components are introduced.

Service Descriptors They provide basic descriptive information about the service (as *ServiceProfile* in OWL-S). This information may include QoS properties, meta-information and descriptions, that are useful in service provisioning processes. The initially proposed list contains properties such as service name, author, version, service description, reliability, cost, etc. Nevertheless,

this list can be further extended to contemplate domain specific properties, for instance.

Process Model In FLOWS, the process model describes the behavior of the service. The available constructs come from the ontology of PSL concepts, adding the notion of atomic processes from OWL-S, and the infrastructure for specifying data flows. The core concepts of the process model ontology are (1) the *Service* concept (a set of service descriptors and an activity that specifies the process model using PSL sub-activities), (2) *Atomic Process* (a PSL activity defined by inputs, outputs, preconditions and effects), (3) *Message* (including its type and body), and (4) *Channel* (an object that holds messages to send and to receive).

Grounding While the rest of the elements provide an abstract definition of the service, its grounding constitutes the link to the concrete details of that service, such as message formats, transport protocols, and network addresses. SWSF supports grounding SWSO service description and concepts to WSDL constructs, defining the mechanism of the mapping.

2.4.1.2 Semantic Web Services Language

SWSL is a formal language to semantically describe WS concepts and descriptions. There are two available variants of the language, namely SWSL-FOL, based on First-Order logic, and SWSL-RULES, based on Logic Programming formalism. Like WSML and OWL-S, both languages rely on XML namespaces, the usage of URIs, and are integrated with XML built-in types. Both languages use a layered approach, integrating different extensions and formalism, so some layer may not be used in certain tasks that do not require such a level of expressiveness.

SWSL-RULES is a logic programming language, that includes different features depending on the layers used, such as *Courteous* logic, *HiLog* and *F-Logic*. Its characteristics allow to use SWSL-RULES as both specification and implementation language, directly supporting tasks such as discovery, contracting, policy specification, and so on. The core of the language consists of the pure *Horn* subset of SWSL-RULES. From that starting point, extra layers can be added to extend expressiveness, like monotonic and non-monotonic reasoning, reification, equality, and other layers enumerated before.

SWSL-FOL is a first-order logic which also features *HiLog* and *F-Logic* support. Some of the extension layers from SWSL-RULES can also be applied to

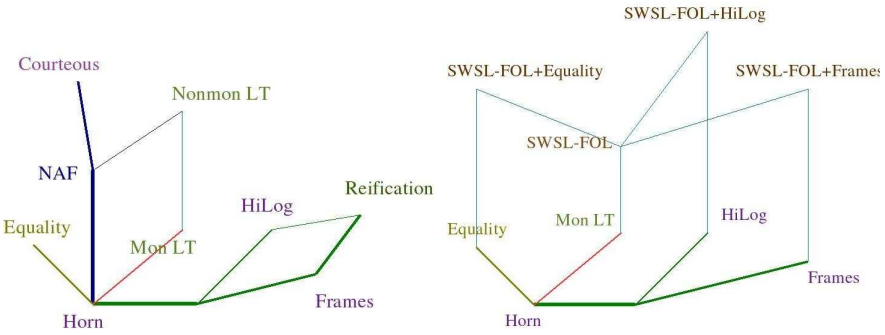


Figure 2.13: SWSL-RULES *and* SWSL-FOL *layers.*

extend SWSL-FOL, with the restriction of the language having monotonic semantics. Fig. §2.13 depicts the different layers of both languages, and their relationship, with the broader lines being the core of each language [9].

2.4.2 Internet Reasoning Service

The Internet Reasoning Service (IRS) project[†2] aims at supporting the automated construction of semantically enhanced systems over the Internet [18]. This project has released three versions of its framework. IRS-I [21] supports the creation of knowledge intensive systems structured according to the UPML framework [68]. IRS-II [64] is similar to WSMF [31], supporting service discovery from a set of demands. It uses descriptions of the reasoning processes called Problem Solving Methods (PSM) from the UPML framework, enhanced with WS technology.

IRS-III [18] updates IRS-II, using the WSMO ontology to model SWS, and providing an architecture to discovery, composition and execution of SWS. This framework and implemented platform acts as a semantic broker, by mediating between clients and service providers. IRS-III is based on WSMO design principles, but there are more principles which have influenced the framework presented below.

2.4.2.1 Design Principles

The following design principles are the foundations of IRS-III framework, whose selection, composition, mediation and invocation processes are based

[†2]http://kmi.open.ac.uk/projects/irs/

on ontological knowledge.

- *Capability driven invocation*. Client applications invoke services simply by providing the desired capability to IRS-III, in terms of WSMO goals. Thus, IRS-III selects the appropriate service, or service composition, invoking it as a response to the client specified goal.

- *Ease of use*. IRS-III design ease the creation of SWS-based applications, hiding the complexity of components to the client.

- *One-click publishing*. In order to support existing systems to be made available through IRS-III, it allows a "one-click" publishing mechanism of standalone code written in Java or Lisp, in addition to publishing through WSDL descriptions.

- *Decoupling with service implementation*. The design of the framework is not coupled with the underlying service implementation platform, although IRS-III assumes that standard WS technologies, as SOAP, are been used.

- *Ubiquitous reasoning*. Reasoning is an essential mechanism of all SWS related activities. IRS-III allows to invoke reasoning queries over WSMO conceptual model or over domain ontologies, in addition to WS status, so that the reasoning system is able to make a WS call to obtain its current status which is needed within the current reasoning query.

- *Open*. IRS-III is based on Java, so it is accessible through standard APIs. Additionally, several components of the semantic broker are SWS represented within IRS-III, so users can replace them with their own services.

- *Inspectability*. SWS descriptions are accessible in a human readable form using graphical representations and understandable ontology languages.

2.4.2.2 Architecture

From the previous design principles, IRS-III architecture has been developed so that ontology-based descriptions are linked with the different components. Figure §2.14 shows this architecture, which is composed by three main components that communicate each other using SOAP protocols: the IRS-III Server, the IRS-III Publisher, and the IRS-III Client.

The kernel of the server contains the WSMO library, that stores the corresponding WSMO definitions using the OCML representation language [65]. Following the inspectability design principle, all the relevant information about services are stored within the library, such as WSMO goals, Web services, mediators and even domain ontologies. Goals, Web services and mediators are stored as knowledge models, with a mediation-centric approach, so the applications usually consists in mediation models that import relevant goals and Web services.

Within WSMO, a Web service is associated with an interface which contains an orchestration and a choreography. Orchestration specifies the control and dataflow of a Web service which invokes other Web services (a composite Web service). Choreography specifies how to communicate with a Web service. The choreography component communicates with an invocation module able to generate the required messages in SOAP format.

The mediation handler is able to interpret WSMO mediator descriptions, including data mediation rules, invoking mediation services, and connecting multiple mediators together. This component also supports the brokering activities of IRS-III, such as selection, composition and invocation based on capabilities. Furthermore, the mediation handler, as well as the choreography and orchestration interpreters are themselves SWS, following the openness principle discussed above. At the bottom of these components, an HTTP server written in Lisp and extended with a SOAP handler is used by IRS-III Server.

The IRS-III Publisher has the role of associating a WS with its corresponding WSMO description. Thus, service providers attach the semantic description to their deployed services, along with handlers to invoke services in a specific platform (using WSDL, Lisp code, Java code, or Web applications). Additionally, IRS-III collects all the information necessary to call the service, such as host, port and paths.

Finally, the IRS-III Client supports the capability-driven invocation of WS by providing a goal-centric invocation mechanism. Thus, the client simply asks for a goal to be solved and the IRS-III broker locates an appropriate Web service semantic description, invoking the underlying deployed WS.

2.4.2.3 Service Ontology

The IRS-III ontology has originally been based on the UPML framework [68], but it has been extended to incorporate the WSMO conceptual model. Specifically, the main incorporated aspects are: non-functional properties, goal-related information, Web service functional capabilities, choreography,

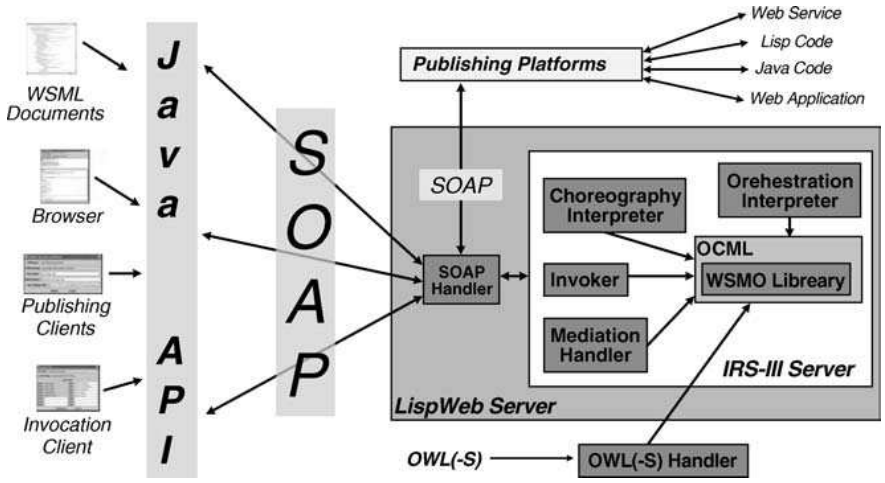


Figure 2.14: IRS-III *architecture.*

grounding, orchestration and mediators. These aspects appear in IRS-III ontology with some differences in order to support capability-driven invocation.

To achieve these capabilities (or goals), Web services are required to have input and output roles. These roles information includes their names, SOAP bindings, and their semantic types. These types are imported form the domain ontology. Moreover, goals are linked to Web services via WG mediators (*cf.* §2.1.4).If a mediator associated with a capability has a goal as a source, then the associated Web service is considered to solve that goal. Additionally, Web services which are linked to goals inherit the goal's input and output roles. This means that input role definitions within a Web service are used to either add extra input roles or to change an input role type.

When a goal is invoked, the broker creates a set of possible Web services using the WG mediators. A specific web service is then selected using an applicability function within the *assumption* expression attribute from the WS capability. In WSMO the mediation service attribute of a mediator may point to a goal that declaratively describes the mapping. Goals in a mediation service context play a slightly different role in IRS-III. Rather than describing a mapping, goals are considered to have associated Web services and are therefore simply invoked.

Finally, the choreography of a WS must be described from the client's point of view. Thus, IRS-III can interpret the choreography to communicate with the deployed service. In contrast, WSMO choreography describes all of the possible interactions that a WS can have.

2.4.3 INFRAWEBS

The INFRAWEBS[†3] research project is focused on developing a Semantic Service Engineering Framework enabling creation, maintenance and execution of WSMO-based SWS, and on supporting semantic Web service applications within their life-cycle [3]. Thus, this ICT European project provides a framework committed to WSMO specification with the following characteristics:

- It hides the complexity of identifying different types of users of SWS technologies.
- It clarifies the different phases of SWS processes.
- It develops a set of tools oriented to the identified types of users and intended to support the different described phases.

Comprising these characteristics, INFRAWEBS is a SOA-based framework composed of coupled INFRAWEBS Semantic Web Units (SWU). Each unit provides tools and components for analyzing, designing and maintaining WSMO-based SWS and applications within the whole life-cycle. Potential users of this framework have been identified and classified as: (1) SWS provider, who publishes SWS; (2) SWS broker or aggregator, who creates and publishes services composed of existing SWS; (3) SWS application provider, who is an organization that design its own application based on SWS; and a (4) WS application consumer, who transparently uses the framework for finding and executing SWS that satisfies its request (a WSMO goal).

Using these categorization of users, the INFRAWEBS framework identifies several tasks that have to be supported and conforms the SWS life-cycle. Definitions for each identified stage in INFRAWEBS framework are the following:

SWS Creation Combines the creation of semantic descriptions of WS, along with needed ontologies and goals. These descriptions are stored and can be published for common usage.

SWS Composition Providers or aggregators can combine several services to provide new services. These composed services are also represented and stored as semantic descriptions for further usage.

SWS Discovery It is a process where user requests (WSMO goals) are matched to service functionality (WSMO capabilities).

[†3]http://www.infrawebs.eu

SWS Selection Choosing which of the discovered services has to be executed can be performed in a pro-active manner, or automatically, by human users or other services.

SWS Execution This stage deals with the actual service, providing the necessary input from the user and obtaining the result after invocation.

SWS Monitoring Information about executed services is gathered in this stage so that it can be used for further service selection.

2.4.3.1 INFRAWEBS Architecture

Figure §2.15 shows the internal architecture of the INFRAWEBS framework, where some components are only used in either runtime or design time (surrounded by a corresponding box). These components responsabilities are briefly described in the following:

SWS-D Semantic Web Service Designer, responsible for the creation of WSMO-based semantic descriptions of WS capabilities and goals.

SWS-C Semantic Web Service Composer, responsible for the static composition of existing WSMO-based SWS.

DSWS-R Distributed Semantic Web Service Repository, responsible for the persistent storage of WSMO-based descriptions and their publication within the framework. In the runtime phase, it is used to retrieve the necessary semantic descriptions.

SIR Semantic Information Router, responsible for registration and annotation of Web services (their WSDL definitions), which are then used in the process of SWS design.

OM Organizational Memory, responsible for indexing and case-based retrieval of WSMO-based descriptions. In runtime, it is used in the first step of the discovery process to retrieve an initial set of SWS matching the current goal, based on ontological keywords similarity.

SAM Service Access Middleware, responsible for guiding the user applications through the steps of SWS usage, including discovery, selection and execution.

SWS-E Semantic Web Service Executor, responsible for executing SWS.

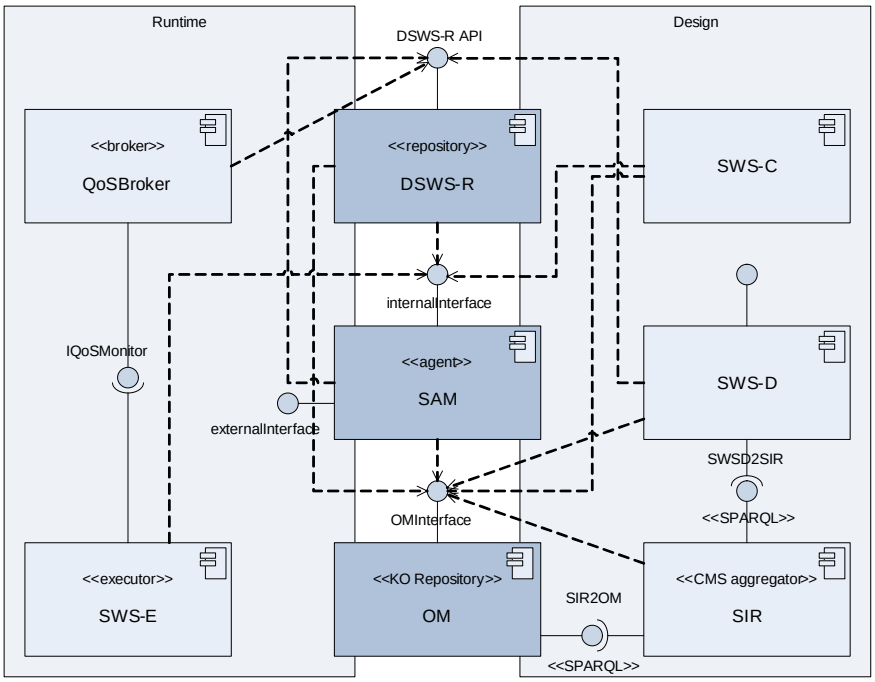


Figure 2.15: *Design and runtime architecture of INFRAWEBS.*

QoS-Broker Quality of Service Broker, responsible for collecting monitored data and calculating the metric values of the SWS being executed.

The described architecture offers a novel, complete approach to solving problems that occurs in the process of creating SWS applications. This architecture supports the interaction scenarios with every potential user type enumerated above that wants to perform one of tasks previously identified.

2.4.3.2 Integrated INFRAWEBS Framework

The Integrated INFRAWEBS Framework (IIF) is the underlying infrastructure for communication and integration of all the INFRAWEBS components, and as a facade to these components in the form of services. It is implemented as an extensible Enterprise Service Bus (ESB) middleware. The IIF is deployed in a peer-to-peer network, possibly integrating components of different technologies within each peer. Thus, any application able to interact with Java APIs or WS can interoperate with INFRAWEBS components.

IIF allows each peer to deploy the whole stack of INFRAWEBS components or only a part of them, in a transparent manner for the user. The use of the underlying ESB middleware provides the IIF with the necessary extensibility to be a Semantic ESB, providing components to create, maintain and execute WSMO-based SWS, so it can be considered as a semantic SOA.

2.4.4 Provisioning Approaches

Several proposals extend the frameworks presented throughout Sec. §2.4 in order to present semantic provisioning approaches. However, there are no relevant proposals that describe provisioning scenarios using SWSF. In the following, some proposals based on IRS-III and INFRAWEBS are discussed.

Using IRS-III framework, Galizia *et al.* [32] propose a selection methodology based on a *trust* ontology. They enhance the capability-driven selection of IRS-III by introducing trust-based selection criteria. This criteria possibly includes QOS policies, as well as reputation ratings and third-party evaluated trust. The selection is made using a classifier.

Hakimpour *et al.* [41] present a structured approach to SWS composition in IRS-III. Each service component of the global composition is automatically discovered using the capability-driven approach from IRS-III and a composition tree, where the global goal is decomposed in several sub-goals, one for each component. However, selection is performed manually by the user, that has to choose the desired combination of discovered services.

A process support applied to a concrete domain using automatic selection of SWS is showed in [24]. In this work, Dietze *et al.* applies IRS-III brokering server to the domain of e-Learning. Thus, they provide a domain ontology that can be linked with IRS-III ontology, in order to perform automatic discovery and selection based on desired capabilities of e-Learning service. Although this proposal dynamically selects the proper service, it do not describe the actual selection mechanism.

Concerning INFRAWEBS composition scenario, Agre and Marinova proposed a discovery process that is aware of the compatibility between the services participating in the composition, i.e. their proposal is able to find a proper orchestration of services [1]. In order to find that orchestration, composite goal templates, which specify some integrity constraints restricting possible compositions, are used. Again, the selection process is done manually by the user.

Finally, Kovács *et al.* presents the discovery engine implemented for INFRAWEBS in [50]. This engine performs the discovery in two phases. Firstly,

a classical keyword filtering are done to filter the candidates. Then, logical matching is performed between goals and services so that the matched services solves the desired goal. Finally, the list of matching services are enhanced with QOS data based on past executions, so that can be used for service selection.

2.4.5 Existing Tools

In order to support the frameworks presented in Sec. §2.4 , there are some tools and implemented software libraries. However, in the case of SWSF, there are no available software that provides a successful usage scenario. Furthermore, as IRS and INFRAWEBS rely on WSMO ontology, several WSMO-based applications can be used within these frameworks, and some tools discussed here can be also used in WSMO-based applications.

2.4.5.1 WebOnto

WebOnto is a Java applet that communicates with a server which allows users to browse and edit ontologies and knowledge models over the Web [28]. It uses OCML to represent ontologies and to provide a direct manipulation interface, displaying ontological expressions using a rich medium. WebOnto is an easy-to-use application, though it has facilities for scaling up to large ontologies. Finally, WebOnto was designed to complement the ontology discussion tool Tadzebao.

Figure §2.16 shows a screenshot of the application. It can be accessed from http://kmi.open.ac.uk/projects/webonto/, and it is useful as an ontology editor for IRS-III.

2.4.5.2 IRS-III Browser

IRS-III Browser is a Java application that serves as a graphical client to a IRS-III server. This server has to be provided, and currently its implementation is not publicly available. Using the Java client, a user may find or create goals, mediators, ontologies and service descriptions, link goals to WS using mediators, publish services within the IRS-III framework instance and invoke such services using goals. This application can be downloaded at http://kmi.open.ac.uk/projects/irs/.

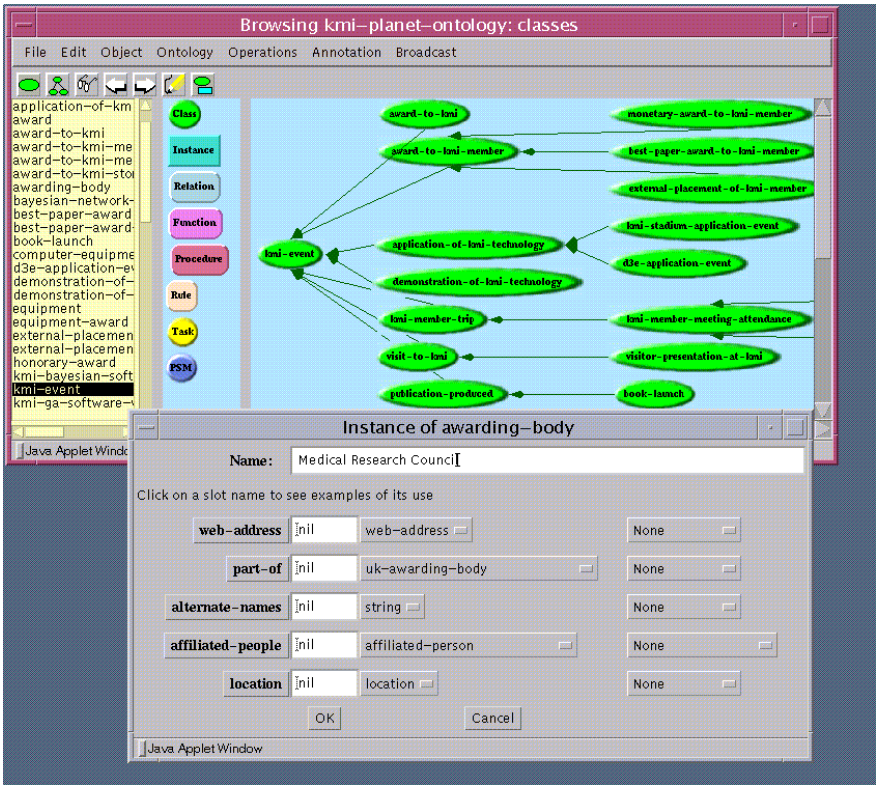


Figure 2.16: *WebOnto application.*

Additionally, a plug-in for WSMO Studio (*cf.* Sec. §2.1.6.1) includes the IRS-III Browser functionality into WSMO Studio. This plug-in is included in the complete version of WSMO Studio.

2.4.5.3 INFRAWEBS Axiom Editor

The INFRAWEBS Axiom Editor is an ontology-driven user-friendly tool for graphical construction of complex logical expressions (called axioms) based on available set of ontologies [2]. It is aimed at constructing and editing WSMO-based SWS capabilities. The graphical models created are internally stored as WSML axioms, and can be used within a WSMO service description.

This Axiom Editor is also included in WSMO Studio as IRS-III Browser. Moreover, it is uses as a graphical editor within INFRAWEBS Designer tool, described below.

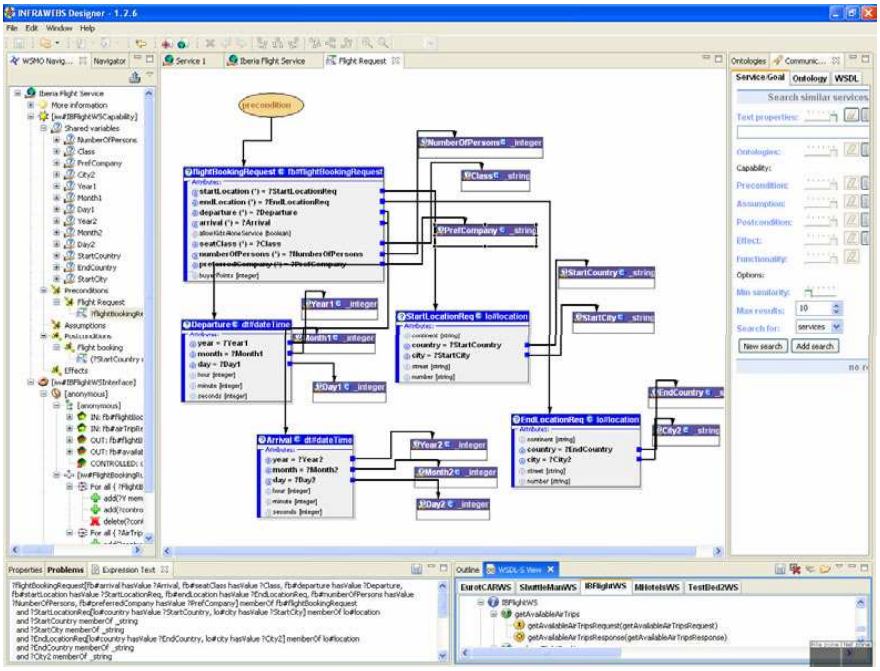


Figure 2.17: *INFRAWEBS Designer.*

2.4.5.4 INFRAWEBS Designer

Instead of an editor only for capabilities within WSMO service descriptions, INFRAWEBS Designer is a complete tool to graphically create WSML-based descriptions of SWS according to WSMO framework [4]. Thus, end users can use this tool to convert provided services into WSMO-based semantic services. Figure §2.17 shows a screenshot of the tool, that is built upon Eclipse rich-client platform.

This tool can be reached at http://www.iit.bas.bg/InfrawebsDesigner/.

2.4.5.5 INFRAWEBS Components

Finally, the INFRAWEBS project has made available the source code of several components from its framework, and from IIF. Altogether with the INFRAWEBS Designer, a complete usage scenario can be implemented and tested by anyone. The source code and relevant documentation and demos are available at the INFRAWEBS project site http://www.infrawebs.eu/index.html?menue=dissemination&site=open_software.

2.5 Analysis and Conclusions

There are several approaches to define SWS in the community. The most important ones have been described in detail in previous sections, including a description of a large set of tools that use them. WSMO, OWL-S and the results of the METEOR-S project, specially SAWSDL, are continuously evolving, trying to become a standard in Semantic Web Service architecture. Each one has advantages and drawbacks that the final user has to be in consideration when choosing a concrete framework on which a provisioning solution is going to be based.

The following comparative analysis only shows alternatives between the three main alternatives, namely WSMO, OWL-S, and METEOR-S, because of they are the frameworks on which semantic provisioning approaches discussed in Sec. §3 rely, and they are the most accepted frameworks in the community. In the METEOR-S case, the comparison is mainly focused on SAWSDL features.

Table §2.1 shows the framework for comparison used in this report. It presents the different approaches that the compared frameworks take in some activities of the SWS life-cycle, and the elements defined to support them. Additional comparisons between SWS proposals can be found in [6, 52, 66]. The compared features of each activity are described in the following sections.

2.5.1 Discovery

The comparison between the different frameworks shows what elements support the discovery of services. In WSMO, goals and capabilities are used to discover SWS. A goal describes the objective that a client want to accomplish when accessing a service, while a capability defines a service functionality. Thus, discovery performs a matching between goals and capabilities.

The Service Profile is used in OWL-S to define the functionality that a service provider offers within a concrete WS. In this case, to perform the necessary matching, a user has to define the required functionality using another instance of the Service Profile class.

The METEOR-S approach is based on an extension of UDDI registries to allow semantic matching. This is also accomplished by using SAWSDL to annotate WSDL documents, so a discovery solution intended for any other framework can be applied to SAWSDL.

Activities	WSMO	OWL-S	METEOR-S
Discovery	Goals and Capabilities	Service Profile	SAWSDL and UDDI
QoS Descriptions	NFP extensions	Service Parameter	Interface Categorization
Selection	Not defined	Not defined	Not defined
Interoperation	Choreography	Process Model	SAWSDL and BPEL4WS
Composition	Orchestration	Control Constructs	BPEL4WS
Invocation	WSDL Grounding	Service Grounding	WSDL

Table 2.1: *Semantic Web Services initiatives.*

2.5.2 QoS Descriptions

Concerning QoS or non-functional properties, each framework has placeholders to associate them to service descriptions. WSMO includes a predefined set of non-functional properties that can be used to describe any first-level component of the framework. Furthermore, that set can be easily extended to support richer QoS descriptions.

Within the OWL-S service profile, there is an extension point called Service Parameter that can be used to describe QoS properties. However, there are some predefined properties, which can be considered meta-information about the service, as well as a service categorization initially intended for discovering, but not used by any provisioning approach.

This categorization is the unique available possibility of describing QoS in the METEOR-S framework, and it is present in WSDL-S, but not in SAWSDL. However, SAWSDL can benefit from QoS descriptions from the imported annotations.

2.5.3 Selection

Selection processes have to take QoS descriptions and a set of discovered services to rank them so the best service in terms of QoS can be chosen. None of the discussed frameworks has specific elements to support selection processes, though there are many extensions presented in Sec. §3 that propose solutions to this task.

2.5.4 Interoperation

The interoperation mechanisms describes how a service works, in terms of communication of inputs and outputs. In WSMO, the choreography elements defines pre and post conditions of the capabilities of each service.

OWL-S envision services as a process, described by their Service Models. Basically, in OWL-S a service is viewed as a transformation of inputs to outputs and as state transition.

Finally, METEOR-S framework add semantics to WSDL by mapping schema elements to ontologies concepts within SAWSDL proposal. Interfaces, operations, and their inputs and outputs are semantically described in order to enable interoperation. Furthermore, BPEL4WS also can be annotated to provide semantics to process models.

2.5.5 Composition

About the service composition, the analysis has been done on how to define the communication process between services. WSMO defines the orchestration processes, but they are not completely defined yet.

Composition processes in OWL-S use control constructs, which provide a process annotation to define data flow between processes (i.e. services). Finally, METEOR-S framework uses BPEL4WS to create a representation of the web process, providing a rich set of tools for modeling the workflow and representing requirements.

2.5.6 Invocation

The mechanism to actually invoke a service is provided by the grounding information. WSMO defines the grounding through the choreography, although it is not completely specified how it can be used to actually invoke a service. This grounding can be connected to WSDL definitions.

In OWL-S the Service Grounding is a top-level element that provides the means and the information to access the service. OWL-S also provides a standard grounding for WSDL, though it does not dictate the grounding mechanism.

METEOR-S framework uses the current standards of WSDL and SOAP for invocation purposes, because of their proposal to semantically annotate WSDL documents.

Chapter 3

Semantic Provisioning Approaches

In this chapter a thorough analysis of the related work on discovery and selection of SWS is presented. Several proposals on discovering and selecting SWS are discussed, along with a survey on how user preferences are described by different approaches. Furthermore, a comparative analysis is showed, presenting some conclusions about all this related work. Although some proposals present approaches on both discovery, selection and user preferences descriptions, each one is discussed in only one of the sections, depending on its focus.

3.1 Discovering Semantic Web Services

An early work on SWS discovery by Paolucci *et al.* propose a DAML-S-based solution to perform semantic matching between user requirements and services provided capabilities [69]. They claim that UDDI registries and languages such as WSDL can not represent capabilities, so they are not able to support semantic matching. Their matching engine, based on DAML-S, return a match between a user request and a service capability (referenced by the authors as advertisement) if they both are "sufficiently similar", *i.e.* the corresponding service provides the capability requested by the user in some degree. Thus, the matching engine has to allow flexible matches. That similarity or *matching degree* is firstly computed between service output of both request and advertisements, and then between their input. The matching degree with regards to output parameters can result one of the following, ordered by preference.

exact: If the request and advertisement output are equivalent, or if the request output is a subclass of the advertisement one.

plug in: If the output of the advertisement subsumes the one of the user request. This is weaker than the above.

subsumes: If the output of the request subsumes the one of the advertisement, then the provider does not completely fulfill the request.

fail: If no subsumption relation between advertisement and request is identified.

These degrees are equivalent in the case of matching the input. A higher matching degree on the output is preferred over input matches. Thus, a simple selection can be performed using this approach. Paolucci *et al.* also provide an architecture to apply their matching engine to a UDDI registry.

Sycara *et al.* apply the previous matching engine in their work, that discuss how semantic information allows automatic discovery, invocation and composition of WS [83]. They provide an early integration of semantic information in a UDDI registry, and propose a matchmaking architecture. Additionally, they present a virtual machine that implements the DAML-S Process Model so it can be used to manage the interaction with WS.

They update their work to OWL-S in [84]. However, their approach are not able to perform discovery and selection taking QoS properties into account, because of the DLs formalism they use.

There are more proposals that perform the discovery of SWS using DLs [36, 53, 56]. Particularly, González-Castillo *et al.* provide a matchmaking algorithm using the subsumption operator between DLs concepts describing user requests and provided services [36]. In order to describe them, they use an *ad hoc* DAML-based ontology, that allow the use of DLs to perform the matchmaking. However, existing DLs reasoners, as RACER [40] and FaCT [45], do not provide a sufficiently expressive DLs variant for their proposed service description, though they are used to partially validate their proposal.

Giving a requested service description S, their algorithm returns matches that are equivalent concepts to S; sub-concepts of S; super-concepts of S that are subsumed by a top-level *serviceDescription* concept; or subconcepts of any direct super-concept of S whose intersection with S is satisfiable. Thus, these matching possibilities are quite similar to the defined by Paolucci *et al.*, though González-Castillo *et al.* do not state any preference order.

On the other hand, Lutz and Sattler [56] do not provide an algorithm, but give the foundations to implement it using subsumption. They further discuss the different DLs variants that are useful to describe services, along with complexity analysis of the associated inference problems within discovery processes. Additionally, they describe services in terms of the actions they perform to transform their concept world.

The same applies to Li and Horrocks work [53], who also give hints to implement a prototype using RACER. Their proposal are based on DAML-S, so it has the same problems about existing implementations of the chosen DLs variant as González-Castillo *et al.* work. However, their matching algorithm extends the degree definition described by Paolucci *et al.*, by adding a "intersection satisfiable" match level. Thus, if the intersection of the advertisement and the request is satisfiable, the match constitutes an *intersection*. This match is considered better than a *fail* (or a *disjoint* match, as referenced by the authors), but weaker than *subsume* match.

All the previously discussed proposals define different matching degrees, from exactly equivalents to disjoint, so they perform a selection based on these degrees. All of them perform this matching by comparing inputs and outputs from the request and the advertisements. However, Benatallah *et al.* propose to use a best profile covering to discover and hence select the requested service [11]. Thus, they take a more flexible approach, performing the discovery process using rewriting techniques and hypergraphs, but it results to be a NP-hard problem, as in any optimization problem [14].

On the other hand, Benbernou and Hacid realise that some kinds of constraints are necessary to discover SWS, including QoS related ones, so they

formally discuss the convenience of incorporating constraints in SWS discovery [12]. However, instead of using any existing SWS description framework, their proposal uses an *ad hoc Services Description Language*, in order to be able to define complex constraints. In addition, the resolution algorithm uses constraint propagation and rewriting, but performed by a subsumption algorithm, instead of a Constraint Programming solver.

An early approach on modeling QoS in the context of SWS discovery are found in [73]. In this work, Ran presents a UDDI extension and a catalog of QoS parameters that can be included in UDDI descriptions. Discovery is performed using queries with functional requirements, as well as conditions on QoS. However, the actual discovery algorithm is not defined, and queries that use QoS are not shown, so their expressiveness is unknown. Additionally, UDDI only supports a keyword based search, so no form of inference or flexible match can be performed [83]. Apart from that, the user have to perform different queries in order to perform a selection of the best service.

While discovery approaches do not normally take QoS preferences into account, selection proposals use them to rank services, in order to choose the best suited service with regards to QoS-aware user preferences. In the following Section, several QoS-aware selection proposals are discussed. Some of them also define discovery processes, but they mainly use similar approaches as the ones discussed in current Section, so the following section is only focused on selection approaches, unless the corresponding discovery process is noteworthy.

3.2 QoS-Aware Selection

Although their proposal is not semantically enabled, Liu *et al.* present a QoS computation model including a selection algorithm [55], which is applied in other approaches [70, 87]. They propose an extensible QoS model that comprises both generic and domain specific criteria. Thus, they do provide a very simple generic QoS model for every WS, but allowing its extension with QoS domain specific properties. The generic QoS criteria provided contains the execution price, execution duration and reputation. Both generic and domain specific criteria define an expression that compute the *quality critera* value used in the selection process.

Liu *et al.* also propose a service selection where the service ranking is performed based on user preferences over QoS requirements. These preferences are expressed by relative weights that come into the QoS computation algorithm at its last stage. Thus, the selection process is reduced to a matrix-based

QoS computation. In this computation, a matrix, whose elements are the obtained from the QoS criteria of each service, is normalized two times, finally obtaining a global QoS value for each service, where user preferences are taken into account. This algorithm suppose that the set of input services have the same functional properties, due to they have been previously discovered.

A semantic framework for service discovery is presented by Pathak *et al.*, which models mappings between ontologies in [70]. They propose to use domain specific ontologies to define QoS properties among user requests and service descriptions. In their work, selection is done using matching degrees at a first stage, such as the degrees defined in [53]. Thus, the set of discovered services is filtered, removing those services that fall under *intersection* and *disjoint* categories.

The second step of their selection process further refines the set of candidate services by QoS parameters. The values from these parameters are collected in a *quality matrix* and normalized, in order to calculate a fixed, weighted utility function for each candidate service. Then, candidates whose utility function is above a given threshold, are passed to the final step of the process, where they are ranked by one QoS parameter that is applied to a ranking function in order to obtain the optimal service with regards to user preferences. These user preferences are specified in two different stagers: (1) weights for each QoS parameter used in the computation of the utility function, and (2) ranking attribute and ranking function used at the final stage.

In addition to the two previous proposal, Wang *et al.* also define a QoS-aware selection model for SWS that makes use of QoS matrices [87]. They provide an extension to WSMO ontology that enables handling QoS parameters. Within this extension, users may define expressions that provide the actual value of a measured QoS parameter, and also the associated preference. This preference is defined in terms of relative weights and preferred tendency, whose value may be one of the following:

small: The value of the QoS parameter is preferred to be as low as possible.

large: The value of the QoS parameter is preferred to be as high as possible.

given: The value of the QoS parameter is preferred to be as close to a given value as possible.

Wang *et al.* define a QoS selection model and an algorithm based on a QoS matrix that contains values of QoS parameters. Before, in a previous step, discovery is performed by matching functional properties. The selection

algorithm first normalize matrix values in order to homogenize them. Then, a uniformity analysis is done applying different formulas depending on the tendency preference of each QoS parameter. Finally, for each service (represented by a row in the normalized matrix), the evaluation result can be computed by adding the resulting values for each parameter applying the corresponding weights.

On the other hand, Zeng *et al.* show a basic QoS model to Web services composition in [88], although it can be applied to discovery and selection. Their QoS model has a limited number of QoS criteria, though it is extensible. These criteria comprise five common properties that can be applied to any WS: execution price, execution duration, reputation, successful execution rate, and availability. They also provide formulas for computing those QoS criteria values, both for atomic and composite services. These formulas can be seen as utility functions.

Furthermore, Zeng *et al.* propose a global service selection process based on these utility functions. Actually, they provide two techniques to perform the selection: (1) using QoS matrix normalization, and (2) obtaining the optimum by a Integer Programming method, which is more reliable. However, this proposal do not take semantics into account and that the utility functions are fixed, so the user can define its preferences only by means of weights.

Another non-semantic, but noteworthy proposal, by Ruiz-Cortés *et al.*, describes an automatic, QoS-aware provisioning using Constraint Programming [76]. They model requests and service descriptions using a language specially devised for transforming these descriptions into Constraint Satisfaction Problems (CSP). Thus, provisioning processes are performed by checking the conformance of both request and candidate services CSPs, in order to discover the conformant services.

Then, the best service is selected by means of the optimization of a weighted composition of utility functions, defined over each relevant QoS parameter. This optimization is solved by Constraint Satisfaction Optimization Problems (CSOP). This approach provides a high level of expressiveness by defining user preferences as utility functions that the CSOP solver can handle with, including inequalities and non-linear expressions.

A semantically-enabled matching using CSPs is presented in [51] by Kritikos and Plexousakis, based on [76]. They propose an ontology similar to the proposed by Maximilien and Singh [60], mixing requests and QoS-aware service descriptions within OWL-S. Moreover, they present a matching algorithm to infer equivalences between differently named QoS parameters that are semantically equivalent, although it is generally undecidable. Their extension ontology, OWL-Q, is separated in several facets that concentrates on a

particular part of their QOS WS description, such as connection with OWL-S instances,the actual QOS descriptions, metrics facets that provide classes to formally define QOS metrics, units, etc.

Concerning discovery and selection, they use CSPs to perform the matchmaking of compatible provided services, and then select the best service by means of a weighted composition of utility functions, which balance the worst and best scenarios to compute the utility value. However, these user preferences are not included in their QOS ontology.

Finally, an approach that merges discovery and selection algorithms execution is presented by Vu *et al.* [86]. They provide an extension to WSMO ontology in order to support QOS properties. Their extension model is based on axioms from the underlying WSMO ontology. They show a QOS-aware discovery framework that takes QOS values of WS based on user feedback and perform the discovery process, ranking the services in terms of QOS compliance. Actually, the ranking of services is obtained with regards to user preferences, defined by relative weights. Additionally, they sketch both a centralized architecture and a scalable one, that can be deployed into a peer-to-peer network.

3.3 User Preferences and QoS Ontologies

The following proposals do not fall exactly into any of the two previous Sections, because they focus on providing QOS or user preferences descriptions. These proposals constitute an important contribution to successfully express QOS within SWS frameworks, that most of them can be applied to. Although it is not the focus of this Section, their discovery and selection approaches are also briefly described in the following.

An extension to DAML-S to include QOS profiles is proposed in [89] by Zhou *et al.*. Their extension is divided in three layers with different roles. Thus, the QOS profile layer provides support for the different roles of the provisioning process: service provider, user requests (*inquiries*), and templates. This layer provide three common superclasses for matchmaking. The constraints that can be defined in each profile can be described by properties definitions and cardinality. The QOS property definition layer allows to specify the domain of the QOS properties, *i.e.* core, input, output, preconditions and effects. Finally, the QOS metrics layer defines how each QOS property is measured, and who is the organization that guarantees that measurement.

Furthermore, Zhou *et al.* provide a basic profile with commonly used QOS properties defined, such as cost, response time, reliability and throughput.

Their proposal only allows order constraint between QoS parameters, so it performs discovery and selection using DLs. The QoS ontology is simple and can be easily linked to the DAML-S service profile. However, its selection algorithm uses matching degrees to order the resulting set of services, so the user preferences can not be expressed, as they are inherent to that selection algorithm. Thus, the different QoS constraints stated in the profile are the only way to influence in the selection process.

Another DAML-based proposal is also presented in [82], where Bilgin and Singh provide a DAML-based query language, instead of just extending OWL-S. Using this Semantic Web Services Query and Manipulation Language, they advertise QoS attributes and perform the selection. They provide a simple ontology for service categories and QoS attributes associated to these categories. The main drawbacks of this approach are the same as in [73], with limitations on the expressiveness of queries, due to the use of DAML as its foundation. Thus, user preferences can not be expressed in those queries, and are inherent to their selection algorithm, as in [89]. Additionally, their ontology is coupled with a UDDI registry that has to be used in their proposal, including the URL of the registry inside the ontology, for instance.

Maximilien and Singh present a framework and a QoS ontology for dynamic selection in [60]. They use an agent-based approach within their Web Services Agent Framework (WSAF). Thus, they provide a complete architecture to perform service selection, that uses a layered QoS ontology. This ontology has the following three layers:

- The QoS upper ontology, which defines basic concepts associated with a QoS parameter, such as measurement, relationships, and aggregate support.
- The QoS middle ontology, which defines the most frequent QoS parameters and metrics encountered in distributed systems, such as availability, capacity, cost, security, interoperability, performance, etc.
- The user-defined lower ontology that defines concepts from the domain of each service.

This framework constitutes a very comprehensive solution to describe QoS in SWS, and it is referenced by other authors [27, 51, 70]. However, user preferences based on QoS parameters from their ontology have to be defined externally to the ontology itself, in a QoS policy description that WSAF apply to perform the selection process.

Finally, Dobson *et al.* presents QoSOnt in [27], which is an ontology that extends OWL-S to describe QoS attributes and metrics. Their approach is similar to the one from Maximilien and Singh, separating QoSOnt ontology in different smaller ontologies. Thus, the base ontology contains basic QoS concepts, such as *Attribute* and *Metric*. Then, concepts specific to some attribute can be built into separate ontologies on top of the base concepts. For instance, a separate ontology for availability can be specified, which contains concepts and metrics that can be only applied to availability specification. Additionally, generic metrics concepts can be defined in another separate ontology, for reuse.

These metrics also define the user preferences applied to them, using the *acceptability* direction, that is the preferred tendency of metric values (*e.g.* the higher the best). However, Dobson *et al.* do not explicitly show how to perform selection, and their proposal suffers from OWL limitations, so they have to use an *ad hoc* XML language to allow custom data ranges.

3.4 Analysis and Conclusions

Once all the different proposals have been described, an analysis and a framework for comparison between their different features can be done. Table §3.1 shows this comparison, merging proposals from the previous sections so they can be compared each other. In this table, ordered by the order of exposition, the following features are shown:

- **QoS Props.**: Whether each proposal takes QoS properties into account when performing its defined processes or not.
- **User Preferences**: How a proposal expresses user preferences.
- **Discovery**: The discovery technique that a proposal uses.
- **Selection**: The selection technique that a proposal uses.

In the following, each feature is independently analyzed, and then global conclusions are discussed.

3.4.1 QoS Properties Definition

Concerning QoS properties, there is a straightforward tendency to support their description in proposals that are mainly focused on selection processes.

Proposal	QoS Props.	User Preferences	Discovery	Selection
Paolucci *et al.* [69]	No	Fixed	DLs matching	Matching degree
Sycara *et al.* [83]	No	Fixed	DLs matching	Matching degree
González-Castillo *et al.* [36]	No	Fixed	DLs matching	Matching degree
Lutz & Sattler [56]	No	Fixed	DLs matching	Matching degree
Li & Horrocks [53]	No	Fixed	DLs matching	Matching degree
Benatallah *et al.* [11]	No	Fixed	Rewriting	Rewriting
Benbernou & Hacid [12]	No	Fixed	DLs + CSP	Not defined
Ran [73]	Yes	Fixed	UDDI	Not defined
Liu *et al.* [55]	Yes	Weights	Not defined	QoS matrix
Pathak *et al.* [70]	Yes	Weights	DLs matching	QoS matrix
Wang *et al.* [87]	Yes	Tendencies	Not defined	QoS matrix
Zeng *et al.* [88]	Yes	Utility & weights	Not defined	Integer Prog.
Ruiz-Cortés *et al.* [76]	Yes	Utility & weights	CSP	CSOP
Kritikos & Plexousakis [51]	Yes	Utility & weights	CSP + Matching	CSOP
Vu *et al.* [86]	Yes	Weights	DLs Matching	Reputation Rank
Zhou *et al.* [89]	Yes	Fixed	DLs Matching	Matching degree
Bilgin & Singh [82]	Yes	Fixed	UDDI	Query language
Maximilien & Singh [60]	Yes	External	DLs Matching	Matching degree
Dobson *et al.* [27]	Yes	Tendencies	DLs Matching	Not defined

Table 3.1: *Semantic provisioning proposals.*

The reason clearly is that SWS discovery is treated as a functional filter, where candidate services are chosen depending on the functionality requested. However, some selection proposals, such as [76, 87], also advocate the use of QoS properties in discovery processes, provided that these properties further filter the set of candidate services. In these cases, QoS properties have to be separated into two groups: the ones that purely filter the candidates, and the ones that define preferences used in selection processes.

Furthermore, the current trend in the SWS provisioning research field is to gradually include QoS definitions within proposals, provided that research on functional discovery is attracting less attention because current proposals are well established.

3.4.2 User Preferences Definition

There are several choices to define user preferences for SWS. First of all, most of the proposals that are focused on discovery processes and/or perform simple selection offer fixed user preferences, *i.e.* they do not provide alternatives in order to allow a user to define his or her own preferences.

Secondly, tendencies and weights constitute a simple, yet powerful alternative to express user preferences. These formalisms are widely extended between SWS selection proposals, because they enable easier and efficient QoS parameter transformations using vectorial operations, providing a reasonable level of precision and recall. Tendency information usually comes with relative weights, too.

Finally, the most expressive alternative to define user preferences seems to be utility functions, which are also used in conjunction with weights. Although they may be complex to define, they constitute a very powerful approach to properly define QoS-aware user preferences. Proposals that make use of utility functions perform selection processes using optimization techniques, such as Integer Programming or Constraint Programming.

3.4.3 Discovery Techniques

Most of the discussed approaches perform the discovery process by means of DLs matching. This is clearly a consequence of DLs spreading among the Semantic Web applications. Actually, this approach is the most straightforward and powerful enough to match requested capabilities and service descriptions, which are mostly defined by functional properties that can be easily described within DLs ontologies.

However, there are other proposals that use different formalisms, including logical ones. Those proposals may offer higher precision, but this may produce efficiency and even decidability problems.

3.4.4 Selection Techniques

The last feature analyzed is the selection techniques that are been used. In this case, most discovery focused proposals compute matching degrees in order to rank candidate services. This approach does not generate many extra computing cost to discovery, but relax the matching algorithm. However, the performed selection results with low precision, and does not allow the user to state his or her preferences.

More powerful and precise solutions are implemented in selection proposals. They propose to rank services by means of user-defined preferences, that are collected and transformed into QoS matrices or optimization problems. On the one hand, QoS matrix based algorithms are easier to implement, performing normalizations, vectorial operations, and applying weighted relations. On the other hand, optimization problems can be solved using several techniques, such as Integer Programming or Constraint Programming. These solvers become a more efficient, expressive and powerful approach to service selection.

3.4.5 Final Conclusions

Several conclusions can be obtained from the discussed comparison. Firstly, there are a few proposals that uses utility functions to express user preferences [51, 76, 88], although only [76] allows the user to define complex utility functions. These three proposals use optimization techniques, as Integer Programming or Constraint Programming, to select the best services, providing more efficiency and precision than other approaches based no matching degrees. Therefore, utility functions become the natural choice to define highly expressive user preferences.

Secondly, there are many proposals that provide a semantic framework to define QoS properties [27, 51, 60, 70, 82, 87, 89], although [60] do not handle user preferences in their ontology and [82, 89] have a fixed definition of user preferences, inherent to their selection algorithm. [51] is the most expressive when defining user preferences, followed by [27, 70, 87], that limit their

preferences to weights and parameter tendencies. According to all those proposals, it is clear that QoS properties have to be defined semantically in order to apply them to SWS provisioning.

Thirdly, it is commonly accepted that functional SWS discovery has to be performed by DLs reasoners, that are able to match user requests and service capabilities. Although there are some proposals that use other approaches, they are mainly non-semantic proposals, or try to extend UDDI registries with semantic information. These other approaches are not successful and efficient enough to take them into account.

Finally, we conclude that none of the above proposals semantically define user preferences, although in [27, 87] the authors include in their ontology extension the tendency of QoS parameters. What is more, most of the proposals that perform selection tasks in terms of user preferences describe them using *ad hoc*, non-semantic descriptions completely decoupled with the ones used to describe service functionality, causing a semantic gap between functional descriptions and user preferences.

The motivation of our research work is precisely to tackle the previous problems. Most recent proposals use utility functions to express user preferences, and there are many QoS ontologies which a novel proposal should be integrated with. Our proposal is to mix the expressiveness of utility functions and weights proposed by Ruiz-Cortés *et al.*, the semantic definition of QoS from Maximilien and Singh or Kritikos and Plexousakis, and an extension to give semantics to utility functions. Thus, this proposal takes full advantage of Semantic Web approaches on selecting the best services, while allowing to define user preferences using the most expressive solution, *i.e.* utility functions. Furthermore, we put functional, non-functional (QoS), and user preferences at the same semantic level, by means of using extensions to current SWS ontologies.

Part III

Final Remarks

Chapter 4

Conclusions and Future Work

In order to sum up this research report, this chapter presents a summary of our research and the main conclusions that show that our initial hypothesis was correct. Additionally, future work areas that arise from our discussion are described, along with a brief review of our ongoing work.

4.1 Conclusions

In this research report, we present a thoroughly review of the state-of-the-art on QoS-aware SWS provisioning. An analysis of current SWS frameworks is discussed, comparing each proposal and showing their approach to service provisioning. Although initially SWS frameworks cannot perform QoS-aware selection, there are several extensions to these frameworks to allow it. These extension approaches are also discussed and analyzed.

These analysis show that our initial hypothesis was correct, and can be summarized in the following statements:

- There is a gap between functional and QoS preferences, with each proposal providing different QoS ontologies that are not properly linked with functional descriptions.
- Selection techniques are highly coupled with the concrete QoS representation chosen by each proposal.
- Expressiveness of QoS user preferences varies between each proposal.

From these statements, that support our work hypothesis, we conclude that there is a need for a holistic solution that offer a QoS-aware SWS provisioning, by providing a generic extension to current SWS ontologies that allows to define QoS preferences at the same abstraction level than service functionality, and that is loosely coupled with the actual selection technique to be used. This solution has to be based on current QoS ontologies, while supporting the most expressive approach to define user preferences, *i.e.* utility functions.

4.2 Future Work

Our future research work is focused on developing such QoS-aware provisioning solution. We have already published two early contributions on this field, and have another one in reviewing process. Thus, our research work is aimed at providing a complete solution to the identified problems in QoS-aware SWS provisioning.

In particular, we are extending our initial approach on semantic description of user preferences, by developing the proposed ontology and evaluating

its soundness and applicability to a concrete SWS provisioning scenario. This ontology has to be decoupled with the actual selection algorithm to be used, and it has to allow to connect it to already proposed QOS ontologies.

In order to decouple user preferences descriptions with the selection algorithm, we plan to develop generic transformations that transform those user preferences into optimization problems that are described so that can be used within specific selection algorithms, such as the discussed ones in Sec. §3.2 . These transformations are being developed using XSLT style sheets.

Part IV

Appendices

Appendix A

Relevant Publications

In this appendix, two published contributions that provide an early approach to QoS-aware semantic provisioning are enclosed along with another paper that is in process of review. These contributions show our first attempt to provide a solution to the identified problems in previous sections. The papers are included in the following preserving their original layout.

Firstly, we present a paper published in the proceedings of the *Fifth International Conference on Service Oriented Computing* (ICSOC 2007), where a hybrid SWS discovery approach that takes care of QoS properties is presented. This conference has been ranked in the *A* category in the CORE ranking. Moreover, that conference edition had an acceptance rate of 78.4 percent, and it has an h-index of 10.

Secondly, our contribution to the *First Non-Functional Properties and Service Level Agreements in Service Oriented Computing Workshop* (NFPSLA-SOC 2007) is showed. In this contribution, we provide an approach to semantically describe user preferences. This workshop had an acceptance rate of 38.8 percent and is going to be published by Springer in its LNCS series.

Finally, a submission to the *2008 International Conference on Semantic Web and Web Services* (SWWS 2008) that is currently been reviewed is also included. This contribution presents a selection model to perform semantic provisioning using an ontology for user preferences that is also provided. This conference is positioned in the Top 80 of the CSCR ranking.

An Hybrid, QoS-Aware Discovery of Semantic Web Services Using Constraint Programming*

José María García, David Ruiz, Antonio Ruiz-Cortés, Octavio Martín-Díaz, and Manuel Resinas

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática
Av. Reina Mercedes s/n, 41012 Sevilla, España
josemgarcia@us.es

Abstract. Most Semantic Web Services discovery approaches are not well suited when using complex relational, arithmetic and logical expressions, because they are usually based on Description Logics. Moreover, these kind of expressions usually appear when discovery is performed including Quality-of-Service conditions. In this work, we present an hybrid discovery process for Semantic Web Services that takes care of QoS conditions. Our approach splits discovery into stages, using different engines in each one, depending on its search nature. This architecture is extensible and loosely coupled, allowing the addition of discovery engines at will. In order to perform QoS-aware discovery, we propose a stage that uses Constraint Programming, that allows to use complex QoS conditions within discovery queries. Furthermore, it is possible to obtain the optimal offer that fulfills a given demand using this approach.

Keywords: Discovery Mechanisms, Quality-of-Service, Semantic Matching, Constraint Programming.

1 Introduction

Most approaches on automatic discovery of Semantic Web Services (SWS) use Description Logics (DLs) reasoners to perform the matching [7,13,15,18,26,27]. These approaches have limitations regarding with the expressiveness of searches, especially when there are Quality-of-Service (QoS) conditions integrated within queries. For instance, a condition like "find a service which $availability \geq 0.9$, where $availability = MTTF/(MTTF + MTTR)$"[1] can not be expressed in DLs. Although there are proposals that extend traditional DLs with concrete domains in many ways [9], they still have limitations on expressing complex conditions [1,14], as in the previous example. These complex conditions usually

* This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472).

[1] $MTTF$ stands for "Mean Time To Failure", while $MTTR$ stands for "Mean Time To Repair". Both of them are QoS parameters often used to define service availability.

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 69–80, 2007.

appear when performing a QoS-aware discovery, so in this case DLs reasoning is not the most suitable choice.

QoS conditions are contemplated in several SWS discovery proposals. For instance, Wang *et al.* extend WSMO framework to include QoS parameters that allow to discover the best offer that fulfills the demanded conditions [30]. Benbernou and Hacid propose the use of constraints, including QoS-related ones, to discover SWS [3]. Moreover, Ruiz-Cortés *et al.* model the QoS conditions as Constraint Satisfaction Problems (CSPs) [23], but in the context of non-semantic Web Services.

Our proposal is an hybrid architecture to discover SWS. Discovery may be split into different stages, each of them using the best suited engine depending on the features of the corresponding stage. We identify at least two stages in this process: QoS-based discovery and functional (non-QoS) discovery. The former may be done using Constraint Programming (CP), as proposed in the case of non-semantic Web Services in [23], while the latter is usually performed by DLs reasoners, although it is not restricted to use other techniques.

Our approach allows to filter offers, stage by stage, using a proper search engine until the optimal offer that fulfills a demand is found. This optimization is accomplished due to the proposed use of CP in the QoS-aware discovery stage, also enabling the definition of more complex conditions than defined ones using DLs. Furthermore, our proposed architecture is loosely coupled and extensible, allowing the addition of extra discovery engines if necessary.

The rest of the paper is structured as follows. In Sec. 2 we introduce related works on discovering SWS, discussing their suitability to perform a QoS-aware discovery. Next, in Sec. 3 we present our hybrid discovery proposal, explaining the proposed architecture and how CP can be used in a QoS-aware semantic discovery context. Finally, in Sec. 4 we sum up our contributions, and discuss our conclusions and future work.

2 Discovering Semantic Web Services

In this Section, we discuss related work on discovering SWS, describing the different approaches and analyzing their suitability to handle QoS parameters and conditions, in order to perform a QoS-aware discovery.

2.1 Preliminaries

Each proposal uses its own terminology to refer to the entities involved in the discovery process, especially its descriptions of the requested and provided services. For the sake of simplicity, we use one single notation along this paper.

We refer to a demand (denoted by the Greek letter delta, i.e. $\delta emand$) as a set of objectives that clients want to accomplish by using a service that fulfills them. It may be composed of functionality requirements and QoS conditions that the requested service must fulfill, such as "find a service which $availability \geq 0.9$, where $availability = MTTF/(MTTF + MTTR)$". The different proposals

refer to demands as goals [22], queries [2,3,13], service request [7,19,27] or user requirements [30].

An offer (denoted by the Greek letter omega, i.e. *ωffer*) of a service is the definition of a SWS that is publicly available from a service provider. An offer may be composed of functionality descriptions, orchestration, choreography, and QoS conditions of the given service. For instance, an offer can consist in a QoS condition like "$MTTF$ is from 100 to 120 inclusive and $MTTR$ is from 3 to 10 also inclusive". Different approaches refer to offers as advertisements [2,13,30], service capabilities [19,22,27], or service profiles [7,15,16].

Most proposals on discovering SWS are built upon one of the following description frameworks. Firstly, OWL-S [16] is a DARPA Agent Markup Language program initiative that defines a SWS in terms of an upper ontology that contains concepts to model each service profile, its operations and its process model. It is based on OWL standard to define ontologies, so it benefits from the wide range of tools available. Secondly, the Web Service Modeling Ontology (WSMO) [22] is an European initiative whose goal, as OWL-S, is to develop a standard description of SWS. Its starting point is the Web Service Modeling Framework [6], which has been refined and extended, developing a formal ontology to describe SWS in terms of four core concepts: ontologies, services, goals and mediators. Finally, the METEOR-S project from the University of Georgia takes a completely different, but aligned approach than the others. Its main target is to extend current standards in Web Services adding semantic concepts [25], among others contributions discussed here. These extensions make use of third party frameworks, including the previous two, to semantically annotate Web Service descriptions. These proposals have extensions to take care of QoS parameters.

2.2 Related Work

In the context of DAML-S (the OWL-S precursor), Sycara *et al.* show how semantic information allows automatic discovery, invocation and composition of Web Services [27]. They provide an early integration of semantic information in a UDDI registry, and propose a matchmaking architecture. It is based on a previous work by Paolucci *et al.*, where they define the matching engine used [19]. This engine matches a demand and an offer when this offer describes a service which is "sufficiently similar" to the demanded service, i.e. the offered service provides the functionality demanded in some degree. The problem is how to define that degree of similarity, and the concrete algorithm to match both service descriptions. They update their work to OWL-S in [28].

Furthermore, there are proposals that perform the matchmaking of SWS using DLs [7,13,15]. Particularly, González-Castillo *et al.* provide an actual matchmaking algorithm using the subsumption operator between DLs concepts describing demands and offers [7]. They use existing DLs reasoners, as RACER [8] and FaCT [11], to perform the matchmaking. On the other hand, Lutz and Sattler [15] do not provide an algorithm, but give the foundations to implement it using subsumption, like Li and Horrocks [13], who also give hints to implement a prototype using RACER.

These three works define different matching degrees as in [27], from exactly equivalents to disjoint. All of them perform this matching by comparing inputs and outputs. Apart from that, neither of them can obtain the optimal offer using QoS parameters. However, Benatallah *et al.* propose to use the degree of matching to select the best offer in [2], but it results to be a NP-hard problem, as in any optimization problem [4].

On the other hand, Benbernou and Hacid realise that some kinds of constraints are necessary to discover SWS, including QoS related ones, so they formally discuss the convenience of incorporating constraints in SWS discovery [3]. However, instead of using any existing SWS description framework, their proposal uses an *ad-hoc Services Description Language*, in order to be able to define complex constraints. In addition, the resolution algorithm uses constraint propagation and rewriting, but performed by a subsumption algorithm, instead of a CSP solver.

Concerning WSMO discovery, Wang *et al.* propose an extension of the ontology to allow QoS-aware discovery [30]. The matchmaking is done by an *ad-hoc* algorithm to add QoS conditions to offers and demands. Their implementation has some limitations, as the algorithm can only be applied to real domain attributes, and is restricted to three types of relational operators.

Ruiz-Cortés *et al.* provide in [23] a framework to perform QoS-aware discovery by means of CP, in the context of non-semantic Web Services. They show the soundness of using CP to improve the automation of matchmaking from both theoretical and experimental points of view. Although CP solving is a NP-hard problem, the results of their experimental study allow to conclude that CP-based matchmakers are practically viable despite of its, theoretical, combinatorial nature. This work is the starting point of our approach on using CP to perform QoS-related stages of our hybrid SWS discovery proposal.

2.3 Frameworks

As an application of their previous work, Srinivasan *et al.* present an implementation to development, deployment and consumption of SWS [26]. It performs the discovery process using the proposals introduced in [19,27]. They show performance results and detail the implementation of OWL-S and UDDI integration, so it can be used as a reference implementation to OWL-S based discovery, but without QoS conditions.

IRS-II [18] is an implemented framework similar to WSMF [6], that is able to support service discovery from a set of demands. It uses descriptions of the reasoning processes called Problem Solving Methods (PSM), similar to OWL. Moreover, IRS-III [5] updates this previous implementation, using WSMO ontology to model SWS, and providing an architecture to discovery, composition and execution SWS. All of them can not handle with QoS conditions, although they are extensible so they may support them.

Another interesting proposal is done in [24], where Schlosser *et al.* propose a graph topology of SWS providers and clients, connected between them as in a peer-to-peer (P2P) network. In this scenario, searching, and specially publishing,

are done very efficiently, without the need of a central server acting as a register of offers and demands. In addition, the network are always updated, due to an efficient topology maintenance algorithm. This structure of decentralized registries is proposed in METEOR-S for semantic publication and discovery of Web Services [29]. The semantic matching algorithm uses templates to search inputs and outputs of services described with ontological concepts, without the use of a specific reasoner, or the possibility to express QoS conditions. Although the matchmaking is too simple, the idea of a P2P network can be adopted in our proposal without troubles.

Our proposal is open to be implemented in the context of any of the presented frameworks in this section. The proposed architecture that we present in the following section does not impose any restriction on the SWS framework used (i.e. OWL-S, WSMO or METEOR-S), and can be composed of any number of the discovery engines discussed in Sec. 2.2, due to its hybrid nature. Furthermore, it can be materialized as the discovery component of implementations like IRS-III [5] or OWL-S IDE [26].

3 Our Proposal

The addition of constraints, specially QoS-related ones, to SWS descriptions, turns most approaches on discovering SWS insufficient, because they mainly use DLs, which are usually limited to logical and relational expressions when describing QoS conditions. CP becomes necessary to manage more complex QoS conditions, so a demand can be matched with the best available offer. Instead of using solely CP to perform the discovery, we present an hybrid solution that splits the discovery into different stages.

3.1 Hybrid Semantic Discovery Architecture

An abstract architecture of our proposal is sketched in Fig. 1, where we show how the different components are connected between them. Here, the dashed line defines the boundaries of our hybrid discovery engine.

Q document corresponds to the query that a client wants to use to discover services, i.e. the demand. This query may be expressed in any desired language that the scheduler can handle, such as a SPARQL query [21], a WSMO goal, a faceted search [20], or even it may be defined visually using a GUI.

R is the result set of offers that fulfill the query Q. It is the output of the discovery process, possibly being an empty set, the best offer, or an ordered list of offers by an optimality criterion. The format of this output should conform the specification of a concrete SWS framework in order to successfully invoke the discovered service(s).

The different stages of the hybrid discovery are performed by the best suited discovery engine. In Fig. 1, $E_1 ... E_n$ represent the engines to be used in each corresponding stage. The core component of our proposed architecture is responsible to send the input data to each engine, by decomposing the query Q

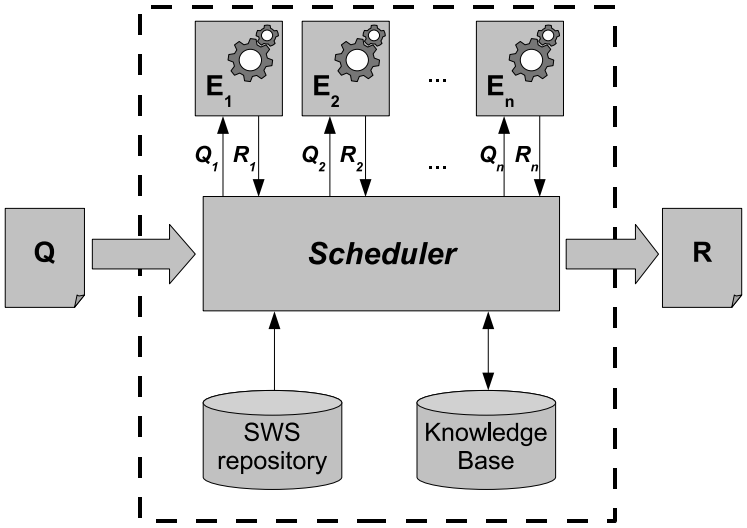


Fig. 1. Architecture of our hybrid discovery proposal

in subqueries (Q_i), and to recover its corresponding output (R_i), joining all of them to output the final result R. These input and output formats depend on the concrete engine of each stage. Thus, if we are performing a QoS-aware stage, the input must be modeled as a CSP, so CP can be applied to perform this kind of stage. Additionally, it is possible to use a DLs engine to perform non-QoS discovery, or a template matchmaker [29], for instance.

Offers have to be published in some kind of repository so they can be matched with demands by means of the different discovery engines used in our approach. This SWS repository may be implemented in different ways: as a semantically-extended UDDI registry [26], as a decentralized P2P registry [29], or as a WSMO repository [5], for instance.

In addition, our architecture takes care of the NP-hard nature of optimization [4], so we propose to include a knowledge-base (KB) that cache already performed discoverings, so the execution of the discovery process becomes faster. Thus, we store executed queries related with their result set of SWS from the repository component, into the KB. IRS-II implementation uses a similar idea to accelerate discovery [18].

Finally, the core component of our proposal is the scheduler. It has to analyze the query Q, split the discovery task into stages, and communicate with discovery engines, in order, providing them with a correct input, and obtaining a corresponding output. These different outputs are processed stage by stage, so the set of matching offers from the SWS repository are gradually made smaller. Each discovery stage may be concurrently or sequentially launched in order, depending on the query nature. Moreover, the scheduler update the KB using the results of discovery process, which is output as R.

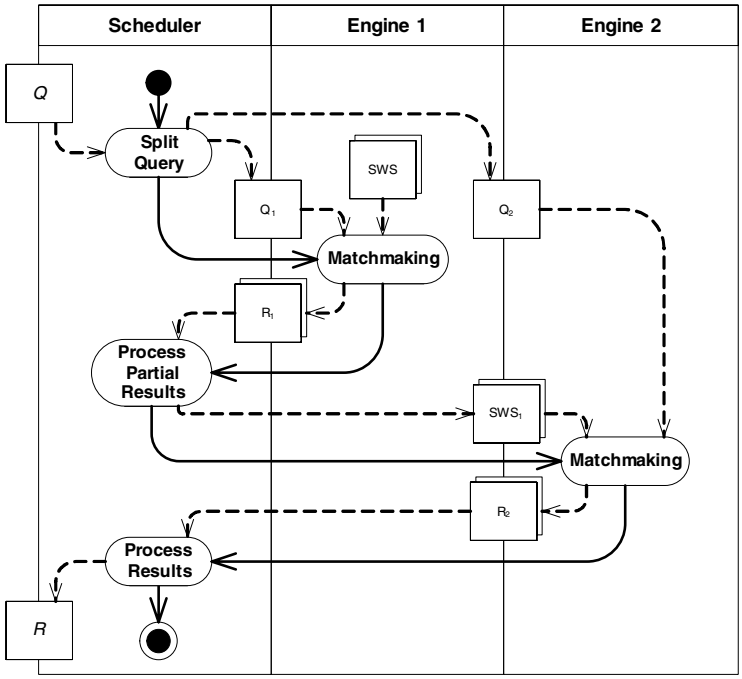


Fig. 2. Activity diagram of our discovery process

Fig. 2 shows the activity diagram of an hybrid discovery process performed in two stages using two different engines. This diagram can be easily extended if we need more stages. For instance, using a similar format from [13], a query $Q = (ServiceProfile \cap A \geq 0.9)$, where A corresponds to availability, is split by the scheduler into two subqueries: $Q_{DL} = (ServiceProfile)$ being the part expressed in DLs, and $Q_{CP} = (\{A\}, \{[0..1]\}, \{A \geq 0.9, A = MTTF/(MTTF + MTTR)\})$ the part modeled by a CSP.[2] $ServiceProfile$ corresponds to the definition of a demand in terms of the OWL-S profile of a service. In this scenario, the scheduler perform a matchmaking firstly using a DLs engine with Q_{DL}, obtaining the offers that satisfy this subquery. Then, with this resulting subset of SWS from the registry, the scheduler performs a matchmaking using a CP engine and Q_{CP}, so the final result is the optimal offer that satisfies the whole query Q. For the sake of simplicity we do not contemplate the KB role in Fig. 2, because it only provides a way to speed up the process.

This hybrid discovery architecture has many advantages. It is loosely coupled, due to the possibility to use any discovery engine. Also, the input query format is not restricted, as the scheduler can analyze a given query, so it can infer the

[2] A CSP consists in a three-tuple of the form (V, D, C) where V is a finite, non-empty set of variables, D is a finite, non-empty set of domains (one for each variable) and C is a set of constraints defined on V. The solution space of a CSP is a set composed of all its possible solutions, and if this set is not empty, the CSP is said to be satisfiable.

concrete engines to use and their order. Moreover, our proposed architecture can be applied to any existing SWS framework and corresponding repositories, taking benefit of the wide range of tools already implemented. Our proposal is able to use the best suited engine to perform the corresponding search of a part of the input query, being used in most cases CP for QoS-related part, and DLs for non-QoS discovery, but without restrictions on adding more engines.

3.2 QoS-Aware Semantic Discovery

Focusing on the QoS-aware discovery stage, the scheduler sends the QoS-related part of the query to a CSP solver, so the set of offers that fulfills the requirements of a given demand can be obtained, or even obtain the optimal offer. To do so, QoS conditions and their involved QoS parameters, defined in demands and offers, must be mapped onto constraints in order to use a CSP solver.

Thus, each parameter must be mapped onto a variable (with its corresponding domain), and each condition must be mapped onto a constraint. At this point, we have to extend the demand and offer concepts previously presented because both of them may contain complementary information. We consider they are composed of two parts: requirements and guarantees. On the one hand, a demand δ is composed of two parts: δ^{γ}, which asserts the conditions that the client meets (i.e. γuarantees), and δ^{ρ}, which asserts the conditions that the provider shall meet (i.e. ρequirements). Similarly, an offer ω can also be considered composed of ω^{γ} (what it guarantees) and ω^{ρ} (what is required from its clients).

For example, consider the demand "The availability shall be less than 0.9, where $A = MTTF/(MTTF + MTTR)$" (δ^{ρ}); and the offer "The mean time to failure is from 100 to 120 minutes inclusive, while the mean time to repair is from 3 to 10 minutes inclusive" (ω^{γ}). Assuming that $MTTF$, $MTTR$ and A range over real numbers, their corresponding CSPs are defined as follows:

$$\begin{aligned}\delta^{\rho} &= (\{A, MTTF, MTTR\}, \{[-\infty, +\infty], [0, +\infty], [0, +\infty]\},\\ &\quad \{A \geq 0.9, A = MTTF/(MTTF + MTTR)\})\\ \omega^{\gamma} &= (\{MTTF, MTTR\}, \{[0, +\infty], [0, +\infty]\},\\ &\quad \{100 \leq MTTF \leq 120, 3 \leq MTTR \leq 10\})\end{aligned}$$

Additionally, the demand may also contain the condition "My host is in Spain" (δ^{γ}); and the offer "For American and British clients only" (ω^{ρ}), so the offer provider requires from its clients some guarantees. Consequently, assuming that $COUNTRY$ variable ranges over the powerset of $\Lambda = \{ES, US, UK, FR\}$, i.e. $\mathcal{P}(\Lambda)$, their corresponding CSPs are defined as follows:[3]

$$\begin{aligned}\delta^{\gamma} &= (\{COUNTRY\}, \{\mathcal{P}(\Lambda)\}, \{COUNTRY = \{ES\}\})\\ \omega^{\rho} &= (\{COUNTRY\}, \{\mathcal{P}(\Lambda)\}, \{COUNTRY \subseteq \{UK, US\}\})\end{aligned}$$

[3] Note QoS parameters can be linked together in order to express more complex conditions, such as $\{COUNTRY = \{ES, UK, FR\} \Rightarrow 5 \leq MTTR \leq 10, COUNTRY = \{US\} \Rightarrow 5 \leq MTTR \leq 15\}$. These conditions can be interpreted as "the $MTTR$ is guaranteed to be between 5 and 10 if client is Spanish, British, or French, else between 5 and 15 if client is American".

The conditions previously expressed in natural language should be expressed in a semantic way, using QoS ontologies such as the one proposed by Maximilien *et al.* in [17]. Thus, semantically defining QoS parameters that take part in such conditions, and integrating these descriptions in any SWS framework, they can be interpreted later as a CSP so a solver can process them in the corresponding discovery stage.

These CSPs allow to check for consistency and conformance of offers and demands. A demand or an offer is said to be consistent if the conjunction of its corresponding CSPs (of requirements and guarantees) are satisfiable. On the other hand, an offer ω and a demand δ are said to be conformant if the solution space of the CSP of the guarantees of the offer (denoted by ψ_ω^γ) is a subset of the solution space of the CSP of the requirements of the demand (ψ_δ^ρ), and vice versa ($\psi_\delta^\gamma \subseteq \psi_\omega^\rho$) [23]. In the previous example, ω and δ are consistent, but they are not conformant, because $COUNTRY$ is guaranteed to be ES, but the offer requires it to be UK or US.

Finally, the ultimate goal of the matchmaking of offers and demands is to find a conformant offer that is optimal from the client's point of view. To do so, it becomes necessary to model the optimization task as a CSP, as with consistency and conformance checks. More specifically, finding the optimal can be interpreted as a Constraint Satisfaction Optimization Problem (CSOP), which requires to explicitly establish a preference order on the offer set. This order can be defined using a weighted composition of utility functions, which can be taken as a global utility function for the client.

Thus, each QoS parameter can have a utility function defined, and an associated weight to successfully describe how important the values that can take are for the client. Fig. 3 shows an example of how to discover optimal offers. In this case, we are assuming that the demand only specifies its requirements (Fig. 3a) and the offer only specifies what it guarantees (Fig. 3b), so the offer is conformant with the demand. The corresponding utility functions of the QoS parameters involved, i.e. $MTTF$ and $MTTR$, ranging over $[0, 1]$, are shown in Fig. 3c and 3d, respectively.

The utility function for $MTTF$ (Fig. 3c) is a piecewise linear function that defines a minimum utility if $MTTF$ is below 60 minutes; the utility grows linearly if $MTTF$ is between 60 and 120 minutes, and the utility reaches its maximum value if $MTTF$ is above 120. On the other hand, the utility function for $MTTR$ showed in Fig. 3d is a decreasing piecewise linear function. In order to obtain a global utility function of the offer, we consider that $MTTF$ has a weight of 70% and $MTTR$ 30%.

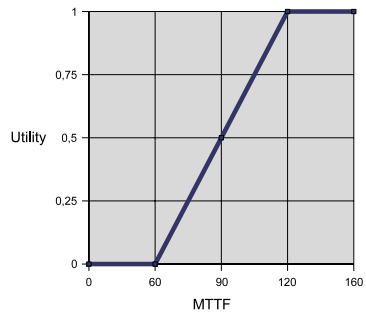
The offer from Fig. 3b must be checked for conformance with the demand from Fig. 3a, supposing that both descriptions have been previously checked for consistency, and that both are based on same QoS parameters, or they are defined using a compatible ontology. In this case, there is only one offer conformant, but there could be more than one, being necessary to obtain the optimal offer. To do so, utility functions for each offer have to be computed in order to compare them and get the maximum utility value, which corresponds with the optimal offer. In Fig. 3e we show the OPL [10] model for the computing of the utility function of the showed offer.

$$\delta^{\rho} \equiv A \geq 0.9 \wedge A = \frac{MTTF}{MTTF + MTTR}$$

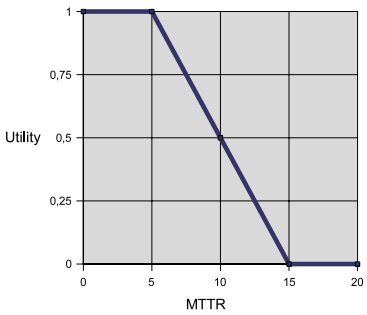
(a) Demand requirements.

$$\omega^{\gamma} \equiv 100 \leq MTTF \leq 120 \wedge 3 \leq MTTR \leq 10$$

(b) Offer guarantees.



(c) $MTTF$ utility function.



(d) $MTTR$ utility function.

```
//variables
range TYPE_MTTF 0..255;
var TYPE_MTTF MTTF;
range TYPE_MTTR 0..255;
var TYPE_MTTR MTTR;
range TYPE_UTILITY 0..100;
var TYPE_UTILITY U_MTTF;
var TYPE_UTILITY U_MTTR;
var TYPE_UTILITY UTILITY;
minimize
 UTILITY
subject to {
 // Offer guarantees
 100<=MTTF<=120;
 3<=MTTR<=10;
 // Utility function of MTTF
 MTTF<=60 => U_MTTF=0;
 60<MTTF<=120 =>60*U_MTTF=MTTF-60;
 MTTF>120=> U_MTTF=1;
 // Utility function of MTTR
 MTTR<=5 => U_MTTR=1;
 5<MTTR<=15 => 10*U_MTTR=15-MTTR;
 MTTR>15 => U_MTTR=0;
 // Utility aggregate of matching
 UTILITY = 70*U_MTTF + 30*U_MTTR;
};
```

(e) OPL model for computing utility.

Fig. 3. An example on obtaining optimal offers

Note that we compute the minimum value of the utility function, taking the worst-case scenario. This way, we say that an offer ω is optimal with regard to a utility function U if the minimum value of this function is the maximum among minimum values of all conformant offers. It is also possible to take other approaches when computing the utility function, like using the maximum value, a mean value, or the more general case of a weighted composition of the maximum and minimum value [12].

4 Conclusions and Future Work

In this work, we show that using a unique engine to discover SWS is not appropriate, due to each engine is usually designed for a concrete kind of search. For

instance, DLs reasoners are well suited when discovering SWS in terms of concepts and relations, but they can not handle complex numerical QoS conditions. Although there are extensions to allow concrete domains in DLs, reasoners have to implement them, and they may bring undecidability results.

We present an hybrid solution that consists in a n-stages discovery process, where each stage is performed using the most appropriate technique. Furthermore, we propose to use CP to perform QoS-aware discovery stages, so the optimal service(s) offered that fulfills a given demand can be found. In addition, our proposed architecture is extensible and loosely coupled, allowing to define complex QoS conditions, and to use utility functions based on QoS parameters to obtain the optimal offer. This architecture does not impose any restriction on the SWS framework and repository to use, allowing its materialization as a discovery component for current SWS implementations.

For future work, we are considering to define more precisely the scheduler and its interaction with the rest of the components. The query split mechanism has to be characterized, so do the results merging for each engine. Thus, a catalog of stages would be defined, including their order of execution. Moreover, we are considering to extend current SWS frameworks using a QoS ontology to define QoS parameters and conditions, allowing to express complex arithmetic, relational, and logical expressions in demands and offers.

Acknowledgments. The authors would like to thank the reviewers of the 5^{th} *International Conference on Service Oriented Computing*, whose comments and suggestions improved the presentation substantially.

References

1. Baader, F., Sattler, U.: Description logics with aggregates and concrete domains. Information Systems 28(8), 979–1004 (2003)
2. Benatallah, B., Hacid, M., Rey, C., Toumani, F.: Semantic reasoning for web services discovery. In: WWW Workshop on E-Services and the Semantic Web (2003)
3. Benbernou, S., Hacid, M.: Resolution and constraint propagation for semantic web services discovery. Distributed and Parallel Databases 18(1), 65–81 (2005)
4. Bonatti, P., Festa, P.: On optimal service selection. In: 14th international conference on World Wide Web, pp. 530–538 (2005)
5. Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Tanasescu, V., Pedrinaci, C., Norton, B.: IRS-III: A broker for semantic web services based applications. In: International Semantic Web Conference, pp. 201–214 (2006)
6. Fensel, D., Bussler, C.: The web service modeling framework WSMF. Electronic Commerce Research and Applications 1(2), 113–137 (2002)
7. González-Castillo, J., Trastour, D., Bartolini, C.: Description logics for matchmaking of services. Technical Report HPL-2001-265, Hewlett Packard Labs (2001)
8. Haarslev, V., Möller, R.: RACER system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 701–706. Springer, Heidelberg (2001)
9. Haarslev, V., Möller, R.: Practical Reasoning in RACER with a Concrete Domain for Linear Inequations. In: Int. Workshop on Description Logics (2002)

10. Van Hentenryck, P.: Constraint and integer programming in OPL. INFORMS Journal on Computing 14(4), 345–372 (2002)
11. Horrocks, I.: FaCT and iFaCT. In: Int. Workshop on Description Logics (1999)
12. Kritikos, K., Plexousakis, D.: Semantic QoS metric matching. In: ECOWS 2006, pp. 265–274. IEEE Computer Society Press, Los Alamitos (2006)
13. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: Int. World Wide Web Conference, pp. 331–339 (2003)
14. Lutz, C.: Description logics with concrete domains – a survey. In: Advances in Modal Logic, pp. 265–296 (2002)
15. Lutz, C., Sattler, U.: A proposal for describing services with DLs. In: Int. Workshop on Description Logics (2002)
16. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., et al.: OWL-S: Semantic Markup for Web Services. Technical Report 1.1, DAML (November 2004)
17. Maximilien, E.M., Singh, M.P.: A framework and ontology for dynamic web services selection. IEEE Internet Computing 8(5), 84–93 (2004)
18. Motta, E., Domingue, J., Cabral, L., Gaspari, M.: IRS-II: A framework and infrastructure for semantic web services. In: Fensel, D., Sycara, K.P., Mylopoulos, J. (eds.) ISWC 2003. LNCS, vol. 2870, pp. 306–318. Springer, Heidelberg (2003)
19. Paolucci, M., Kawamura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 333–347. Springer, Heidelberg (2002)
20. Prieto-Díaz, R.: Implementing faceted classification for software reuse. Commun. ACM 34(5), 88–97 (1991)
21. Prudhommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. Technical Report Working Draft, W3C (March 2007)
22. Roman, D., Lausen, H., Keller, U.: Web Service Modeling Ontology (WSMO). Technical Report D2 v1.3 Final Draft, WSMO (October 2006)
23. Ruiz-Cortés, A., Martín-Díaz, O., Durán Toro, A., Toro, M.: Improving the automatic procurement of web services using constraint programming. Int. J. Cooperative Inf. Syst. 14(4), 439–468 (2005)
24. Schlosser, M., Sintek, M., Decker, S., Nejdl, W.: A scalable and ontology-based P2P infrastructure for semantic web services. In: Peer-to-Peer Computing, pp. 104–111 (2002)
25. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding semantics to web services standards. In: Intl. Conference on Web Services, pp. 395–401 (2003)
26. Srinivasan, N., Paolucci, M., Sycara, K.: Semantic web service discovery in the OWL-S IDE. In: Hawaii International Conference on Systems Science (2006)
27. Sycara, K., Paolucci, M., Ankolekar, A., Srinivasan, N.: Automated discovery, interaction and composition of semantic web services. J. Web Sem. 1(1), 27–46 (2003)
28. Sycara, K., Paolucci, M., Soudry, J., Srinivasan, N.: Dynamic discovery and coordination of agent-based semantic web services. IEEE Internet Computing 8(3), 66–73 (2004)
29. Verma, K., Sivashanmugam, K., Sheth, A., Patil, A., et al.: METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services. Inf. Tech. Management 6(1), 17–39 (2005)
30. Wang, X., Vitvar, T., Kerrigan, M., Toma, I.: A QoS-aware selection model for semantic web services. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 390–401. Springer, Heidelberg (2006)

On User Preferences and Utility Functions in Selection: A Semantic Approach.*

José María García, David Ruiz, and Antonio Ruiz-Cortés

Universidad de Sevilla
Escuela Técnica Superior de Ingeniería Informática
Av. Reina Mercedes s/n, 41012 Sevilla, España
josemgarcia@us.es

Abstract. Discovery tasks in the context of Semantic Web Services are generally performed using Description Logics. However, this formalism is not suited when non-functional, numerical parameters are involved in the discovery process. Furthermore, in selection tasks, where an optimization algorithm is needed, DLs are not capable of computing the optimum. Although there are DLs extensions that can handle numerical parameters, they bring decidability problems. Other solutions, as hybrid approaches which use DLs in functional discovery and other formalisms in non-functional selection, do not provide a semantic framework to describe user preferences based on non-functional properties. In this work, we propose to semantically describe user preferences, so they can be used to perform selection within a hybrid solution. By using semantically described utility functions in order to define user preferences, our proposal enables interoperability between service offers and demands, while providing a high level of expressiveness in these preferences and including them within SWS descriptions.

Keywords: NFP-based Selection, Quality of Services, Utility Functions, Semantic Web Services.

1 Introduction

Concerning Semantic Web Services (SWS), discovery is one of the main research topics that have been widely studied and discussed, among others like composition. Description Logics (DLs) usually have become the natural choice when discovering SWS. Traditionally, discovery tasks have been interpreted as a functional filter, where demands are matched with compatible offers in terms of functionality. However, including non-functional properties (NFP) in the discovery process leads to an optimization problem. Selection of the best offer by means of their NFP has not been contemplated as a main task in discovering, so DLs reasoners are not well suited to select optimal offers. However, there are some proposals to perform NFP-based discovery, as the ones discussed in this work.

* This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472).

Optimization problems can be handled by solvers based on different formalisms, like Linear Programming, Constraint Programming, or Dynamic Programming, among others. Thus, it is possible to split discovery and selection tasks in terms of functional and non-functional requirements, so the former task can be performed by DLs reasoners, while the latter can be performed by solvers, taking a hybrid approach [3].

Focusing on selection, service demands have to state an optimality criterion, i.e. user preferences, so a solver can obtain the best offer in terms of these preferences. We propose to describe these user preferences by means of utility functions, whose domains are the different QoS parameters used to define NFP of service offers. In a SWS context, these utility functions have to be semantically described, allowing to match demands and offers described by different, but possibly equivalent, QoS parameters, enabling semantic interoperability between these descriptions.

The paper is structured as follows. In Sec. 2 we analyze current approaches on selecting SWS. Then, in Sec. 3 we show our proposal, describing what is a utility function and how to give semantics to it, showing an example. Finally, in Sec. 4 we discuss our conclusions.

2 Selecting Semantic Web Services

Once a set of services are discovered using a functional filter, the next step is to select the best offer in terms of NFP and user preferences. Thus, selection is modeled as an optimization problem. This kind of problem refers to a minimization or maximization of a real function, choosing the appropriate values of the involved variables.

In this context, the function to optimize is frequently called *utility function* or *objective function*. There are different techniques to obtain the optimal value of these functions, like Linear Programming, Constraint Programming or Dynamic Programming, for instance. In the following, we present the different approaches on selecting offers by means of NFP, characterizing their features and limitations with respect to user preferences.

2.1 Current Approaches

An early approach on modeling QoS in the context of SWS discovery are found in [10]. In this work, Ran presents a UDDI extension and a catalog of QoS parameters that can be included in UDDI descriptions. Discovery is performed using queries with functional requirements, as well as conditions on QoS. However, the actual discovery algorithm is not defined, and queries that use NFP are not shown, so their expressiveness is unknown. Additionally, UDDI only supports a keyword based search, so no form of inference or flexible match can be performed [15]. Apart from that, user preferences can not be expressed in the query and the resultant services are not ranked, so the user have to perform different queries in order to find the best suited service.

Although their proposal is not semantically defined, Liu *et al.* present a QoS computation model including a selection algorithm [5], which is adapted in other approaches [9,16]. They propose an extensible QoS model that comprises both generic and domain specific criteria. Selection is performed using an algorithm based on matrices normalization, where services are ranked in terms of their QoS matrix description and a vector of relative weights between QoS parameters, which express user preferences.

Pathak *et al.* also model mappings between ontologies in [9]. They propose to use domain specific ontologies to define NFP among offers and demands. In their work, selection is done using matching degrees at a first stage. Then, QoS parameters values are collected in a quality matrix, which is used to calculate a fixed, weighted utility function for each offer. Finally, offers whose utility function is above a given threshold, are ranked by one QoS parameter to obtain the optimal offer.

Wang *et al.* provide an extension to WSMO ontology [11] to handle QoS parameters [16]. They define a QoS selection model and an algorithm based on a quality matrix that contains values of QoS parameters. The user preferences are described in terms of tendencies, i.e. a demand may prefer parameters to be as small as possible, as large as possible, or around a given value.

Maximilien and Singh present a framework and a QoS ontology for dynamic selection in [8]. They use an agent-based approach where NFP are modeled via a three-layer ontology: an upper ontology which defines basic concepts associated with a quality parameter, a middle ontology which defines the most frequent QoS parameters and metrics, and a user-defined lower ontology that depends on the domain of the service. Although it constitutes a well-defined framework to semantically describe NFP and it is referenced by many authors [2,4,9], it lacks of a way to semantically describe user preferences.

An extension to DAML-S[1] to include QoS profiles is proposed in [18] by Zhou *et al.* This proposal only allows order conditions between QoS parameters, so it performs discovery and selection using DLs. The QoS ontology is simple and can be easily linked to the DAML-S service profile. However, its selection algorithm uses matching degrees to order the resulting set of services, so the user preferences can not be expressed, as they are inherent to that selection algorithm.

Another DAML-based proposal is also presented in [14], where S. Bilgin and Singh provide a DAML-based query language, instead of just extending OWL-S. Using this Semantic Web Services Query and Manipulation Language, they advertise QoS attributes and perform the selection. The main drawbacks of this approach are the same as in [10], with limitations on the expressiveness of queries, due to the use of DAML as its foundation. Thus, user preferences can not be expressed in those queries, and are inherent to their selection algorithm, as in [18].

Dobson *et al.* presents QoSOnt in [2], which is an ontology that extends OWL-S to describe QoS attributes and metrics. However, they do not explicitly explain how to perform selection, and their proposal suffers from OWL limitations, so

[1] DAML-S is an early version of OWL-S [6]

they have to use an ad-hoc XML language to allow custom data ranges. User preferences are modeled using the acceptability direction, that is the preferred tendency of metric values (e.g. the higher the best).

On the other hand, Zeng *et al.* show a basic QoS model to Web services composition in [17], although it can be applied to discovery and selection. They propose an algorithm based on utility functions, which are already defined for all the contemplated QoS parameters. The optimization is implemented using Integer Programming, providing weights to the different QoS parameters involved. The main drawbacks of this proposal are that it do not take semantics into account and that the utility functions are fixed, so the user can define its preferences only by means of weights.

Ruiz-Cortés *et al.* describe a QoS-aware discovery using Constraint Programming, where optimization is modeled as a Constraint Satisfaction Optimization Problem that minimize a weighted composition of utility functions, which are defined by the client using QoS parameters from a catalog [12]. As in [17], this proposal does not provide semantics, but user preferences, described by utility functions, can be defined by the user with high expressiveness.

An extension to [7] is presented in [4] by Kritikos and Plexousakis. They propose an ontology similar to the proposed by Maximilien and Singh [8], mixing offers and demands within an OWL-S description. Moreover, they present a matching algorithm to infer equivalences between different named QoS parameters that are semantically equivalent, although it is generally undecidable. Concerning discovery and selection, they use CSPs to perform the matchmaking of compatible offers, and then select the best service by means of a weighted composition of utility functions, which balance the worst and best scenarios to compute the utility value. However, these user preferences are not semantically defined in their QoS ontology.

2.2 Analysis

We show an analysis of the features of the different approaches introduced in the section before in Table 1. In this table, ordered by the order of exposition, we analyze if the given proposal semantically defines NFP, how it express user preferences, and the selection algorithm used.

We obtain several conclusions from this comparison. Firstly, there are a few proposals that uses utility functions to express user preferences [4,12,17], although only [12] allows the user to define complex utility functions. These three proposals use optimization techniques, as Integer Programming or Constraint Programming, to select the best offers. Therefore, utility functions become the natural choice to define highly expressive user preferences.

Secondly, there are many proposals that provide a semantic framework to define NFP [2,4,8,9,14,16,18], although [8] do not handle user preferences in their ontology and [14,18] have a fixed definition of user preferences, inherent to their selection algorithm. [4] is the most expressive when defining user preferences, followed by [2,9,16], that limit their preferences to weights and parameter ten-

Proposal	Semantic Defs.	User Preferences	Selection
Ran [10]	No	Not defined	Not defined
Liu et al. [5]	No	Weights	Quality matrix
Pathak et al. [9]	Yes	Weights	Quality matrix
Wang et al. [16]	Yes	Tendencies	Quality matrix
Maximilien & Singh [8]	Yes	External	Matching degree
Zhou et al. [18]	Yes	Fixed	Matching degree
S. Bilgin & Singh [14]	Yes	Fixed	Query lang.
Dobson et al. [2]	Yes	Tendencies	Not defined
Zeng et al. [17]	No	Utility and weights	Integer Prog.
Ruiz-Cortés et al. [12]	No	Utility and weights	Constraint Prog.
Kritikos & Plexousakis [4]	Yes	Utility and weights	Constraint Prog.

Table 1. Comparison between discussed proposals

dencies. According to all those proposals, it is clear that NFP have to be defined semantically.

Finally, we conclude that none of the above proposals semantically define user preferences, although in [2,16] the authors include in their ontology extension the tendency of QoS parameters. What is more, most of the proposals that perform selection tasks in terms of user preferences describe them using ad-hoc, non-semantic descriptions completely decoupled with the ones used to describe service functionality, causing a semantic gap between functional descriptions and user preferences.

The motivation of our work is precisely to tackle the previous problems. Most recent proposals use utility functions to express user preferences, and there are many NFP ontologies which our proposal can be integrated with. This proposal comes from mixing the expressiveness of utility functions and weights proposed by Ruiz-Cortés *et al.*, the semantic definition of NFP from Maximilien and Singh or Kritikos and Plexousakis, and an extension to give semantics to utility functions. Thus, we take full advantage of Semantic Web approaches on selecting the best services, while allowing to define user preferences using the most expressive solution, i.e. utility functions. Furthermore, we put functional, non-functional, and user preferences at the same semantic level, by means of using extensions to current SWS ontologies.

3 Our Proposal

Utility functions are the most expressive approach presented to describe user preferences that are used when selecting the best offers among a set. Although there are proposals that semantically describe QoS parameters and NFP, no one contemplates the conceptualization of utility functions. In this Section, we firstly give an overview of utility functions. Then, an ontology of user preferences is proposed to be used in the context of discovery and selection of SWS, showing a concrete example.

3.1 Utility Functions

An utility function is a normalized function (ranging over $[0, 1]$) whose domain is a given QoS parameter, that gives information about which values of that QoS parameter are preferred by the user. Fig. 1 shows an example of a utility function for the mean time to failure ($MTTF$) parameter. This function is a piecewise linear function that defines a minimum utility (0) when $MTTF$ is below 60 minutes, and a maximum utility (1) when $MTTF$ is above 120 minutes. Between these two limits, the function grows linearly.

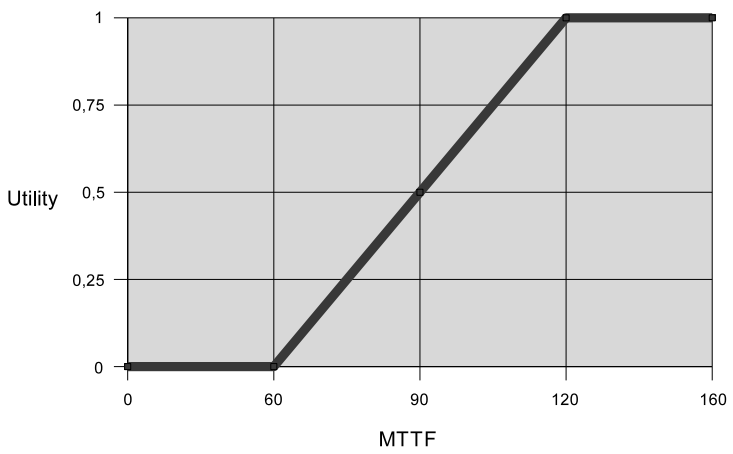


Fig. 1. Utility example for Mean Time To Failure.

When selecting the best offers, a composition of different utility functions (one for each QoS parameter involved in NFPs) is often used to compute a global utility value, which serves to sort the service offers. In this composition, each utility function has an associated weight in the global function, so the user can specify how important is each QoS parameter when selecting offers. The general form of this weighted composition of utility functions $\mathcal{U}$ is as follows [12]:

$$\mathcal{U}(p_1, \ldots, p_n) = \sum_{i=1}^{n} k_i \, U_i(p_i) \quad , k_i \in [0, 1] \sum_{i=1}^{n} k_i = 1 \tag{1}$$

where each p_i denotes a QoS parameter, each k_i its associated weight ranging over $[0, 1]$, and each U_i its associated utility function also ranging over $[0, 1]$ with the semantics previously defined.

3.2 Giving Semantics to Utility Functions

In order to provide semantic interoperability between utility functions defined on differently named QoS parameters, we propose to model these functions,

or more generally, user preferences, as an ontology. This ontology has to be instantiated by each user preference describing utility functions, so equivalences between QoS parameters can be inferred. Furthermore, this conceptualization allows the user to describe the whole service, including functional descriptions, at the same semantic level, without coupling user preferences descriptions with the selection algorithm.

Our proposed model is shown in Fig. 2. The main concept (or *class*) is UserPreference, which references a Quality concept via the *hasReference* object property. This Quality concept is analogous to the defined in [8], and represents the QoS parameter which is used in the definition of the corresponding user preference. Furthermore, the UserPreference concept has a key datatype property, *hasDefinition*, which links the more generic preference concept with the utility function that defines it. Note that Quality class is the link to QoS parameters used in the semantic definition of NFP. This definition can be performed using the ontology from Kritikos and Plexousakis [4] or from Maximilien and Singh [8], for instance.

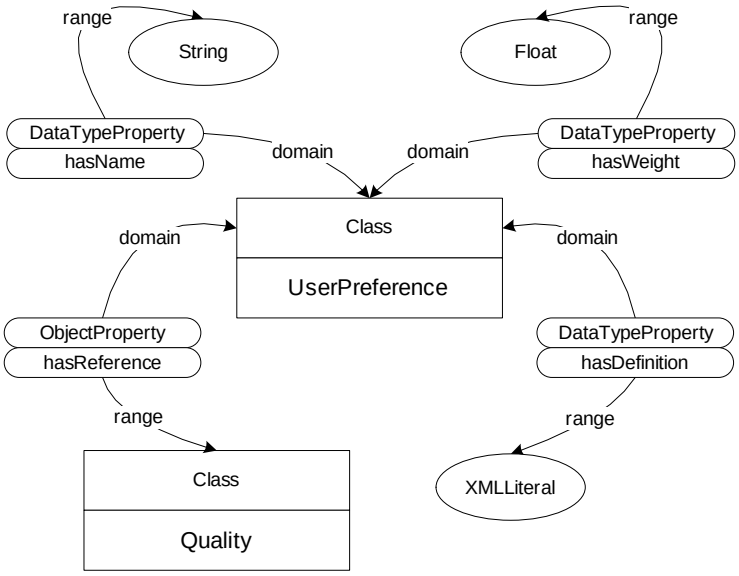


Fig. 2. Proposed ontology to model user preferences.

The utility function is initially modeled as a property that contains an XML expression that describe the definition of each function in terms of OpenMath standard [1], as used in [4,13], allowing the evaluation of the function with a proper compiler or a mathematical tool, such as Mathematica.

Finally, our main concept UserPreference has two datatype properties: *hasName* and *hasWeight*. The former is used as an identifier of a given instance. The

latter is a real number which corresponds to the relative weight associated with the corresponding QoS parameter, used to compute the global utility function of an offer.

Fig. 3 shows an instance of our proposed ontology, in the case of an user preference about $MTTF$, with an associated weight of 1. Thus, the instance MTTFUserPreference references an instance UserMTTF of MTTF class, that is a subclass of Quality class from [8]. Moreover, the concrete utility function is specified as an OpenMath object that represent the one showed in Fig. 1, using XML.

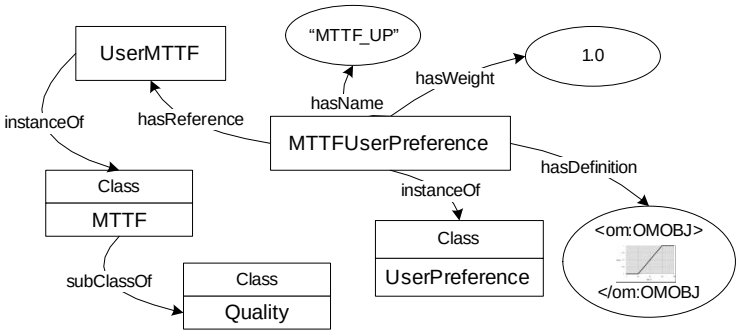


Fig. 3. Instance of our proposed ontology.

The link with the rest of the semantic description of a service, including both functional and non-functional properties, is the QoS parameter $MTTF$, i.e UserMTTF instance in our example. In this case, the engine that perform the selection only has to be aware of the part showed in Fig. 3 and the corresponding NFPs that involve the $MTTF$ parameter, but generally, there are more parameters and user preferences involved in the selection process.

4 Conclusions

In this work, we provide a semantic framework to define user preferences on semantically defined QoS parameters, provided that it is used in conjunction with another proposal that semantically defines NFP, like [4,8]. Thus, all facets of SWS description (functional, non-functional, and user preferences) are described at the same semantic level, so discovery and selection tasks are completely done within a single semantic framework, allowing interoperability between different service definitions.

Furthermore, our proposal uses a very expressive solution to define user preferences, i.e. utility functions and weights, as in [4,12,17]. This formalism becomes more generic and powerful that ones used in other approaches. Additionally, we propose the use of a hybrid discovery engine to perform the different tasks in

discovery and selection using the best suited technique in each case. Thus, we optimize these tasks without compromising expressiveness.

In conclusion, our proposal allows a semantic definition of the whole discovery and selection process, using a hybrid approach without losing expressiveness. These facts allow to decouple the definition of user preferences from the concrete selection algorithm used.

Acknowledgments. The authors would like to thank the reviewers of the *NFPSLA-SOC'07 Workshop*, whose comments and suggestions improved the presentation substantially.

References

1. S. Buswell, O. Caprotti, D. P. Carlisle, M. C. Dewar, M. Gaëtano, and M. Kohlhase. The OpenMath standard. Technical Report Version 2.0, The OpenMath Society, 2004.
2. G. Dobson, R. Lock, and I. Sommerville. Qosont: a qos ontology for service-centric systems. In *EUROMICRO-SEAA*, pages 80–87. IEEE Computer Society, 2005.
3. J. M. García, D. Ruiz, A. Ruiz-Cortés, O. Martín-Díaz, and M. Resinas. An hybrid, QoS-aware discovery of semantic web services using constraint programming. In B. Krämer, K.-J. Lin, and P. Narasimhan, editors, *ICSOC 2007*, volume 4749 of *LNCS*, pages 69–80. Springer, 2007.
4. K. Kritikos and D. Plexousakis. Semantic QoS metric matching. In *ECOWS 2006*, pages 265–274. IEEE Computer Society, 2006.
5. Y. Liu, A. H. H. Ngu, and L. Zeng. Qos computation and policing in dynamic web service selection. In *WWW (Alternate Track Papers & Posters)*, pages 66–73, 2004.
6. D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. Mcdermott, et al. OWL-S: Semantic Markup for Web Services. Technical Report 1.1, DAML, 2004.
7. O. Martín-Díaz, A. Ruiz-Cortés, D. Benavides, A. Durán, and M. Toro. A quality-aware approach to web services procurement. In *4th. VLDB Workshop on Technologies for E-services TES'03*, pages 42–53, 2003.
8. E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *Internet Computing, IEEE*, 8(5):84–93, 2004.
9. J. Pathak, N. Koul, D. Caragea, and V. G. Honavar. A framework for semantic web services discovery. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 45–50, New York, NY, USA, 2005. ACM Press.
10. S. Ran. A model for web services discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.
11. D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology (WSMO). Technical Report D2 v1.3 Final Draft, WSMO, 2006.
12. A. Ruiz-Cortés, O. Martín-Díaz, A. Durán-Toro, and M. Toro. Improving the automatic procurement of web services using constraint programming. *Int. J. Cooperative Inf. Syst*, 14(4):439–468, 2005.
13. A. Sánchez-Macián, D. López, J. E. López de Vergara, and E. Pastor. A framework for the automatic calculation of quality of experience in telematic services. In *13th HP-OVUA Workshop*, Sophia Antipolis, France, May 2006.

14. A. Soydan Bilgin and M.P. Singh. A DAML-based repository for QoS-aware semantic Web service selection. In *IEEE International Conference on Web Services*, pages 368–375, 2004.
15. K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *J. Web Sem.*, 1(1):27–46, 2003.
16. X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A QoS-Aware Selection Model for Semantic Web Services. In A. Dan and W. Lamersdorf, editors, *ICSOC 2006*, volume 4294 of *LNCS*, pages 390–401. Springer, 2006.
17. L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.
18. C. Zhou, L. Chia, and B. Lee. DAML-QoS ontology for web services. In *IEEE International Conference on Web Services*, pages 472–479, 2004.

Semantic Discovery and Selection: A QoS-Aware, Hybrid Model

(Submitted to SWWS'08)

José María García*, David Ruiz and Antonio Ruiz-Cortés
Universidad de Sevilla
Dept. Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Av. Reina Mercedes s/n, 41012 Sevilla, Spain
Telephone: (+34) 9545 59814. Fax: (+34) 9545 57139
*Contact Author. Email: josemgarcia@us.es

***Abstract*—Most Semantic Web Services discovery approaches are based on Description Logics, allowing a limited expressiveness when describing Quality-of-Service preferences. Furthermore, DLs is not suited to perform selection tasks, because these are modeled as optimization problems. In this work, we present a hybrid discovery and selection model for Semantic Web Services that takes care of QoS preferences. Our approach splits the whole process into two stages, using the most suited engine in each one, depending on its search nature. In order to perform QoS-aware discovery and selection, user preferences have to be semantically described as the rest of the service description. Our model provides an ontology for user preferences, so instances can be transformed into optimization problems that can be solved by using the most suited engine.**

***Index Terms*—Service Discovery, Quality-of-Service, Semantic Web Services, Ontology Languages, QoS-Aware Selection.**

I. Introduction

Most approaches on automatic discovery of Semantic Web Services (SWS) use Description Logics (DLs) reasoners to perform the matching [1]–[6]. These approaches have limitations regarding with the expressiveness of searches, especially when there are Quality-of-Service (QoS) conditions integrated within user preferences. For instance, a condition like "find a service which $availability \geq 0.9$, where $availability = MTTF/(MTTF + MTTR)$"[1] can not be expressed in DLs. However, there are some proposals that extend DLs with concrete domains [7], though they still have limitations on expressing complex conditions [8], [9]. Furthermore, selection tasks lead to optimization problems, so DLs are not suited in that stage of the whole discovery process.

Optimization problems can be handled by solvers based on different techniques, like Linear Programming (LP), Constraint Programming (CP), or Dynamic Programming (DP), among others. Thus, it is possible to split discovery and selection tasks in terms of functional and QoS preferences, so the former task can be performed by DLs reasoners, while the latter can be performed by solvers, taking a hybrid approach.

[1] $MTTF$ stands for "Mean Time To Failure", while $MTTR$ stands for "Mean Time To Repair". Both of them are QoS parameters often used to define service availability.

Our proposal presents a hybrid architecture to discover SWS. Thus, our solution splits that process into two stages: (1) functional discovery, which is usually performed by DLs reasoners using functional preferences; and (2) QoS-driven selection, where a solver obtain the best service in terms of QoS preferences that define an optimality criterion. Both *user* preferences have to be semantically described at the same level, so different, but possibly equivalent, service descriptions can be matched.

This model does not restrict the concrete technique to be used in any of its stages. Our proposal provides a user preference ontology that can be linked to any current SWS framework. Moreover, it uses a very expressive solution to define these preferences, i.e. utility functions and weights. Finally, our hybrid architecture is extensible, so it is possible to add more stages to the process, provided that the descriptions needed by that stage are described at the same semantic level than the rest of the service description.

The rest of the paper is structured as follows. In Sec. II we analyze current approaches on discovering and selecting SWS. Then, in Sec. III we present our hybrid discovery and selection model, explaining the proposed architecture, an ontology of user preferences using utility functions, and how to use that ontology to perform QoS-aware semantic selection. Finally, in Sec. IV we sum up our contributions, and discuss our conclusions.

II. Related Work

In this Section, we discuss related work on discovery and selection of SWS, describing the different approaches and analyzing their suitability to handle QoS-aware user preferences. Firstly, we review discovery-related proposals, which use DLs to match services with functional preferences, and then we present different approaches on selecting services by means of QoS-aware user preferences.

Concerning these preferences, in the following, service descriptions are considered as their *provider preferences* (what a provider offers and possibly its requirements to the user), and user requirements are referenced as *user preferences*. Nevertheless, each preference can be broken in two main parts:

(1) functional preferences, that refer to what a service has to do; and (2) QoS preferences, that are used to rank services in terms of one or more QoS parameters.

A. Discovering SWS

In the context of DAML-S (the precursor of OWL-S [10]), Sycara *et al.* show how semantic information allows automatic discovery, invocation and composition of Web Services [6]. They provide an early integration of semantic information in a UDDI registry, and propose a matchmaking architecture. It is based on a previous work by Paolucci *et al.*, where they define the matching engine used [11]. This engine matches a demand and an offer when this offer describes a service which is "sufficiently similar" to the demanded service, i.e. the offered service provides the functionality demanded in some degree. The problem is how to define that degree of similarity, and the concrete algorithm to match both service descriptions. They update their work to OWL-S in [12].

Furthermore, there are proposals that perform the matchmaking of SWS using DLs [1]–[3]. Particularly, González-Castillo *et al.* provide an actual matchmaking algorithm using the subsumption operator between DLs concepts describing demands and offers [1]. They use existing DLs reasoners, as RACER [13] or FaCT [14], to perform the matchmaking. On the other hand, Lutz and Sattler [3] do not provide an algorithm, but give the foundations to implement it using subsumption, like Li and Horrocks [2], who also give hints to implement a prototype using RACER.

These three works define different matching degrees as in [6], from exactly equivalents to disjoint, so they perform a selection. All of them perform this matching by comparing inputs and outputs. However, Benatallah *et al.* propose to use the degree of matching to select the best offer in [15], but it results to be a NP-hard problem, as in any optimization problem [16].

On the other hand, Benbernou and Hacid realise that some kinds of constraints are necessary to discover SWS, including QoS related ones, so they formally discuss the convenience of incorporating constraints in SWS discovery [17]. However, instead of using any existing SWS description framework, their proposal uses an *ad-hoc Services Description Language*, in order to be able to define complex constraints. In addition, the resolution algorithm uses constraint propagation and rewriting, but performed by a subsumption algorithm, instead of a CSP solver.

B. Selecting SWS

An early approach on modeling QoS in the context of SWS discovery are found in [18]. In this work, Ran presents a UDDI extension and a catalog of QoS parameters that can be included in UDDI descriptions. Discovery is performed using queries with functional requirements, as well as conditions on QoS. However, the actual discovery algorithm is not defined, and queries that use QoS parameters are not shown, so their expressiveness is unknown. Additionally, UDDI only supports a keyword based search, so no form of inference or flexible match can be performed [6]. Apart from that, the resultant services are not ranked, so the user have to perform different, successive queries, filtering the result set in order to find the best suited service.

Although their proposal is not semantically defined, Liu *et al.* present a QoS computation model including a selection algorithm [19], which is adopted in other approaches [20], [21]. They propose an extensible QoS model that comprises both generic and domain specific criteria. Selection is performed using an algorithm based on matrices normalization, where services are ranked in terms of their QoS matrix description and a vector of relative weights between QoS parameters, which express user preferences.

Pathak *et al.* also model mappings between ontologies in [20]. They propose to use domain specific ontologies to define QoS preferences among users and providers. In their work, selection is done using matching degrees at a first stage. Then, QoS parameters values are collected in a quality matrix, which is used to calculate a fixed, weighted utility function for each offer. Finally, offers whose utility function is above a given threshold, are ranked by one QoS parameter to obtain the optimal offer.

Wang *et al.* provide an extension to WSMO ontology [22] to handle QoS parameters [21]. They define a QoS selection model and an algorithm based on a quality matrix that contains values of QoS parameters. The user preferences are described in terms of tendencies, i.e. a demand may prefer parameters to be as small as possible, as large as possible, or around a given value. Thus, in conjunction with weights, they rank services to select the best one within a given set.

Maximilien and Singh present a framework and a QoS ontology for dynamic selection in [23]. They use an agent-based approach where QoS are modeled via a three-layer ontology: an upper ontology which defines basic concepts associated with a quality parameter, a middle ontology which defines the most frequent QoS parameters and metrics, and a user-defined lower ontology that depends on the domain of the service. Although it constitutes a well-defined framework to semantically describe QoS and it is referenced by many authors [20], [24], [25], it is not aimed at semantically describing user preferences.

An extension to DAML-S to include QoS profiles is proposed in [26] by Zhou *et al.* This proposal only allows order conditions between QoS parameters, so it performs discovery and selection using DLs. The QoS ontology is simple and can be easily linked to the DAML-S service profile. However, its selection algorithm uses matching degrees to rank the resulting set of services, so user preferences can only be expressed as ordering relations, which are inherent to that selection algorithm.

Another DAML-based proposal is also presented in [27], where S. Bilgin and Singh provide a DAML-based query language, instead of just extending OWL-S. Using this *Semantic Web Services Query and Manipulation Language*, they advertise QoS attributes and perform the selection. The main drawbacks of this approach are the same as in [18], with

limitations on the expressiveness of queries, due to the use of DAML as its foundation. Thus, user preferences can not be expressed in those queries, and are inherent to their selection algorithm, as in [26].

Dobson *et al.* presents QoSOnt in [24], which is an ontology that extends OWL-S to describe QoS attributes and metrics. However, they do not explicitly explain how to perform selection, and their proposal suffers from OWL limitations, so they have to use an ad-hoc XML language to allow custom data ranges. User preferences are modeled using the preferred tendency of metric values (e.g. the higher the best).

On the other hand, Zeng *et al.* show a basic QoS model to Web services composition in [28], although it can be applied to discovery and selection. They propose an algorithm based on utility functions, which are already defined for all the contemplated QoS parameters. The optimization is implemented using Integer Programming, providing weights to the different QoS parameters involved. The main drawbacks of this proposal are that it do not take semantics into account and that the utility functions are fixed, so the user can define its preferences only by means of weights.

Ruiz-Cortés *et al.* describe a QoS-aware discovery using Constraint Programming, where optimization is modeled as a Constraint Satisfaction Optimization Problem that minimize a weighted composition of utility functions, which are defined by the client using QoS parameters from a catalog [29]. As in [28], this proposal does not provide semantics, but user preferences, described by utility functions, can be defined by the user with high expressiveness.

An extension to [30] is presented in [25] by Kritikos and Plexousakis. They propose an ontology similar to the proposed by Maximilien and Singh [23], mixing offers and demands within an OWL-S description. Moreover, they present a matching algorithm to infer equivalences between different named QoS parameters that are semantically equivalent, although it is generally undecidable. Concerning discovery and selection, they use CSPs to perform the matchmaking of compatible offers, and then select the best service by means of a weighted composition of utility functions, which balance the worst and best scenarios to compute the utility value. However, these user preferences are not semantically defined in their QoS ontology.

C. Motivation

Several conclusions can be obtained from the analysis of the related work. The main ones are the following, which conform the motivation of this work.

1) Discovery proposals, all of them based on DLs, generally use matching degrees to select the best service [1]–[3], [6]. However, they do not support QoS preferences, so they can not perform any optimization based on preferences.
2) There are a few proposals that uses utility functions to express user preferences [25], [28], [29], although only [29] allows the user to define complex utility functions. These three proposals use optimization techniques, as Integer Programming or Constraint Programming, to select the best offers. Therefore, utility functions become the natural choice to define highly expressive user preferences.
3) There are many proposals that provide a semantic framework to define QoS [20], [21], [23]–[27], although [23] do not handle user preferences in their ontology and [26], [27] have a fixed definition of user preferences, inherent to their selection algorithm. [25] is the most expressive when defining user preferences, followed by [20], [21], [24], that limit their preferences to weights and parameter tendencies. According to all those proposals, it is clear that QoS have to be defined semantically.
4) None of the analyzed proposals semantically define user preferences, although in [21], [24] the authors include in their ontology extension the tendency of QoS parameters. What is more, most of the proposals that perform selection tasks in terms of user preferences describe them using ad-hoc, non-semantic descriptions completely decoupled with the ones used to describe service functionality, causing a semantic gap between functional descriptions and user preferences.

These conclusions motivate this paper, because it is necessary to tackle the previous problems. Most recent proposals use utility functions to express user preferences, although they are not semantically defined, and there are many QoS ontologies which any solution should be able to linked with. Our discovery and selection model takes into account all these problems, providing a hybrid and QoS-aware solution.

III. A QoS-Aware, Hybrid Model

The addition of QoS preferences to SWS descriptions, turns most approaches on selecting SWS insufficient, because they mainly use DLs, which are usually limited to logical and relational expressions when describing QoS conditions. Discovery and selection have to become independent, so the former can be performed by DLs reasoners, but the latter has to use an optimization technique, although functional and QoS-aware preferences have to be described at the same level. Thus, a hybrid solution arise as the most adequate option. In the following, this solution is presented, along with an ontology of user preferences that allows to describe both discovery and selection at the same semantic level. Although in [31] we present a generic n-stages hybrid discovery, in this paper we specify that early approach by using two concrete stages, as well as providing an user preference ontology that is used in the selection process.

Our selection proposal comes from mixing the expressiveness of utility functions and weights proposed by Ruiz-Cortés *et al.* [29], the semantic definition of QoS from Maximilien and Singh [23] or Kritikos and Plexousakis [25], and an extension to give semantics to utility functions. Furthermore, this QoS preferences ontology can be linked with functional preferences, so the whole discovery and selection process are performed within a hybrid architecture, where DLs is used

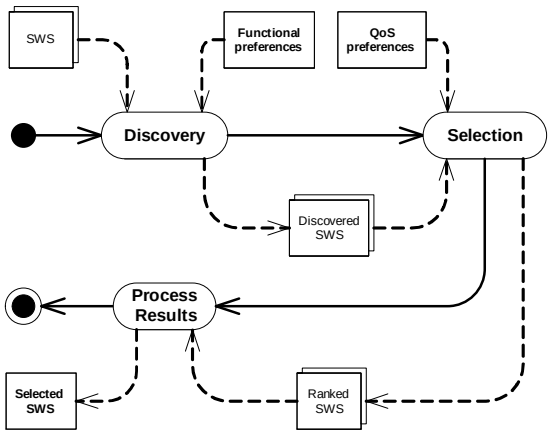


Figure 1: Activity diagram of the hybrid process.

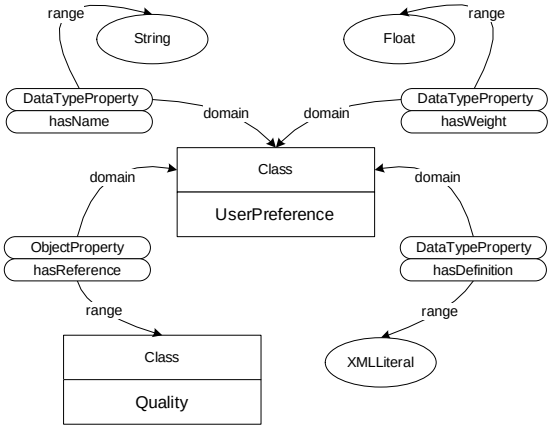


Figure 2: Proposed ontology to model user preferences.

to discover services as in the proposals from Sec. II-A, and selection is treated as an optimization problem.

A. Hybrid Semantic Discovery

Fig. 1 shows the activity diagram of a hybrid discovery process performed by two different engines. This process begins with the discovery stage, where a set of SWS descriptions are matched with the user functional preferences. Thus, at this point, only compatible services, in terms of functionality, are returned to the next stage. That matching can be performed using available DLs reasoners that can take SWS descriptions and return those SWS which match with the functional preferences, that can be expressed as a service profile in OWL-S or as service capabilities in WSMO, for instance.

Then, the *discovered SWS* are further processed, ranking them in order to select the best service. In this case, selection process uses QoS-aware user preferences to rank services. These preferences are also semantically described, and linked with functional preferences to conform the whole user preference. However, QoS preferences are not semantically described in current proposals (cf. Sec. II-B), so it is necessary to provide a conceptualization of QoS-related user preferences. Furthermore, in order to perform the selection stage, an optimization technique has to be used. Thus, discovered SWS descriptions and QoS preferences are transformed into an optimization problem that can be performed by different techniques, such as CP, LP or DP solvers. This stage is further explained in Sec. III-B.

Finally, the list of *ranked SWS* are processed at the second stage, where the best service in terms of user preferences are returned, so it can be invoked or composed with others. This service is referenced as *selected SWS* in Fig. 1.

This hybrid discovery architecture has many advantages. It is loosely coupled, due to the possibility to use any discovery and selection engines. Also, user preferences expressiveness are not constrained to a specific selection technique, provided that a transformation from our conceptualization is available. Moreover, our proposed architecture can be applied to any existing SWS framework and corresponding repositories, taking benefit of the wide range of tools already implemented.

B. QoS-Aware Semantic Selection

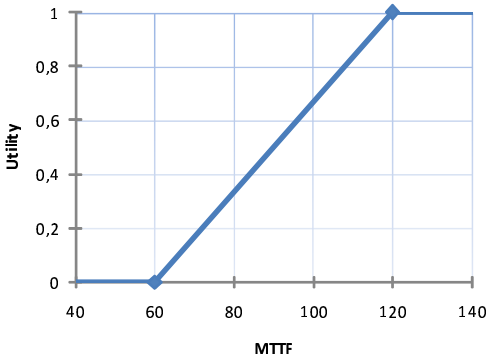
In order to decouple QoS-aware preferences descriptions with the concrete selection algorithm used, we propose to model these preferences as an upper ontology. This conceptualization allows the user to describe the whole service, including functional descriptions from a SWS framework, at the same semantic level. Furthermore, it provides semantic interoperability between user preferences based on differently named QoS parameters, because equivalences between those QoS parameters can be inferred.

Our proposed model is shown in Fig. 2. The main concept (or *class*) is UserPreference, which references a Quality concept via the *hasReference* object property. This Quality concept is analogous to the defined in [23], and represents the QoS parameter which is used in the definition of the corresponding user preference. Furthermore, the UserPreference concept has a key datatype property, *hasDefinition*, which links the more generic preference concept with the utility function that defines it. Note that Quality class is the link to QoS parameters used in the semantic definition of QoS. This definition can be performed using the ontology from Kritikos and Plexousakis [25] or from Maximilien and Singh [23], for instance.

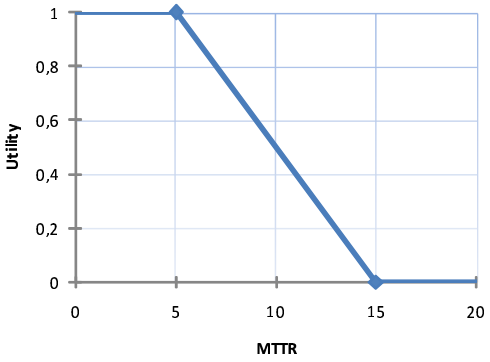
The utility function is initially modeled as a property that contains an XML expression that describe the definition of each function in terms of OpenMath standard [32], as used in [25], allowing the evaluation of the function with a proper compiler or a mathematical tool, such as Mathematica.

Finally, our main concept `UserPreference` has two datatype properties: *`hasName`* and *`hasWeight`*. The former is used as an identifier of a given instance. The latter is a real number which corresponds to the relative weight associated with the corresponding QoS parameter, used to compute the global utility function of an offer.

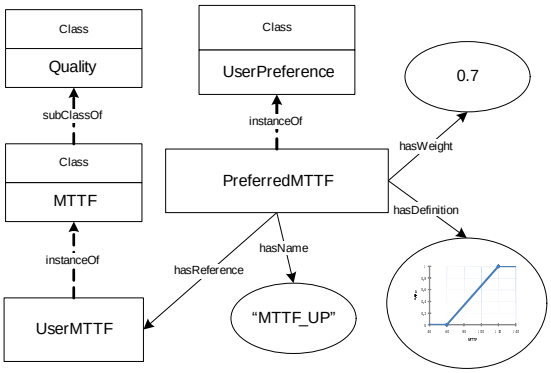
Fig. 3 shows two instances of our proposed ontology, in the case of a composed user preference about $MTTF$, with



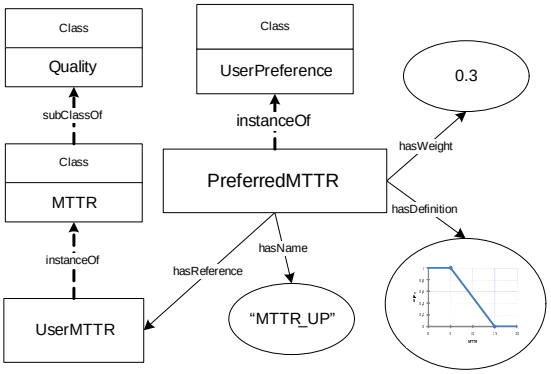
(a) $MTTF$ utility function.



(b) $MTTR$ utility function.



(c) User preference instance about $MTTF$.



(d) User preference instance about $MTTR$.

Figure 3: Conceptualization of $MTTF$ and $MTTR$ user preferences.

an associated weight of 0.7 (Fig. 3c), and $MTTR$ with its corresponding weight of 0.3 (Fig. 3d). On the first hand, the instance `PreferredMTTF` references an instance `UserMTTF` of `MTTF` class, that is a subclass of `Quality` class from [23]. On the other hand, `PreferredMTTR` references an instance `UserMTTR` of `MTTR` class. Moreover, concrete utility functions are specified as OpenMath objects that represent the showed in Fig. 3a and Fig 3b, respectively, using XML.

Selection process has to take all the instances from the proposed ontology to compose a global user preference. Thus, in the example from Fig. 3, the concrete optimization technique used has to take both user preferences into account to perform the actual selection, according to the relative weights associated with each QoS preference.

IV. Conclusions

In this work, we show that using a unique engine to discover SWS is not appropriate, due to each engine is usually designed for a concrete kind of search. For instance, DLs reasoners are well suited when discovering SWS in terms of concepts and relations, but they can not handle complex numerical QoS preferences. Although there are extensions to allow concrete domains in DLs, reasoners have to implement them, and they may bring undecidability results.

We present a hybrid solution that consists in a two-stages discovery process, where each stage is performed using the most appropriate technique. Furthermore, we provide a semantic framework to define user preferences on semantically defined QoS parameters, provided that it is used in conjunction with another proposal that semantically defines those QoS parameters, like [23], [25]. Thus, all facets of SWS description (functional, non-functional, and user preferences) are described at the same semantic level, so discovery and selection tasks are completely done within a single semantic framework, allowing interoperability between different service definitions.

In addition, our proposed architecture is extensible and loosely coupled, allowing to define complex QoS conditions, and to use utility functions based on QoS parameters to obtain the best service. This architecture does not impose any restriction on the SWS framework and repository to use, allowing its materialization as a discovery component for current SWS implementations. Moreover, it is independent on the concrete optimization technique used in the selection stage.

Acknowledgment

This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project Web-Factories (TIN2006-00472) and by the Andalusian Government under project ISABEL (TIC-2533).

REFERENCES

[1] J. González-Castillo, D. Trastour, and C. Bartolini, "Description logics for matchmaking of services," Hewlett Packard Labs, Tech. Rep. HPL-2001-265, 2001.

[2] L. Li and I. Horrocks, "A software framework for matchmaking based on semantic web technology," in *Int. World Wide Web Conference*, 2003, pp. 331–339.

[3] C. Lutz and U. Sattler, "A proposal for describing services with DLs," in *Int. Workshop on Description Logics*, 2002.

[4] E. Motta, J. Domingue, L. Cabral, and M. Gaspari, "IRS-II: A framework and infrastructure for semantic web services," in *Int. Semantic Web Conference*, 2003, pp. 306–318.

[5] N. Srinivasan, M. Paolucci, and K. Sycara, "Semantic web service discovery in the OWL-S IDE." in *Hawaii International Conference on Systems Science*, 2006.

[6] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated discovery, interaction and composition of semantic web services." *J. Web Sem.*, vol. 1, no. 1, pp. 27–46, 2003.

[7] V. Haarslev and R. Möller, "Practical reasoning in RACER with a concrete domain for linear inequations," in *Int. Workshop on Description Logics*, 2002.

[8] F. Baader and U. Sattler, "Description logics with aggregates and concrete domains," *Information Systems*, vol. 28, no. 8, pp. 979–1004, December 2003.

[9] C. Lutz, "Description logics with concrete domains – a survey." in *Advances in Modal Logic*, 2002, pp. 265–296.

[10] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. Mcdermott *et al.*, "OWL-S: Semantic markup for web services," DAML, Tech. Rep. 1.1, 2004.

[11] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara, "Semantic matching of web services capabilities." in *Int. Semantic Web Conference*, 2002, pp. 333–347.

[12] K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan, "Dynamic discovery and coordination of agent-based semantic web services." *IEEE Internet Computing*, vol. 8, no. 3, pp. 66–73, 2004.

[13] V. Haarslev and R. Möller, "RACER system description." in *Automated Reasoning, First International Joint Conference, IJCAR 2001*, 2001, pp. 701–706.

[14] I. Horrocks, "FaCT and iFaCT." in *Int. Workshop on Description Logics*, 1999.

[15] B. Benatallah, M. Hacid, C. Rey, and F. Toumani, "Semantic reasoning for web services discovery." in *WWW Workshop on E-Services and the Semantic Web*, 2003.

[16] P. Bonatti and P. Festa, "On optimal service selection." in *14th international conference on World Wide Web*, 2005, pp. 530–538.

[17] S. Benbernou and M. Hacid, "Resolution and constraint propagation for semantic web services discovery," *Distributed and Parallel Databases*, vol. 18, no. 1, pp. 65–81, 2005.

[18] S. Ran, "A model for web services discovery with QoS," *SIGecom Exch.*, vol. 4, no. 1, pp. 1–10, 2003.

[19] Y. Liu, A. H. H. Ngu, and L. Zeng, "Qos computation and policing in dynamic web service selection," in *WWW (Alternate Track Papers & Posters)*, 2004, pp. 66–73.

[20] J. Pathak, N. Koul, D. Caragea, and V. G. Honavar, "A framework for semantic web services discovery," in *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*. New York, NY, USA: ACM Press, 2005, pp. 45–50.

[21] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma, "A QoS-aware selection model for semantic web services." in *ICSOC 2006*, ser. LNCS, A. Dan and W. Lamersdorf, Eds., vol. 4294. Springer, 2006, pp. 390–401.

[22] D. Roman, H. Lausen, and U. Keller, "Web service modeling ontology (WSMO)," WSMO, Tech. Rep. D2 v1.3 Final Draft, 2006.

[23] E. M. Maximilien and M. P. Singh, "A framework and ontology for dynamic web services selection," *Internet Computing, IEEE*, vol. 8, no. 5, pp. 84–93, 2004.

[24] G. Dobson, R. Lock, and I. Sommerville, "QoSOnt: a QoS ontology for service-centric systems," in *EUROMICRO-SEAA*. IEEE Computer Society, 2005, pp. 80–87.

[25] K. Kritikos and D. Plexousakis, "Semantic QoS metric matching," in *ECOWS 2006*. IEEE Computer Society, 2006, pp. 265–274.

[26] C. Zhou, L. Chia, and B. Lee, "DAML-QoS ontology for web services," in *IEEE International Conference on Web Services*, 2004, pp. 472–479.

[27] A. Soydan Bilgin and M. Singh, "A DAML-based repository for QoS-aware semantic web service selection," in *IEEE International Conference on Web Services*, 2004, pp. 368–375.

[28] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for web services composition," *IEEE Transactions on Software Engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[29] A. Ruiz-Cortés, O. Martín-Díaz, A. Durán-Toro, and M. Toro, "Improving the automatic procurement of web services using constraint programming," *Int. J. Cooperative Inf. Syst*, vol. 14, no. 4, pp. 439–468, 2005.

[30] O. Martín-Díaz, A. Ruiz-Cortés, D. Benavides, A. Durán, and M. Toro, "A quality-aware approach to web services procurement," in *4th. VLDB Workshop on Technologies for E-services TES'03*, 2003, pp. 42–53.

[31] J. M. García, D. Ruiz, A. Ruiz-Cortés, O. Martín-Díaz, and M. Resinas, "An hybrid, QoS-aware discovery of semantic web services using constraint programming," in *ICSOC 2007*, ser. LNCS, B. Krämer, K.-J. Lin, and P. Narasimhan, Eds., vol. 4749. Springer, 2007, pp. 69–80.

[32] S. Buswell, O. Caprotti, D. P. Carlisle, M. C. Dewar, M. Gaëtano, and M. Kohlhase, "The OpenMath standard," The OpenMath Society, Tech. Rep. Version 2.0, 2004.

Appendix B

Curriculum vitae

The Curriculum vitae of the author of this research report is enclosed in the following in Spanish. Further information can be provided at request if necessary.

DATOS PERSONALES

Apellidos: García Rodríguez
Nombre: José María
DNI: 31724461-D
Fecha de nacimiento: 10/10/1981
Teléfono: 954559814
Correo electrónico: josemgarcia@us.es

AUTODEFENSA

El doctorando comenzó su carrera profesional, tras obtener su titulación de Ingeniero en Informática con Premio Extraordinario, en Sadiel, S.A. Desde octubre de 2004 hasta octubre de 2006 estuvo ejerciendo funciones de programador para distintos proyectos realizados para la Junta de Andalucía, incluyendo desarrollos Web con tecnología Java e integradas en la plataforma *w@ndA*. En el curso 05/06 comenzó sus estudios de doctorado en el programa de Tecnología e Ingeniería del Software. En junio de 2006 comenzó a trabajar para el proyecto AgilWeb (IP: Miguel Toro) en tareas de apoyo a la investigación, para finalmente en octubre de 2006 comenzar a trabajar en el Departamento de Lenguajes y Sistemas Informáticos como profesor.

Actualmente se encuentra participando como investigador en los proyectos WebFactories (CICYT, IP: Antonio Ruiz) e ISABEL (Junta de Andalucía, IP: Antonio Ruiz), en tareas relacionadas con el aprovisionamiento de servicios Web semánticos. En este contexto, el doctorando cuenta con dos aportaciones, una en ICSOC'07 (Congreso A-Core) y otra en el taller NFPSLA-SOC'07. Asimismo, se encuentra en proceso de revisión otra aportación enviada al SWWS'08 (Congreso incluido en el Top-80 del ranking CSCR)

Finalmente, la experiencia docente cosechada en estos dos años como profesor universitario se ha centrado en asignaturas troncales en las tres titulaciones impartidas en la ETSII, completando 21 créditos en el curso 06/07 en las asignaturas de Introducción a la Programación I y II; y 24 créditos en el curso 07/08 en las asignaturas de Ingeniería del Software I y III, y en Ingeniería del Software de Gestión I y II.

FORMACIÓN ACADÉMICA

Titulación Superior: Ingeniero en Informática
Centro: E.T.S de Ingeniería Informática de Sevilla
Fecha: 07/04/2005
Calificación: Premio Extraordinario (2.5)

EXPERIENCIA LABORAL

07/2006-10/2006 (3 meses): Personal Investigador Titulado Superior de la Universidad de Sevilla (Proyecto AgilWeb).
10/2004-10/2006 (24 meses): Programador en Sadiel, S.A.

IDIOMAS

Inglés: Diplomado por el Instituto de Idiomas (Universidad de Sevilla)
Alemán: Primer Curso del Instituto de Idiomas (Universidad de Sevilla)

PARTICIPACIÓN EN PROYECTOS DE I+D FINANCIADOS EN CONVOCATORIAS PÚBLICAS

Título del proyecto: ISABEL. Ingeniería de Sistemas Abiertos Basada en LínEas de productos
Entidad financiadora: Consejería de Innovación Ciencia y Empresa de la Junta de Andalucía (TIC-2533)
Entidades participantes: Universidad de Sevilla, Universidad Politécnica de Valencia, Universidad de Loyola (EEUU), Universidad del Ulster, Actúan de EPOS: ISOTROL, TELVENT, INGENIA, Hospital Universitario Virgen del Rocío.
Universidades colaboradoras: Universidad Politécnica de Valencia, Universidad de Loyola (EEUU), Universidad del Ulster.
Duración: 2008 – 2011
Investigador responsable: Antonio Ruiz Cortés
Número de investigadores participantes: 18 (15 del grupo de la Universidad de Sevilla)
Financiación: 410.421 €
Tipo de participación: Investigador participante a tiempo completo en tareas de aprovisionamiento de servicios Web semánticos.

Título del proyecto: WEB-FACTORIES. Fábricas Software para Sistemas con Arquitectura Orientada a Servicios Web
Entidad financiadora: Ministerio de Ciencia y Tecnología (TIN2006-00472)
Entidades participantes: Universidad de Sevilla y Universidad de Huelva. Actúan de EPOs: XimetriX network Thoughts, Telvent Interactiva S.A, Acromática S.L, Isotrol, Laboratorio de Ingeniería del Software de la NASA y Consejería de Innovación Ciencia y Empresa de la Junta de Andalucía.
Duración: 2007 – 2009
Investigador responsable: Antonio Ruiz Cortés
Número de investigadores participantes: 16 (15 del grupo de la Universidad de Sevilla)
Financiación: 229.200 €
Tipo de participación: Investigador participante a tiempo completo en tareas de aprovisionamiento de servicios Web semánticos.

Título del proyecto: WEB-MADE. Desarrollo de aplicaciones basadas en servicios WEB, subproyecto del proyecto coordinado Desarrollo de aplicaciones basadas en servicios WEB (AgilWeb).
Entidad financiadora: Ministerio de Ciencia y Tecnología. TIC2003-02737-C02-01 (18.13.03.30.06 2003/1210)
Entidades participantes: Universidad de Sevilla y Universidad de Castilla–La Mancha
Duración: 2003 – 2006
Investigador responsable: Miguel Toro Bonilla
Número de investigadores participantes: 12
Financiación: 191.200 €
Tipo de participación: Titulado Superior contratado a tiempo parcial en tareas de apoyo en las actividades de diseño e implementación de un emparejador de servicios Web.

ESTANCIAS EN CENTROS EXTRANJEROS

Centro: Semantic Technology Institute Innsbruck
Localidad: Innsbruck
País: Austria
Fecha: 23/06/2008-22/09/2008 (Estancia confirmada pendiente de llevar a cabo)
Duración: 3 meses
Tema: Selección de Servicios Web Semánticos teniendo en cuenta propiedades no funcionales y preferencias de usuario, en el contexto de la iniciativa europea WSMO/WSMX.

CONTRIBUCIONES A CONGRESOS

Autores: José María García, David Ruiz, Antonio Ruiz-Cortés, Octavio Martín-Díaz, Manuel Resinas.
Título: An Hybrid, QoS-Aware Discovery of Semantic Web Services Using Constraint Programming.
Congreso: International Conference on Service Oriented Computing (ICSOC'07)
Publicación: Springer Verlag, LNCS 4749
DOI: 10.1007/978-3-540-74974-5_6 ISSN: 0302-9743
Páginas: 69 - 80
Fecha: Septiembre 2007
Indicios de Calidad:

- Índice de rechazo del 78.4%
- Categoría A en el ranking CORE.
- Índice H del congreso: 10.

Autores: José María García, David Ruiz, Antonio Ruiz-Cortés
Título: On User Preferences and Utility Functions in Selection: A Semantic Approach.
Congreso: Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC'07).
Publicación: Springer Verlag, LNCS. Por publicar.
ISSN: 0302-9743
Páginas: Por publicar.
Fecha: Septiembre 2007
Indicios de Calidad: Índice de rechazo del 38.8%

Autores: Carlos Müller, Octavio Martín-Díaz, Antonio Ruiz-Cortés y José M. García.
Título: Consistencia y conformidad en un contexto temporal.
Congreso: ZOCO'06 – Métodos y Herramientas para el Desarrollo de Aplicaciones (taller organizado en el contexto del congreso JISBD'06)
Publicación: Libro de Actas
ISBN: 978-84-690-5792-6
Páginas: 15 - 24
Fecha: Octubre 2006

OTROS MÉRITOS (RESUMEN)

- Premio Extraordinario Fin de Carrera - Universidad de Sevilla curso 2004/2005.
- Premio Real Maestranza de Caballería de Sevilla al mejor expediente del curso 2004/2005 en Ingeniería en Informática.
- Premio Ayuntamiento de Sevilla al mejor expediente del curso 2004/2005 en Ingeniería en Informática.
- Finalizado el periodo docente de estudios de tercer ciclo en el programa con mención de calidad concedida por el Ministerio de Educación y Ciencia con referencia MCD2005-00261.
- Diplomado en Inglés por el Instituto de Idiomas de la Universidad de Sevilla en 2005.
- Cursando actualmente Primer Curso de Alemán en el Instituto de Idiomas de la Universidad de Sevilla.
- Alumno interno del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla en el curso 2001/2002.
- Asistencia a los siguientes cursos y seminarios:
    - JSP's Avanzado - CLE Formación (10 horas)
    - Oracle 9i: XML Develop Applications (DXML9) - Oracle Educación (15 horas)
    - Flash MX 2004 para diseñadores - Confederación de Empresarios de Andalucía (50 horas)
    - Programación avanzada en ASP.NET - Confederación de Empresarios de Andalucía (50 horas)
    - Introducción a OpenCMS y Cocoon - Sadiel, S.A. (32 horas)
    - Creación de aplicaciones gráficas en lenguaje C/C++ mediante entornos visuales de programación en sistemas Unix/Linux - Universidad de Sevilla (40 horas)
    - II Curso Linux Avanzado: Administración y Servidores - Universidad de Sevilla (40 horas)
    - Seminario sobre Orientación Sociolaboral - Schlumberger – Sema (8 horas)
- Proyecto Fin de Carrera: Diseño e Implementación de un Framework para el Desarrollo de Servicios de Intermediación; Tutor: Antonio Ruiz Cortés. Calificación obtenida: 9,5.

Appendix C

Bibliography

[1] G. Agre and Z. Marinova. An INFRAWEBS Approach to Dynamic Composition of Semantic Web Services. *Cybernetics and Information Technologies*, 7(1), 2007.

[2] G. Agre, P. Kormushev, and I. Dilov. INFRAWEBS Axiom Editor - A Graphical Ontology-Driven Tool for Creating Complex Logical Expressions. *International Journal Information Theories and Applications*, 13(2):169–178, 2006.

[3] G. Agre, Z. Marinova, T. Pariente, and A. Micsik. Towards Semantic Web Service Engineering. In *Proceedings of the SMR2 2007 Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web*, volume 243 of *CEUR WS*, 2007.

[4] G. Agre. INFRAWEBS Designer - A Graphical Tool for Designing Semantic Web Services. In J. Euzenat and J. Domingue, editors, *AIMSA 2006*, volume 4183 of *LNAI*, pages 275–289. Springer, 2006.

[5] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, M. T. Schmidt, A. Sheth, and K. Verma. Web service semantics - WSDL-S. Technical Note v. 1.0, IBM and University of Georgia, April 2005.

[6] A. Ankolekar, D. Martin, D. Mcguinness, S. Mcilraith, M. Paolucci, and B. Parsia. OWL-S' relationship to selected other technologies. Technical report, DAML, 2004.

[7] G. Antoniou and F. van Harmelen. *A Semantic Web Primer*. Cooperative Information Systems. MIT Press, April 2004.

[8] F. Baader, D. Calvanese, D. L. Mcguinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook : Theory, Implementation and Applications*. Cambridge University Press, January 2003.

[9] S. Battle, A. Bernstein, H. Boley, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. Mcilraith, D. Mcguinness, J. Su, and S. Tabet. Semantic web services framework (SWSF) overview. Technical report, World Wide Web Consortium, September 2005.

[10] C. Becker, K. Geihs, and J. Gramberg. Representation of Quality of Service Preferences by Contract Hierarchies. 1999.

[11] B. Benatallah, M. Hacid, C. Rey, and F. Toumani. Semantic reasoning for web services discovery. In *WWW Workshop on E-Services and the Semantic Web*, 2003.

[12] S. Benbernou and M. Hacid. Resolution and constraint propagation for semantic web services discovery. *Distributed and Parallel Databases*, 18(1):65–81, 2005.

[13] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[14] P. Bonatti and P. Festa. On optimal service selection. In *14th international conference on World Wide Web*, pages 530–538, 2005.

[15] Web services architecture. Technical report, World Wide Web Consortium, February 2004.

[16] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison Wesley, September 2000.

[17] C. Bussler, D. Fensel, and A. Maedche. A conceptual architecture for semantic web enabled web services. *SIGMOD Rec.*, 31(4):24–29, December 2002.

[18] L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, V. Tanasescu, C. Pedrinaci, and B. Norton. IRS-III: A broker for semantic web services based applications. In *International Semantic Web Conference*, pages 201–214, 2006.

[19] Web Services Description Language (WSDL) Version 2.0. Technical report, World Wide Web Consortium, June 2007.

[20] UDDI Version 3.0.2. Technical report, OASIS, October 2004.

[21] M. Crubézy, M. A. Musen, E. Motta, and W. Lu. Configuring online problem-solving resources with the internet reasoning service. *IEEE Intelligent Systems*, 18(2):34–42, 2003.

[22] J. Davies, R. Studer, and P. Warren, editors. *Semantic Web Technologies: Trends and Research in Ontology-based Systems*. John Wiley & Sons, July 2006.

[23] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel. *D16.1v0.21: The Web Service Modeling Language WSML*. WSMO, October 2005.

[24] S. Dietze, A. Gugliotta, and J. Domingue. Context-aware process support through automatic selection and invocation of semantic web services. In *Service-Oriented Computing and Applications, 2007. SOCA '07. IEEE International Conference on*, pages 199–206, 2007.

[25] M. Dimitrov, D. Ognyanov, and A. Simov. wsmo4j programmers guide. Technical report, OntoText Lab, November 2004.

[26] M. Dimitrov, A. Simov, V. Momtchev, and M. Konstantinov. WSMO Studio - a semantic web services modelling environment for WSMO. In *The Semantic Web: Research and Applications (ESWC 2007 Proceedings)*, volume 4519 of *LNCS*, pages 749–758. Springer, 2007.

[27] G. Dobson, R. Lock, and I. Sommerville. QoSOnt: a QoS Ontology for Service-Centric Systems. In *EUROMICRO-SEAA*, pages 80–87. IEEE Computer Society, 2005.

[28] J. Domingue. Tadzebao And Webonto: Discussing, browsing, editing ontologies on the web. In *11th Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1998.

[29] Semantic annotations for WSDL and XML schema. Technical report, World Wide Web Consortium, August 2007.

[30] C. Feier and J. Domingue. WSMO primer. WSMO Final Draft D3.1v0.1, DERI, April 2005.

[31] D. Fensel and C. Bussler. The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[32] S. Galizia, A. Gugliotta, and J. Domingue. A trust based methodology for web service selection. In *Semantic Computing, 2007. ICSC 2007. International Conference on*, pages 193–200, 2007.

[33] J. M. García, D. Ruiz, A. Ruiz-Cortés, O. Martín-Díaz, and M. Resinas. An hybrid, QoS-aware discovery of semantic web services using constraint programming. In B. Krämer, K.-J. Lin, and P. Narasimhan, editors, *ICSOC 2007*, volume 4749 of *LNCS*, pages 69–80. Springer, 2007.

[34] J. M. García, D. Ruiz, and A. Ruiz-Cortés. On user preferences and utility functions in selection: A semantic approach. In *1st Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop*, LNCS. Springer, 2008. To appear.

[35] K. Gomadam, K. Verma, D. Brewer, A. Sheth, and J. Miller. Radiant: A tool for semantic annotation of web services. In *4th International Semantic Web Conference*, 2005.

[36] J. González-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. Technical Report HPL-2001-265, Hewlett Packard Labs, 2001.

[37] T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993.

[38] M. Grüninger and C. Menzel. The process specification language (PSL) theory and applications. *AI Magazine*, 24(3):63–74, 2003.

[39] SOAP Version 1.2. Technical report, World Wide Web Consortium, April 2007.

[40] V. Haarslev and R. Möller. RACER system description. In *Automated Reasoning, First International Joint Conference, IJCAR 2001*, pages 701–706, 2001.

[41] F. Hakimpour, D. Sell, L. Cabral, J. Domingue, and E. Motta. Semantic web service composition in IRS-III: the structured approach. In *E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference on*, pages 484–487, 2005.

[42] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-Oriented Architecture. In *IEEE International Conference on Web Services (ICWS'05)*, volume 0, pages 321–328, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[43] A. Heß, E. Johnston, and N. Kushmerick. ASSAM: A tool for semi-automatically annotating semantic web services. In *3rd International Semantic Web Conference*, 2004.

[44] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Technical report, W3C Member Submission, 2004.

[45] I. Horrocks. FaCT and iFaCT. In *Int. Workshop on Description Logics*, 1999.

[46] M. Jaeger, G. Rojec-Goldmann, C. Liebetruth, G. Mühl, and K. Geihs. Ranked matching for service descriptions using OWL-S. *Kummunikation in Verteilten Systemen (KiVS)*, pages 91–102, 2005.

[47] F. Kaufer and M. Klusch. WSMO-MX: A logic programming based hybrid service matchmaker. pages 161–170, 2006.

[48] U. Keller, R. Lara, A. Polleres, I. Toma, M. Kifer, and D. Fensel. WSMO web service discovery. Technical Report D5.1v0.1, DERI, November 2004.

[49] M. Kerrigan. Web Service Modeling Toolkit (WSMT). Technical report, DERI, April 2005.

[50] L. Kovács, A. Micsik, and P. Pallinger. Two-phase semantic web service discovery method for finding intersection matches using logic programming. In *Workshop on Semantics for Web Services (SemWS'06)*, 2006.

[51] K. Kritikos and D. Plexousakis. Semantic QoS metric matching. In *ECOWS 2006*, pages 265–274. IEEE Computer Society, 2006.

[52] R. Lara, D. Roman, A. Polleres, and D. Fensel. A conceptual comparison of WSMO and OWL-S. In *ECOWS 2004*, volume 3250 of *LNCS*, pages 254–269. Springer, 2004.

[53] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Int. World Wide Web Conference*, pages 331–339, 2003.

[54] K. Li. *Lumina: Using WSDL-S for Web Service Discovery*. PhD thesis, University of Georgia, 2005.

[55] Y. Liu, A. H. H. Ngu, and L. Zeng. QoS computation and policing in dynamic web service selection. In *WWW (Alternate Track Papers & Posters)*, pages 66–73, 2004.

[56] C. Lutz and U. Sattler. A proposal for describing services with DLs. In *Int. Workshop on Description Logics*, 2002.

[57] RDF Primer. Technical report, World Wide Web Consortium, February 2004.

[58] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. Mcdermott, and Others. OWL-S: Semantic Markup for Web Services. Technical Report 1.1, DAML, 2004.

[59] 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. In J. Cardoso and A. Sheth, editors, *SWSWPC 2004*, volume 3387 of *LNCS*, pages 26–42. Springer, 2004.

[60] E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *Internet Computing, IEEE*, 8(5):84–93, 2004.

[61] OWL web ontology language overview. Technical report, World Wide Web Consortium, February 2004.

[62] S. McIlraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[63] A. Mocan, E. Cimpian, M. Stollberg, F. Scharffe, and J. Scicluna. WSMO mediators. WSMO Working Draft D29v0.2, DERI, December 2005.

[64] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A framework and infrastructure for semantic web services. In *Int. Semantic Web Conference*, pages 306–318, 2003.

[65] E. Motta. *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*. IOS Press, 1999.

[66] M. Moyano, A. Buccella, and A. Cechich. Semantic resources to support web services selection, mediation, and composition. In *Proceedings of the Third Latin American Web Congress*. IEEE, 2005.

[67] N. Oldham, K. Verma, A. Sheth, and F. Hakimpour. Semantic WS-agreement partner selection. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 697–706. ACM, 2006.

[68] B. Omelayenko, M. Crubézy, D. Fensel, R. V. Benjamins, B. J. Wielinga, E. Motta, M. A. Musen, and Y. Ding. UPML: The language and tool support for making the semantic web alive. In D. Fensel, J. A. Hendler, H. Lieberman, and W. Wahlster, editors, *Spinning the Semantic Web*, pages 141–170. MIT Press, 2003.

[69] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Int. Semantic Web Conference*, pages 333–347, 2002.

[70] J. Pathak, N. Koul, D. Caragea, and V. G. Honavar. A framework for semantic web services discovery. In *WIDM '05: Proceedings of the 7th annual ACM international workshop on Web information and data management*, pages 45–50, New York, NY, USA, 2005. ACM Press.

[71] A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S web service annotation framework. In *International Conference on World Wide Web*, pages 553–562, 2004.

[72] SPARQL query language for RDF. Technical report, World Wide Web Consortium, January 2008.

[73] S. Ran. A model for web services discovery with QoS. *SIGecom Exch.*, 4(1):1–10, 2003.

[74] M. Resinas. Propuesta para la provisión de servicios web usando criterios de calidad. Master's thesis, Universidad de Sevilla, June 2004.

[75] D. Roman, H. Lausen, and U. Keller. Web service modeling ontology (wsmo). Technical Report D2 v1.3 Final Draft, WSMO, 2006.

[76] A. Ruiz-Cortés, O. Martín-Díaz, A. Durán, and M. Toro. Improving the automatic procurement of web services using constraint programming. *Int. J. Cooperative Inf. Syst.*, 14(4):439–468, 2005.

[77] C. Schröpfer, M. Schönherr, P. Offermann, and M. Ahrens. A flexible approach to service management-related service description in SOAs. pages 47–64. 2007.

[78] J. Scicluna, C. Abela, and M. Montebello. Visual modelling of OWL-S services. In *IADIS International Conference WWW/Internet*, 2004.

[79] G. Silver, J. Miller, A. Sheth, J. Myers, A. Maduko, and R. Jafri. Modeling and simulation of quality of service for composite web services. In *7th World Multiconference on Systemics, Cybernetics and Informatics*, 2003.

[80] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *Intl. Conference on Web Services*, pages 395–401, 2003.

[81] K. Sivashanmugam, J. Miller, A. Sheth, and K. Verma. Framework for semantic web process composition. *International Journal of Electronic Commerce*, 9(2):71–106, 2004.

[82] A. Soydan Bilgin and M.P. Singh. A DAML-based repository for QoS-aware semantic Web service selection. In *IEEE International Conference on Web Services*, pages 368–375, 2004.

[83] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan. Automated discovery, interaction and composition of semantic web services. *J. Web Sem.*, 1(1):27–46, 2003.

[84] K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan. Dynamic discovery and coordination of agent-based semantic web services. *IEEE Internet Computing*, 8(3):66–73, 2004.

[85] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A scalable P2P infrastructure of registries for semantic publication and discovery of web services. *Inf. Tech. Management*, 6(1):17–39, 2005.

[86] L. H. Vu, M. Hauswirth, F. Porto, and K. Aberer. A search engine for QoS-enabled discovery of semantic web services. *International Journal of Business Process Integration and Management*, 1(4):244–255, 2006.

[87] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma. A QoS-Aware Selection Model for Semantic Web Services. In A. Dan and W. Lamersdorf, editors, *ICSOC 2006*, volume 4294 of *LNCS*, pages 390–401. Springer, 2006.

[88] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.

[89] C. Zhou, L. Chia, and B. Lee. DAML-QoS ontology for web services. In *IEEE International Conference on Web Services*, pages 472–479, 2004.

This document was typeset on 2008/4/4 using RC-BOOK α2.11 for LaTeX$_{2\epsilon}$.
Should you want to use this document class, please send mail to
contact@tdg-seville.info.