Pitfalls of OWL-S – A Practical Semantic Web Use Case

Steffen Balzer
Dept. of Artifical Intelligence
University of Ulm
Ulm, Germany
balzer@informatik.uni-ulm.de

Thorsten Liebig
Dept. of Artifical Intelligence
University of Ulm
Ulm, Germany
liebig@informatik.uni-ulm.de

Matthias Wagner
DoCoMo Communications
Laboratories Europe GmbH
Munich, Germany
wagner@docomolab-euro.com

ABSTRACT

OWL-S is a combined effort of the Semantic Web and the Web Service community to facilitate an intelligent service provisioning on the Semantic Web. The vision of OWL-S includes automatic service discovery, invocation, composition, orchestration and monitoring of Web-Services through their semantic descriptions. In this paper, we investigate the practical applicability of the current OWL-S specification and show that, in spite of the large momentum of OWL-S, significantly more work needs to be done before the vision of truly intelligent Semantic Web Services can become true. We therefore study the case of an autonomous travel agent that helps users with online hotel arrangements. The aim of our work is twofold: on the one side, we show step-by-step how a prototypical implementation can be realized based on current semantic technologies around UDDI, WSDL, and SOAP. On the other hand, we reveal pitfalls in the current version of OWL-S that severely limit its support for mechanizing service discovery, configuration, combination and automated execution. Throughout the paper, we present practical solutions and workarounds to existing OWL-S shortcomings and hope to therewith further stimulate the ongoing work on Semantic Web Services.

Categories and Subject Descriptors: H.4 [Information Systems Applications]: Miscellaneous

General Terms: Experimentation, Languages

Keywords: OWL-S, Semantic Web Services

1. INTRODUCTION

Web services are Internet-based, distributed modular applications that provide standard interfaces and communication protocols aiming at efficient and effective service integration. In times of the ubiquitous Internet, the Web Service paradigm is expected to substantially alter the world of modern business processes. By now web services have started to show their usefulness in a wide variety of domains including applications in business-to-business integration, business process integration and management, e-sourcing and content distribution.

In terms of basic information about service providers as well as service invocation and integration details, web service advertisements are made available on an Web-wide network of UDDI registries for Universal Description, Discovery and Integration [14]. On the basis of the web services Description Language (WSDL) [4], UDDI is designed to function in a fashion similar to white pages or yellow pages, where businesses and services can be looked up by name and/or by standardized service taxonomies. Though UDDI and WSDL have become the de facto standard in the field they still suffer from conceptual shortcomings: UDDI is limited to keyword matching and does not support any kind of semantic matching that would be necessary for a flexible matching of service advertisements to service requests. On the other hand, research in the area of the so-called Semantic Web seeks a solution to this unsatisfying situation. Generally speaking, the Semantic Web encompasses efforts to populate the Web with content having formal semantics and rich service descriptions. OWL-S [1] is the most prominent amongst different semantic efforts around UDDI, WSDL and SOAP that try to enable automated agents to reason about web service descriptions and to perform intelligent service discovery, comparison and composition.

In this paper, we study the practical applicability of the current OWL-S specification through the case of an autonomous travel agent . This paper is organized as follows: Section 2 briefly introduces the current OWL-S 1.0 standard. Section 3 sketches the running examples of our paper and gives further insights into major design issues. Section 4 gives an overview of our testbed implementation. Section 5 highlights the major pitfalls of OWL-S revealed during prototyping. Here, we address issues concerned with the validity of OWL-S documents, profile matchmaking and process execution. We present workarounds to current OWL-S shortcomings before we wrap-up our critical discussions in section 6 and summarize our results in section 7. Finally, section 8 presents IRS-II and WSMO as two major related approaches.

2. OWL-S

OWL-S is an ontology-based approach to the semantic description of web services that is inspired by other research

ICSOC'04, November 15–19, 2004, New York, New York, USA.

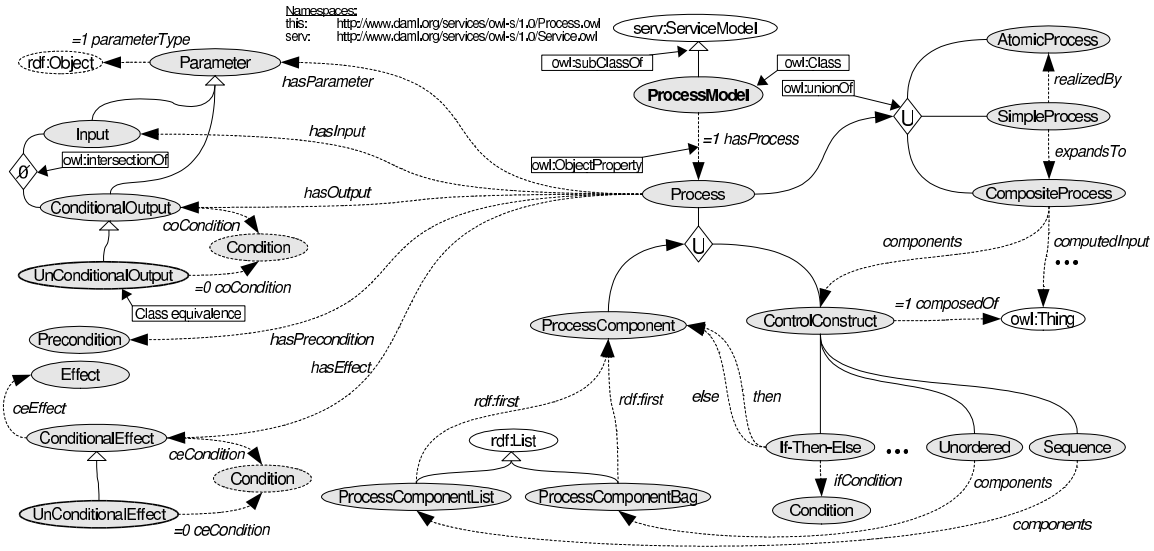

Figure 1: OWL-S process ontology

in the area of the semantic web that encompasses efforts to populate the web with content and services having formal semantics. The ultimate goal of OWL-S is to provide an ontology that allows software agents to discover, execute and compose web services automatically. Currently the structure of the OWL-S ontology is threefold and consists of a service profile for advertising and discovering services, a process model which gives a detailed description of a service's operation and a service grounding which provides details on how to interoperate with a service via message exchange. To enable automatic discovery and execution an OWL-S description must consist of one or more profiles, a process model and one or more groundings. The following sections briefly describe the particular elements. For a detailed description see [1].

2.1 Profiles

In service discovery profiles are applied in two ways. On one hand, they are used by service providers to publish web services. These profiles are called *advertisements*. On the other hand, profiles are used by a service requester to describe the web service to be searched for. During discovery this *request* is compared with published advertisements to find suitable services.

The profile class of the OWL-S profile ontology specifies web service descriptions based on their functional and non-functional parameters. The functional description is based on the transformation of data and states during the execution of a web service. The profile specifies the inputs that a web service requires, the outputs that it generates, the preconditions that must hold in order to execute the web service and the effects that an execution generates. The results may depend on the execution because of potential conditional statements. IOPEs[1] are specified in profiles by referring to the classes Input, ConditionalOutput, Precondition and ConditionalEffect of the process ontology (see figure 1). Therefore, process models can be viewed as an abstraction of service execution details tailored to discovery.

Non-functional parameters are divided into two sections. First as semi-structured information intended for human users that is of no relevance for semantic service discovery e.g. serviceName, textDescription etc. Second non-functional parameters can be specified as sub-classes of ServiceParameter to incorporate additional requirements regarding service capabilities into the discovery process, e.g. the geographic scope of a web service, security or quality-of-service requirements etc.

2.2 Process Models

The Process ontology as shown in figure 1 is used to define process models that describe the execution of a web service in detail by specifying the flow of data and control between the particular methods of a web service. In order to achieve the results defined in a profile an agent has to execute the corresponding process model step by step considering all defined dependencies between IOs and PEs.

The execution graph of a process model can be composed using different types of processes and control constructs. OWL-S defines three classes of processes. Atomic processes (AtomicProcess) are directly executable and contain no further sub-processes. From the view of the caller atomic processes are executed in a single step which corresponds to the invocation of a web service method. Simple processes (SimpleProcess) are not executable. They are used to specify abstract views of concrete processes by hiding certain IOPEs. Composite processes (CompositeProcess) are specified through composition of atomic, simple and composite processes recursively by referring to control constructs (ControlConstruct) using the property composedOf. Control constructs define specific execution orderings on the contained processes.

2.3 Groundings

The previously described profiles and process models serve as abstract specifications of web service characteristics. The

[1]combinations of functional parameters are denoted using their first letters.

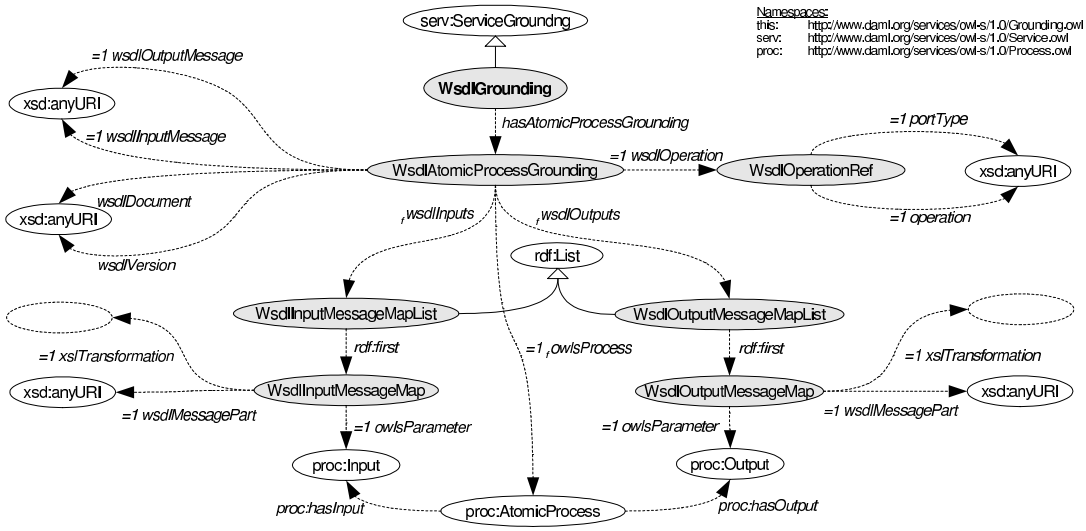

Figure 2: OWL-S grounding ontology

grounding now enables communication with a concrete web service by binding abstract IOs of atomic processes to concrete message formats.

OWL-S defines exemplarily an ontology for grounding process descriptions to WSDL as shown in figure 2 which is based on three corresponding descriptive elements of OWL-S and WSDL. Firstly, an atomic process in OWL-S corresponds to a wsdl:operation. Secondly, IOs of an atomic process are referred to wsdl:parts of input and output message definitions in WSDL. Thirdly, the types (i.e. OWL classes) of IOs in OWL-S correspond to the concept of abstract types in WSDL. A WsdlGrounding contains one WsdlAtomicProcessGrounding for each atomic process of the corresponding process model. WsdlOperationRef defines the access to the web service method using the properties operation and portType. Additionally, the WSDL messages are referenced using Wsdl{Input|Output}Message.

The mapping of parameter types is defined using rdf:Lists of Wsdl{Input|Output}MessageMaps. OWL-S considers two ways of referring the appropriate OWL representations. If the web service is an "OWL native speaker" the OWL class representing the parameter type is referred directly by owlsParameter. For contemporary web services OWL-S uses XSLT to convert parameter descriptions from OWL-S to XML Schema and vice versa.

3. APPLICATION SCENARIO

Making hotel arrangements is an essential part of travel planning. The internet provides uncountable web sites where users can search for suitable hotels and make their reservations. But finding the best offer is still a very time consuming task that must be performed manually. The user has to query the web using search engines, must pick suitable hits, query each web site with the users preferences and dates, pick the best offer and finally make the reservation.

In the application scenario of this work the two most time-consuming tasks, i.e. discovery and execution of hotel booking web services, are performed automatically by a software agent. All information needed by this agent will be provided either by querying the user's profile or by interacting with the user during execution. The next two sections describe in more detail how automated discovery and execution are realized by the hotel booking agent.

3.1 Service Discovery

The OWL-S profile is the main data structure used for service discovery. It's role is twofold. Both service requests and service advertisements are described as profiles. In order to search for suitable web services the hotel booking agent creates a request containing functional and non-functional properties of the web service looked for and sends it to the service registry. The service registry compares the request to the registered advertisements and returns the matching results. This process is called *matchmaking of services*. In a realistic application scenario it must be assumed that an advertisement does not match a request exactly in most cases. A web service e.g. that has only registered explicitly for booking lodgings in general can also be used for booking hotel rooms. As a consequence, it should appear in the result set of the query. Such implicit relationships between profiles can be derived using subsumption. [9] describes the basic concept how matchmaking can be realized with subsumption. Accordingly, two profiles are compatible, iff their intersection is satisfiable. On the basis of subsumption relationships between requests and advertisements [9] defines five different *degrees of match*. In our application scenario all specified results in a request must be met by a single advertisement, because the hotel booking agent is not capable of composing web services. Thus, only exact and subsume matches are appropriate. Matches of this kind are called *usable*. All remaining matches are called *unusable*.

The most important non-functional parameter that has been considered in the scenario is the *geographic scope* of web services. E.g. a service for booking hotel rooms in France is not usable for booking hotel rooms in Germany. A service covering all hotels of a specific group world-wide can be used in both countries. A major observation while defining the profiles was that functional parameters can depend on non-functional ones. E.g. no city must be defined to invoke a

booking service of a single hotel because it is already defined by its location resp. geographic scope. In order to keep the scenario simple but yet expressive the following inputs have been considered: ArrivalDate and DepartureDate denote the beginning and end of the stay. City references the required hotel location. Customer contains customer information like address, name etc. CreditCard contains credit card information that is used by the agent to pay for transactions. The only output produced by a successful execution which is relevant for further tasks is a BookingCode specific to each booking service. It can be used e.g. to cancel a reservation again. Preconditions such as "Customer must have a credit card" are implicitly modeled by inputs. RoomBooked is the only effect that has been considered. All IOPEs have been modeled as part of domain-specific OWL ontologies[2].

3.2 Service Execution

Very likely, different web services which generate the same results may differ in their execution. E.g. a booking service could require to register for a customer account before a reservation can be made whereas others could require full customer information for every reservation. The information how a web service must be executed in detail is provided by the process model. To be able to execute such a process model the hotel booking agent must generate a semantically equivalent execution model by transforming the defined data flow and control flow with the associated conditions. After instantiating this model the agent executes it step by step.

To be able to execute an atomic process by invoking a web service method the agent must first allocate values for the required parameters and provide appropriate containers for the results to be returned. All necessary information for this task is either derived from the agent's knowledge base or enquired by interaction with the user. Hereafter, the agent generates the appropriate messages for communication with the web service using its grounding definition. As part of this, the parameter values must be transformed into semantically lower representations, i.e. XML schema values in the case of SOAP. After successfully invoking the particular web service method the agent must process the results by asserting the generated outputs and specified effects in its knowledge base.

Figure 3 shows the process model of a virtual HotelKroneBookingService as activity diagram with associated control flow. Aside from the IOs referenced by the profile additional temporary data has been modeled that is necessary for a correct execution. The white node represents an agent activity, that has to be executed to generate suitable inputs. Here, the agent asks the user to select the preferred VacantRoomDescription that includes the VacantRoomCode required to confirm the booking (not part of the model).

4. SYSTEM ARCHITECTURE

This section gives an overview of the system architecture that has been used to implement the application scenario. The design and implementation of the system is based on the *Java* programming language using *JDK 1.4.2*.

4.1 Main Components

Figure 4 shows the static system structure with its main

[2]http://www.informatik.uni-ulm.de/ki/trap/ontologies/domain/

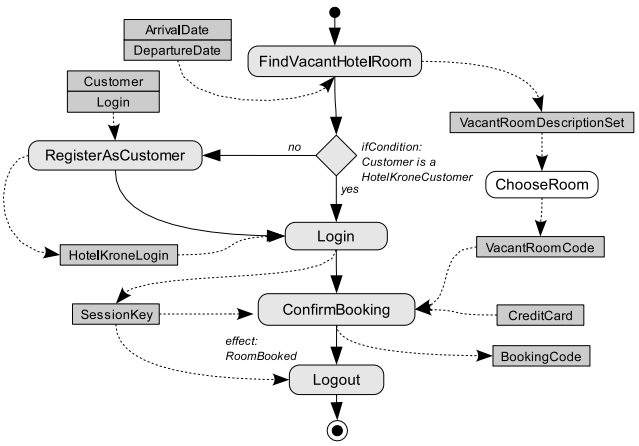

Figure 3: Booking agent process

components. The *JBoss*[3] application server provides the framework for implementing the web services. All web-services specific communication is performed by the Apache AXIS web services framework. The OWL documents that describe the OWL-S ontologies, domain ontologies as well as the semantic web services are published by an *Apache*[4] web server. The description logics (DL) system *RACER* [7] provides the infrastructure and mechanisms to reason about OWL knowledge. Finally, the *booking agent* implements the interactive selection and booking of hotel rooms. It uses the *Jena*[5] semantic web framework to gain graph-based access to OWL and RDFS documents. Information about the user is stored and retrieved by the agent in OWL format using the *user profile*.

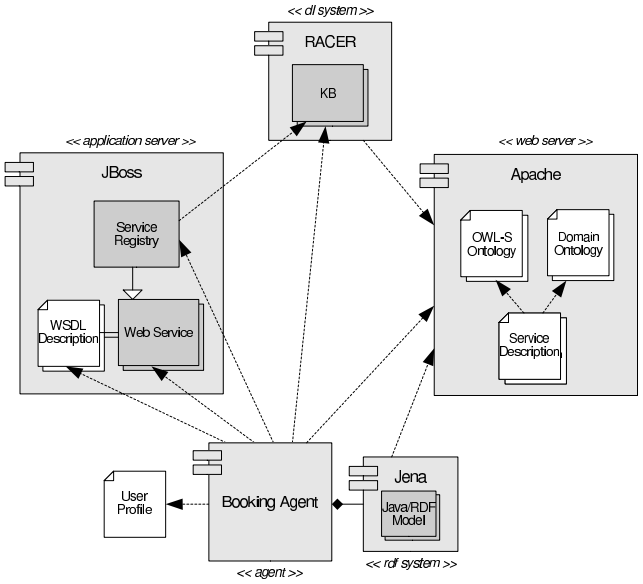

Figure 4: Main components

[3]http://www.jboss.org
[4]http://httpd.apache.org
[5]http://www.hpl.hp.com/semweb/jena-top.html

4.2 Agent Structure

Figure 5 shows booking agent containing the classes and packages that are responsible for executing an OWL-S process model. The agent's knowledge base `agentKB` is managed by the `MainProcess`. It is initialized with the user profile at startup and is used to derive and store parameter values during execution. The `ProcessExecutionModel` class serves as a facade for the `pem` package and encapsulates all functionality required to execute a Semantic Web Service. The `MainProcess` creates one instance of a `ProcessExecutionModel` for each process model. The OWL-S descriptions required to instantiate and execute a process model are maintained in the `serviceKB` knowledge base. It is used by an instance of the `RacerParser` class during instantiation to generate the execution graph of the process model. All structural elements of the execution graph are implemented by the `graph` package. The `TypeMapperFactory` is used by the `RacerParser` to create `TypeMapper` instances that are responsible for the transformation of parameter values during execution (see 5.3.2). While parsing the process model `RacerParser` fills the `GoalCache` with expected execution results, i.e. certain outputs and effects. It is used by the `ProcessExecutionModel` later to determine successful process execution.

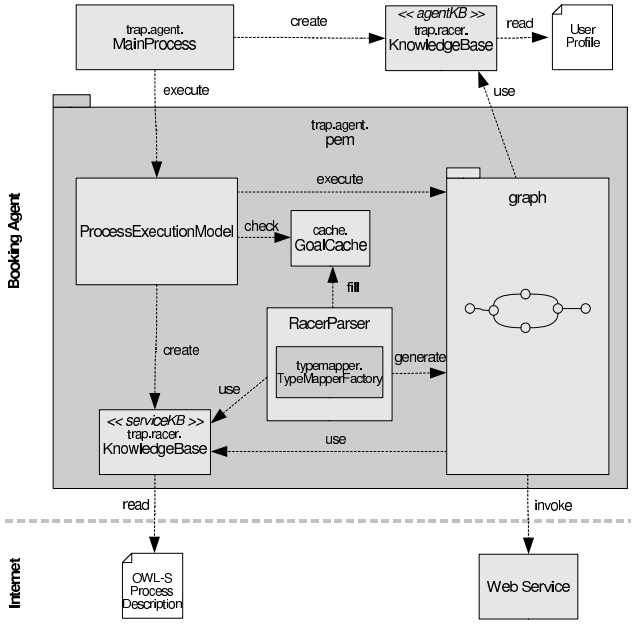

Figure 5: Execution model structure

5. PITFALLS

This section describes the major difficulties and shortcomings of the OWL-S specification that have been revealed during the realization of the application scenario. Most of them were discovered while putting process execution to work. So the focus is kept mainly on this part.

5.1 Validity of OWL-S Documents

To be able to use the OWL-S ontologies in a practical environment several corrections had to be done. In order to enable their use in DL reasoners like RACER it was necessary to tailor OWL-S down to OWL DL. Furthermore, changes in the fundamental model have been made to eliminate semantic inconsistencies and to introduce own concepts. The OWL [12] and OWL-S 1.0 specifications [3] provide the syntactic and semantic foundations for all corrections.

5.1.1 Syntactic Corrections

This section describes the necessary syntactic corrections for tailoring OWL-S' expressiveness down to OWL DL in order to enable their use in DL reasoners that provide sound and complete inference. This means in particular that language constructs of OWL Full must be avoided. However, the OWL-S ontologies contain both OWL Full constructs and fundamental syntactic errors[6]. The correction that are described consecutively refer to all OWL-S ontologies.

Cardinality constraints in OWL must be defined according to the *RDF Datatyping Schema*, i.e. their types must be specified explicitly. In all cardinality constraints defined in the OWL-S ontologies these type specifications are missing.

In some places of the OWL-S ontologies rdf:Property is used instead of owl:ObjectProperty to define properties with OWL classes as range. However, rdf:Property doesn't belong to the allowed vocabulary of OWL DL. Thus, its occurrences have been replaced with owl:ObjectProperty.

According to the OWL language reference [3], functional properties must be defined by an additional type qualification. Definitions of functional properties just using owl:FunctionalProperty like practiced in the OWL-S ontologies are therefore syntactically not correct and must be replaced.

Throughout the OWL-S ontologies further language constructs are used that are not allowed in OWL DL, e.g. rdf:List, rdf:Resource etc. However, corrections of these constructs can not be performed on a syntactic level but only by altering semantics (see next section).

Finally, it must be mentioned that in the process model definitions of the examples accompanying OWL-S the construct `rdf:parsetype="Collection"` to describe sets resp. lists. This is neither allowed in OWL DL nor OWL Full. Therefore, in the definitions of the hotel booking process an own list construct has been used, which will be presented in the next section.

5.1.2 Semantic Modifications

There are three major reasons which have lead to changes in the ontological model of OWL-S in the context of our realization. Firstly, like mentioned before, invalid language constructs had to be replaced with semantically similar structures unless a syntactic correction was not possible. Secondly, modification an extension of the model has been necessary to introduce own concepts. Thirdly, the correction of semantic inconsistencies defined in the OWL-S ontologies have caused changes in the model. These modifications are discussed now in more detail.

All definitions used to replace invalid OWL DL language constructs have been modelled in the *Base* ontology[7]. The constructs rdf:Literal, rdf:Resource have simply been replaced with base:Resource and base:Literal. The set of resources denoted by rdf:Set has been replaced by a set of OWL instances

[6]Validity was checked using the OWL Validator from the University of Manchester (`http://phoebus.cs.man.ac.uk:9999/OWL/Validator`).

[7]`http://www.informatik.uni-ulm.de/ki/trap/ontologies/Base.owl`

modelled with base:Set and its property hasSetElement that links owl:Things to the set. Lists are expressed by the LISP-like list structure base:List, i.e. a list can be either an empty list or a non-empty list that consists of a list element and a rest also representing a list.

Attention must be paid to the fact that in contrast to OWL Full OWL classes cannot be used as property fillers in OWL DL which is in particular useful for referencing semantic types in instance definitions. As a consequence, a definition of profiles and processes as intended by the OWL-S examples is not feasible anymore. To overcome this problem, special instances of OWL classes denoting parameter types have been used as representatives in OWL-S descriptions. With help of this changes, modelling profiles and processes as instances can be maintained.

In order to bring the application scenario to work two own concepts had to be integrated into the OWL-S model. Firstly, a simple model for representing conditions has been developed. Secondly, a semantic concept for the bidirectional type mapping has been integrated. These concepts are discussed in more detail in section 5.3. The fundamental class for representing conditions is cond:Condition. It replaces the proc:Condition class that functions as a placeholder for a future conditional model. proc:Precondition and proc:Effect are now defined as disjoint subclasses of cond:Condition. proc:Effect reduces the model to simple conditions by replacing proc:ConditionalEffect. Finally, the range of the hasEffect property in the profile ontology has been changed to proc:Effect. Figure 6 shows how the process ontology has been modified.

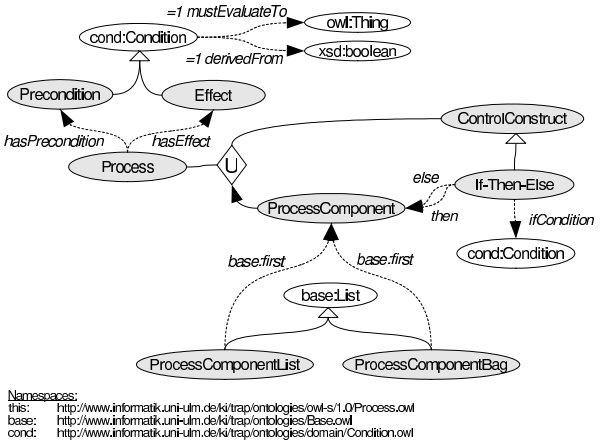

Figure 6: Modifications of the process ontology

The type mapping concept has been integrated in a very simple way. Merely a new attribute named rdfMapping with the type xsd:anyURIand a cardinality of one has been added to the grounding ontology. This way, the particular RDF document which describes the type mapping can be bound to the corresponding WsdlMessageMap.

In the process ontology only the namespace of the grounding has been corrected. In the grounding ontology the data types of the two attributes operation and wsdlMessagePart have been modified from xsd:anyURI to xsd:NMTOKEN. According to the WSDL specification [4] the referenced elements are only unique within the containing definitions and therefore specified as xsd:NMTOKEN.

Furthermore, another correction can be carried out on the definition of WsdlMessageMap lists in the grounding ontology. Actually, this is no semantic inconsistency, but the definition can be considerably simplified, due to the fact that parameter order of an operation can be derived from its WSDL bindings. Thus, WsdlMessageMap are bound directly to their corresponding WsdlAtomicProcessGrounding. Figure 7 shows the modifications of the grounding ontology exemplarily for inputs.

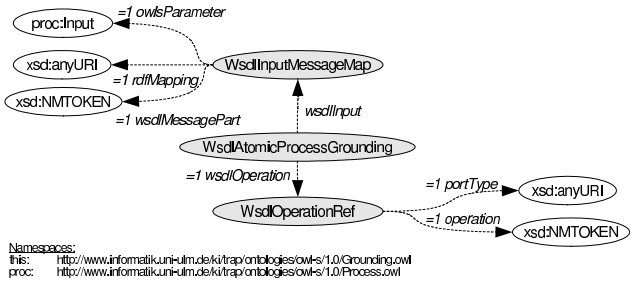

Figure 7: Modifications of the grounding ontology

5.2 Matchmaking of Profiles

OWL-S does not provide any concrete concepts for publication and discovery of services. It just claims to be universally applicable due to its declarative descriptions. But the requirements of a matchmaking algorithm strongly depend on the capabilities of the requester and the way functional and non-functional parameters are modelled. In this section we show with a string of examples how modelling alternatives of parameters and requester capabilities influence matchmaking strategies and vice versa.

5.2.1 Task-oriented Matchmaking

A fundamental problem of defining a whole set of IOPEs in a request is *over-specification*. While matching strictly with IOPEs is practically suitable to enact static workflows based e.g. on BPEL4WS [10] it is cumbersome for agents that plan their actions dynamically. Even if advertisements would fulfill the requested task (i.e. OEs) the HotelKroneBookingService e.g. would not match in the desired way with a request that defines the input city because this input parameter simply is missing in its advertisement. Another approach that we call *task-oriented matchmaking* is derived from AI planning and seems to be more suitable.

The main objective of service discovery is to find services that fulfill certain tasks, i.e. OEs, under certain circumstances, i.e. non-functional parameters. IPs are treated by the requesting agent afterwards by applying techniques like regression planning. Thus, the service registry is treated as a repository of plan operators. In combination with subsumption this leads to significant conceptual advantages. The problem of over-specified requests disappears, because only OEs and service parameters are used in requests. Profiles could be classified automatically in a predefined subsumption-hierarchy of tasks. This hierarchy could also be used for relaxation of requests as proposed in [2]. If profiles would be modelled as class terms, even the predefined task-hierarchy could be omitted.

The OWL-S profile hierarchy[8] does not establish any connection to web-service parameters. Thus, it must be assumed that classification of services is done manually by the provider. As a consequence of the lack of automatic classification, no implicit relationships between properties will be discovered. Furthermore, inconsistencies can easily be introduced into the repository by providers.

5.2.2 Matchmaking of Service Parameters

In this section we show how modelling alternatives of service parameters influence the matchmaking algorithm. Figure 8 shows the geographic scope modelled as subsumption hierarchy. The major advantage here is that subsumption is sufficient for matchmaking, i.e. a request can simply be posted as class definition. On the other hand, this variant can only serve for special "views" on the geographic scope. It cannot be used for our scenario, because e.g. a request for hotels in Ulm does not return services that encompass a wider scope. Here, the subsumption relationship si contrary to our intended semantics of a geographic scope. The subsumption relationship is rather interpreted as *located-in* than as *is-a* relation.

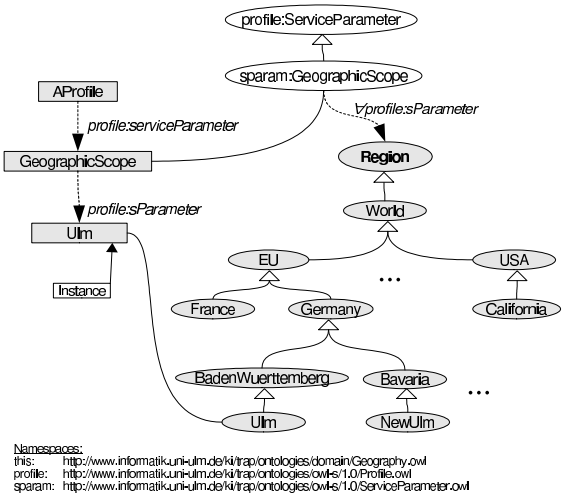

Figure 8: Geographic scope based on subsumption

A solution to these shortcomings shows figure 9. Here, the geographic scope is modelled using a *transitive property*. Regions that are contained by another region can easily be derived with the transitive closure of subRegionOf. Scope definitions simply refer to existing instances of Region. However, the tradeoff here is that subsumption on its own is not sufficient for matchmaking anymore, because subregions first must be derived and then treated in separate queries.

The main result is, that subsumption can be used effectively in matchmaking to derive polymorphic relationships of functional parameters. However, for matchmaking of non-functional parameters special algorithms must be provided.

5.3 Execution of Process Models

During the realization of the execution component two main shortcomings of OWL-S have been revealed. Firstly,

[8] `http://www.daml.org/services/owl-s/1.0/ProfileHierarchy.owl`

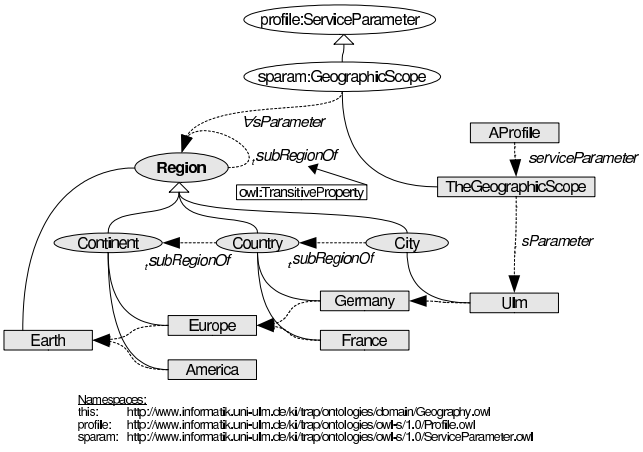

Figure 9: Geographic scope with transitive roles

OWL-S lacks the specification of a model for representing conditions. Secondly, the concept of XSL transformations for mapping data types is not sufficient. These shortcomings will be described in this section in more detail and some workarounds will be presented.

5.3.1 Conditional Model

Actually, it should be straight forward to derive and assert conditions in description logics. Conditions could simply be seen as assertions that exist in the knowledge base or not. However, the main problem is that conditions often refer to concrete parameter instances which are not known before execution. Thus, these instances must be referred to via variables in the definitions of conditions. However, OWL does not support the concept of variables. The only solution is to extend OWL DL by reification of additional concepts in a similar way as it is done by OWL-S to define the data flow in a process model. As a consequence, special algorithms are required to verify such definitions and to derive knowledge from them. Subsumption is not sufficient anymore. This raises the question of OWL being a suitable formalism at all for representing procedural knowledge as intended by OWL-S. Moreover, when deriving conditions from an ABox implicitly a closed world is assumed. This can potentially lead to wrong conclusions. Without changing the ABox, the so called *Local Closed World Assumption* can be used as workaround as described in [8]. Furthermore, the assertion of negated conditions requires retraction of knowledge from the ABox. Once more, the agent does not know which concrete instances must be considered due to the missing concept of variables in OWL. Another problem occurs with the definition of negated conditions. The class-complement operator ¬ obviously cannot be used due to an inappropriate semantics. Thus, a reified concept must be introduced therefore, too.

To be able to realize execution anyhow, a simple conditional model has been introduced as shown in figure 6. Conditions are defined as representative instances of Condition. The property derivedFrom refers to the class description that subsumes the assertions representing the condition. The attribute mustEvaluateTo is used to define a negated condition. E.g. set to false, the condition evaluates to true, if no instances were found, i.e. the condition could not be met.

To evaluate the conditions a hybrid algorithm is used that first checks with an instance query, if the condition can be derived from the ABox. After this a potential negation will be considered.

5.3.2 Datatype Mapping

This section presents a semantically based approach using so called RDF mappings. XSLTs are not suitable to be used for transforming data types as intended by OWL-S. XSLT uses pattern matching rules to identify syntactic structures in an XML document which are then replaced by others. This poses no problem while the same serialization is processed. Normally, this is the case for *output* mappings, i.e. mapping from XML schema definitions to OWL instances. Changes in the serialization must be considered explicitly in the stylesheet. In the worst case, every serialization needs its own stylesheet. This problem occurs with *input* mappings, i.e. mapping from OWL to XML Schema. The same OWL model can have many different serializations. E.g. the fact that relations between instances can be defined nested or with references leads to an exponentially growing number of serializations.

To overcome this problem a transformation must be based on semantics, i.e. not depending on syntactic structures. Our approach of the so called RDF mappings uses a simple RDFS ontology for representing semantic relations as shown in figure 10.

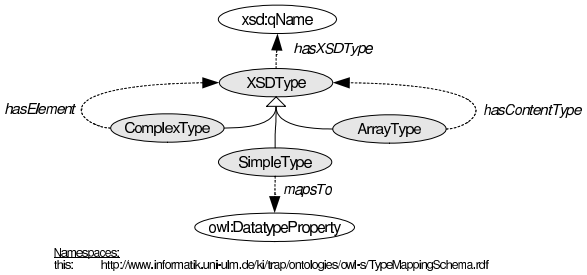

Figure 10: RDFS type mapping ontology

RDF mappings are defined with the same nested structure as the XML Schema types they refer to. The RDF property mapsTo maps a SimpleType to an OWL attribute which belongs to the OWL type of the WsdlMessageMap the RDF mapping is bound to. Thus, RDF Mappings exploit the fact that every instance of an xsd:SimpleType corresponds to a filler of an owl:DatatypeProperty. This also clarifies the use of RDFS. In OWL properties cannot be property fillers themselves. Figure 11 shows an example RDF mapping. It can be seen, that even attributes can be referred which are indirectly bound to the OWL type. Transformations based on RDF mappings require four sources of information. WSDL documents contain the XML Schema type definitions. OWL ontologies define the OWL types. RDF mappings themselves link the two type definitions on a semantic basis. Groundings connect RDF mappings to the corresponding OWL types.

Due to the use of semantics for type mapping, only one RDF mapping needs to be defined for an OWL type and its corresponding XML Schema type. The resulting RDF document is small, simple and can even be generated semi-automatically. Unfortunately, the consistency of RDF mappings cannot be verified completely on the basis of RDFS semantics. Thus, they share the same problem with OWL-S that an adequate verification of definitions based on the semantics of the representation formalism is not possible.

6. EVALUATION

This sections recalls the major benefits as well as deficits of OWL-S by discussing those parts of the specification which are most critical with respect to automatic discovery as well as execution of Web Services.

6.1 Semantics

In contrast to other conventional approaches (e.g. WSDL, UDDI, and BPEL4WS) OWL-S has the advantage of providing a formal semantics for describing Web Services. This allows for meaningful and rich Service descriptions which promise to enable automatic discovery and execution of Web Services. Since OWL-S is (supposed to be) layered on top of OWL DL it has to adopt its formal Description Logic foundation. However, this heritage is twofold with respect to the broad objectives of the OWL-S initiative.

An obvious advantage of the OWL DL layering is the fact, that that sound and complete inference services can be used in order to make implicit subsumption relationships between different advertisements or requests explicitly available. More concrete, Service provider as well as requester do not have to explicitly classify their advertisement or request according to given keywords or schemata; instead their descriptions will be classified automatically.

Unfortunately, OWL-S version 1.0 does not comply with the OWL DL language specification – even when not taking into account those obvious syntactical errors which are typos or were caused due to an incorrect translation from the language predecessor DAML-S. However, sound and complete reasoning is an important premise in order to achieve at least a subset of the proposed OWL-S objectives of facilitating the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring. It is therefore required to transfer all OWL-S ontologies from OWL Full to OWL DL. The necessary adaptions and limitations of this mapping have been discussed in section 5.1.

Another problem of Web Service composition is related with the lack of variables within OWL. In order to combine chains of processes of one ore more services, IOs of atomic processes have to be related with each other. Without being able to reference particular parameters for describing the flow of data with help of variables this can not be done in OWL-S. This seems to be an example showing the limits of the declarative foundation of OWL-S, namely OWL.

In addition, the OWL-S specification conflicts with the OWL language semantics in some other ways. A pragmatic assumption of the profile description is that of a closed world. However, on the level of concrete profile descriptions roles are not closed (this holds at least for the examples provided with the specification). In fact this closed world assumption apparently conflicts with the open world assumption of the OWL language definition. Beyond that, parameter types are implicitly assumed to be pairwise disjoint without making this assumption explicit in the given modeling.

The validation of the different parts of the OWL-S specification also remains as an open issue. As noted above, dy-

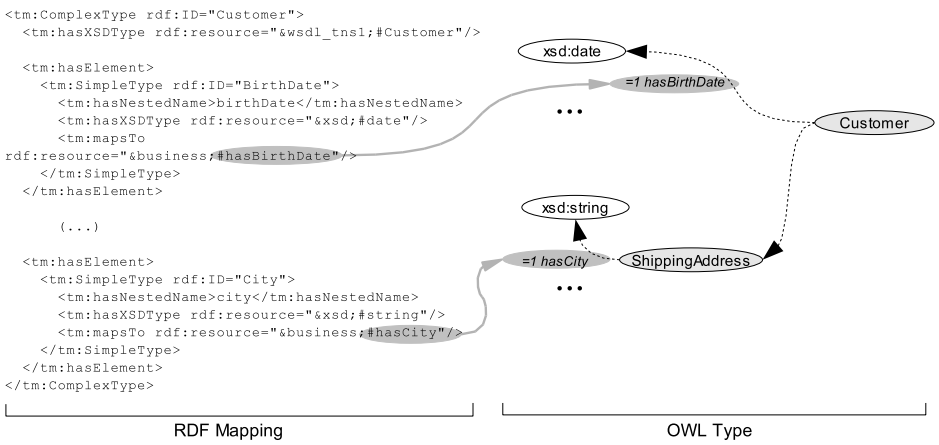

Figure 11: Example of an RDF-mapping

namic process models with cycles and conditions are outside the scope of a representation formalism like OWL. Validation concerning deadlocks or applicability are therfore not possible within this framework.

6.2 Conceptual Model

A goal of the OWL-S specification is to provide a broad conceptual model in order not to narrow its applicability. However, when implementing a concrete scenario this model showed up to be not sufficient in terms of a straight forward realization (e.g. see [13]). Matchmaking strongly depends on the concrete way modeling the service domain. Here, a more detailed and concrete framework would definitely simplify and encourage the usage of OWL-S.

The partitioning of Web Services into profiles, process models and grounding seems to be very valuable with respect to the different phases of finding and execution of Services. The disjunction between functional and non-functional parameters within profiles is consistent with the principles known from software techniques. However, an exception handling within process execution hasn't been taken into account.

A clear conceptual shortcoming is that of using XSL transformations for mapping data structures from XML to OWL and vice versa. A pure syntactical conversion is not feasible and needs to be replaced by a more sophisticated mapping like the one presented in section 5.3.2.

Beyond that, automatic composition of Web Services currently seems to be not practicable with respect to variety of constructs provided by OWL-S. The resulting process models would require planing algorithms for non-deterministic, conditional, hierarchical task networks with loops.

6.3 Practical Applicability

The lack of adequate tools has been one of the major obstacles for a fast and easy implementation of our prototype. This concerns both fundamental tools for the Semantic Web and specific tools to create OWL-S descriptions.

Standard ontology editors like OilEd and Protégé lack the ability of generating "clean" and "trustable" code. Verification support of modelled ontologies is still basic. Creating the descriptions manually using an XML editor has shown to be much faster for evaluation purposes. In terms of reasoning support RACER and JENA have proven to be valuable to integrate ontological knowledge and inference services into Java. However, only a combination of both systems has covered all required functionality.

A fundamental problem in specifying OWL-S descriptions is the enormous complexity of OWL-S the user is exposed to. Besides OWL Full, OWL DL and the concepts of OWL-S comprehensive knowledge of WSDL, SOAP and XSLT is necessary to be able to create OWL-S descriptions. Professional frameworks for the development of web services like Axis hide a huge part of their complexity by providing development tools. This is not the case with OWL-S. Especially tools which support the modelling and verification of OWL-S descriptions on a high semantic level are missing.

7. SUMMARY AND CONCLUSION

This work has clearly shown the potential of using Semantic Web Services to automate complex tasks. However, during the realization enormous difficulties have been encountered especially concerning the implementation of essential OWL-S concepts. The reason for this is twofold. Firstly, it is obvious that OWL is not suitable for representing processes or procedural knowledge based on formal semantics due to the missing concept of variables. As a consequence, essential elements of process models like parameter bindings and conditions cannot be expressed sufficiently within OWL semantics. This can only be achieved by reifying additional formalisms.

Secondly, OWL-S lacks a holistic conceptual model which could facilitate a rapid and consequent realization of Semantic Web Services. Detailed use cases are needed for guiding users through implementation. A vertical prototype which demonstrates techniques for service discovery and execution for own implementations would be highly desirable.

OWL-S undoubtedly demonstrates its potential concerning the integration of proprietary data on the basis of ontologies. This could be of relevance in the area of *Enterprise Application Integration* within business boundaries an beyond. On the restricted basis of only inputs and outputs it would be conceivable to automate interweaving of web services to a certain degree. However, it is highly questionable, if OWL and especially OWL DL is able to provide the foundations

for further reaching ambitions like condition-based service discovery and execution or composition of web services.

8. RELATED WORK

The *Internet Reasoning Service IRS-II* [5] is a holistic Java framework for publishing, locating, composing and executing semantic web services. It is being developed in the Knowledge Media Institute at the Open University London, UK. IRS-II distinguishes explicitly between *tasks* that must be fulfilled which can be viewed as OWL-S Profiles and concrete *problem-solving-methods (PSMs)* that can be used to solve specific tasks. Web services containing a single method are equivalent to PSMs. A likewise notion of OWL-S process models is not supported. In IRS-II web services can be published easily by a "one-click" procedure. IRS-II supports different service implementation platforms like Java, Lisp and SOAP. The internal service resp. PSM descriptions are generated automatically by tools using the native service implementations. The user merely has to assign the published web service to a specific task that it solves. Form a conceptual point of view IRS-II is not as complex as OWL-S. However, all features are implemented in a prototypical framework which is ready to use. The formal language that IRS-II utilizes internally is called *Operational Conceptual Modelling Language* (OCML) [11]. It supports the specification and operationalization of functions, relations, classes, instances and rules. Therefore, OCML is more suitable for representing procedural knowledge than OWL. To be compatible to a certain degree with OWL-S IRS-II provides import functionality for OWL-S descriptions.

The *Web Service Modelling Framework - WSMF* [6] aims at providing an industry scale framework for semantic web service discovery, execution and composition. It is a joint effort of European research projects on the Semantic Web and Semantic Web Services and consists of three working groups each contributing to one of the development areas concerning the ontology and the fundamental conceptual model (WSMO[9]) the representation language (WSML[10]) and the execution framework (WSMX[11]). WSMF has gained an enormous momentum in the last half year. A first implementation of the execution framework realizing the core features was released recently. WSMO is built on four key concepts: *Ontologies*, *web services*, *goals* and *mediators*. Ontologies are used to formally conceptualize properties and *capabilities* of web services. Web services are selected – similar to IRS-II – based on their capabilities to fulfill certain goals. In WSMO web services only interact via mediators which not only translate data representations but also provide for a seamless integration of process models. WSML consists of four language definitions with different expressive power. *WSML-FOL* is the most expressive language and is equivalent to first order logic. *WSML-DL* and *WSDM-RL* are representations for description logic and horn logic fragments of FOL. The language with the least expressive power but therefore the best computational properties is *WSML-Core*. It combines language elements from WSML-DL and WSML-RL while maintaining decidability. Due to the enormous effort put into the formal and conceptual model of Semantic Web Services in WSMO/WSML as well as the aim to provide a complete execution framework WSMO seems to be a very promising candidate to proof the feasibility of Semantic Web Services in realistic scenarios.

[9]Web Service Modeling Ontology `www.wsmo.org`
[10]Web Service Modeling Language `www.wsmo.org/wsml`
[11]Web Service Execution Environment `www.wsmo.org/wsmx`

9. REFERENCES

[1] A. Ankolekar. OWL-S: Semantic Markup for Web Services, 2003. `http://www.daml.org/services/owl-s/1.0/owl-s.pdf`.

[2] W. Balke and M.Wagner. Cooperative Discovery for User-centered Web Service Provisioning. In *Proc. of the First Int. Conference on Web Services (ICWS 2003)*, Las Vegas, USA, 2003.

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL Web Ontology Language Reference, February 2004. W3C Recommendation.

[4] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, Word Wide Web Consortium, 2001. `http://www.w3c.org/TR/wsdl`.

[5] J. D. E. Motta, L.Cabral, and M.Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proc. of the 2nd Int. Semantic Web Conference (ISWC2003)*, Sanibel Island, USA, October 2003.

[6] D. Fensel and C. Bussler. The Web Services Modelling Framework. In *Proc. of the NSF-EU Workshop on Database and Information Systems Research for Semantic Web and Enterprises*, Georgia, USA, April 2002.

[7] V. Haarslev and R. Möller. Description of the Racer System and its Applications. In *Proc. Int. Workshop on Description Logics (DL-2001)*, Stanford, USA, August 2001.

[8] V. Haarslev and R. Möller. Incremental Query Answering for Implementing Document Retrieval Services. In *Proc. Int. Workshop on Description Logics (DL-2003)*, Rome, Italy, September 2003.

[9] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proc. of the Twelfth Int. World Wide Web Conference (WWW 2003)*, pages 331–339. ACM, 2003.

[10] D. Mandell and S. McIllraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In *Proc. of the Second Int. Semantic Web Conference (ISWC 2003)*, Sanibel Island, FL, USA, 2003.

[11] E. Motta, editor. *Reusable Components for Knowledge Modelling*. IOS Press, 1999.

[12] P. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax, February 2004. W3C Recommendation.

[13] M. Sabou, D. Richards, and S. van Splunter. An experience report on using DAML-S. In *Proc. of the Workshop on E-Services and the Semantic Web*, 2003.

[14] UDDI.org. UDDI Technical White Paper, 2000. `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`.