Using Assumptions in Service Composition Context

Zheng Lu
School of IT & Computer Science
University of Wollongong, Australia
zl07@uow.edu.au

Peter Hyland
School of IT & Computer Science
University of Wollongong, Australia
phyland@uow.edu.au

Aditya K. Ghose
School of IT & Computer Science
University of Wollongong, Australia
aditya@uow.edu.au

Ying Guan
School of IT & Computer Science
University of Wollongong, Australia
yg32@uow.edu.au

ABSTRACT

Service composition aims to provide an efficient and accurate model of a service, based on which the global service oriented architecture (SOA) can be realized, allowing value added services to be generated on the fly. Unlike a traditional software module, which runs within a predictable domain, Web Services are autonomous software agents running in a heterogeneous execution environment. Because of distributed responsibilities, ownership and control, it is often not feasible to acquire all information needed for the service composition. Thus, there is no guarantee that the service execution has the anticipated effects. Full automation of this process poses challenges to reliable service composition by raising questions such as how to deal with incomplete knowledge during the dynamic service composition, and how to ensure consistent service execution result without human intervention.

Categories and Subject Descriptors

I.2 [**ARTIFICIAL INTELLIGENCE**]: Miscellaneous

General Terms

Design, Management, Reliability

Keywords

Web Service, Service Composition, Assumption

1. INTRODUCTION

OWL-S [10], formerly known as DAML-S, is an upper ontology for services, aimed at achieving the automation of service discovery, invocation, composition and interoperation. OWL-S leverages the rich expressive power of OWL [3] together with its well-defined semantics to provide richer descriptions of Web Services. Service ontologies can be used to map service functional descriptions and domain properties into a standardized logic so that they can be machine understandable and interpretable. Recently, semantic web rules language (SWRL) [4, 17] has been proposed to define service process preconditions and effects, process control conditions and their contingent relationships in OWL-S. Though OWL-S is endowed with more expressive power and reasoning options when combined with SWRL, the description provided by a combination of OWL-S and SWRL about service composition is still only a partial picture of the real world. Most of what we know about the world, when formalized, will yield an incomplete theory precisely because we cannot know everything - there are gaps in our knowledge [15]. Similarly, the ontology of services, is finite and incomplete. Thus, a service composition specified by OWL-S has to deal with partial or incomplete knowledge.

In this paper, we are going to bridge the gap between the semantic service description and multiple operational domains involved by introducing the representation of "service assumptions". Currently, OWL-S has no mechanism for handling the explicit description of service assumptions and no method for reasoning about their side-effects. We will extend the current OWL-S and try to define a formal mechanism for reasoning about incomplete knowledge in dynamic service composition context.

The paper is structured as follows: in Section 2 we explain an atomic service and composite service in general. In Section 3, we define the basic semantics for the planning-based service composition domain. In Section 4, we present a framework for reasoning about incomplete knowledge in the service composition context. In Section 5, we use a real world scenario to illustrate our framework. In Section 6, we extend current OWL-S to provide a richer service description schema by introducing service assumptions. We also provide an example of using assumptions in service composition. Then in Section 7, we summarize the reasoning process in service composition. Finally, we present related work and our conclusions and discuss future research in Section 8 and Section 9 respectively.

IW-SOSE'06, May 27–28, 2006, Shanghai, China.

2. ATOMIC SERVICE AND SERVICE SELECTION

2.1 Atomic Service

An atomic service only performs a single function, each atomic service ws is described by a tuple $\langle p, e, a\rangle$, where

- p is a set of sentences representing the preconditions that must be true for the atomic service to execute.
- e is a set of sentences representing the change of world state, which include both positive and negative effects.
- a is a set of service assumptions.

Note that p and a are different. It must be able to establish that p is true for ws to be invoked. On the other hand, we only need to establish that a is consistent with what is known i.e. nothing is known that contradicts a. p is a strong condition which must be true in order to execute the service ws, while a is a weak condition. Initially we assume a to be true, unless we get additional information which is explicitly contradictory to a.

2.2 Service Selection

Compared to conventional software module compositions, the automated process of Web Service compositions holds some additional critical issues, such as service matching, selection and retrieval. UDDI [6] provides a mechanism for a Web-wide service registry, in which descriptions of Web Service in UDDI are stored and searched by Category. OWL-S allows us to semantically describe the capabilities of Web Services, thus it is possible to perform logical inference for service matching. [8] Provides one way to combine these two efforts, registering Web Services defined in OWL-S with UDDI and allowing UDDI engines to exploit OWL-S semantic information to facilitate the retrieval of Web Services. In this proposed framework:

- ws_i represents an atomic service.
- WS is the set of all Web Services, $ws_i \in WS$.
- All Web Service descriptions are held in their corresponding categories $\{cat_1, cat_2, \ldots, cat_n\}$. cat_i is a tangible area split from the service registry, for example downloadable Multimedia.
- CAT is the set of all service categories $cat_i \in CAT$, $cat_i \in WS$, $cat_i = \{ws_1, \ldots, ws_m\}$.
- Service selection function $sel : CAT \rightarrow WS$ which takes a certain service category as its input and give us an atomic service based on the service matching i.e. $sel(cat_i) = ws$.

Every atomic service in the rest of this paper refers to a Web Service which is produced by the service selection function defined above. For more details about the service matching, interested readers may refer to [8, 9].

2.3 Composite Service

Intuitively, a composite Web Service which performs combined functions may include multiple atomic services. A composite service is the combination of the multiple atomic services ws_i, where $0 < i < n$, a composite service $CompWS$ is represented by:

$$CompWS = \{sel(cat_1), \ldots, sel(cat_n)\}$$

Because participants of the service composition do not necessarily share the same objectives and background, without reasoning about incomplete knowledge and its side effects during the service execution, conflicts easily arise in the service composition context.

3. WEB SERVICE COMPOSITION AS PLANNING

It is often assumed that a business process or application is associated with some explicit business goal definition that can guide a planning-based composition tool to select the right service [16]. Typically, classical planners presuppose complete and correct information about the world. However, this simplified assumption is neither suitable and nor realistic in the context of Web Service composition. Each node of service composition is designed, owned, or operated by distinct parties, thus the agent may not have complete information about the world. To reason about incompleteness of information in the service composition context, we extend the current semantic Web Service description in OWL-S by introducing service assumptions. Assumptions, in this framework, together with states, preconditions, effects, and goals are all specified in Description Logic L [2].

Now we are prepared to define the semantics of a service composition domain. A state S is a snapshot which describes the world with respect to the service composition context. The state S in this work is extensionally defined as a set of positive or negative ground atomic formulas (atoms). Those atoms which may change their values during the state transition are called *fluent*, while those which don't change are called *state invariant*. Unlike traditional planning, S here is a partial description of the world. A state transition t is represented as a tuple $t = \langle s, ws, s'\rangle$, where s, s' are states and ws is an atomic service. It is also worthwhile to mention that the initial state s_0 is also a partial description of the world. A goal G is set of conjunctions of atoms which need to hold in a desired world state or say final state. A service composition plan for a goal is a sequence of state transitions of atomic services, and the transitions lead from an initial state to a final state where all ground atomic formulas in the goal are true.

In the process of the state transition, there are three types of knowledge about the current world which represent the state transition. In rest of the paper, we will use symbol $\models$ to represent logical entail. Let SEN_i denote a set of sentences used to change the state S_i. This set of sentences can be partitioned into three categories, namely, state invariant, state expansion and state update. The set of sentences is defined as:

$$SEN_i = \{Inv_i \mid Exp_i \mid Upd_i\}$$

where:

1. State invariant Inv_i denotes a set of sentences which can be entailed by the knowledge in the previous state, defined as:

$$S_{i-1} \models Inv_i$$

2. State expansion Exp_i denotes a set of sentences which cannot be entailed by the knowledge in the previous state and its negation also cannot be entailed by the knowledge in the previous state, defined as:

$$S_{i-1} \nvDash Exp_i$$

and

$$S_{i-1} \nvDash \neg Exp_i$$

3. State update Upd_i denotes a set of sentences whose negation can be entailed by the knowledge in the previous state, defined as:

$$S_{i-1} \models \neg Upd_i$$

In our framework, for any atomic service ws, and where WS is the set of all Web Services, E is the set of all service effects, P is the set of all service preconditions, we define the following extraction functions:

1. Effect extraction function $f_e : WS \rightarrow E$ which takes an arbitrary atomic service ws_i as an input, and extracts the effect e_i of ws_i as its output. e_i is a set of primitive effects of ws_i and every primitive effect is a partition with the state invariant, state expansion and state update i.e.

$$f_e(ws_i) = e_i$$

and

$$e_i = \{eInv_i \mid eExp_i \mid eUpd_i\}$$

in which $eInv_i, eExp_i, eUpd_i$ denote state invariant, state expansion and state update respectively.

2. Precondition extraction function $f_p : WS \rightarrow P$ which takes an arbitrary atomic service ws_i as an input, and extracts the precondition p_i of ws_i as its output i.e. $f_p(ws_i) = p_i$. Similar to the effect extraction function:

$$p_i = \{pInv_i \mid pExp_i \mid pUpd_i\}$$

Following the definitions above, we can define the generic state transition operator as:

$$S_i = \Delta(pUpd_i, eUpd_i, S_{i-1}) + eExp_i + pExp_i$$

which means the state transition from S_{i-1} to S_i is completed by means of applying $pUpd_i$ and $eUpd_i$ to the state S_{i-1} in order, then adding the two types of expansion of knowledge $(eExp_i, pExp_i)$ to the state S_{i-1} . Note that the order of applying state update must be strictly followed.

4. DEFEASIBLE REASONING FOR SERVICE COMPOSITION

Comparing with the traditional software development, a dynamic service composition is an automated process with less human intervention. Usually, it does not have a predefined boundary, based on which the problems of uncertainty and incompleteness of information could be tackled. Unpredictable service executions and a dynamic changing context complicate dynamic service composition in many ways. Inspired by Non-monotonic logic [5], the following subsection will attempt to provide a formal framework for reasoning about incomplete knowledge in service composition context.

4.1 Defeasible Reasoning Framework

In this work, our conflict checking and reasoning about incompleteness of information work with three kinds of rules, namely absolute rules, defeasible rules and defeaters [5]. The absolute rule, which is interpreted in the classical sense, means that whenever the premises are indisputable then so is the conclusion. On the other hand, the defeasible rule is one whose conclusion is normally true when its premises are, but certain conclusions may be defeated in the face of new information. Defeasible rules can be defeated by contrary evidence or by defeaters. Defeaters represent knowledge which is able to prevent defeasible inference from taking place. We use the operator $\Rightarrow$ for absolute rules, $\sim>$ for defeasible rules and $\mapsto$ for defeaters.

So, ws_i represents a Web Service which is produced by the service selection function (see section2.2), a_i represents the assumptions of ws_i, p_i represents the precondition of ws_i, e_i represents the effects of ws_i and $isValid(ws_i)$ represents a Web Service whose preconditions can be satisfied. Based on Nute's defeasible reasoning [5],

- **Absolute rule** (Rule A):

$$p_i \Rightarrow isValid(ws_i)$$

Which means that only precondition holds, and then the service is a valid candidate service to participate in service composition.

- **Defeasible rule** (Rule B):

$$isValid(ws_i) \sim> e_i$$

Which means that normally e_i is the conclusion of a valid candidate service ws_i, but that e_i may be defeated in the face of new information.

- **Defeater** (Rule C):

$$\neg a_i \mapsto \neg e_i$$

Which means that given an assumption a_i, if the negation of the assumption is entailed by a given state of knowledge, it will prevent the Rule B from making the conclusion e_i.

4.2 Outdated Assumptions and Assumption Database

When the negation of a service effect is the logic consequence of a current state, we refer to the assumption associated with this service as the outdated assumption. In other words, if the negation of all sentences in e_i is entailed by some states S_j, where e_i is an effect of Web Service ws_i and $j > i$, then the assumption a_i associated with e_i called the outdated assumption. Formally, the outdated assumption can be represented as: $\forall x \in e_i, \exists j > i$ *such that* $\neg x \in Cn(S_j)$, where $Cn(S_j)$ denotes logical closure of S_j.

The outdated assumptions are not allowed to participate in defeasible reasoning. A simple example of an outdated assumption is: a book borrowing service assumes that the borrower is in same city as the library. When the borrowed book is returned, we say this assumption is outdated.

To conduct the defeasible reasoning about the current state of the world, it is necessary to describe and record various assumptions generated during the service composition planning. In this framework, we maintain an assumption

Figure 1: Car Renting Service

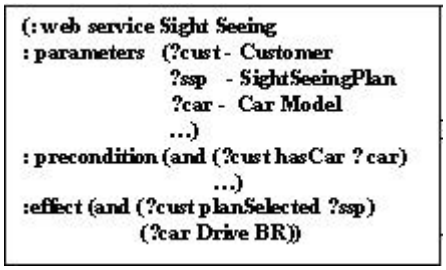

Figure 2: Sightseeing Service

database D_α to store these assumptions and their relevant effects as a pair $\langle a_i : e_i \rangle$. Same as preconditions and effects, assumptions are represented as ground literals.

In this framework, it is not only possible to make tentative conclusions when the information available is insufficient but also to revise these conclusions in face of additional information. In the context of service composition, our framework has the following kinds of advantages:

1. Making the decision upon partial or incomplete information easily fails to achieve consistency. This framework bridges the gap between user requirements and the consistent service composition.

2. The assumptions are used as justifications of the effects in service composition context. And the knowledge of the state can be revised over the time to incorporate new knowledge, thus it enhances the ability to deal with exceptions.

5. SCENARIO

In our Web Service examples (Fig.1 and Fig.2) we use syntax similar to that of the Planning Domain Definition Language (PDDL [7]). The first example Fig.1 is about a car rental service which rents a car to a customer. The precondition of this service is that customer has a visa card and this visa card is valid e.g. has enough credit and not expired. As the effect of this car rental service, the customer rented his preferred car. And the assumption of this service made by its service provider is that the car is used in proper way e.g. a city car is not used for a mountain or desert dune exploration.

NotDriveOn (see Fig.1) is our example is a service assumption which can be expressed as a property atom by SWRL. A property atom consists of a property name and two elements that can be individual names, variable names or data values [17]. In this case, BR(Bad Road conditions, see Fig.1) is an individual constant and represents a road condition on which the rented car must not drive on according to the car rental company's service policy.

Satisfaction of the Service Precondition: To participate in the service composition, the selected Web Service must be a valid candidate service, here Rule A is fired. In the example above, to be a valid candidate service, the precondition associated with this Web Service must be satisfied (A customer has a visa card and this visa card is valid).

Applying the Effects: If the car rental service is a valid candidate Web Service, normally the effects associated with this Web Service can be applied to the current state as Rule B. However, these effects may be defeated in the face of new information.

Making assumptions: Certainly, the rented car should be used in proper way. For instance, if the rented car is a city car, in common sense, this car should not be used for a mountain or desert dune exploration. To deal with exceptions which may result from incomplete information, the car rental service provider makes an assumption here that the car is not used for certain road conditions. The assumptions are stored in D_α. To clarify the usage of the assumption in our framework, here we give a simplified example of a sightseeing plan service(see Fig.2), which is supposed to integrate with the car rental service together as a travel agency package.

If the customer Alice chose a city car as her preferred car and registered to the sightseeing plan service. The desert dune exploration as a sightseeing plan is generated and as a result, the car will be used for this dune exploration (Fig.2). Now the state holds the following the sentences:
$rented(Alice, cityCar)$,
$hasPlan(Alice, duneExploration)$,
$driveOn(cityCar, dune)$
The assumption made by the car rental service is stored in D_α, which is: $notDriveOn(cityCar, dune)$. Clearly, the negation of the car usage assumption now is entailed by the current state, that is $S \models \neg a$, then we can reach the conclusion that Rule C is fired. In this case, the violation against the car usage is detected, and the car rental company could not rent a city car to this customer for his desert dune exploration. This example explains how the service assumption can help us to detect potential conflicts in service composition context, especially when there is only incomplete information available.

6. EXTENDING OWL-S

Because different enterprises have distinct business objectives, rules and assumptions about using or providing a Web Service, especially in some rule or policy intensive enterprises, it is unrealistic to acquire complete information from all parties involved during dynamic service composition. Making the decision upon partial or incomplete information often fails to achieve consistency. The solution to this problem is to use the assumptions as justifications of the effects in service composition context, which bridge the gap between user requirements and consistent service composition. The inability to make assumptions therefore translates into an inability to deal with exceptions [13].

6.1 Syntax

Currently, there is no way for OWL-S to describe the various assumptions about the multiple independent application domains involved in service composition. And no mecha-

nism to guarantee that the service execution has the anticipated effects when there is only insufficient knowledge available during the service composition execution. By adopting the service assumptions into OWL-S, we can conduct reasoning about what is known in the composite service execution context against various domain assumptions. Thus the ontology for Web Service becomes more complete and closer to the real world. We proposed to add the assumptions as the properties of service process, which allows various assumptions to be captured and recorded in readily accessible fashion. Syntax as follows:

```
<owl:Class rdf:ID="Assumption">
   <owl:subClassOf rdf:resource="&expr;
#Expression"/>
</owl:Class>

<owl:ObjectProperty
        rdf:ID="hasAssumption">
   <rdfs:domain rdf:resource="#Process"/>
   <rdfs:range rdf:resource="&expr;#
       Condition"/>
</owl:ObjectProperty>
```

6.2 Example of Using Assumption

The proposed mechanism gives OWL-S the ability to describe the various assumptions of the service domain. Hence, the assumption can be defined as one of the properties of the service process together with input, output, preconditions and effects. We also propose to use SWRL expressions in OWL-S assumptions, thus we can use the expressive power of rules to facilitate service conflict reasoning. Example as follows:

```
<process:hasAssumption>
<expr:SWRL-Condition rdf:ID="Road-
    Restriction-Assumption">
<rdfs:label>
        the car assumed must not drive on
            certain road conditions
</rdfs:label>
<expr:expressionLanguagerdf:resource="http
    ://www.daml.org/services/owl-s/1.1/
    generic/Expression.owl#SWRL"/>
<expr:expressionBody rdf:parseType="
    Literal">
<swrl:AtomList><rdf:first>
<swrl:IndividualPropertyAtom>
<swrl:propertyPredicate rdf:resource="#
    NotRunningOn"/>
<swrl:argument1 rdf:resource="#car"/>
<owl:Individual rdf:ID="dune"/>
</swrl:IndividualPropertyAtom></rdf:first>
<rdf:rest rdf:resource="http://www.w3.org
    /1999/02/22-rdf-syntax-ns#nil" />
</swrl:AtomList> </expr:expressionBody>
</expr:SWRL-Condition>
</process:hasAssumption>
```

In our proposed framework, one can make two kinds of assumptions in OWL-S. Here we denote variable name by x, y, z, concept atom by C and property atom by P, then we have:

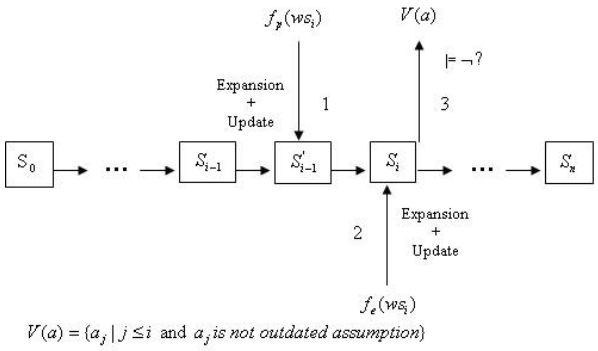

Figure 3: Defeasible Reasoning Process

1. Concept assumptions $C(x)$, which asserts x belongs to concept C;

2. Property assumptions $P(y, z)$, which asserts z is value of the property P for y.

7. DEFEASIBLE REASONING PROCESS

In this section, we are prepared to illustrate the process of constructing the service composition plan based on our proposed framework. Service composition planning can be viewed as a process of resolving conflicts and gradually refining a partially specified plan, until it is transformed into a complete plan that satisfies the goal.

Service Composition planning is similar to the classical planning in that each world state is represented by a conjunction of literals and each Web Service is related to a transition between those states. However, unlike classical AI planning techniques, in this proposed framework, the planner is the rule based system which allows making tentative conclusions and revising them in the face of additional information. In other words, the planner is endowed with the ability to reasoning about incomplete information in the service composition context.

For any state S_{i-1}, Web Service ws_i is not applicable to the state until certain minimal criteria are met. ws_i is specified in terms of the precondition p_i, effect e_i and assumption a_i, where p_i must be satisfied for ws_i to be valid (Rule A) , the effect may be concluded (Rule B) and the negation of the a_i plays the role of being the defeaters (Rule C).

A state in our framework is not a complete view of the world. Usually, an agent is forced to perform sensing operations which is aiming at finding out the information which could satisfy the precondition p_i. Like "1" showed in the fig 3, the sensing operation may lead to knowledge expansion and update of the state S_{i-1}. When the sensing operations complete, if p_i is satisfied, we can conclude that ws_i is applicable to the current state (Rule A). Due to the expansion and update of knowledge to state S_{i-1}, before the transition to state S_i, we get an intermediate state S'_{i-1} which holds the current knowledge of the world after the agent's sensing operation. Following the sensing operations, effect e_i is applied to the current world state to simulate the action. Again, the effect e_i may expand and update the knowledge of the current state (Rule B). This process can be presented as generic state transition operation as we defined in Section 3.

One of the main features in this proposed framework is the ability to describe various service assumptions and support defeasible reasoning with these assumptions. Assumptions generated from the service composition planning are represented as a set of ground literals stored in the assumption database D_α. After the effect is applied to the current state, the knowledge in the state may be expanded or updated. For the new state of knowledge, the planner needs to carefully perform the checking to see whether any outdated assumption is in D_α. Because the outdated assumptions are not allowed to participate the defeasible reasoning, all outdated assumptions are deleted from D_α. Next, it is to find the defeaters by the mean of checking whether any negation of assumptions can be entailed by the current state. The negation of service assumption which is not outdated plays the role of being a defeater, which prevents the effects associated with this assumption being applied to the state (Rule C). Up to now, the process of state transition from S_{i-1} to S_i is completed. We have illustrated that how the new world state is reached in the presence of possibly conflicting rules.

8. RELATED WORKS

WSMO [18] extended and refined Web Service Modeling Framework (WSMF), aimed at complementing current Web Service standards. WSMO provides a conceptual model and language for the relevant aspects of the semantic Web Service. Web Service descriptions consist of functional, non-functional and the behavioral aspects of a Web Service. The capability of a Web Service is specified in terms of precondition, postcondition, assumption, effect and some others properties. However, the assumption we proposed in this work is different from the assumption in WSMO. The assumption proposed in this work extends the semantics of OWL-S for the purpose of explicitly supporting defeasible reasoning and trying to tackle the problem of incomplete information in service composition context. The current Web Service standards [1, 6] only focused on modularization of service layers and static service compositions, while some issues about dynamic service composition have not been addressed. In particular, how to reason about incomplete information during the dynamic service composition, in other words, how to ensure the consistency of service composition when there is only incomplete knowledge available in service composition context.

9. CONCLUSIONS AND FUTURE WORK

The goal of dealing with incomplete information in the service composition context is certainly a challenging task. The proposed framework is an attempt at tackling the problem of how to achieve consistent service composition when information available is insufficient.

In this work, we have extended the OWL-S to a richer service description representation schema by introducing the service assumptions. We also adopted defeasible rules for reasoning with various assumptions. We illustrated how knowledge based planning could reason about incomplete knowledge in service composition context and construct the service composition plan. During the process of the service composition, we showed that absolute rules could be used for service precondition satisfaction, especially defeasible rules and defeaters could be employed to make tentative conclusion based on the available information, and to detect potential conflicts in service composition when further suitably information about the problem is available.

As part of our future work, we are working on the incorporation of other service composition technologies, such as matching of Web Services capabilities, quality of service etc, which would result in further reliable automation of the consistent service composition.

10. REFERENCES

[1] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., und Weerawarana, S. "Business Process Execution Language for Web Services", Version 1.1. Specification. BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems. 2003.

[2] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., Eds. 2003. The Description Logic Handbook:Theory, Implementation and Applications. Cambridge University Press.

[3] Dean, M. and Schreiber G. "OWL Web Ontology Language". Reference W3C Recommendation, http://www.w3.org/tr/owl-ref/. Feb 2004.

[4] Benjamin N. Grosof, Ian Horrocks "Description Logic Programs: Combining Logic Programs with Description Logic" ACM 1581136803/03/0005.

[5] Donald Nute. "Defeasible logic." In D. Gabbay and C. Hogger (eds.), Handbook of Logic for Artificial Intelligence and Logic Programming, Vol. III, Oxford University Press, 1994:353-395.

[6] Kreger, H. Web Services Conceptual Architecture (WSCA 1.0). http://www-4.ibm.com/software/solutions/webservices/ , 2001

[7] M. Fox and D. Long. Pddl2.1: An extension to pddl for expressing temporal planning domains, 2002. http://www.dur.ac.uk/d.p.long/pddl2.ps.gz

[8] M. Paolucci, Takahiro Kawamura, Terry R. Payne, Katia Sycara. "Importing the Semantic Web in UDDI". In Proceedings of Web Services, E-business and Semantic Web Workshop.

[9] M. Paolucci, T. Kawmura, T. Payne and K. Sycara. Semantic Matching of Web Services Capabilities. In First Int. Semantic Web Conf., 2002

[10] OWL-S White Paper "OWL Services Coalition. OWL-S: Semantic markup for Web Services", 2005. http://www.daml.org/services/owl-s/1.1/overview/#1

[11] Paulo F. Pires, Mario R.F. Benevides, Marta Mattoso "Building Reliable Web Services", Web, Web Services and Database Systems 2002, LNCS 2593

[12] R. E. Fikes and N. J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. In J. Allen, J. Hendler, and A. Tate, editors, Readings in Planning, pages 88-97. Kaufmann, San Mateo, CA, 1990.

[13] R. V. Guha. Contexts: A Formalization and Some Applications. PhD thesis, Stanford University, 1991.

[14] Reiter R. "A logic for default reasoning", Artif Intell 1980; 13:81-132.

[15] Reiter R. "ON REASONING BY DEFAULT", Proceedings of the theoretical issues in natural language processing-2 July 1978

[16] S. McIlraith and T. C. Son. Adapting Golog for composition of Semantic Web Services. In Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France, April 2002.

[17] "Semantic Web Rule Language", May 21, 2004. http://www.w3.org/Submission/2004/03/.

[18] WSMO working group. WSMO homepage, since 2004. http://www.wsmo.org/.