

2006

# Extending Semantic Web Service Description by Service Assumption

Z. Lu

*University of Wollongong, lu@uow.edu.au*

S. Li

*University of Wollongong*

Aditya K. Ghose

*University of Wollongong, aditya@uow.edu.au*

P. Hyland

*University of Wollongong, phyland@uow.edu.au*

---

## Publication Details

This paper was originally published as: Lu, Z, Li, S, Ghose, A & Hyland, P, Extending Semantic Web Service Description by Service Assumption, IEEE/WIC/ACM International Conference on Web Intelligence 2006 (WI2006), Hong Kong, China, December 2006, 637-643. Copyright IEEE 2006.

---

# Extending Semantic Web Service Description by Service Assumption

## **Abstract**

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. These characteristics of autonomy and heterogeneity are fundamental to service oriented computing but make it inherently difficult to avoid service conflicts. To reason about and adapt to a changing environment, in this work, we will extend current OWL-S by introducing the concept of service assumptions which allow reasoning with incomplete information. Furthermore, together with the proposed service assumptions, a sequence of rules is proposed to describe all permitted behaviors in service composition context.

## **Disciplines**

Physical Sciences and Mathematics

## **Publication Details**

This paper was originally published as: Lu, Z, Li, S, Ghose, A & Hyland, P, Extending Semantic Web Service Description by Service Assumption, IEEE/WIC/ACM International Conference on Web Intelligence 2006 (WI2006), Hong Kong, China , December 2006, 637-643. Copyright IEEE 2006.

# Extending Semantic Web Service Description by Service Assumption

Zheng Lu, Shiyang Li, Aditya Ghose and Peter Hyland  
School of IT & Computer Science  
University of Wollongong, Australia  
{zl07, sl562, aditya, phyland}@uow.edu.au

## Abstract

*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. These characteristics of autonomy and heterogeneity are fundamental to service oriented computing but make it inherently difficult to avoid service conflicts. To reason about and adapt to a changing environment, in this work, we will extend current OWL-S by introducing the concept of service assumptions which allow reasoning with incomplete information. Furthermore, together with the proposed service assumptions, a sequence of rules is proposed to describe all permitted behaviors in service composition context.*

## 1 Introduction

The basic motivation of service oriented computing is to allow a high degree of flexibility to create the value-added composite service in a dynamic fashion. Web Services are running in a distributed environment, and often, are provided by a large number of independent parties. However, these independent parties do not necessarily share the same objectives and background. Thus the ignorance of one another may result in incompleteness and uncertainty of the information during the process of service composition. Hence, to achieve reliable service composition, it is critical for Web Services to have the ability to adapt to a changing environment.

The current OWL Web Ontology Language for services specification (OWL-S [1]) leverages the rich expressive power of OWL [4] together with its well-defined semantics to provide richer descriptions of Web Services. In addition, Semantic Web Rule Language (SWRL) [2] has been proposed to define service process preconditions and effects, process control conditions and their contingent re-

lationships 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 [7]. 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. Currently, OWL-S has no mechanism for handling incomplete knowledge during the process of dynamic service composition.

In this paper, we are going to bridge the gap between semantic service description and multiple operational domains involved by introducing "service assumptions". In addition, based on our proposed extensions, we will try to define a formal framework for reasoning about incomplete knowledge and to address the service conflict issues.

The paper is structured as follows: In Section 2, we give examples of Web Service composition problems that we propose to address. In Section 3, we extend the current version of OWL-S by adopting service assumption and explain the semantics of the service assumption. In Section 4, we define the service selection and the composite service in general. In Section 5, we define the basic semantics for the planning-based service composition domain. In Section 6, we present a framework for reasoning about incomplete knowledge in service composition context. Finally, in Section 7, we present related work and our conclusions respectively.

## 2 Motivating Examples

The following example of a travel agency is used to explain the service conflicts which may be caused by incompleteness of information during the dynamic service composition. Our example uses the often presented travel agency service package. A typical use case could involve arranging a trip consisting a hotel booking, a car rental and

a sightseeing service. To simplify this use case, we assume this composite service is executed in a sequential manner (i.e. hotel booking service, then car rental service, finally sightseeing service). Assume that, when requesting this composite travel agency service, the user specifies his preferred car model, for example, a city car. Obviously, this car will be used for sightseeing, which is also generated as part of this composite service. If the functionality matches the user's requirement, then the car rental service is invoked. In the real world, it is most likely that the car rental service providers have some service policy about usage of rental cars. However, when the car rental service is invoked, we don't have any information about what kinds of sightseeing plan might have been generated from the execution of the service, in other words, we don't know how the rented car will be used. The point here is that different sightseeing plans may be associated with different roads, and it may not be allowable for a rented car to drive on certain roads. For example, a desert dune exploration plan is dynamically generated from the sightseeing service and a city car is used for the desert dune exploration. Clearly, this is not an acceptable situation for either the car rental company or the customer.

Often, Web Service composition involves multiple independent parties, and during this process, the interactions between these independent parties have to be carried out to locate, invoke services. However, 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, thus we can assume any service composition involving more than one independent service providers will be subject to typical group conflicts [9]. Thus, to ensure integrity of service composition, there should be a mechanism to deal with incompleteness of information during dynamic service composition. The solution to this problem is to use service assumptions. Before formally introducing our framework in the succeeding section, we will clarify some basic definitions about Web Services.

### 3 Extending OWL-S by Service Assumption

#### 3.1 Atomic Service

Results from the study of default logic [8, 10] serve as a basis for understanding service assumptions. By extending the current OWL-S, an atomic service  $ws_i$  in this proposed work is described by a tuple  $\langle p_i, e_i, a_i \rangle$ , where

- $p_i$  is a set of sentences representing the precondition, i.e.  $p_i = \{p_i^1, \dots, p_i^n\}$ .  $p_i$  must be true for the atomic service to execute. Each sentence in  $\{p_i^1, \dots, p_i^n\}$  is defined as a *primitive precondition*.

- $e_i$  is a set of sentences representing the change of world state, i.e.  $e_i = \{e_i^1, \dots, e_i^n\}$ .  $e_i$  may include both positive and negative effects. Each sentence in  $\{e_i^1, \dots, e_i^n\}$  is defined as a *primitive effect*.
- $a_i$  is a set of sentences representing service assumption, i.e.  $a_i = \{a_i^1, \dots, a_i^n\}$ . Each sentence in  $\{a_i^1, \dots, a_i^n\}$  is defined as a *primitive assumption*.

Because of the heterogeneous nature of the Web Service execution environment, incomplete information in the process of service composition may occur either because of the unavailability of certain information or to keep the formulation simple at the start. Given a service  $ws_i = \langle p_i, e_i, a_i \rangle$ , informally, its semantics can be interpreted as: if  $p_i$  can be satisfied, and if it is consistent to assume  $a_i$ , then we may conclude that  $e_i$  can be applied. Service assumptions can be used to define a collection of default conditions regarding service policies. The service assumptions are believed when information is incomplete, but these assumptions also can be revised over time to incorporate new knowledge. Thus the ontology for Web service becomes more precise and closer to the real world. To be consistent with current OWL-S specification, in this work the chosen logical language to represent the service assumption is the Semantic Web Rule Language (SWRL)[2].

#### 3.2 Classification of Service Assumptions

To adopt the assumption to the real world application in more flexible way, there are two distinct usages of service assumptions which need to be taken into consideration. The first case is the restriction about the usage of web service, while the second case makes assumptions to provide warning information, aiming to ensure the service requester gets the satisfactory result. Informally, the classification of service assumptions as follows (also See Fig 1):

- **Hard Assumption:** this kind of assumptions are used to strengthen the service policy made by each atomic service node in service composition, which cannot be violated or the service composition is running into failure.
- **Soft Assumption:** this kind of assumptions are used for the purpose of prototypical reasoning, which means that typically, most instances of a service composition have some property.

The example provided in our motivation section (see Section 2) is an example of using the hard assumption, which aims to enforce the service policy - car usage. In other words, if there is conflicting information against the car usage assumption in the context of the service composition, then this car rental service will not become part of

the generated composite service. For the second kinds of service assumptions, like the first kinds of service assumptions, if conflicting information appears, it is considered to be the conflict of the web service composition. Unlike the first kinds of service assumptions, in face of conflict information, soft assumption will provide the warning information, but the option is available to the service requester, i.e. the service could choose either ignore this conflict information or discard the chosen Web Service which produce the conflicting information. Using soft assumption is quite normal in our real life, return to our car rental service, one example of using soft assumption can be that suppose one particular car model normally is rented as the wedding courtesy car, but service requester wants to rented this car for a long distance trip. Renting this car may be very costly, thus the service provider kindly provides such warning information. However, the service requester can make his decision whether or not rent this car in face of the warning information.

### 3.3 Example of Using Assumption

Here we give examples to show a simple case of service assumption. The example is taken from the car rental service, which has the policy “the rented city car cannot drive on certain road conditions”, and this policy is enforced by the service assumption. SWRL expressions are proposed to represent assumptions, thus we can use the expressive power of rules to facilitate service conflict reasoning. Generally, a service assumption is represented as a rule, which has the form:  $antecedent \Rightarrow consequent$ , where the symbol  $\Rightarrow$  denotes the logical imply and both antecedent and consequent are generally defined as conjunctions of atoms, having the form of  $a_1 \wedge \dots \wedge a_2$ . Using this syntax, a rule stating that the composition of “city car not drive on dune” and “city car not drive on unsealed road” properties implies the “DriveCarInProperWay” property would be written:

$$\begin{aligned} & \neg DriveOn(?cityCar, dune) \wedge \\ & \neg DriveOn(?cityCar, unsealedRoad) \\ & \Rightarrow DriveCarInProperWay(?cityCar) \end{aligned}$$

Informally, the example can explained as: if both “city car not drive on dune” and “city car not drive on unsealed road” are *consistent* with what is known in the context of the service composition, then it is assumed that the car will drive in the proper way, where consistent means without the information to the contrary.

The goal of adding service assumptions is to enable Web services applications: a). to be more flexible and intelligent which result from commonsense inference nature of service assumption. b). to be executed in a consistent manner. The more accurate and precise service description of the problem, the more reliable the decisions we make.

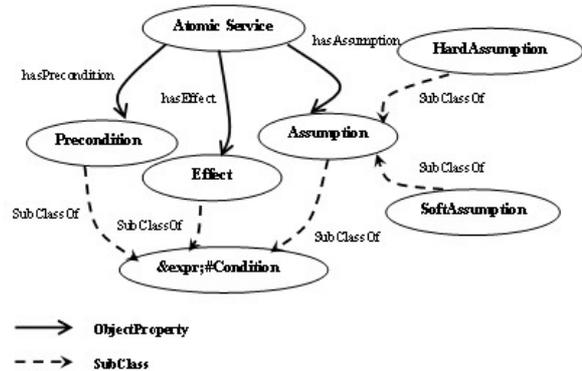


Figure 1. Extended Atomic Service

## 4 Service Selection and Composite Service

### 4.1 Service Selection

The process of dynamic Web Service composition over that of software component composition holds some additional critical issues, such as service matching, selection and retrieval. 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  $\{cat_1, cat_2, \dots, cat_n\}$ .  $cat_i$  is a tangible areas split from the service registry, for example downloadable Multimedia.
- $CAT$  is the set of all service categories  $cat_i \in CAT$ ,  $cat_i \in WS$ ,  $cat_i = \{ws_1, \dots, ws_m\}$ .
- 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 the Web Service which is produced by the service selection defined above.

### 4.2 Composite Service

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

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

Because participants of the service composition do not necessarily share the same objectives and background, conflicts easily arise in a dynamic service composition environment.

## 5 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 [6]. Typically, classical planners presuppose complete and correct information about the world. However, in terms of the service composition, this simplified assumption is not suitable and unrealistic. Each service node is designed, owned, or operated by different parties, thus the planning agent may not have a complete view about the world. To make more precise service description in a dynamic service composition environment, we have extended the current semantic Web Service description OWL-S by introducing the service assumption. The service assumptions together with states of knowledge, preconditions, effects, and goals are specified in Description Logic  $\mathcal{L}$  [3].

Now we are prepared to define the semantics of a service composition domain. A state  $S$  is not a complete view of the world, which describes the partial state with respect to the service composition context. The state  $S$  is extensionally defined as a set of positive or negative ground atomic formulas (atoms). In addition, the initial state  $S_0$  here is a partial description about the world, i.e. a partial state. A goal  $G$  is a set of conjunctions of atoms which need to hold in a desired state or say final state. 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. A service composition plan for a goal is a sequence of state transitions which lead from an initial state to a final state where all ground atomic formulas in the goal are true. In rest of the paper, we will use symbol  $\models$  to represent logical entail.

In the process of service composition planning, there are three types of knowledge produced by state transitions about the current world. 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 invariants, expansion and update, which is defined as:

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

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:

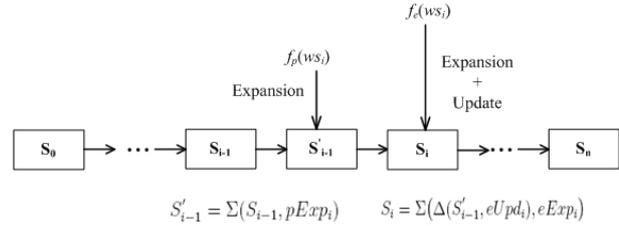


Figure 2. Generic State Transition Operators

$$S_{i-1} \not\models Exp_i \text{ and } S_{i-1} \not\models \neg Exp_i$$

3. State update  $Upd_i$  denotes a set of sentences whose negation can be entailed by the knowledge in the previous state, defined as:  $S_{i-1} \models \neg Upd_i$

Let  $ws_i$  be an atomic service,  $WS$  be the set of all Web Services,  $E$  be the set of all service effects,  $P$  be 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, expansion and 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, expansion and 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. Here we assume that the information contained in the state of knowledge is incomplete but correct. Clearly, the precondition evaluation either depends on the current state of knowledge or is based on sensing operation [5] which adds new knowledge to the current state. Thus the knowledge generated from the sensing operation for the purpose of the precondition evaluation can only expand the current state of knowledge. i.e.

$$f_p(ws_i) = p_i \text{ and } p_i = \{pInv_i \mid pExp_i\}$$

Following the definitions above, we can define the generic state transition operators (See Fig 2) as:

1.  $S'_{i-1} = \Sigma(S_{i-1}, pExp_i)$
2.  $S_i = \Sigma(\Delta(S'_{i-1}, eUpd_i), eExp_i)$

which means the state transition from  $S_{i-1}$  to  $S_i$  is completed by means of performing sensing operations for the precondition evaluation, then applying the service effect. In

step one, the knowledge  $pExp_i$  generated from the sensing operations is used to expand previous knowledge of the state  $S_{i-1}$ . The operator  $\Sigma$  takes the  $S_{i-1}$  and  $pExp_i$  as its input, expands knowledge of the  $S_{i-1}$  and produces the intermediate state  $S'_{i-1}$ .  $S_i$  is reached at step two, in which the operator  $\Delta$  takes the  $S'_{i-1}$  and  $eUpd_i$  as its input and performs an update to knowledge of the  $S'_{i-1}$ . Finally, applying the effect may also lead to knowledge expansion.

In the knowledge representation literature [11], incomplete of the knowledge has been classified as: either *absence* or *uncertainty*. Adopting this classification about the incomplete knowledge, in the service composition planning process, we refer to the missing facts as the *absence of information*. On the other hand, *uncertainty* is the a subjective measure of the certainty about service interactions, which may be caused by ignorance of one another when multiple independent parties are involved in the process. Clearly, uncertainty and absence are essentially different, thus we use different techniques to handle these two distinct types of incomplete information. The absence of information is handled by the sensing operation. What makes dynamic service composition complicated is the fact that, during the process, services interact in complex ways. The proposed service assumption can be used to describe the service composition environment which may be not specifically known. As a consequence of this more precise description of the service composition environment, it is possible for us to deal with exceptions and resolve the inconsistencies which are caused by uncertainty. From now on, we will concentrate on the service composition consistency problem which may be caused by the interactions of multiple independent parties.

## 6 Default Reasoning in Service Composition

### 6.1 Assumption Database and Outdated Assumptions

To conduct the default reasoning about the partial state of knowledge, it is necessary to describe and record various assumptions generated during the service composition planning. In this framework, we maintain an assumption database  $\mathcal{M}$  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.

For a web service  $ws_i = \langle p_i, e_i, a_i \rangle$ , which is selected by the service selection function and has participated in a service composition, its effect  $e_i$  is a set of sentences, i.e.  $e_i = \{e_i^1, \dots, e_i^n\}$ . We define  $\varphi$  as the negation of the service effect  $e_i$ , if  $\{\neg e_i^1, \dots, \neg e_i^n\} \subseteq \varphi$ . Here, we use  $\neg e_i \subseteq \varphi$  to denote that  $\varphi$  is the negation of the service  $e_i$ .

If  $\neg e_i \subseteq \varphi$  and  $\varphi$  is the logical consequence of a current state of knowledge, we refer to the assumption  $a_i$  which is associated with service  $ws_i$  as *outdated assumption*. 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.

Notice that outdated assumptions are not allowed to participate in reasoning process for the service composition. For any service  $ws_i$  which has participated a given service composition, if its assumption  $a_i$  is not outdated assumption, we refer  $a_i$  as *active assumption*, and use  $\Pi(\mathcal{M})$  to denote the set of all active assumptions maintained by  $\mathcal{M}$  for a given service composition.

### 6.2 Default Reasoning Framework

The state transition function takes previous state of knowledge  $S_{i-1}$  and Web Service  $ws_i$  as the input and produces the new state  $S_i$ . To get the legal state transition, inspired by default logics [8, 10], our conflict checking contains three transition conditions:

1. Precondition Satisfaction (Cond-A): means that only when a precondition holds, and then the service is a valid candidate service to participate service composition. Formally, if  $S_{i-1} \models p_i$ , then we define  $ws_i$  as *precondition satisfied service*. Note that  $S_{i-1}$  here also contains the knowledge acquired by the sensing operation for the purpose of the precondition evaluation.
2. Consistency of State and Assumptions: which means that after the effect  $e_i$  of Web Service  $w_i$  is applied to the current state, the new state of knowledge must be consistent with the set of all active assumptions  $\Pi(\mathcal{M})$  maintained in  $\mathcal{M}$ . Formally,  $S_i \cup \Pi(\mathcal{M}) \not\models \perp$ . Normally,  $e_i$  is the conclusion of a precondition satisfied service  $ws_i$ , but  $e_i$  may need to be retracted in face of new evidence. Note that here we intentionally make the design decision that joint consistency of service assumptions is required. Thus checking of consistency between the state and the assumptions has two steps:
  - Joint Consistency of Assumptions (Cond-B): which means the conjunction of all active service assumptions must be consistent. Formally,  $\Pi(\mathcal{M}) \not\models \perp$
  - Consistency between State and Assumptions (Cond-C): which means that in addition to the conjunction of all active service assumptions being consistent, it is also required that the new state of knowledge should be consistent with this set of service assumptions. Formally  $S_i \cup \Pi(\mathcal{M}) \not\models \perp$

The building of consistent value-added services on a heterogeneous environment is not a trivial task, in next section, we will prepare to illustrate the process of constructing the service composition plan and explain how these proposed state transition conditions should be used during this reasoning process.

### 6.3 Default Reasoning Process

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 state of knowledge 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 reason about and adapt to a changing environment. As the result of the applying the state transition rules, the generated plan represents an applicable or consistent solution to the service composition problem even with insufficient information during the process. 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 to be the precondition satisfied service (Cond-A), the effect may be concluded, however the joint consistency of assumptions (Cond-B) and consistency of new state of knowledge and various service assumptions (Cond-C) are required.

A state in our framework is not a complete view of the world. Usually, an agent is forced to perform sensing operations which aim at finding out the information which could satisfy the precondition  $p_i$ . Like "1" shown in Fig 3, the sensing operation may lead to knowledge expansion of the state  $S_{i-1}$ . When the sensing operations complete, if  $p_i$  is satisfied, we can conclude that  $ws_i$  may be applicable to the current state  $S_{i-1}$  (Cond-A). Due to the knowledge expansion to the state  $S_{i-1}$ , before the transition to state  $S_i$ , we get an intermediate state  $S'_{i-1}$ . This intermediate state holds the current state of knowledge after the agent's sensing operation, which is shown as the operation step "2". Following the sensing operations, effect  $e_i$  is applied to the current state to simulate the action. Like we mentioned before, the effect  $e_i$  may expand and update the knowledge of the current state, which is shown as the operation step "3". This process can be presented as generic state transition operation as we defined in page 4.

One of the main features in this proposed framework is the ability to describe various service assumptions and sup-

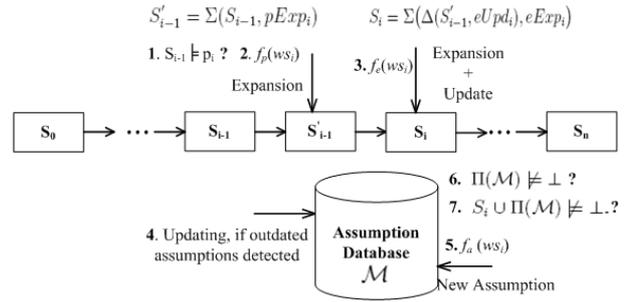


Figure 3. Reasoning Process

port default reasoning with these assumptions. The service assumptions generated from the service composition planning are represented as a set of ground literals stored in the assumption database  $\mathcal{M}$ . After expanding and updating the knowledge of the current state, the planner needs to carefully perform checking to see whether any outdated assumption is in  $\mathcal{M}$ . Because the outdated assumptions are not allowed to participate in the default reasoning, all outdated assumptions will be deleted from  $\mathcal{M}$ , which is shown as operation step "4".

After updating the assumption database  $\mathcal{M}$ , the service assumption  $a_i$  is added to assumption database  $\mathcal{M}$ . Initially, the status of this new service assumption  $a_i$  is set to be *active*, which is shown as the operation step "5". Service assumptions are made about things that may not specifically be known during the process of service composition, which also represent the environment of an underlying service composition. A particular service composition environment is described by the set of all active assumptions  $\Pi(\mathcal{M})$  maintained in the assumption database  $\mathcal{M}$ . Logically, this environment refers to a conjunction of service assumptions. Clearly, a service composition context is uniquely identified the combination of the environment and the current knowledge state. To achieve consistent service composition, we intentionally make the design decision that the service composition environment is required to be consistent, which means that no contradiction can be inferred from  $\Pi(\mathcal{M})$ . A consistent service composition environment is enforced by Cond-B which is shown as the operation step "6". Note that the checking of joint consistency of assumptions is performed after both the effect  $e_i$  is applied to the current state of knowledge and the updating of all the detected outdated assumptions in the assumption database  $\mathcal{M}$  is complete. If a contradiction appears, it means that the service composition environment is no longer consistent and corrections to these assumptions must be made in the face of this contradicting information. The conclusion of applying the  $e_i$  of  $w_i$  to the state of knowledge must be revoked.

On the other hand, if the the service composition en-

environment is described by a consistent set of service assumptions, the next reasoning task is to check the consistency between the new state of knowledge and the set of all active service assumptions  $\Pi(\mathcal{M})$  (Cond-B), which is shown as operation step “7”. This task is completed by means of checking whether the negation of any active assumptions can be entailed by the current state of knowledge. The negation of a service assumption plays the role of being a defeater, which prevents the effects associated with this assumption being applied to the state. Similarly, if the contradicting information is detected at this step, it means that the previous conclusions are not appropriate in the face of this additional information and the old conclusions must be discarded in order to incorporate new knowledge and adapt to a changing environment. Up to now, the process of state transition from  $S_{i-1}$  to  $S_i$  is completed. We have illustrated that how the new state of knowledge is reached in the presence of possibly incomplete or conflicting information.

Notice that, although the are treated as the same during the process of service composition planning, there is a fundamental difference between the ways of handling the conflicts caused by hard assumptions and soft assumptions respectively. Typically, when the conflict is detected, and the conflict is caused by a soft assumption, the choice will be left to client. The client could choose to ignore the detected conflict. However, if the detected conflict is produced by a hard assumption, it is considered to be a conflict of the Web Service composition anyway, and the client does not have control over it.

## 7 Conclusions

In this work, we have extended OWL-S to a richer service description representation schema by introducing service assumptions. The general goal of adding service assumptions as one property of a Web Service is to allow making plausible inferences in the process of service composition and ensure consistent service composition, which might be seen as: a). accurately describing the service composition environment, in which most instances of a concept generally have some property, but not always. b). presenting the hypothetical guesses about incompleteness and uncertainty. c). some combination of both.

The goal of dealing with incomplete information in the service composition context is certainly a challenging task. In our proposed framework, together with the proposed service assumption, we developed a sequence of rules for reasoning with various assumptions during the process of service composition planning. We also illustrated how knowledge based planning could reason about incomplete knowledge in the service composition context and construct a service composition plan. During the planning process, we showed that only when a precondition holds, then the ser-

vice is a valid candidate service to participate service composition. Specially, by adopting service assumptions, the framework supports default reasoning in the presence of incomplete knowledge. The service assumptions are made about the things that may not specifically know during the process of service composition, thus what service assumptions represent is the environment of a underlying service composition. Logically, this environment refers to a conjunction of service assumptions. To achieve the consistent service composition, we intentionally make the design decision that the service composition environment is required to be consistent. Finally, consistency between the state of knowledge and the set of all active service assumptions is required. This consistency checking task is completed by the means of checking whether the negation of any active assumptions can be entailed by the current state of knowledge. The negation of a service assumption plays the role of being a defeater, which prevents the effects associated with this assumption being applied to the state. Briefly, this proposed framework allows us to make tentative conclusions based on the available information, and to detect potential conflicts in service composition when further suitable information about the problem is available.

## References

- [1] OWL Services Coalition. OWL-S: Semantic markup for Web Services, OWL-S White Paper. 2005. <http://www.daml.org/services/owl-s/1.1/overview/#1>
- [2] “Semantic Web Rule Language”, May 21, 2004 <http://www.w3.org/Submission/2004/03/>.
- [3] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P. F., Eds. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press. 2003.
- [4] Dean, M. and Schreiber G. “OWL Web Ontology Language”. Reference W3C Recommendation, <http://www.w3.org/tr/owl-ref/>. Feb 2004.
- [5] Golden, K., Etzioni, O. & Weld, D. 1996. Planning with Execution and Incomplete Information, UW Technical Report TR96-01-09, February 1996.
- [6] 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.
- [7] Reiter R. “ON REASONING BY DEFAULT”, Proceedings of the theoretical issues in natural language processing-2 July 1978
- [8] Reiter R. “A logic for default reasoning”, Artif Intell 1980; 13:81-132.
- [9] Robbins, S. P. Organizational Behavior: Concepts, Controversies and Applications. Englewood Cliffs, NJ: Prentice-Hall. 1989
- [10] SCHAUB, T. 1992. On constrained default theories. In Proceedings of the 10th European Conference on Artificial Intelligence (ECAI92, Vienna, Austria, Aug. 3-7), B. Neumann, Ed. John Wiley and Sons, Inc., New York, NY, 304-308.
- [11] M. Smithson. Ignorance and Uncertainty. Emerging Paradigms. Springer Verlag. New York. NY. 1989