University of Wollongong
Research Online

Faculty of Informatics - Papers (Archive)

Faculty of Engineering and Information Sciences

2006

Adopting Default Reasoning in Service Composition Context

Z. Lu
University of Wollongong, lu@uow.edu.au

Aditya K. Ghose
University of Wollongong, aditya@uow.edu.au

P. Hyland
University of Wollongong, phyland@uow.edu.au

Publication Details

This article was originally published as: Lu, Z, Ghose, A & Hyland, P, Adopting Default Reasoning in Service Composition Context, 4th European Conference on Web Services (ECOWS '06), Zurich, Switzerland , December 2006, 243-254. Copyright IEEE 2006.

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

Adopting Default Reasoning in Service Composition Context

Abstract
Web Service composition is the ability of one business to provide value-added services to its customers through the composition of basic Web services, possibly offered by different companies [12]. Because of distributed responsibilities, ownership and control, it is often not feasible to acquire all information needed for service composition. These characteristics 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 article was originally published as: Lu, Z, Ghose, A & Hyland, P, Adopting Default Reasoning in Service Composition Context, 4th European Conference on Web Services (ECOWS '06), Zurich, Switzerland , December 2006, 243-254. Copyright IEEE 2006.

This conference paper is available at Research Online: http://ro.uow.edu.au/infopapers/494

Adopting Default Reasoning in Service Composition Context

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

Abstract

Web Service composition is the ability of one business to provide value-added services to its customers through the composition of basic Web services, possibly offered by different companies [12]. Because of distributed responsibilities, ownership and control, it is often not feasible to acquire all information needed for service composition. These characteristics 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 Service composition shares many similarities with traditional component-based software system. They both provide aggregated functionality via reassembling various existing objects, and emphasize [11] same design principles such as reusability, replaceability, flexibility and extensibility. However, Web Services are provided by a large number of independent parties. Often, these independent parties do not necessarily share the same objectives and background. [8] has pointed out that requirements engineering has traditionally assumed that the system to be designed is under the control of a single stakeholder who (at least in principle) determines a consistent set of requirements. Modern distributed systems, however, do not fit this mold, so requirements engineering must adapt to handle them. A multi-stakeholder distributed system (MSDS) is a distributed system in which subsets of the nodes are designed, owned, or operated by distinct stakeholders. The nodes of the system may, therefore, be designed or operated: in ignorance of one another or with different, possibly conflicting goals.

Clearly, Web Services are running in a distributed environment, and 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 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 [14]. 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 problem 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

Proceedings of the European Conference on Web Services (ECOWS'06)
0-7695-2737-X/06 $20.00 © 2006 IEEE

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

Conflict has been defined in [13] as "the interaction of interdependent people who perceive opposition of goals, aims, and values, and who see the other party as potentially interfering with the realization of these goals ...". This highlights some general characteristics of conflict: interaction, interdependence, and incompatible goals. In addition, it has been demonstrated that conflict is common in group interactions [5, 16]. Often, Web Service composition involves multiple independent parities, and during this process, the interactions between these independent parities 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 that any service composition involving more than one independent service providers will be subject to typical group conflicts [16].

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.

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. In this example, to prohibit the illegal usage of the rental car, the car rental service could make the assumption that "city cars do not drive on dune, beach, unsealed road...". If the contrary evidence appears (e.g. a dune exploration) from the succeeding service executions, then we can conclude that there is a violation to the car usage policy. The inability to make assumptions therefore translates into an inability to deal with exceptions [7]. 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 [15, 17] serve as a basis for understanding service assumptions. Default logics provide formalism to deal with assumptions or beliefs. Default logics perform the retraction of beliefs when new information is presented which contradicts those beliefs. 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. $p_i = \{p_i^1, \ldots, p_i^n\}$. p_i must be true for the atomic service to execute. Each sentence in $\{p_i^1, \ldots, p_i^n\}$ is defined as a *primitive precondition*.
- e_i is a set of sentences representing the change of world state, i.e. $e_i = \{e_i^1, \ldots, e_i^n\}$. e_i may include both positive and negative effects. Each sentence in $\{e_i^1, \ldots, e_i^n\}$ is defined as a *primitive effect*.
- a_i is a set of sentences representing service assumption, i.e. $a_i = \{a_i^1, \ldots, a_i^n\}$. Each sentence in $\{a_i^1, \ldots, a_i^n\}$ is defined as a *primitive assumption*.

Note that p_i and a_i are different. p_i is a **strong** condition which must be true in order to execute the service ws_i, while a_i is a **weak** condition. We only need to establish that a_i is consistent with what is known, i.e. nothing is known that contradicts a_i. Initially we assume a_i to be true, unless we get additional information which is explicitly contradictory to a_i. 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, where each assumption is believed in the lack of evidence to the contrary, and is taken to be true until the contrary is proved. For instance, the assumption made by a car rental service could be "city car does not run on dune". The service assumptions are believed when information is incomplete, but these assumptions also can be revised over time to incorporate new knowledge. 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. Service assumptions here allow reasoning with incomplete information by the default settings and then revising conclusions ever made to reflect new information about the problem. Thus the ontology for Web service becomes more precise and closer to the real world. To use the service assumption in unified manner with other properties of Web Service specified by current OWL-S, together with *hasPrecondition* and *hasEffect*, *hasAssumption* is also defined as the one of the functional properties of a Web Service or say a process. The syntax is proposed 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>
```

Extending OWL-S with Service Assumptions

Same as the functional property *hasPrecondition* already defined in OWL-S, *hasAssumption* is also represented as logical expressions and denotes conditions that are evaluated with respect to the service composition environment. 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. In this sub-section, we will present an example which demonstrates the need to classify the service assumptions based on the relation between a service assumption and its associated service effect. In Section 3.1, a Web Service ws_i has been defined as: $ws_i = \langle p_i, e_i, a_i \rangle$. In rest of the paper, we will use symbol $\models$ to represent logical entail. Here, let a_i^i be a primitive assumption, which is a sentence in a_i, i.e. $a_i^i \in a_i$. Let $\{\neg e_i^1, \ldots, \neg e_i^n\}$ denote the negation of a service effect e_i. Because both service assumption and effect are generally defined as conjunctions of atoms, if a_i^i is also a sentence in $\{\neg e_i^1, \ldots, \neg e_i^n\}$, i.e. $a_i^i \in \{\neg e_i^1, \ldots, \neg e_i^n\}$, then clearly, $a_i \cup \{\neg e_i^1, \ldots, \neg e_i^n\} \neq \emptyset$, in other words, a contradiction can be inferred from ws_i itself, i.e. $a_i \cup e_i \models \perp$. In this case, the service assumption a_i of service ws_i contradicts its associated service effect e_i. One might hold the view that such a service represents nonsense. However, considering the following real world example

```
(: Web Service AAA Shopping Online
   Member Reg
 :parameters  (?appl - Applicant
                    ...)
 :precond ( and (?appl  OlderThan 18)
                    ...)
 :effect (?appl     isAAAmember)
 :assumption (?appl isNotAAAmember))
```

Self-Defeating Service Example

For the description of the sample service above, we use syntax similar to that of the Planning Domain Definition Language [10]). The example is about AAA Shopping Online Member Registration Service, the precondition of the service is that "the applicant must be older than 18", the effect is that "the applicant is a member" and the assumption is that "the applicant is not a current member". One interpretation is that as long as the fact of the applicant older than 18 years old can be proved, and so far there is no known evidence that the applicant is a current member, then after applying this service, the applicant is a member. The service assumption in this example is used to prevent the same applicant having two memberships. However, this simple service also can be interpreted in another way: as long as the fact that the applicant is older than 18 years can be proved, and assuming that the applicant may **never** be a member (even after the service), then after applying this service, the applicant is a member. The second interpretation makes nonsense of this sample service description, because the service description itself is self-defeating. The issue here is the lifetime of a service assumption, which assumption explicitly contradicts its associated service effect. As stated earlier, service assumption represent hypothetical guess that is believed in the lack of evidence to the contrary, thus it also acts as one of the consistency conditions needs to be tested under specific contexts during the process of service composition. Typically, the consistency condition has to be met both before and after the service is applied. So, to avoid the self-defeating problem in the real application of the Web Service, based on the relation between service assumption and its associated service effect, we classify ser-

vice assumptions as:

1. **Transient Assumptions**: for any Web Service ws_i, if a contradiction can be inferred from the union of a service assumption a_i and its associated effect e_i, i.e. $a_i \cup e_i \models \bot$, then we refer to a_i as the *transient assumption*. When a transient service assumption plays the role of being the consistency condition in a service composition context, this condition only needs to be tested before the given service ws_i is applied, but **not after**.

2. **Persistent Assumptions**: for any Web Service ws_i, if there is no contradiction inferred from the union of a service assumption a_i and its associated effect e_i, i.e. $a_i \cup e_i \not\models \bot$, then we refer to a_i as the *persistent assumption*. When a persistent service assumption plays the role of being the consistency condition, this condition has to be tested both **before** and **after** the given service ws_i is applied.

3.3 Example of Using Assumption

In a service composition context, the service assumption can be viewed as a hypothesis. However, this hypothesis is about *individuals* rather than the *terminology*, where terminology is about how concepts or roles are related to each other in a given application domain and individuals are instances of classes or properties. Terminology represents the characteristics of the world, for instance, MasterCard is always subclass of Credit Card, while the facts about individuals represent our current state of knowledge that may change over time, for instance, a particular MasterCard may have expired. The reason to exclude the usage of terminology as the assumption is intuitive, because the terminology in ontologies is used to model the world as we know it.

Moreover, the proposed service assumption can represent two types of different knowledge in the service composition context. Let x, y, z denote either a variable, an OWL individual or an OWL data value, $\mathcal{C}$ denote an OWL class description and $\mathcal{P}$ denote OWL property, then we have:

1. Concept assumptions $\mathcal{C}(x)$, which asserts x is an instance of the OWL class description $\mathcal{C}$.

2. Property assumptions $\mathcal{P}(y, z)$, which asserts z is value of the OWL property $\mathcal{P}$ for y.

The proposed extensions to the current OWL-S make it possible to capture the various assumptions of the service domain. We also propose to use SWRL expressions to represent OWL-S assumptions, thus we can use the expressive power of rules to facilitate service conflict reasoning. 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. The example is as follows:

```
<process:hasAssumption>
<expr:SWRL-Condition rdf:ID="
   DriveCarInProperWay">
<rdfs:label>notDriveOn(car,
   roadCondition) & notDriveOn(car,
   anotherRoadCondition)
</rdfs:label>
<rdfs:comment>Typically this condition
   should also include more road
   conditons the car cannot drive on,
to keep this example simple, all other
   details are left out for this
   example.
</rdfs:comment>
<expr:expressionBody rdf:parseType="
   Literal">
<swrl:AtomList>
<rdf:first>
<swrl:IndividualPropertyAtom>
<swrl:propertyPredicate rdf:resource="#
   NotDriveOn" />
<swrl:argument1 rdf:resource="#cityCar"
    />
<owlx:Individual owlx:name="Dune" />
</swrl:IndividualPropertyAtom>
</rdf:first><rdf:rest>
<swrl:AtomList>
<rdf:first>
<swrl:IndividualPropertyAtom>
<swrl:propertyPredicate rdf:resource="#
   NotDriveOn" />
<swrl:argument1 rdf:resource="#cityCar"
    />
<owlx:Individual owlx:name="
   Unsealed_Road" />
</swrl:IndividualPropertyAtom>
</rdf:first>
<rdf:rest rdf:resource="http://www.w3.
   org/1999/02/22-rdf-syntax-ns#nil" />
</swrl:AtomList>
</rdf:rest>
</swrl:AtomList>
</expr:expressionBody>
</expr:SWRL-Condition>
</process:hasPrecondition>
```

Example of Using Service Assumption

The example is written by using SWRL syntax which extends the abstract syntax of OWL-S described in the OWL Semantics. Unfortunately, rules written in SWRL are not particularly human-readable. Thus the example is provided here to explain this abstract syntax. 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 $\psi_1 \wedge \ldots \wedge \psi_n$. Using this syntax, a rule states that the

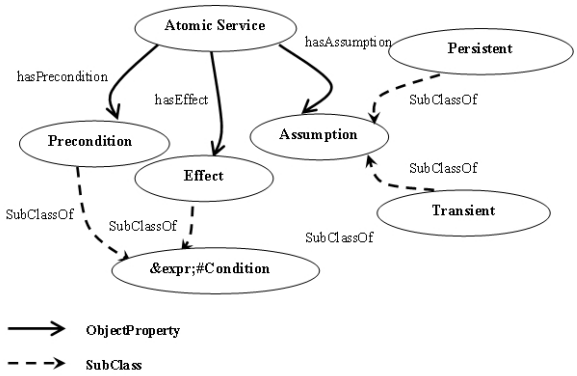

Figure 1. Extened Atomic Service

composition of "city car not drive on dune" and "city car not drive on unsealed road" properties implies the "DriveCarInProperWay" property would be written:

$\neg DriveOn(?cityCar, dune) \wedge$
$\neg DriveOn(?cityCar, unsealedRoad)$
$\Rightarrow DriveCarInProperWay(?cityCar)$

According to the example, the antecedent of the rule consists of two primitive assumptions, i.e.

$\neg DriveOn(?cityCar, dune)$
$\neg DriveOn(?cityCar, unsealedRoad)$,

and the consequent of the rule is

$DriveCarInProperWay(?cityCar)$

Since we have defined that service assumptions can only contain OWL *individuals*, possibly with variables, an assumption expression becomes equivalent to a conjunctive query. Informally, the example can explained as: if both "city car may not drive on dune" and "city car may 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. In this example, the contrary information will be "city car drives on dune" or "city car drives on unsealed road".

In this Section, we have extended the OWL-S to a richer service description representation schema by introducing the service assumption (See Fig 1) and have explained the semantics of the service assumption. To adopt the service assumption in more flexible way, we also further classify the service assumption into two categories: transient assumptions and persistent assumptions. This proposed extensions to current semantic web service description aim to bridge the gaps between the semantic service descriptions and multiple operational domains. The goal of adding service assumptions as one of the functional properties of service description is to provide the mechanisms needed to enable Web services applications:

- to be more flexible and intelligent. The flexibility and intelligence result from default service assumption which has the defeasible nature of commonsense inference.
- to be executed in a consistent manner. When Web Services execute in an open-ended environment, uncertainties can easily lead to conflicts. Service Assumptions attempt to make precise statement about the intended behavior of the Web Service and its environment. The more accurate and precise service description of the problem, the more reliable the decisions we make.

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 $\{cat_1, cat_2, \ldots, cat_n\}$. cat_i is a tangible areas split from the service registry, for example downloadable Multimedia.
- CAT is the set of all service categories $cat_i \in CAT$, $cat_i \in WS$, $cat_i = \{ws_1, \ldots, ws_m\}$.
- 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 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:

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

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 [9]. 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 a 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 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:

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

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

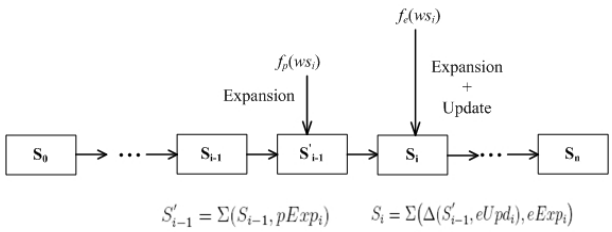

Figure 2. Generic State Transition Operators

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 [6] 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. $S_i = \Sigma\big(\Delta(S^{'}_{i-1}, eUpd_i), eExp_i\big)$

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 Σ 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 Δ 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 [18], 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. However, 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 parities.

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 $ws_i =< p_i, e_i, a_i >$, which is selected by the service selection function and has participated in a service composition. There are two cases, in which we refer to a service assumption a_i as an outdated assumption. The first case is very simple, as defined in Section 3.2. Based on the relation between e_i and a_i, service assumptions have been classified as transient assumptions and persistent assumptions. If a contradiction can be inferred from the union of a service assumption a_i and its associated effect e_i, i.e, $a_i \cup e_i \models \perp$, then a_i is classified as a transient assumption. For any transient assumption a_i, to avoid nonsensical service descriptions (due to the problem of self-defeating), after its associated effect e_i is applied to the current state of knowledge, a_i is outdated, i.e. we only need to check for consistency beforehand for a transient assumption.

In the second case, the effect e_i of service ws_i is a set of sentences, such that $e_i = \{e_i^1, \ldots, e_i^n\}$. We define φ as the negation of the service effect e_i, if $\{\neg e_i^1, \ldots, \neg e_i^n\} \subseteq \varphi$. Here, we use $\neg e_i \subseteq \varphi$ to denote that φ is the negation of the service e_i. If $\neg e_i \subseteq \varphi$ and φ 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*. Formally, the assumption is outdated, if $\forall x \in e_i, \exists j > i$ such that $\neg x \in Cn(S_j)$, where $Cn(S_j)$ denotes logical closure of S_j. 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, thus its status will be set to *inactive*. 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 [15, 17], 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 ser-

vice 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 $S_i \cup \Pi(\mathcal{M}) \not\models \perp$

A state transition $t = \langle S_{i-1}, ws_i, S_i \rangle$ is called legal, if ws_i is a precondition satisfied service with respect to S_{i-1}, the conjunctions of the set of all current active service assumptions is consistent and there is no contradiction which can be inferred from the corresponding set of active assumptions with respect to the new state S_i. However, the building of consistent value-added services on a heterogeneous environment is not a trivial task, we have to take in the consideration that

- the current set of service assumptions must be continually updated over time to incorporate new knowledge during this process.
- the conclusions which have been drawn must sometimes be revoked.
- corrections must be soothly accommodated to its corresponding assumptions.

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 for it 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.

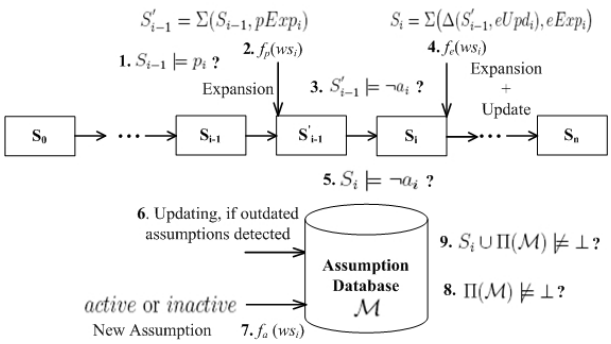

Figure 3. Reasoning Process

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, beforehand checking will be performed by means of issuing the query $S'_{i-1} \models \neg a_i$. If $\neg a_i$ is entailed by the knowledge of state S'_{i-1}, which means that applying service ws_i lead to an inconsistency with its service assumption. On the other hand, if $\neg a_i$ cannot be entailed by the knowledge of state S'_{i-1}, we say the beforehand checking is successful. This step is shown as operation step "3". After the successful beforehand checking, effect e_i is applied to the current state to simulate the action. As we mentioned before, the effect e_i may expand and update the knowledge of the current state, which is shown as the operation step "4". This process can be presented as generic state transition operation as we defined in page 6.

After the effect e_i is applied to the current state of knowledge, for this new state of knowledge S_i, it is the time to perform the afterward checking, which could distinguish the type of service assumption a_i. As defined in Section 3.2, based on the relation between e_i and a_i, the service assumptions has been classified as transient assumption and persistent assumption. Because the beforehand checking

must have been successful before reaching to this step, i.e. $S'_{i-1} \not\models \neg a_i$, after applying the e_i to the current state of knowledge, if the $\neg a_i$ is entailed by the new state S_i, i.e, $S_i \models \neg a_i$, which indicates that for a Web Service $ws_i = \{p_i, e_i, a_i\}$, $a_i \cup e_i \models \perp$. In this case, the a_i is classified as the transient assumption, otherwise, the a_i will be classified as a persistent assumption. This step is shown as operation step "5".

One of the main features in this proposed framework is the ability to describe various service assumptions and support 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, the status of all outdated assumptions will be set to $inactive$, which is shown as operation step "6". After updating the assumption database $\mathcal{M}$, the service assumption a_i is added to the assumption database $\mathcal{M}$. Based on the operation of step "3" and "5", we have completed the assumption type identification checking. If the service assumption belongs to the type of persistent assumption, then, initially, the status of this new service assumption a_i is set to be $active$, while if this new service assumption is a transient assumption, its status will be set to inactive. which is shown as the operation step "7".

Service assumptions are made about things that may not specifically be known during the process of service composition. Thus what the service assumptions represent is the environment of an underlying service composition. Clearly, the combination of the environment and the current knowledge state uniquely identify a service composition context. 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. 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 "8". 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 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 "9" . 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 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.

7 Conclusions

OWL-S [1] is a formal language which aims to provide precise and rich declarative specification of a wide variety of properties about Web Services in order to support automation of a broad spectrum of activities across the Web Service life cycle, such as discovery, selection, composition, negotiation and contracting, invocation and monitoring of progress. In the current version of OWL-S, the vast majority of efforts and techniques focus on the modeling and specification of the Web Service alone. Certainly, the declarative specifications of the prerequisites, consequences of application of individual services and data flow interactions need to be defined precisely and related to each other. In current version OWL-S, these properties of services have been defined by means of $\langle I, O, P, E \rangle$. However, in the absence of these essential service assumptions about a changing environment, the service composition specification offered by OWL-S is incomplete or inaccurate. Thus, in parallel, the assumptions made about a changing environment also need to be explicitly represented and documented as an indispensable part of service composition specification.

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:

- accurately describing the service composition environment, in which most instances of a concept generally have some property, but not always.

- presenting the hypothetical guesses about incompleteness and uncertainty.

- 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 service 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. This proposed work also leaves many opportunities for future improvements, which include:

1. Moving beyond sequential service composition, how the underlying service assumptions can be used to deal with incomplete knowledge and uncertainty in distributed parallel processing.

2. To facilitate the further automation of web service composition, it is desirable to have priorities among various service assumptions. These priorities indicate the different levels of the preference that the service requester would have on any given service composition. With these explicitly specified priorities, the planning agents may be able to re-compile their knowledge in response to conflicting information or partial failures.

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] Easterbrook.S. M. 1991. Handling Conflict Between Domain Descriptions With Computer Supported Negotiation. Knowledge Acquisition: An International Journal, Vol 3, No 4, pp 255-289.

[6] Golden, K., Etzioni, O. & Weld, D. 1996. Planning with Execution and Incomplete Information, UW Technical Report TR96-01-09, February 1996.

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

[8] R.J. Hall, Open Modeling in Multi-stakeholder Distributed Systems: Model-based Requirements Engineering for the 21st Century, in 2002 Workshop on the State of the Art in Automated Software Engineering

[9] 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.

[10] D. McDermott and AIPS'98 Committee. PDDL the planning domain definition language. Technical report, Available at: www.cs.yale.edu/homes/dvm, 1998.

[11] M.P. Papazoglou and J. Yang, Design Methodology for Web Services and Business Pro- cesses, Procs. of the 3rd VLDB-TES Workshop, August, Hong Kong, Lecture Notes in Computer Science Vol. 2444, Springer, 2002

[12] Paulo F. Pires, Mário R. F. Benevides, and Marta Mattos, Building Reliable Web Services Compositions, Net.Object Days - WS-RSD'02, page 551-562, 2002

[13] Putnam, L. L., and M. S. Poole (1987) Conflict and Negotiation. In L. W. Porter, Ed., Handbook of Organizational Communication: An Interdisciplinary Perspective, pp. 549-599, Newbury Park: Sage.

[14] Reiter R. On Reasoning by default, Proceedings of the theoretical issues in natural language processing-2 July 1978

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

[16] Robbins, S. P. Organizational Behavior: Concepts, Controversies and Applications. Englewood Cliffs, NJ: Prentice-Hall. 1989

[17] 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.

[18] M. Smithson. Ignorance and Uncertainty. Emerging Paradigms. Springer Verlag. New York. NY. 1989