Semantic Web Services: description requirements and current technologies

Rubén Lara
ruben.lara@uibk.ac.at

Holger Lausen
holger.lausen@uibk.ac.at

Sinuhé Arroyo
sinuhe.arroyo@uibk.ac.at

Jos de Bruijn
jos.de-bruijn@uibk.ac.at

Dieter Fensel
dieter.fensel@uibk.ac.at

Universität Innsbruck
http://deri.semanticweb.org/
Technikerstrasse, 13
6020, Innsbruck, Austria

ABSTRACT

Semantic Web Services aim at providing a new level of functionality on top of the current Web and current services, by enabling automatic discovery, composition, invocation and interoperation of Web Services. Different efforts are addressing some of the requirements to enable such next generation services, with different degree of success. Nevertheless, to achieve the main goals addressed by Semantic Web Services, an appropriate semantic description, supporting automation of discovery, composition, invocation and interoperation, must be defined. In this paper, a set of requirements on the information a Semantic Web Service must expose in order to fulfill these major objectives is presented. These requirements are related to the different initiatives in the area, and proposals for useful extensions and combinations of these efforts are discussed.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *web-based services, commercial services.*

General Terms

Standardization, Languages.

Keywords

Web services, semantics, semantic web, semantic web services, semantic web services description, service ontology, WSMF, DAML-S, BPEL4WS, BPML, WSCI.

1. INTRODUCTION

Web services extend the Web from a distributed source of information to a distributed source of service. Semantic Web has added machine-interpretable information to Web content in order to provide intelligent access to heterogeneous and distributed information. In a similar way, Semantic Web concepts are used to define intelligent web services, i.e., services supporting automatic discovery, composition, invocation and interoperation. This joint application of Semantic Web concepts and web services in order to realize intelligent web services is usually referred as Semantic Web Services.

Due to the huge potential impact of semantic web services (SWSs for short) in areas like enterprise application integration and electronic commerce, several efforts, both academic and industrial, have jumped into the arena with the purpose of bringing semantic web services to its full potential. These initiatives are addressing different aspects of the requirements needed to realize semantic web services. They are sometimes complementary, but conflicts between the different approaches also appear. These efforts try to improve current web service technology around SOAP, WSDL and UDDI, which provides very limited support for real automation of services.

One of these initiatives is the Web Services Modeling Framework (WSMF), which aims at providing an appropriate conceptual model for developing and describing services and their composition, based on the principles of maximal decoupling and scalable mediation [8].

Another running project is DAML-S, a DARPA effort to describe an ontology of web services with the objective of making web services computer-interpretable and hence enabling discovery, invocation, interoperation, composition, verification and execution monitoring of services [5].

BPEL4WS [1] and BPML [4]/WSCI [20] have similar functionalities, both aiming at defining a language to describe process models, as well as public process interfaces and service

choreography support, to provide conversational and interoperation means for web services.

Regarding W3C activities in the area, its initiative to define a set of requirements on service description pays little or no attention to semantic support, hence offering a weak basis to make automation of functionality possible [21].

Among the presented approaches, WSMF is the one with the widest scope, as it describes a full-fledged framework with the purpose of making the use of semantic web services a reality. Nevertheless, a concrete realization of the conceptual requirements it presents is still under development in the context of the EU-funded project SWWS[1]. DAML-S focuses on providing semantics to web services descriptions, although some caveats, limitations and lacks have been identified within the proposed ontology. The initiatives focusing on the modeling of business processes, BPEL4WS and BPML/WSCI, do not incorporate any semantics to their modeling primitives, neither for private nor for public processes, thus providing limited support for dynamic discovery, composition and invocation [18].

Whatever the approach and intended purpose, every initiative relies on a specific way to describe web services. Discovery, composition, invocation and interoperation strongly depend on how services are described and exposed for subsequent use. The way a service is described determines to what extent other constructs can provide automation support.

WSMF proposes a service description framework which fulfills the requirements for semantic web services. Nevertheless, it needs some refinements and a specific grounding of the proposed description concepts. Therefore, the approach presented in this paper takes this framework as a starting point to determine the requirements for a meaningful service description.

In this paper, and taking WSMF as a basis, we present a set of requirements for web services description and present grounding guidelines considering the features provided by current efforts. The paper is structured as follows. In section 2, capabilities are presented as the central description support for discovery and composition. Section 3 presents description needs for interoperation of services. Section 4 addresses the concrete grounding of services for invocation. In Section 5, other issues such as service compensation are discussed. Finally, section 6 presents conclusions and future work.

2. Capabilities and description requirements for discovery and composition

Automatic discovery and composition of services are probably the biggest challenges Semantic Web Services are facing. Finding a suitable way to put these two features together has become one of the key points to convert the Web in a distributed source of computation, as they enable the location and combination of distributed services to perform a required functionality.

Automatic Web service discovery involves automatically locating Web Services that provide a particular functionality and that adhere to requested properties [11]. To provide such an automatic location, the discovery process should be based on the semantic match between a declarative description of the service being sought, and a description of the service being offered. This problem requires not only an algorithm to match these descriptions, but also a language to declaratively express the capabilities of services [12].

Furthermore, composition of Web Services requires something more than representing combinations of services where flow and bindings to the services are known a priori. It also must allow the combination of services to provide a given functionality when a request can not be fulfilled by using available services individually [15].

Discovering and composing services needs an approach based on semantic descriptions, as the requester required functionally has to be expressed in a high-level and abstract way to enable reasoning procedures.

Composition and discovery work together in different ways. The following examples depict different scenarios where location and combination of services take place:

- A user wants to book a flight from Innsbruck to Madrid, for next Wednesday and with a fixed maximum price. With this information, discovery will look for a service accepting origin, destination, date and maximum price and providing a seat for an appropriate flight. In the case where such a service is not available, but only a service to look for flight information given trip data and another service to book a flight given flight information can be found, combination must be performed. By combining these two services, the requested functionality can be provided.

- A travel agency models its business process to provide a “make a trip” service. The agency explicitly models control and data flow between the different services involved (book a flight, book a hotel, book a car…). In this case, composition is explicitly modeled at provider side. But in the situation where the agency does not want to limit the book flight service to a given company, a high-level description of the required service must be provided in the process model to enable dynamic discovery (and composition if not a single service can fulfill the requirements) of the best available service to book the flight based on requester criteria.

These two examples illustrate the main roles of composition and discovery to provide a required functionality. Although these use cases can be modified and extended in a number of ways and different examples can be depicted, all of them rely on a high-level description of the functionality being sought.

2.1 Capability description

The required high-level functionality description can be viewed as the capability of the service. Different services can provide the same capability, e.g., book a flight, and the same service can provide different capabilities, e.g., search a book and search a movie.

In this sense, capabilities must be naturally described separately from specific service descriptions [8] for several reasons:

[1] Semantic Web enabled Web Services. http://swws.semanticweb.org/

- *Express generic functionalities*: Several services offering the same functionality but with different specific refinements should be related to the same generic high-level capability. Refinements can then be specified by different services. This approach is related to concepts taken from Problem Solving Methods research, and it inherits some of their advantages, as the ones highlighted in [7] and [9].

- *Use different terminologies*: Refinements done by a given web service can be expressed using a different terminology from the one used to describe their capability, thus increasing flexibility, as requiring the use of the same terminology is sometimes unrealistic.

- *Allow a given service to present two different capabilities* while exposing only one service description.

- *Support discovery process*: Discovery first needs capability descriptions. Refinement based on actual input, output and requirements is performed in subsequent steps. Thus, separating capabilities and referring service refinements to them establishes a natural link to the discovery process.

Declarative means to define capabilities, as well as specific service refinements and the link between refinements and capabilities are needed. This description must allow reasoning about the information presented by the service. Several works in the area use subsumption reasoning, i.e., determining whether a given concept is more general than another, to support discovery and composition, either looking for just one service providing the required functionality [12] or composing different services [2], like in the travel agency example exposed before.

To support dynamic discovery and composition, a capability must include the following information:

- *Pre-conditions*: High-level inputs to the service together with conditions over these inputs. These inputs are concepts of a given domain ontology. Each pre-condition will include an identifier to allow future references. High-level input means that more specific concepts in the ontology can be found, e.g. indicating payment information as a pre-condition, instead of credit card information or bank information or even data types. If a too specific concept is given as a pre-condition, then the capability will hardly express generic functionalities. It is important to notice that the pre-conditions of a given capability are not independent of each other, as they all define the functionality expressed in the capability.

- *Post-conditions*: High-level results of the service execution together with conditions over these results. The results are also concepts of a given domain ontology. Identifiers are also defined for post-conditions, and as with pre-condition, they cannot be considered independent, as the removal of one of them changes the functionality expressed by the capability.

- *Textual description*: To allow human interpretation.

- *Services*: References to the services presenting the described capability.

- *Identifier*: Identifier to allow references to the capability.

Pre and post-conditions define the capability of the service in terms of the information needed to use the service and the results of its invocation. Describing capabilities by expressing their functionalities in terms of required high-level input and high-level results covers the following requisites:

- Modeling a process at design time. In this case, the workflow and data flow is defined a priori, at least partially. Thus, the declaration of the use of a capability must enable the specification at design time of the input and the result of the service. This information is needed to model data and control flow. Nevertheless, this information must be kept general enough to describe generic service functionalities, allowing dynamic location and combination of services. For example, in a business process using a service to buy some goods and another service to ship them, the result of the buy_goods service must be used by the ship_goods service, and the required data and control flow must be designed. Other approaches like relating the capability to a task ontology describing possible requested functionalities cannot be used in this context, as they don't provide enough information to define flow at design time.

- Dynamic discovery: Subsumption algorithms, as the ones presented in [12] and [2] are supported by relating pre and post-conditions to the appropriate domain ontology and by using the specific services refinements, presented in the next subsection.

- Dynamic composition: Combination of services to fulfill a given functionality can be performed by expressing functionality in terms of pre and post-conditions, as described in [2].

- n to m mappings: Describing a capability using pre and post-conditions and not including low-level input and output information, enables n to m mappings between capabilities and services, thus allowing the description of generic functionalities and the declaration of different generic functionalities by the same service. Lower level inputs and outputs must not be included in the capability description as this would prevent these features and would imply several modeling problems, as explained in [14].

Textual information is used to let the human user browse capabilities. Capabilities must be understandable by humans and machines [13], as a process designer may need to search suitable capabilities to include in a process model at design time.

References to the services presenting the capability are specified to enable the location of refinements of the described generic functionality, i.e., the location of specific services during discovery and composition process.

2.2 Capability refinements

Once a capability is described, different services presenting this capability can refine it. In this way, specific requirements, constraints and results of an individual service can be expressed. To avoid redundancy, the service will only explicitly describe the refinements it introduces, not repeating the information already enclosed in the capability description. Figure 1 depicts the relationship between capabilities and services:

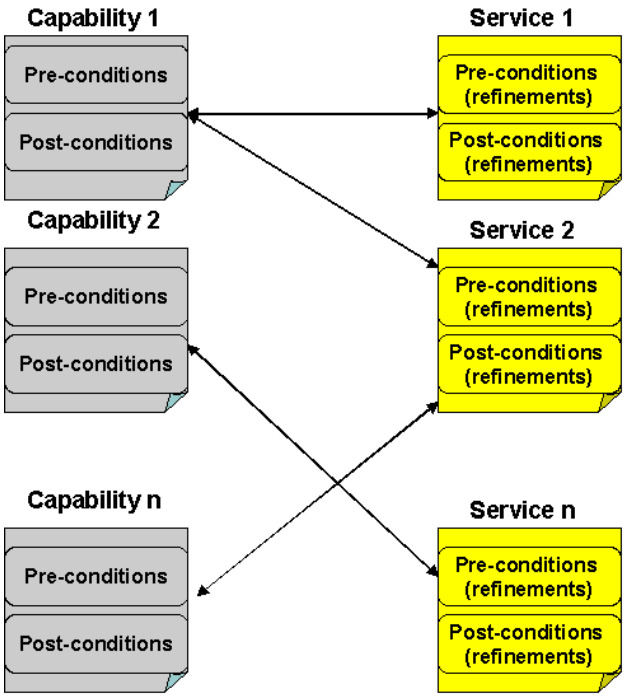

Figure 1. Relationship between capabilities and services

By defining the service refinements, a complete high-level description of the input required by the service and the result of its execution is given. Nevertheless, actual input and output data is needed to express the low-level details of service functionality. These inputs and outputs are naturally related to pre and post-conditions, as they constitute the realization in terms of data of high-level conditions.

Therefore, the information to be exposed by an individual service to refine generic functionalities is the following:

- *Identifier*: information for service references.

- *Textual description*: human-understandable information.

- *Capability references*: references to the capability or capabilities presented by the service.

- *Pre-condition refinements*: pre-conditions refining the ones presented by the capability. If it is a refinement of one or more pre-conditions defined in the capability, a reference to them will be included. This is the case of a service referring to a capability requiring general payment information as pre-condition, while the service accepts only information about credit card. Also new pre-conditions are allowed, and identifiers for every new pre-condition or refinement must be included. In general, pre-condition refinements reflect the strengthening of pre-conditions.

- *Inputs*: actual input data information. Inputs are grouped and referred to the pre-condition the group realizes, either from the generic capability or from the service refinements. To allow polymorphism, different sets of inputs can be defined for the same pre-condition. For example, in the case of a service accepting different ways of payment (credit card, bank transfer…), different input data is required depending on how the requester wants to pay. However, it is natural to define only one interface for the service. Therefore, this service will have a pre-condition "payment information", with different sets of inputs associated to it for credit card payment, bank transfer payment, etc. Which specific set of inputs is used will be decided at run-time.

- *Post-condition refinements*: refinements or new post-conditions. They are defined following the same mechanism as pre-conditions. Refinements of existing post-conditions reflect the weakening of the capability post-conditions, whereas adding new ones reflects the strengthening of the capability post-conditions

- *Outputs*: actual output data, described in the same way as inputs.

In figure 2 the refinement of pre-conditions and how the inputs realize them is depicted:

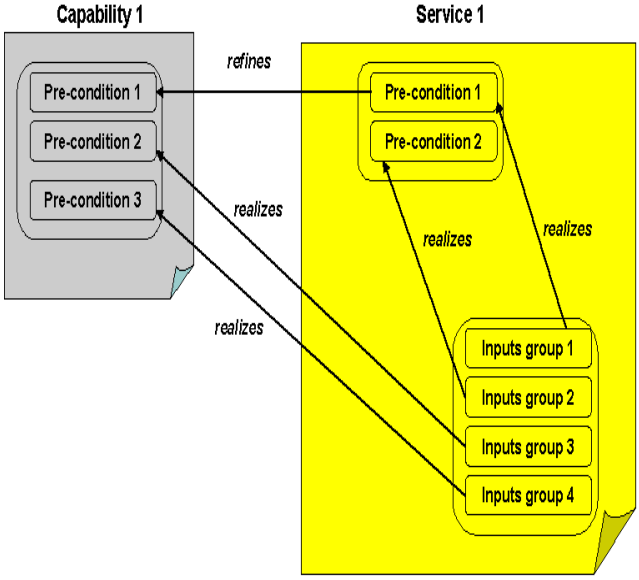

Figure 2. Pre-conditions refinements and pre-conditions realization

In the figure above, service pre-condition 1 refines the first pre-condition defined in the capability. Inputs group 1 gives actual data for the refined pre-condition. Service pre-condition 2 is a new pre-condition, not existing in the capability exposed, and realized by inputs group 2. Capability pre-conditions 2 and 3 are not modified, so inputs groups 3 and 4 directly refer to these pre-conditions. As explained before, the refinement of more than one pre-conditions together is also possible.

By defining capabilities, refinements and actual input and output data using appropriate ontologies, service description exposes enough information to enable automatic discovery and composition. Refinements and information about input and output data can also be used to locate a required service, as well as for composition, thus providing appropriate expressivity power for the requester. Therefore, the requester is not limited to locate and combine generic capabilities, but he can also express more detailed needs.

2.3 Relation to current technologies

Although the requirements presented above for describing service functionalities reflect the ideas contained in the WSMF approach, they extend and refine the description means outlined in the framework.

Once these extensions and refinements are defined, they must be expressed using an appropriate ontology to provide semantics to the information exposed by the service. In this sense, neither BPEL4WS nor BPML/WSCI include any suitable mechanism, as they do not use similar concepts to capabilities or refinements and they do not add any semantic information.

Therefore, DAML-S is the only potentially reusable work to define the desired ontology. The existing ontology of services, currently at version 0.9, includes profiles and service models, which purposes are similar to the ones of capabilities and refinements respectively. However, the DAML-S ontology presents serious limitations if left as it is. First, input and output is included in the profile, preventing polymorphism and n to m mappings between profiles and specific service models. Second, pre and post-conditions (pre-conditions and effects in DAML-S terminology) of the concrete service model are not related to the ones presented at the profile. Third, inputs and outputs are not related to pre-conditions and effects. Fourth, using different sets of inputs and outputs for a given pre or post-condition is not allowed. All these limitations imply service modeling problems, as analyzed in [14].

In conclusion, the DAML-S ontology can be used as a basis to define semantics for service functionality descriptions, but it must be considerably changed and extended to present the properties and requirements presented in this section.

3. Interoperation and current technologies

One of the main purposes of web services is the automation of application integration within and across organizational boundaries. This implies necessarily the need for interoperation between services. This interoperation can be between services in an organization or crossing different organizational boundaries. To ensure automatic interoperation, description means must be defined declaratively using explicit semantics.

Business collaborations require long-running interactions driven by an explicit process model [17]. Thus, a service must explicitly model its business process which will contain decision mechanisms for the execution of the service. But following one of the main principles of WSMF, no internal details of the organization business logics should be made publicly visible. Therefore, while a process model and its data and control flow must be designed explicitly to ground the execution and public behavior of a given service, it must not be exposed.

Nevertheless, the external behavior of the service in terms of message interchange must be made public in order to enable automatic interoperation of the service with any other service. In this sense, the public description of a service must include a conversational interface which allows interoperation while not revealing any private detail.

Figure 3 illustrates the relationship between the private process model and the public process model in terms of visibility. Private process model grounds the public model and drive its actual behavior, but only the public process model is made public.

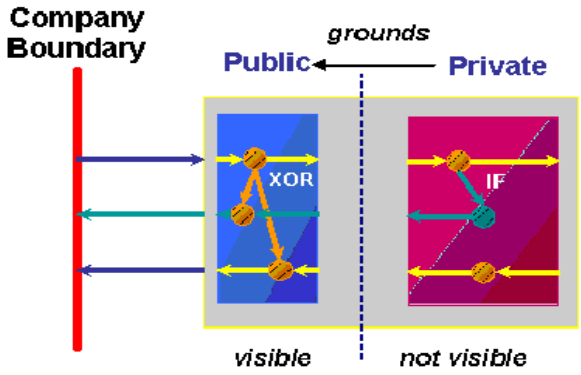

Figure 3. Public and private process models visibility[2]

Although the main purpose of this paper is to define public description requirements for semantic web services, private process modelling deserves some analysis given its close relationship with public process models.

Concerning private process models, both BPEL4WS and BPML offer a rich set of primitives to model the workflow of the service, supporting composite processes based on web services. In [19] and [16] a pattern based analysis of both languages can be found. This work analyzes BPEL4WS and BPML/WSCI using a set of workflow and communication patterns to clarify if they provide sufficient modeling primitives for any possible abstract situation. The result is similar for both, as they support most of the patterns described, either directly or using workarounds.

Both approaches clearly separate private and public process models. BPEL4WS introduces the concept of executable process for private processes and abstract process for public processes. Similarly, BPML is used to model private processes while WSCI is concerned with the choreography and public interoperation of services. However, as stated before, both languages lack semantics to expose its public interface as well as the possibility of expressing the use of a service within the private process model in terms of the capability it presents.

DAML-S must be analyzed for this purpose, as it is the only initiative including explicit semantics. However, an appropriate service ontology requires a richer set of process modeling primitives, as the concepts described in the DAML-S process model are not powerful enough to support some of the communication and workflow patterns required. And what is more, DAML-S does not distinguish between private and public processes, allowing internal details to be exposed via a composite process. Thus, although DAML-S provides ontological support for service modeling, its limitations prevent his direct use to publish interoperation information.

Our proposal is to add semantics to either BPEL4WS or BPML/WSCI modelling mechanism for conversational interface

[2] From the presentation "Principles of integration: Are Web Services heading in the right direction?", Christoph Bussler, Innsbruck 19.05.2003

and integrate these semantics into the service ontology. That means replacing the process model in DAML-S by a BPEL4WS or BPML/WSCI-based model, including the necessary public information to expose the behaviour of the service in terms of message interchange.

Summarizing, the public description requirements for interoperation, which will be grounded in the way previously discussed, are the following:

- External behavior of the service in terms of message interchange and message sequencing must be described.

- No information about internal business logics should be exposed.

- The public process exposed must be grounded by an appropriate private process, which must allow the use of capabilities and refinements at design time to specify the service to be used and its dynamic location and composition.

Deciding whether BPEL4WS or BPML/WSCI can be used to fulfill these requirements is out of the scope of this paper. In any case, they must be semantically grounded and extended to use generic capabilities and refinements at design time in the way described in the previous section, and included in the service ontology with the necessary extensions.

Also the use of Abstract State Machines (ASMs) [3] to model business processes is being analyzed, as they present several interesting properties, namely: express functionally complete but abstract description that can be understood by a human reader, define every system features as far as it is semantically relevant for the required functionality and contain only what the logic of the problem requires for the system behavior. Furthermore, the grounding model is implemented following a refinement process, trough a hierarchy of intermediate models, and ASMs also allow structuring the system horizontally by building it from components with abstract definitions of behavior and interactions trough interfaces.

Though ASMs properties make them suitable for its use to describe service conversational interfaces and their corresponding groundings, none of current efforts use ASMs. As this paper tries to ground description requisites using existing technologies or extensions and combinations of them, introducing ASMs is out of the scope of this paper, although it will be part of future work.

4. Service invocation

The requirements presented so far deal with the expression of declarative functionality and the publication of conversational interface. This information will support discovery, composition and interoperation, but declarative means to enable invocation of a given service are still missing in the picture.

Invocation information presented by a given service must be agnostic in principle with respect to the specific technologies which will ground it. Nevertheless, details must be available at run-time for the service requester in order to perform a real invocation. These details must relate every aspect of the declared functionality to a ground mechanism, e.g., SOAP on HTTP.

Grounding mechanisms are provided within BPEL4WS, BPML/WSCI and DAML-S, although most of the examples available are focused on WSDL and SOAP grounding.

Considering the need for an effective and platform independent invocation, input and output data, messages and message sequencing must be declaratively related to a specific technology and exposed in the public service description. In this sense, service ontology must include concepts to express this relationship, as the “grounding” concept defined in DAML-S ontology.

Due to the semantic link to the required grounding, DAML-S should naturally be used as a starting point as it already contains a declarative grounding mechanism [6]. Nevertheless, the DAML-S ontology relates grounding directly to a given service, hence not supporting polymorphism and encountering problems while grounding a real service, as the ones highlighted in [14]. As a consequence, extensions to DAML-S grounding mechanism are to be introduced in order to support the grounding of the description means introduced in sections 2 and 3. Different groundings for every set of inputs must be defined, and the use of a concrete grounding should be decided at run-time. Furthermore, conversational interface, not defined in DAML-S ontology, must be related in a similar way to a concrete technology.

However, the basic DAML-S grounding mechanism can be reused and refined make it usable for automatic invocation. After the outlined refinements are performed, services presenting all the properties required in this paper can be grounded using such ontology.

5. Compensation and other requirements

Until now, the optimistic assumption “everything works well” has been implicitly made. No errors were considered, although they appear in computer systems more frequently than desired. Due to this fact, an intelligent service description must take into account possible errors and how to deal with them.

For this purpose, error data must be described in addition to input and output data. A SWS description should include one or more error ports, to provide error information to the requester potentially at different points of execution. These error ports must refer to an appropriate ontology in the same way inputs and outputs do. Error ports can be thought as special types of outputs, so the same requirements apply to them, although they are not referred to any pre or post-condition. Error ports, as well as inputs and outputs, will be used in the same way in the definition of the conversational interface, establishing at which point of execution a concrete input is required, when outputs are delivered to the requester, and where specific error ports may report error information. These ports will also be included in the service grounding information.

In the context of semantic web services, dynamic location and combination of services implies that no a priori assumptions can be made about the duration of a service invocation. For this reason, the use of traditional ACID transactions [10] to deal with errors is not useful in this context, as they require blocking resources for an undefined amount of time. Therefore, the concept of compensation has appeared to substitute classic transactions. Compensating a service means to invoke one or more services to undo the actions of the former one. For instance, a service to book

a flight can be compensated by invoking a service to cancel a flight.

Compensation is based on the assumption of the existence of a reverse service for the invoked one. If this assumption does not hold, the effect of the service can not be compensated.

Due to the need for a service reversing the action of the service invoked, the location of such service plays an essential role in compensation. To locate this service, three different situations can be thought: 1. the invoked service explicitly points to a service compensating its action ("book a flight" service from an airline points to its cancellation service), 2. the invoked service describes the capability of the service needed to compensate its invocation to make its location (and possible composition) possible (although several services may be able to perform the compensation) or 3. the invoked service does not give any information about the compensation service or the capability it should present. In the latter case, reasoning mechanisms must be developed to figure out the required capability and constraints to compensate the service. The first two cases can include information via the described capability (second case) or the pointed service(s) capabilities (first case) about any compensation constraint, e.g., economic penalization when canceling a flight ticket.

Therefore, a SWS description must allow the inclusion of compensation information, although it can not be compulsory, as a given service may not want to define information about compensation or such compensation may not be viable. This information will be defined in terms of the service compensating its action or necessary capabilities. Different degrees of complexity, as the ones discussed before, can be modeled. In this way, expressing explicit compensation has enough flexibility to contemplate every case and can be thought as modeling a compensation service at different possible levels of detail (just one service, just one capability, or reasoning mechanisms to figure out the required capability).

DAML-S does not include any error or compensation information, so the ontology must be extended again to reflect error and compensation data. BPEL4WS and BPML/WSCI do contemplate this information, so they must be taken into account while changing the ontology to include the discussed data.

Security and reputation are also required, although due to the complexity of the mechanisms they require, this discussion is considered out of the scope of this paper and part of future work.

Information such as quality of service, geographical area, time of response, and other non-functional information must be reflected in the service description. The list of non-functional properties of the service must be extendable, but DAML-S non-functional properties can be used at a first moment, as they can be extended and changed easily. The elaboration of a complete set of non-functional properties will be accomplished in the future.

Other requirements on service description are the inclusion of contact information for the service, i.e., the organization providing the service, and the classification of the service according to a given taxonomy (the taxonomy must be referenced). DAML-S already includes this information, so it can be used as it is.

All these properties must be related to the concrete service description, and not to the capability, as they are dependent on concrete providers.

6. Conclusions and future work

Semantic description is the base to realize automation of web services. Although WSMF is the conceptually more complete approach to enrich services with semantics, it needs a refinement and grounding work as the one presented. In addition, none of the other approaches is complete enough to make next generation services a reality. The different description requirements presented serve as a basis to extend and refine current efforts and to come out with an appropriate ontology for services. Considering the state of the art, an advisable approach is to take WSMF concepts and DAML-S ontology as a basis. This ontology can be refined based on the reviewed WSMF concepts presented and the work already done for BPEL4WS and BPML/WSCI.

The need for semantically enriched web services has become clear, and therefore a suitable ontology must be developed. In the context of the SWWS and centoYo[3] projects such ontology will be defined, taking into account the requirements stated in this paper.

Future work contemplates the definition of a complete set of non-functional properties, the analysis of security and reputation needs, and the decision about the use of BPEL4WS and BPML/WSCI to extend DAML-S ontology. The use of DAML-S or OWL-S will also be analyzed, although the decision will be based on which ontology language (DAML+OIL or OWL) is more suitable, as it does not affect to the definition of the service ontology. The service ontology defined will be considered to be applied to different use cases in both SWWS and centoYo projects. The possible application of Abstract State Machines to describe service conversational interfaces will also be analyzed in the future. In addition, as all the work presented relies on a strong ontology support and expecting all service requesters and providers to use the same ontology is unrealistic, mediation between different ontologies is needed and will be part of the future work.

7. ACKNOWLEDGMENTS

The research presented in this paper was partially funded by European Commission in the context of the SWWS project (http://swws.semanticweb.org) under contract number IST-2001-37134.

Our thanks to the centoYo team for providing the necessary discussion for this work.

8. REFERENCES

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana. Business Process Execution Language for Web Services, version 1.1, available at ftp://www6.software.ibm.com/software/developer/library/ws-bpel11.pdf, 2003.

[2] B. Benatallah, M. Hacid, C. Rey, F. Toumani: Semantic reasoning for Web Services Discovery, WWW 2003 workshop on E-services and the Semantic Web (ESSW'03), Budapest, Hungary, 2003.

[3] http://informatik.uibk.ac.at/infweb/projects/centoYo/

[3] E. Börger: High Level System Design and Analysis using Abstract State Machines, Current Trends in Applied Formal Methods (FM-Trends 98). Springer LNCS 1641, 1999, pp 1-43, 1999.

[4] Business Process Modeling Language (BPML). Accessed june 2003 from www.bpmi.org.

[5] The DAML services coalition: DAML-S: Semantic Markup for Web Services (version 0.9), available at http://www.daml.org/services/daml-s/0.9/daml-s.pdf, 2003.

[6] The DAML Services Coalition: Describing Web Services using DAML-S and WSDL. DAML-S Coalition working document, May 2003. http://www.daml.org/services/daml-s/0.9/daml-s-wsdl.html

[7] D. Fensel, E. Motta: Structured development of problem solving methods, IEEE Transactions on Knowledge and Data Engineering, 13(6):9131-932, 2001.

[8] D. Fensel, C. Bussler: The Web Service Modeling Framework WSMF, Electronic Commerce Research and Applications, 1(2), 2002.

[9] D. Fensel, E. Motta, V.R. Benjamins, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, B. Wielinga: The Unified Problem-solving Method Development Language UPML, Knowledge and Information Systems (KAIS): An international journal, 5(1), 2003.

[10] J. Gray, A. Reuter: Transaction processing: Concepts and Techniques. Morgan Kaufmann Publishers, San Mateo, California, 1993.

[11] S. McIlraith, T.C. Son, H. Zeng: Semantic Web Services. IEEE Intelligent Systems, Special Issue on the Semantic Web, 16(2): 46-53, March/April 2001.

[12] M. Paloucci, T. Kawamura, T. R. Payne, K. Sycara: Semantic Matching of Web Services Capabilities. In Int. Semantic Web Conference, Sardinia, Italy, pages 333-347, June 2002.

[13] T. Pilioural, A. Tsalgatidoul, A. Batsakis: Using WSDL/UDDI and DAML-S in Web Service Discovery, WWW 2003 workshop on E-services and the Semantic Web (ESSW'03), Budapest, Hungary, 2003.

[14] M. Sabou, D. Richards, S. Splunter: An experience report on using DAML-S, WWW 2003 workshop on E-services and the Semantic Web (ESSW'03), Budapest, Hungary, 2003.

[15] E. Sirin, J. Hendler, B. Parsia: Semi-automatic composition of Web Services using Semantic Descriptions, to appear in "Web Services: Modeling Architecture and Infrastructre" workshop in conjunction with ICEIS2003, 2002.

[16] W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, P. Wohed. Pattern based analysis of BPML (and WSCI), QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.

[17] W.M.P. van der Aalst. Don't go with the flow: Web services composition standards exposed. IEEE Intelligent Systems, Jan/Feb 2003. Electronically accessible from http://www.tm.tue.nl/it/research/patterns/ieeewebflow.pdf, 2003.

[18] Web Service Composer Project, Maryland Information and Network Dynamics Lab, University of Maryland, USA; http://www.mindswap.org/~evren/composer/

[19] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede. Pattern based analysis of BPEL4WS. QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

[20] W3C. Web Service Choreography Interface (WSCI) 1.0. Accessed June 2003 from www.w3.org/TR/wsci

[21] Web Service description requirements, W3C working draft 28 October 2002. http://www.w3.org/TR/ws-desc-reqs/