DISCOVERING SEMANTIC RELATIONS BETWEEN WEB SERVICES USING THEIR PRE AND POST-CONDITIONS

by

LIN LIN

(Under the Direction of I. Budak Arpinar)

ABSTRACT

Discovering and assembling individual Web services into more complex yet new and more useful Web processes has received significant attention from academia recently. In this thesis, we explore using pre and post-conditions of Web services to enable their automatic composition. Also, we present a novel technique for discovering semantic relations between pre and post-conditions of different services using their ontological descriptions. This enables determining services with complementary functions and generating a semantic Web of services. Our technique takes semantic similarity of pre and post-conditions into account and builds on our earlier work on discovering semantic relationships between interfaces (input and output) of Web services. A comprehensive classification of existing composition techniques is also included.

INDEX WORDS: Web Service, Discovery, Composition, Conditions and Semantics.

DISCOVERING SEMANTIC RELATIONS BETWEEN WEB SERVICES USING THEIR PRE AND POST-CONDITIONS

by

LIN LIN

Bachelor of Science, The University of Georgia, 2002

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2005

© 2005

Lin Lin

All Rights Reserved

DISCOVERING SEMANTIC RELATIONS BETWEEN WEB SERVICES USING THEIR PRE AND POST-CONDITIONS

by

LIN LIN

Major Professor:	I. Budak Arpinar
Committee:	Krzysztof J. Kochut Eileen T. Kraemer

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2005

DEDICATION

This is dedicated to my loving parents Mr. and Mrs. Hongzhou Lin. Thank you for all your support and encouragement.

ACKNOWLEDGEMENTS

I am very thankful for the members of my thesis committee. Their helpful suggestions and reviews on my thesis are greatly beneficial. Especially, it is my major professor, Dr. Arpinar, to whom I owe the most overwhelming debt of gratitude for his support and his invaluable guidance for my thesis research. I would also like to thank Ruoyan Zhang, who started the research on interface matching approach for our previous work and kindly provided me with some inputs for my work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Motivation

Web services (WSs) extend the current Web from a distributed source of information to a distributed source of services. WSs are designed to provide interoperability between diverse applications, i.e., the platform and language independent interfaces of WSs allow the easy integration of heterogeneous applications. For example, the languages such as Universal Description, Discovery, and Integration [UDDI], Web Services Description Language [WSDL] and Simple Object Access Protocol [SOAP] define standards for service discovery, description and messaging protocols. UDDI, WSDL and SOAP are current industry standards developed for e-commerce. Services are described according to an XML schema, defined by the UDDI specification and registered by the service providers along with keywords for their categorizations. In complementary roles, WSDL and SOAP describe WSs as a set of endpoints operating on messages, and a protocol for exchange of these messages between the services respectively. As more and more WSs become available on the Web, discovery and assembly of individual Web services into more complex yet new and more useful Web processes has received significant attention from both industry and academia. Finding services by using a keyword-based search engine (e.g., expedia.com or google.com) or by looking them up in a WSs registry (e.g., a UDDI registry) is time-consuming and ineffective. Also, manual composition of discovered services is highly inconvenient. Thus, languages and techniques that can automate discovery and composition of services are needed. However, the current Web service description languages like WSDL suffer from their inability to provide constructs and concepts that enable reasoning about service behavior. Many researchers look for answers using the contributions of the semantic Web.

The semantic Web is also an extension of the current Web in which information is given well-defined meaning, consequently better enabling computers and humans to work in cooperation. This is realized by marking up Web content, its properties, and its relations, in a reasonably expressive markup language with a well-defined semantics. For example, the Resource Description Framework [RDF] is a general-purpose language for representing information on the Web. The RDF specifications provide a lightweight ontology system to support the exchange of knowledge on the Web. In our experiments we use ontologies specified in RDF. Another example is the Web Ontology Language [OWL], which is a W3C specification that supersedes the earlier DARPA Agent Markup Language [DAML+OIL]. OWL is an extension to XML and RDF enabling the creation and population of ontologies for any domain. Thus, ontologies are a key enabling technology for the semantic Web. They provide a shared and common understanding of a domain, which can be communicated across people and application systems.

The semantic Web aims to add machine-interpretable information to Web content in order to provide intelligent access to heterogeneous and distributed information by using ontology description languages. In a similar way, ontological concepts are used to define semantic Web services i.e., services supporting automatic discovery, composition, invocation and interoperation. As part of the DARPA Agent Markup Language program, OWL-S (formerly DAML-S), an ontology of services, is developed as a set of language features arranged in those ontologies to establish a framework within which the Web services may be described in this semantic Web context. This ontology describes three types of information for a service: service profile, service process and service grounding. The service profile provides a representation of properties and capabilities that can be used by a service requester to specify their needs and services providers to advertise their services for what is does. The process model describes how the service works. The grounding describes how an agent can access the service. In previous work, we primarily focused on the collection of inputs and outputs, described in service profile, for composition. In this work, we exploit the use of pre and post-conditions in service profile to enhance the quality of discovery during the service composition process.

1.2 Black Box Description for Web Service

Many WS description languages distinguish between elementary and complex WSs. Elementary WSs are simple input/output boxes, whereas complex WSs break down the overall process into sub-tasks that may call other WSs. Strictly speaking, such a distinction is wrong and may lead to mis-conceptualizations in a WS modeling framework. It is not the complexity of the WS that makes an important distinction. It is rather the complexity of its description or its interface that makes a difference [Fensel02]. We need to make an explicit distinction between an internal description of a WS and its external visible description. Often, the internal complexity of a WS reflects the business intelligence of a service provider, and its internal description describes the workflow of the service. Therefore, it is essential for the provider not to make it publicly accessible. In our work, we strictly limit ourselves to dealing with the external aspects of a service, i.e., its interface that can be accessed via a network. In this case, we refer to WSs as black boxes, i.e., we do not care about the internal aspects of how such a service is achieved.

In a black box description of service, first, a service has a name, i.e., a unique identifier to refer to it. Second, a service has some description to describe the goal a service can achieve. Third, service descriptions contain pre-conditions and post-conditions. The pre-condition is the condition that has to be true for the input in order for the WS to successfully execute. The post-condition is the condition that holds once the service has been executed successfully. Fourth, a service description describes the structure of its input data and output data. These are the input parameters that a service expects to receive for its execution and the output parameters that a service returns after its successful execution.

1.3 Previous Work

Developing efficient automatic discovery and composition techniques is among the most important challenges for the success of semantic Web services. Finding a suitable way to put these two features together has become one of the key points to convert the Web into a distributed source of computation, as they enable the location and combination of distributed services to provide a required functionality. To contribute towards this goal, we developed an Interface-Matching Automatic (IMA) composition technique earlier [Arpinar04a & Arpinar04b]. The possible compositions are obtained by checking semantic similarities between interfaces (inputs and outputs) of individual services without any predefined template and user's involvement in specification and adaptation. An optimum composition which can best satisfy a user's needs considering the semantic similarity and quality is selected. This technique also addresses issues, such as composition of services with multiple inputs and outputs parameters and Quality of Service (QoS). More details about this work can be found in Chapter 3.

Our experiments show that without functionality constraints, IMA technique is more appropriate for the information-retrieval services, which always return relatively simple results based on the user-supplied inputs [Zhang03 & Arpinar04a]. This is mainly due to the fact that WSs with the same interfaces could have different functions and the difficulty of mapping of input and output parameters for many services.

1.4 Contributions

In order to overcome the problems we encountered in our previous work (described in Section 1.3), we propose a discovery technique based on pre and post-conditions of WSs. By exploiting the use of pre and post-conditions, we can distinguish services that provide different functionalities even though they have the same interfaces. Also, some services that are not able to be described by just inputs and outputs can be modeled using pre and post-conditions.

Currently OWL-S provides concepts like pre-conditions and post-conditions (effects), but the formal specification for expressing pre and post-conditions is not there yet. We propose that the pre or post-condition of a service can be expressed as a subject-predicate-object triple (or a set of triples) and such a triple can be semantically described using RDF (Resource Description Framework) [RDF]. We believe that the pre and post-conditions can semantically express the capabilities of services in a simple manner if they are expressed as a set of RDF triples. In this work, we present a novel technique to identify possible relations between pairs of WSs by checking semantic similarities between their pre and post-conditions. Using a *condition ontology*, we can discover relations between two services even the conditions don't match each other syntactically. This technique also addresses the issue of relaxed matching in the sense that pre-condition of one service can be satisfied by two or more WSs.

The rest of this thesis is organized as follows: Chapter 2 reviews the related work on WS composition techniques. Chapter 3 reviews our earlier work on IMA technique. Chapter 4 presents the technique for discovering semantic relations between pre and post-conditions of WSs. Chapter 5 describes the system architecture and experiments, and finally Chapter 6 provides the conclusion and future work.

CHAPTER 2

RELATED WORK

Several techniques have been proposed for realizing WS composition. Based on how they model and implement WS composition, we classify the exiting techniques into several categories, such as template-based techniques, interface-based techniques, logic-based techniques, ontology-driven techniques, quality-driven techniques, automata-based techniques and Petri net-based techniques. The following criteria are used to conceptually evaluate different techniques:

- **Flexibility:** The ability to compose large varieties of applications from a set of available components.
- **Adaptability:** The ability to compose an application that best matches the given user preference and context.
- **Extensibility:** The ability to allow service providers to implement new services with as little restraint as possible.
- **Scalability:** The ability to compose an application efficiently in a distributed environment.
- **Usability:** The ability to require as little user involvement as possible in a WS composition process. Also, the ability to provide such composition requests and results that are easy for users to generate and understand.

2.1 Template-based Techniques

Template-based techniques compose an application from a given service template. A service template defines types or rules of the components required for composing an application, as well as structure of the application (i.e., control-flow). A user can choose a service template from a repository or create it him/herself. The requested application is composed through selecting the components specified in the template and combining them according to the structure described in the template.

An adaptation of these template-based techniques, called process-driven techniques, is emerging as a promising approach to integrate business applications within and across organizational boundaries. In this approach, individual WSs are federated into composite WSs whose business logic is expressed as a process model, and a process template is often used to describe this process model. The process model underlying a composite service identifies the functionalities required by the services to be composed (i.e., the tasks of the composite service) and their interactions (e.g., control-flow, data-flow, and transactional dependencies). Component services that are able to provide the required functionalities are then associated to the individual tasks of the composite services and invoked during each execution of the composite service [Zeng03]. In eFlow [Casati00] the process is modeled as a graph that defines the control and data flow. MWSCF [Sivashanmugam03] captures semantics of the activities in the process. The activities are not bound to WS implementations but defined using semantic descriptions.

- **Advantages:** The template-based systems provide some flexibility and adaptability as different applications may emerge from a single template by choosing different components.
- **Disadvantages:** The adaptability of the template-based systems is limited since they cannot compose the applications whose templates are not available. Also, creating service templates

requires technical knowledge and experience, thus is not a user-friendly option. In addition, many of the existing template-based systems, such as eFlow [Casati00], adopt a centralized architecture where centralized servers store and process service templates. This centralized architecture lacks scalability, efficiency and robustness.

2.2 Interface-based Techniques

Interface-based techniques use interface information of WSs (i.e., a set of inputs and outputs) instead of service templates in order to compose an application. Interface information of a service consists of operations, each of which is a set of inputs and outputs. A user requests an application by submitting interface information of the application s/he needs. The requested application is composed through combining services such that the combination of the services accepts the requested inputs and generates the requested outputs.

- **Advantages:** The interface-based techniques provide higher adaptability than the template-based systems because any application can be composed as long as there exist proper component services and an appropriate request is provided.
- **Disadvantages:** The interface-based techniques have limited flexibility since certain services cannot be represented through a set or inputs and outputs. For example, a service that sends a text message to a specified email address cannot be modeled as a set of inputs and outputs since it does not output any data. Also, services may have more than one input and output parameters, and their interfaces may not match syntactically. [Zhang03 and Arpinar04a] addressed this issue by proposing ontology-driven techniques, which will be discussed in Chapter 3. Furthermore, interface information usually provides little or no semantic

information about the internal functionality of WSs, which degrades the flexibility and usability.

2.3 Logic-based Techniques

Logic-based techniques extend the interface-based approach by usually adding first-order formula as pre and post-conditions into interface information. A user requests an application by submitting a formula representing the logic that must be satisfied by the application. The requested application is composed through combining components such that the conjunction of the logics specified in the components is equivalent to the logic specified by the user. SWORD [Ponnekanti02] and SHOP2 [Wu03] follow this approach.

- **Advantages:** The logic-based techniques provide higher flexibility than the interface-based systems since the logic-based systems support the applications where interface information may not be sufficient.
- **Disadvantages:** The logic-based techniques do not achieve extensibility because they require all service providers and users to agree and share the same logical definition. This requirement prohibits a service provider from implementing a new type of service if the service cannot be represented in the existing logical definition, leading the logic-based systems to be application domain specific.

2.4 Ontology-driven Techniques

Ontology is a formal representation of the specification of concepts and relationships of different concepts that exist for different communities. Ontology-driven techniques extend the interface-based approach by bridging the concept gaps in interface parameters and other parts of the descriptions of services. For example, MWSAF (METEOR-S Web Services Annotation Framework) was designed to annotate WSDL files with relevant ontologies [Patil04]. MWSCF (METEOR-S Web Services Composition Framework) makes use of ontologies in template definition to allow much richer description of activity requirements [Sivashanmugam03].

- **Advantages:** Ontologies provide high extensibility by allowing a service designer to define his/her own ontology. Since different ontologies can coexist by supplying a mapping between them, ontology-based approaches are also suitable for a distributed environment.
- **Disadvantages:** The use of ontologies for matching interface parameters leads to different degree of similarity matches. For example, if the output parameter of the former service subsumes the input parameter of the succeeding service, the properties of the parameters could be partially satisfied. Thus, this kind of weak matches may not always guarantee the correctness of the composition.

2.5 Quality-driven Techniques

The quality-driven techniques extend the process-driven techniques by adding the selections of component services based on a set of quality criteria during execution of a composite service. The number of services providing a given functionality, although with different levels of pricing and quality, may be large and constantly changing. Consequently, it may be inappropriate to compose composite services that require the identification of the exact services at the design-time. A WS can be selected during execution based on some operational metrics of non-functional properties, such as reliability, execution price, duration, reputation, availability, and security. The runtime selection of component services during the execution of a composite service has been put forward as an approach to address this issue. [Zeng03] proposes an extensible multi-dimensional WSs quality model. [Cardoso02] develops a QoS (Quality of Service) model for workflow components.

- **Advantages:** Quality-driven techniques take into account constraints and preferences set by the user. They aim to optimally select component services during execution of a composite service. Thus they provide higher adaptability than the process-driven techniques.
- **Disadvantages:** Quality-driven techniques require increasing computational cost of selecting services due to the increase of the number of tasks in a process and the number of candidate services. Furthermore, in order to evaluate services according to certain standard, global QoS metrics or QoS ontologies need to be defined and agreed upon, which may be a challenging task to accomplish.

2.6 Automata-based Techniques

The automata-based techniques use Finite State Automata (FSA) to model a WS composition. FSA are widely known as a simple but powerful formalism, which allow us to model the behavior of a system as a sequence of transitions. In [Bultan03] a service is modeled as a Mealy machine, with input and output messages, and a queue is used to buffer messages that were received but not yet processed. [Berardi03] has developed a composition model involving activity-focused FSA. One input to this approach is a set of descriptions of component WSs, each given as an automaton. The second input is a desired global behavior, also specified as an automaton, which describes the possible sequences of activities. The output is a subset of the component services, and a mediator that will coordinate the activities of those services through a form of delegation.

- **Advantages:** Automata-based techniques provide some flexibility and adaptability as long as proper component services are available and can be modeled as FSA.
- **Disadvantages:** Automata-based techniques have limited extensibility since they required all service providers and users to agree and share the same modeling specification. Also, modeling services as automata requires technical knowledge and experiences, thus is not a user-friendly option.

2.7 Petri net-based Techniques

These techniques use Petri nets to model processes. Petri nets [Peterson81] are well founded process modeling technique that has formal semantics. A Petri net is a directed, connected, and bipartite graph in which each node is either a place or a transition. Tokens occupy places. When there is at least one token in every place leading to a transition, the transition is enabled. Enabled transition may fire by removing one token from every input place, and depositing one token in each output place. A WS behavior can be mapped into a Petri net. At any given time, service can be in one of the following states: not instantiated, ready, running, suspended or completed. After a Petri net is defined for each service, composition operator, such as sequence, selection and iteration, are used to perform composition. [Narayanan02] encodes WS descriptions in a Petri net formalism and provides decision procedures for WS simulation, verification and composition. [Hamadi03] proposes a Petri net-based algebra for composing Web services. The formal semantics of the composition operators is expressed in terms of Petri nets by providing a direct mapping from each operator to a Petri net construction. Thus, any service expressed using the algebra constructs can be translated into a Petri net representation. A colored Petri net-based approach proposed by [Yi04] captures both complex conversation protocols and process compositions of WSs.

- **Advantages:** The Petri net-based approaches provide some adaptability since any application can be composed as long as there exist proper component services.
- **Disadvantages:** These approaches have limited extensibility and usability due to the complexity of modeling and composing services as Petri nets.

CHAPTER 3

SEMANTIC RELATIONS BETWEEN INPUTS AND OUTPUTS

This chapter describes our earlier work on investigating the semantic relations between inputs and outputs of WSs by checking their semantic similarities. We developed an Interface-Matching Automatic (IMA) composition technique, described in [Arpinar04b], which aims for generation of complex WS compositions automatically. This requires capturing user's goals (i.e., expected outcomes), and constraints, and matching them with the best possible composition of existing services. Therefore, inputs and outputs of the composite service should match the user-supplied inputs, and expected outputs, respectively. Furthermore, the individual services placed earlier in the composition should supply appropriate outputs to the following services in an orchestrated way similar to an assembly line (i.e., pipe-and-filter) in a factory so they can accomplish the user's goals.

In IMA, we navigate the WS network to find the sequences starting from the user's input parameters and go forward by chaining services until they deliver the user's expected output parameters. The composition terminates when a set of WSs that matches all expected output parameters is found, or the system fails to generate such a composition of services.

The goal of this technique is to find a composition that produces the desired outputs within shortest execution time and better data-flow (i.e., better matching of input and output parameters). The composition starts from the service that provides one or more input parameters specified by the user. If this WS does not produce all of the expected outputs, more WSs need to be found to provide the expected outputs. This process continues until we find a sequence of WSs that will produce the expected composition outputs from the input specified by the user.

Figure 3.1 shows a sample process ontology involving relations representing input and output parameter matching. Nodes represent services and edges connect services if the output of a service can be "feed-into" the input of a service. Edges shown with dashed line connect parameters that do not match exactly yet are semantically equivalent. In the figure, different service outputs can feed into other service inputs. For example, Service 6 requires two input parameters, one of which can be provided by either Service 1 or Service 3 and the other comes from Service 4.

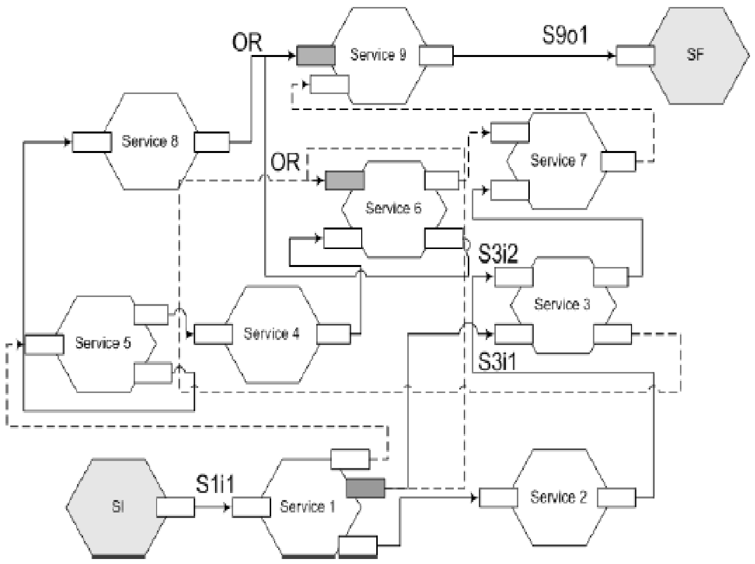


Figure 3.1 A Process Ontology Example

In an example scenario, the user provides input parameter S1i1 and expects the output S9o1 as indicated in the graph. The composition goal is to find a shortest sequence of services from

Service 1 to Service 9. In this graph the source node SI and SF represent the virtual initial and final services respectively, which are added for computing convenience. The weight of every edge is calculated using a function of quality rate and semantic similarity value [Zhang03 & Zhang04]. To assign weights using quality, five generic QoS criteria can be used: (1) execution prices, (2) execution duration, (3) reputation, (4) reliability, and (5) availability [Zeng03].

The trade-off formula weighting quality criteria versus similarity is defined by the following formula, with a user adjustable parameter λ:

$$W = (1-\lambda) * \textit{quality rate} + (\lambda) * \textit{similarity value.}$$

We considered four cases to check similarity (i.e., matching) of an output and input parameter from the same ontology:

- **Case 1:** If they are same, their similarity is maximal.
- **Case 2:** If output parameter of the former service is subsumed by the input parameter of the succeeding service, this is the second best matching level. The similarity value depends on their distance in the ontology.
- **Case 3:** If the output parameter of the former service subsumes the input parameters of the succeeding service, the properties of the parameters could be partially satisfied.
- **Case 4:** When two parameters have no subsumption relation or they are from different ontologies, the similarity value can be obtained by using Tversky's feature-based similarity model [Cardoso02], which is based on the idea that common features increase the similarity of two concepts, while feature difference decreases the similarity.

However, our experiments show that sometimes IMA technique can fail to produce correct compositions due to the fact that some WSs, with same input and output parameters, provide quite different functionalities. Thus we have developed a (Human-Assisted Automatic)

HAA composition technique to help users in selecting appropriate WSs among a ranked list, and build a composition incrementally [Arpinar04b]. The first step is to consider all inputs by semantically matching them with all WSs that take one or more of them as input. A list of these WSs is provided to the user with the WSs ranked based on their similarity matching score. The user can select WSs s/he considers best for the desired composition. Furthermore, to facilitate a better selection process, each listed WS includes a description of its functionality as well as the output(s) it produces. Then the system determines whether all output parameters of the desired composition are produced by the services selected so far. If that is the case, the composition is completed. Otherwise the interactive composition process continues with more stages, or can be terminated by the user. Figure 3.2 shows the first stage of a composition for a vacation trip arrangement, where the user has selected two WSs so far.

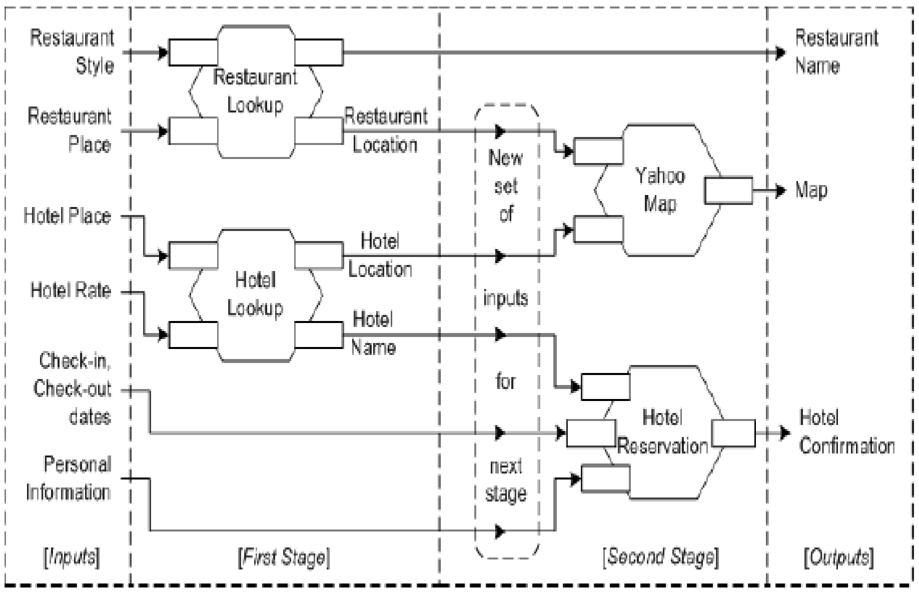


Figure 3.2 Stages of an Interactive Composition

For the second stage, a new set of input parameters is generated by the system. This set includes the input parameters considered and also includes all outputs of the WSs selected by the user. The user is given the option to discard elements in this new set of input parameters that may no longer be needed. The user may also mark some of the new input as "optional". This helps in the ranking of the list of WSs that will be shown to the user in subsequent stages. The list of ranked results is grouped by input parameters to facilitate selection when the list is large. In the example of Figure 3.2, *Restaurant Style*, *Restaurant Place*, *Hotel Place*, and *Hotel Rate* are no longer considered as inputs in next stages. The inputs *Check-in* and *Check-out dates* and *Personal information* are still considered; outputs *Restaurant Location*, *Hotel Location*, and *Hotel Name* from the selected services are now considered as inputs in next stages; and output *Restaurant Name* from one of the selected services is no longer considered as input (unless stated otherwise by the user), because it satisfies an output parameter *Restaurant Name*. The smaller dashed box highlights the new set of inputs for the next stage, and in this way the composition problem has been reduced. Figure 3.2 also shows the second and final stage of the composition.

CHAPTER 4

SEMANTIC RELATIONS BETWEEN PRE AND POST-CONDITIONS

The problem of composing autonomous WSs automatically to achieve new functionality is generating considerable interest in recent years in academia and industry as mentioned earlier. The prerequisite for correct and precise semantically driven discovery is provision of sufficient and well-defined knowledge. Thus, automatic services discovery and composition require an approach based on semantic descriptions, as the required functionality has to be expressed in a high-level and abstract way to enable reasoning procedures.

The required high-level functionality description can be viewed as the capability of the service. However, different services can provide the same capability, e.g., booking a flight and the same service can provide different capabilities, e.g., searching for a book or a CD through the same service. In this sense, capabilities must be described separately from specific service descriptions so that generic functionalities can be expressed [Lara03]. For example, several services offering the same functionality but with different inputs and outputs should be related to the same generic high-level capability. However, several services having same inputs and outputs but offering different functionalities should be distinguished from each other.

In order to compose services based on functionalities, there has to be a way to express the functionality of a service. Since pre and post-conditions can be used to define the capability of the service in terms of the information needed to use the service and the results of its invocation, functionalities of services can be expressed in terms of these conditions. So the problem of identifying the degree of interoperability of WSs based on semantic relations of their pre-conditions and post-conditions becomes very important during WS composition. Thus, our work targets the following problem: given a set of WSs, the semantic relations between pre and post-conditions of these services need to be established, and then a semantic network of services with complimentary functions can be constructed according to these relations. Therefore, the subsequent step of services composition according to a specific task at hand can be viewed as a path traversal problem.

4.1 Condition Ontology

As we have mentioned earlier, finding a suitable way to express pre and post-conditions semantically becomes a very important issue. It should be simple yet expressive enough for machine processing and capturing the functionality of services respectively. Current OWL-S specification provides some prior agreement on how a service is represented in its service profile. However, there are no existing standards for representing pre and post-conditions in current OWL-S specification. SWRL [SWRL] is a preliminary proposal for a rule language designed to express rules and constraints in the framework of the OWL language. Even though several proposals have been discussed for expressing conditions in many different formats, currently SWRL does not specify any such formats. It must be noted that SWRL is evolving and significant changes can be expected in the near future. The related work for specifying rules within the domain of semantic Web are initiatives like RuleML [RuleML] and DRS [DRS].

Since we want to remain within the framework of OWL-S in this work, the temporary solution for expressing pre and post-conditions as suggested in OWL-S is to specify them as variable terms. For example, to express a condition that a class is open for registration, a service provider can define this condition as a variable term *classOpen*, and another service provider can define the same condition as a variable term *courseAvailable*. If we treat these two variable terms in isolation from their context, they are two completely different terms even though they are defined to have same meaning. Therefore, we need to rely on semantic information to evaluate pre and post-conditions of services. Hence, we propose to model pre and post-conditions of a WS as two sets of RDF triples from a *condition ontology*. Each pre-condition or post-condition can be seen as a subject-predicate-object triple or a conjunction of triples. Each triple can be defined in the condition ontology that specifies the relation between subject and object as predicate.

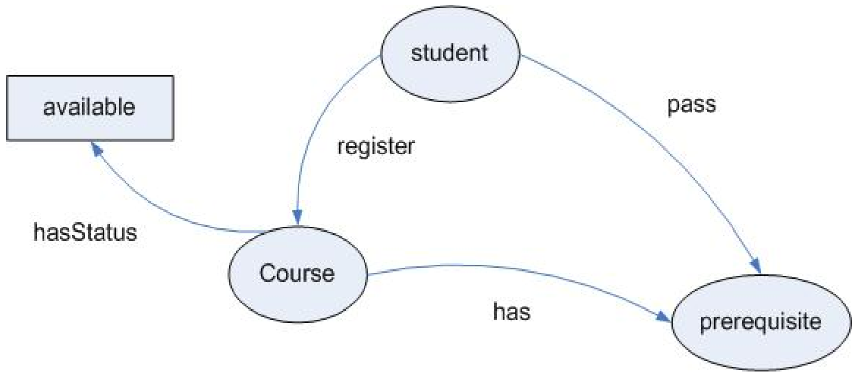


Figure 4.1 A Simple Example of a Condition Ontology

The Figure 4.1 depicts a very simple condition ontology that specifies relations among *student*, *course* and *prerequisite course*. This kind of condition ontology often need to model states or statuses of concepts, which usually are overlooked in many domain ontologies. Hence, it could be developed as an ontology stands by itself or as an extension to the domain ontology so that service developers can refer to the same ontology for specifying the pre-conditions and post-conditions while describing their services in service profile. Designing and developing this kind of condition ontology for any domain could be a part of the future work, and it is not in the scope of this thesis.

4.2 Modeling Pre and Post-conditions

The future realized condition ontology can be made available to service users and service providers in such a way that they can browse through the whole ontology quickly and visually to find suitable triples for expressing pre and post-conditions of the services they are requesting or developing respectively. Pre-conditions can be expressed as, but not limited to, high-level inputs to the service together with conditions over these inputs. High-level input means that more specific concepts in the ontology can be found to replace more abstract concepts in the input. For example, in stead of the availability of credit card information or bank information, users can indicate the availability of payment information as a pre-condition. In the condition ontology, the payment information is defined as a relatively abstract concept since more specific concepts, such as credit card information and bank information, can be found as its grandchildren. It is important to notice that the pre-conditions of a service are not independent of each other, as they all define the functionality. Post-conditions can also be expressed as, but not limited to, high-level results of the service execution together with conditions over these results. As with pre-condition, they cannot be considered independent, as the removal of one of them changes the functionality.

The problem of determining the relationship between two services can be addressed through discovering semantic relations between the pre and post-conditions of these services using the condition ontology. For example, as shown in Table 4.1, the pre-condition for a course registration service can be expressed as a set of triples: (*course hasStatus available, course has* *prerequisite, student pass prerequisite*), and the post-condition can be expressed as another triple: (*student register course*). Both pre and post-conditions in this simplistic example are defined at the schema level of the condition ontology shown in Figure 4.1. Hence, the pre-condition of this course registration service expresses that the course should be available, the course has a prerequisite course, and the student should pass the prerequisite course. The post-condition expresses that the student should register the course if the pre-condition is satisfied. The inputs and outputs can be defined as concepts in the domain ontology. The roles of input and output parameters in a service composition have been discussed in IMA technique earlier in Chapter 3, and we will leave them out in the rest of this thesis.

Table 4.1 A Course Registration Service

Web Service Name:	Course Registration Service
Description:	This service can be used by students to register courses.
Pre-condition:	(course hasStatus available, course has prerequisite, student pass prerequisite)
Inputs:	studentID, password, courseID
Post-condition:	(student register course)
Outputs:	confirmationMessage

4.3 Relationships among Web Services

We define two WSs to have a relationship as either these two services can be somehow plugged together to perform a value added service or one of the services can be substituted by the other. Let service S*m* and service S*n* be two services, and a relationship R between services S*m* and S*n* can be identified as follows:

- **R1:** Prerequisite Relationship (→)

$$Sm \rightarrow Sn$$

The prerequisite relationship means that one service has to finish before the other starts. Service S*m* has to finish before service S*n* starts. For example, the booking service has to be done before the payment service.

- **R2:** Parallel Relationship (//)

$$Sm \mathbin{//} Sn$$

Here services S*m* and S*n* can execute in parallel but the results of each service need to be combined to enable further execution.

- **R3:** Substitute Relationship (⇔)

$$Sm \Leftrightarrow Sn$$

Here service S*m* and service S*n* can be substituted with each other. The services S*m* and S*n* seem to provide the same functionality but they have some different attributes. For example, in the case of delivery service, service S*m* can be an air courier delivery service while service S*n* is a ground delivery service.

- **R4:** Include Relationship (Э)

$$Sm \ni Sn$$

The include relationship means that one service provides services that includes the services offered by the other. The service S*m* includes the service S*n*. For example, service S*m* can be an express delivery service that offers both ground and air delivery while service S*n* is a ground delivery service.

4.4 Semantic Relations between Conditions

As mentioned earlier, we model pre or post-condition as a conjunction of RDF triples. Each set of RDF triples in a pre or post-condition is simply called a condition. Thus, a WS S can be viewed as Cond*1*→Cond*2*, in which condition Cond*1* and condition Cond*2* represent the pre and post-condition of this service S respectively. The relationship between two services can be identified by checking the semantic relations between these conditions of WSs.

Let service S*m* and service S*n* be two services, and service S*m* can be represented graphically as *Condm1*→*Condm2* in Figure 4.2(a), and service S*n* can also be represented graphically as *Condn1*→*Condn2* in Figure 4.2(b). Several semantic relations between conditions can be identified as follows:

- Condition Cond*m2* **exactly matches** to condition Cond*n1*:

  $$Condm2 \rightarrow Condn1.$$

  In Figure 4.2(c), the exact match relation between conditions Condm2 and Cond*n1* is represented graphically, and services S*m* and S*n* have the prerequisite relationship (e.g., →).

- Condition Cond*m2* is semantically stricter than condition Cond*n1*. Figure 4.2(d) shows that condition Cond*m1* can be a **plug-in (PI) match** to condition Cond*n1*:

  $$Condm2\ \boxed{\text{PI}}\!\rightarrow Condn1.$$

In this case, services S*m* and S*n* have the prerequisite relationship. For example, condition Cond*m2* can specify the availability of payment by MasterCard only, and condition Cond*n1* can allow the availability of payment by all major credit cards.

- Condition Cond*m2* only partially satisfies condition Cond*n1* so that some other condition(s) are needed together with Cond*m2* to completely satisfy condition Cond*n1*. We say that condition Cond*m2* can be a **plus match** to condition Cond*n1* as shown in Figure 4.2(e):

  $$Condm2 \boxed{+}\!\rightarrow Condn1.$$

  Service S*m* has the parallel relationship (i.e., //) with some other services since their results together enable the execution of service S*n*. For example, service S*n* can be a protein search results comparison service that accepts two protein identification results from two different protein search services as its inputs and generates a results comparison report as its output. Service S*m*, one of two protein search services, can only partially satisfies service S*n*. These two protein search services have the parallel relationship.

- Condition Cond*m2* compliments condition Cond*n1*. For example, condition Cond*m2* can specify that book is available to be sold, and condition Cond*n1* can specify that book is available to be bought. Buying denotes an action that is compatible to selling. While [Stollberg05] defines this compatible relation in their action-resource ontology, we can define this kind of relations in condition ontology. We say that condition Cond*m2* can be a **complimentary match** to condition Cond*n1*:

  $$Condm2 \boxed{\text{CP}}\!\rightarrow Condn1.$$

  In this case, services S*m* and S*n* also have the prerequisite relationship.

Figure 4.2(f) shows that services S*m* and S*n* have the substitute relationship (i.e., ⇔) since condition Cond*n1* exactly matches condition Cond*m1*, and condition Cond*n2* exactly matches condition Cond*m2*. Figure 4.2(g) depicts the include relationship (i.e., Э) between services S*m* and S*n*. In this case, condition Condn1 is a plug-in match to condition Cond*m1* while condition Cond*n2* exactly matches to condition Cond*m2*. For example, as mentioned earlier, condition Cond*m1* can specify that express delivery, including air and ground deliveries, is available while condition Cond*n1* expresses that only ground delivery is available. Post-conditions of two services Cond*m2* and Cond*n2* can be the same as they both express the effect of executing a delivery service.

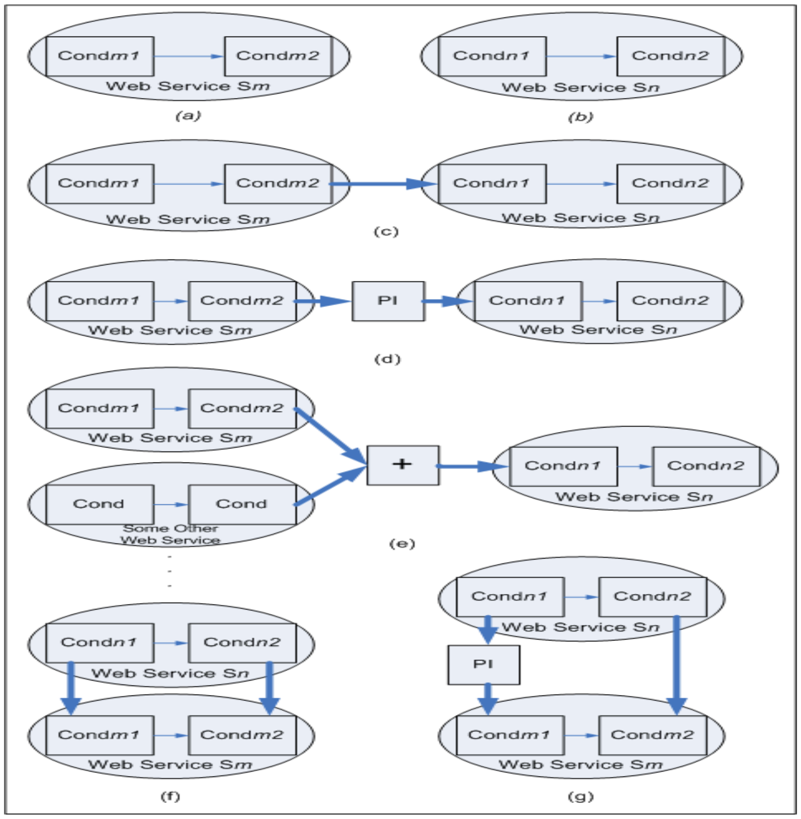


Figure 4.2 Semantic Relations between Conditions

4.5 Discovery of Semantic Relations between Conditions

As discussed earlier, we define a condition as a set of RDF triples, and each RDF triple is made up of three components: subject, predicate, and object. The degree of similarity between two conditions is assessed through comparing similarity between triples of these two conditions. Thus, the semantic relations, if they exist, between two conditions can be discovered depending on the similarity value. In the following, we explain the relation discovery algorithm in several steps:

1) evaluating the similarity of two triples,
2) calculating the similarity value between two conditions,
3) identifying the semantic relation between two conditions using similarity value, and
4) identifying the semantic relations between pre and post-conditions among services.

Table 4.2 Algorithm for Finding Similarity Value between Two Triples

```
01. procedure similarity_bw_triples(T1, T2)
02.   if similarity of T1 and T2 falls into Case 1
03.       return the maximal similarity value
04.   else if similarity of T1 and T2 falls into Case 2
05.       return the second level similarity value // the second level similarity value is less then
                                                     the maximal similarity value
06.   else if similarity of T1 and T2 falls into Case 3
07.       return the third level similarity value // the third level similarity value is less then the
                                                    second level similarity value
08.   else if similarity of T1 and T2 falls into Case 4
09.       return the fourth level similarity value // the fourth level similarity value is less then the
                                                     third level similarity value
10.   else if similarity of T1 and T2 falls into Case 5
11.       return the fifth level similarity value // the fifth level similarity value is less then the
                                                    fourth level similarity value
12.   else
13.       return the minimum similarity value // the minimum value means not similar
```

The first step is to evaluate the similarity of two triples. Each component from the first triple is compared to the corresponding component from the second triple, and a similarity value is assigned depending on the type of similarity. Since each component in a triple is defined in a condition ontology, our previous method of checking similarity of an output and input parameter described in Chapter 3 can be adopted to check similarity of a pair of corresponding components between two triples. For simplicity, we consider five cases of similarity measure for a pair of triples:

- **Case 1:** If all three corresponding components between two triples are same, the similarity value for these two triples is maximal.
- **Case 2:** If one or more components from the first triple are subsumed by the corresponding component(s) from the second triple, their similarity value is the second best. For example, the two triples, *book hasAuthor person* and *mediaObject hasAuthor agent*, fall into this case since *book* from the first triple is subsumed by *mediaObject* from the second triple, and *person* from the first triple is also subsumed by *agent* from the second triple. In the condition ontology for this example, *book* is defined as a subclass of *mediaObject*, and *person* is defined as a subclass of *agent*.
- **Case 3:** If one or more components from the first triple subsume the corresponding component(s) from the second triple, their similarity value is the third best. In this case, the order of two triples from the previous example is switched so that two components from the first triple, *mediaObject hasAuthor agent*, subsume the corresponding two components from the second triple, *book hasAuthor person*.
- **Case 4:** If at least one component from first triple is subsumed by the corresponding component from second triple (case 2) while the other one or more components from first

triple subsume the corresponding component(s) from second triple (case 3), their similarity value is the fourth best. An example for this case can be two triples: *book hasAuthor agent* and *mediaObject hasAuthor person*.

- **Case 5:** If they have no subsumption relation, the similarity value can be obtained by using Tversky's feature-based similarity model [Cardoso02]. Usually, in this case, their similarity value is very small.

It must be noted that in our prototype we do not implement the case of compatibility of components between two triples specified in complimentary match relation in previous Section 4.4, we leave it as a part of out future work. Suppose there are two triples T1 and T2, the algorithm for finding similarity value between two triples is described in detail in Table 4.2: a similarity value is returned according to which case the similarity of two triples falls into.

Table 4.3 Algorithm for Finding Similarity Value between Two Conditions

```
01. procedure similarity_bw_conditions(cond1, cond2)
02.   for i = 1 to m // m is the number of triples in cond1
03.       for j = 1 to n // n is the number of triples in cond2
04.           if Tj is not yet marked similar to any triples in cond1 // Tj is the jth triple in cond2
05.               value = similarity_bw_triples(Ti, Tj) // Ti is the ith triple in cond1
06.               record the (value, j) pair into a list L
07.           end if
08.       end for
09.       Find the highest value in list L and mark corresponding jth triple in cond2 as similar to
          the ith triple in cond1 with this highest value
10.   end for
11.   Linearly combine all values that are greater than the threshold value
12.   Calculate the normalized similarity value
13.   return the normalized similarity value
```

The second step is to obtain the similarity value between two conditions. We use a linear combination of the similarity values of each triple in the condition. Currently we manually set the threshold value that distinguishes “good” matches from “bad” matches. A triple with similarity value greater than the threshold value is considered as a good-matched triple, otherwise it is considered as a bad-matched triple. In our experiments, the threshold value is set to be the fourth best similarity value described in case 4 in the previous step so that a triple with case 4 or case 5 similarity value would be considered as a “bad” match. The reason for the choice of the threshold value is that this high threshold value will guarantee the correct semantic relations being identified in the following steps. Thus, for a pair of conditions, the similarity values of good-matched triples are combined linearly. To further obtain the normalized similarity value for two conditions, the similarity value is divided by the total number of good-matched triples in the conditions. Table 4.3 describes the algorithm for finding the similarity value for two conditions. The two conditions cond1 and cond2 are two parameters passed into the procedure.

Table 4.4 Algorithm for Mapping Similarity Value to Relations

```
01. procedure map_value_to_relation(V)
02.    if V is too small
03.        return noMatch
04.    else if V is maximal // maximal indicates that every triple in the first condition
                                    has a exactly matched triple in second condition
05.    if size1 = size2 // the two conditions have equal number of triples
06.        return exactMatch
07.    else if size1 < size2 // the first condition has less number of  triples than the second
08.        return plusMatch
09.    else
10.        return plusInMatch
```

The third step is to map this normalized similarity value to one of three semantic relations between conditions identified in Section 4.4 or to ignore it because the value is too small to be considered. Let the numbers of triples in these two conditions be size1 and size2, respectively, and the normalized similarity value be V. Table 4.4 shows a simplified algorithm for mapping similarity values to relations.

Table 4.5 Algorithm for Finding Semantic Relations between Pre and Post-conditions

```
01. procedure relation_bw_services(S1, S2)
02.    for i = 1 to N // N is the number of services
03.        for j = i+1 to N
04.            if i not equal to j
05.                sValue1 = similarity_bw_conditions(Pre1, Pre2)
06.                sValue2 = similarity_bw_conditions(Post1, Post2)
07.                sValue3 = similarity_bw_conditions(Post1, Pre2)
08.                sValue4 = similarity_bw_conditions(Post2, Pre1)
09.                if relation = map_value_to_relation(sValue1) not equal to noMatch
10.                    record the relation between Pre1 and Pre2
11.                if relation = map_value_to_relation(sValue2) not equal to noMatch
12.                    record the relation between Post1 and Post2
13.                if relation = map_value_to_relation(sValue3) not equal to noMatch
14.                    record the relation between Post1 and Pre2
15.                if relation = map_value_to_relation(sValue4) not equal to noMatch
16.                    record the relation between Post2 and Pre1
17.        end for
18.    endfor
19.    Generate a semantic network of services according to those relations recorded
```

The final step is to find the semantic relations between pre and post-conditions among a set of Web services. For every pair of services, their pre and post-conditions need to be cross matched to obtain all possible relations. Suppose service S1 has pre-condition Pre1 and post-condition Post1, and service S2 has pre-condition Pre2 and post-condition Post2. Then, the relations need to be identified between four pair of conditions: (i) Pre1 and Pre2, (ii) Post1 and Post2, (iii) Post1 and Pre2, and (iv) Post2 and Pre1. Here, identifying relations in (i) and (ii) may discover possible substitute and include relations between two services; and possible prerequisite and parallel relations between two services can be discovered by identifying relations in (iii) and (iv). According to the identified relations between pre and post-conditions, a semantic network of WSs can be generated. The algorithm is shown in Table 4.5.

4.6 Multiple Pre and Post-conditions

In the previous sections, we discuss how we model pre and post-conditions. However, it must be noted that we only address this issue for services having only a single pre-condition and a single post-condition. However, services may have multiple pre-conditions associated with different input scenarios and multiple post-conditions associated with different outcomes. For example, a book buying service may have one post-condition that specifies that book is sold and the other post-condition that specifies that book is in back order since it is out of stock. If we are to consider those services, the problem of matching multiple pre or post-conditions of one service with multiple pre or post-conditions of the other service becomes a very complex problem. Hence, we may identify multiple relationships between a pair of services both having multiple pre and post-conditions. Consequently, the problem of selecting the right service for a composition becomes more difficult. Our current technique might not be effective due to the complexity of the problem. Thus, our technique can only be applied to a sub set of services that have single pre and post-condition. Finally, specification of multiple pre and post-conditions for composite WSs in particular could be a future research area, and it is out of scope of this thesis.

CHAPTER 5

SYSTEM ARCHITECTURE AND EXPERIMENTS

For the experimental purposes, we have developed a Semantic Relations Discovery System prototype. Figure 5.1 shows the architectural overview of this prototype. The system architecture involves three components: (i) ontology and service storage component, (ii) discovery engine, and (iii) visualization component.

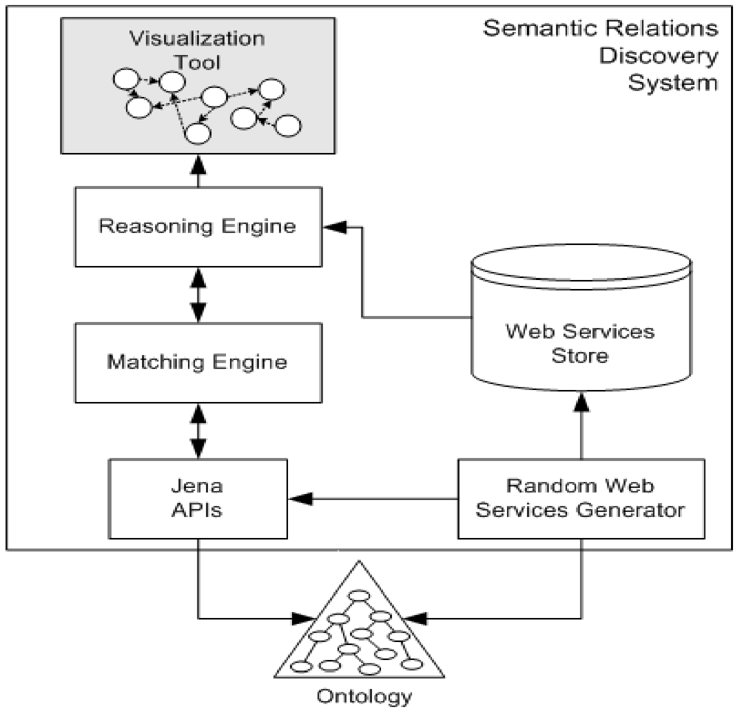
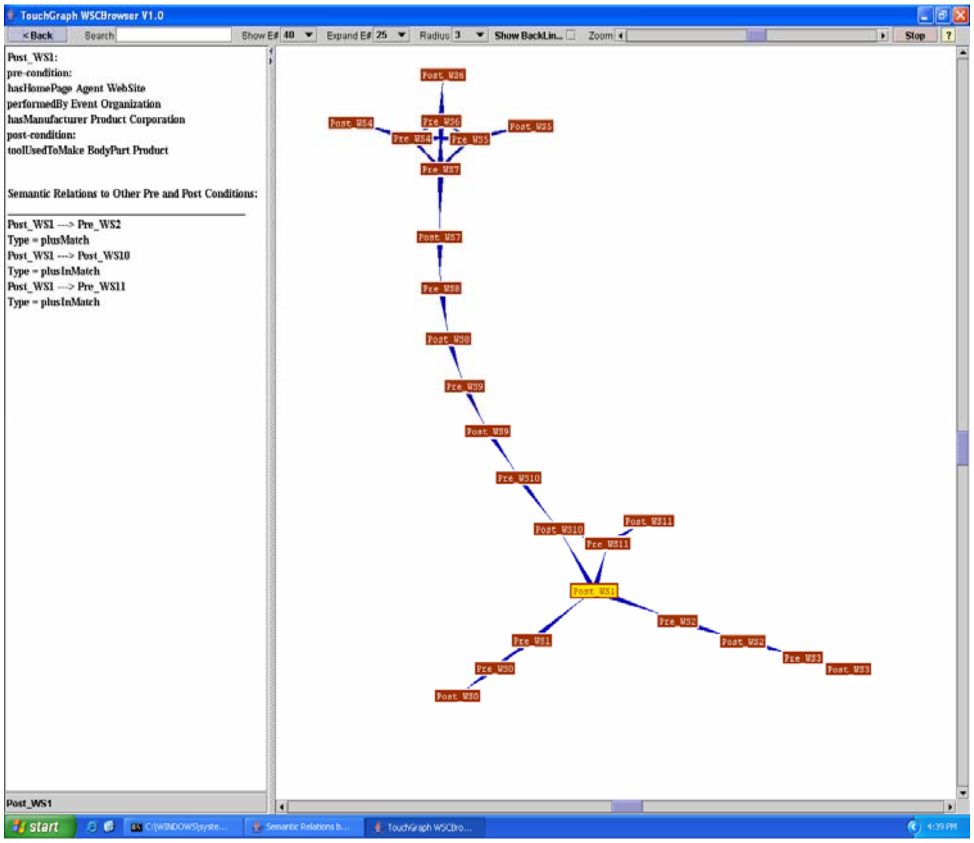


Figure 5.1 An Overview of System Architecture

The service storage component hosts a synthetic data set of WSs each with pre and post-condition randomly extracted from a domain ontology (see Figure 5.1). The domain ontology can be any user defined ontology in RDF specification.



Post_WS1:
pre-condition:
hasHomePage Agent WebSite
performedBy Event Organization
hasManufacturer Product Corporation
post-condition:
toolUsedToMake BodyPart Product

Semantic Relations to Other Pre and Post Conditions:

Post_WS1 ---> Pre_WS2
Type = plusMatch
Post_WS1 ---> Post_WS10
Type = plusInMatch
Post_WS1 ---> Pre_WS11
Type = plusInMatch

Figure 5.2 User Interface for Semantic Web of Services

The discovery engine component includes the Reasoning Engine and the Matching Engine. The Reasoning Engine takes the pool of randomly generated WSs as input, and for each pair of WSs, identifies the semantic relation between the pre and post-conditions of these two services. In order to do this, the Reasoning Engine sends the RDF triples from pre-conditions and the post-conditions of services to the Matching Engine for identifying similarity values. The returned similarity values are used in reasoning process. To determine the similarity values, the

Matching Engine uses Jena APIs [Jena], which is a Java framework for building semantic Web applications, to load and query the ontology to locate relations between two given concepts (i.e., classes). The resulting semantic Web of services is visualized through a Touch Graph based graphical user interface (see Figure 5.2), which is the visualization component.

Figure 5.2 shows a screen shot of the user interface for our system. There are two panels in the user interface: left panel and right panel. On the right panel, there are boxes connected with arrows. Each box represents a pre-condition or a post-condition for a service, together with service id showing as the text label in the box. Each arrow represents a type of semantic relation between two connected conditions. When a box is clicked by the user, a text description shows up on the left panel. It describes the types of semantic relations identified between this condition and all possible other conditions.

The user interface for our system can only allow users to browse through the generated semantic network of pre and post-conditions of services. This user interface can be improved by adding more functionalities. For example, some selection alternatives for possible services (e.g., composable services) in a next composition step can be provided to a user once the user selects a service. Thus, it limits the number of services that the user needs to look through before finding a suitable service. This could be a part of the future work.

Our choice of generating a random dataset to perform the experiments is because currently, there are no standard dataset that is used for evaluating the semantic WS discovery or composition techniques. We are running our experiments using TAP [TAPKB], which is an experimental knowledge base (i.e., populated ontology) about people, places, products, etc. Several synthetic WSs generated using TAP are shown in following Table 5.1. The first column shows service id, and the second column shows pre and post-conditions for each service. Each condition in parentheses is a set of triples, separated by comma, and each triple is ordered as predicate, subject, and object. The post-condition of service WS135 exactly matches the pre-condition of service WS71. Thus, services WS135 and WS71 have the prerequisite relationship. The services WS123 and WS86 have the include relationship since the pre-condition of WS86 is a plug-in match to the pre-condition of WS123, and their post-conditions exactly match.

Table 5.1 Sample Pre and Post-conditions

WS ID	**Pre- and Post-conditions**
WS135	**Pre-condition:** (occurrenceTimeOfDay Event TimeOfDay) **Post-condition:** (hasCapitalCity Country City, representsPlace Politician Place, hasIngredient Food Food)
WS71	**Pre-condition:** (hasCapitalCity Country City, representsPlace Politician Place, hasIngredient Food Food) **Post-condition:** (subBrand Brand Brand, hasComponent Tangible Tangible)
WS123	**Pre-condition:** (hasAuthor MediaObject Agent) **Post-condition:** (price Product MoneyQuantity)
WS86	**Pre-condition:** (hasAuthor Book Agent) **Post-condition:** (price Product MoneyQuantity)
WS93	**Pre-condition:** (chiefExecutive Organization Person, hasAddress Place GeoAddress) **Post-condition:** (activityOfEvent Event ActivityType)

Due to the lack of the standard semantic WSs dataset, traditional evaluation metrics such as precision and recall can not be performed. Hence, we evaluated our discovery results with respect to those obtained by a panel of ten human subjects, who are graduate students in computer science and not familiar with the research presented here. The human subjects were given thirty randomly generated services, each consisting of pre and post-condition. Together with the service dataset, all subjects were provided with the discovery criteria for three types of relations between conditions. The human objects were also provided with a graphical representation of partial domain ontology used to generate pre and post-conditions for the service dataset, thus allowing them to evaluate the similarity between triples within conditions. They then identified all possible relations between pre and post-conditions of all thirty services according to the criteria provided.

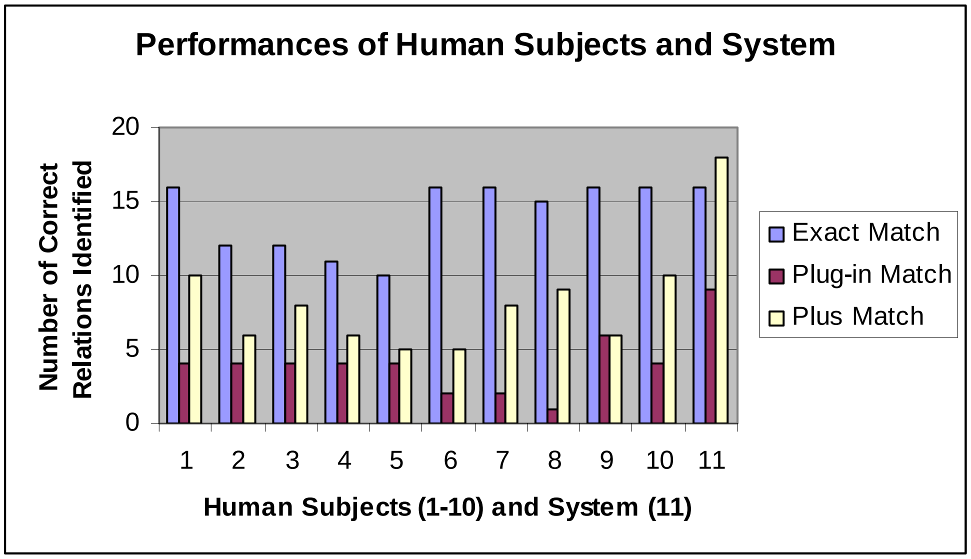


Figure 5.3 Performances of Human Subjects and System

In order to demonstrate the effectiveness of our discovery scheme, we illustrate in Figure 5.3 the performances of human subjects and system. The x-axis represents ten human objects and system, each having three columns indicating its performances on exact match, plug-in match, and plus match relations, respectively. On the other hand, the y-axis represents the number of correct relations identified by the system and human subjects. It is evident in the figure that there are varying levels of performances in human subjects' ability to identify semantic relations between conditions. Especially, human subjects had difficulty of identifying plus match relation. Note that the system identifies more relations than those human subjects. Here the threshold value for the system is set to the same value discussed in the second step of algorithm in the previous Section 4.5. Figure 5.4 shows the performances of the system with different threshold values. The average performance is also calculated and displayed in Figure 5.4. It is a clear indication that varying threshold values can produce varying levels of performances. To obtain a good system performance, we recommend that the threshold value is set to the value previously discussed in Section 4.5.

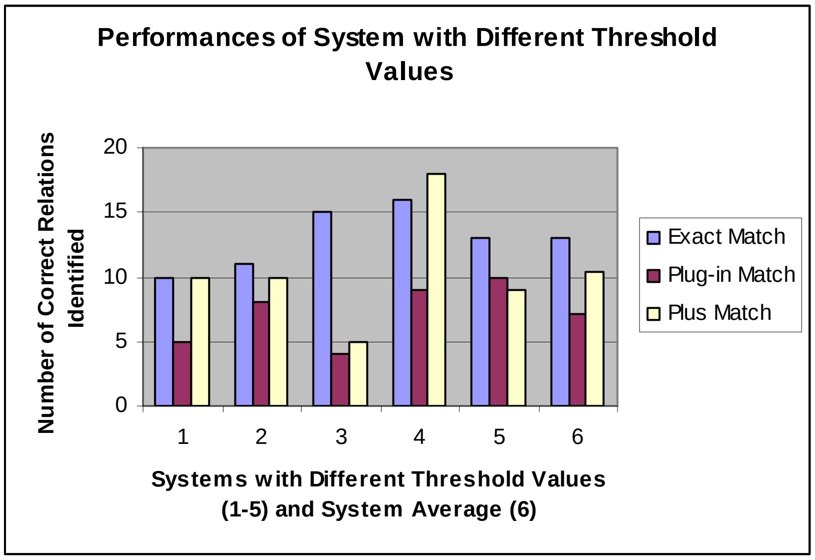


Figure 5.4 Performances of System with Five Different Threshold Values

5.1 Discussion

Our experiments are based on a synthetic dataset of services with only pre and post-condition specifications and numerical identifiers. It is not surprising to see that the performance of the system is better than those of human subjects in the experiments since they can not apply their real world experiences to a synthetic dataset that has no real meanings. If real services, with a full description of service name, service goal, and input and output parameters as well as pre and post-conditions, were to be used in the experiments, human subjects might perform better than the system. Human subjects are good at identifying associations, for example, using WS name, when various sources of information are provided. However, there is a limit for the amount of WS specifications human subjects can handle at a time. As the number of services in the dataset increases significantly, it becomes difficult for them to identify all possible relations among pre and post-conditions. In this case, our technique will give a better performance with large number of services in a dataset. Also, frequently WS names may not be descriptive enough so that human subjects may have difficulties in identifying possible associations in this case as well.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, we propose a novel technique for discovering semantic relations between pre and post-conditions of different services using their ontological descriptions. We propose to model pre and post-conditions of WSs as sets of RDF triples, which allow us to express conditions in a simple but flexible way. We also address the issue of relaxed matching in the sense that the pre-condition of one service can be satisfied by the post-conditions of more than one service. A prototype of proposed approach is implemented so that a semantic Web of services can be generated, and their semantic relations can be displayed visually. Currently, we do not test the actual compositions in our prototype. However, users can visually browse the final generated semantic Web of pre and post-conditions of services to select any service of interest for checking possible compositions involving this service. In the future, improvements on the user interface can be made so that the interface can be more useful and user-friendly. Our technique can be used in conjunction with IMA technique to enhance the quality of the produced composition. In the future, potential path traversal algorithms can be applied to our semantic Web of services to obtain service compositions. In our future work, we may need to address the issue of multiple pre and post-conditions for one service. Our future work may also include conducting experiments to measure performance of our scheme if one standard semantic WSs dataset becomes available, and developing a condition ontology with respect to the dataset.

REFERENCES

[Arpinar04a]	B. Arpinar, R. Zhang, B. Aleman-Meza, and A. Maduko., "Ontology-driven Web Services Composition Platform", IEEE Intl. Conf. on e-Commerce Technology, San Diego, California, July 6-9, 2004.
[Arpinar04b]	B. Arpinar, R. Zhang, B. Aleman-Meza, and A. Maduko, "Ontology-driven Web Services Composition Platform", Journal of Information Systems and e-Business Management, to appear.
[Berardi03]	D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella, "Automatic Composition of e-services that export their behavior", Proceedings of the 1st International Conference on Service Oriented Computing, 2003.
[Bultan03]	T. Bultan, X. Fu, R. Hull, and J. Su, "Conversation Specification: A New Approach to Design and Analysis of E-Service Composition", Proceeding of WWW Conference, 2003.
[Cardoso02]	J. Cardoso, "Quality of Service and Semantic Composition of Workflows", Ph.D. Dissertation, Department of Computer Science, University of Georgia, Athens, GA, 2002.
[Casati00]	F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan, "Adaptive and dynamic service composition in eFlow", Proceedings of the

Intl. Conf. on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, 2000.

[DAML+OIL] DARPA Agent Markup Language, 2001, URL: http://www.daml.org/language/

[DRS] Drew McDermott and Dejing Dou, "Representing Disjunction and Quantifiers in RDF Embedding Logic in DAML/RDF", International Semantic Web Conference, 2002.

[Fensel02] D. Fensel and C. Bussler, "The Web Service Modeling Framework WSMF", Electronic Commerce Research and Applications, 1(2), 2002.

[Jena] Jena Semantic Web Framework, URL: http://jena.sourceforge.net/index.html

[Hamadi03] R. Hamadi and B. Benatallah, "A Petri net-based Model for Web Service Composition", Conferences in Research and Practice in Information Technology Series**,** in Proc. of the 14th Australasian Database Conf., 2003.

[Lara03] R. Lara, H. Lausen, S. Arroyo, J. de Bruijn, D. Fensel, "Semantic Web Services: description requirements and current technologies", Intl. Workshop on Electronic Commerce, Agents, and Semantic Web Services, Pittsburgh, PA, September 2003.

[Narayanan02] S. Narayanan and S. Mcllraith. Simulation, "Verification and Automated Composition of Web Services", Proc. of the 11th Intl Conf. on WWW, Honolulu, Hawaii, 2002.

[OWL] Web Ontology Language, 2004, URL: http://www.w3.org/TR/owl-ref/

[Patil04] A. Patil, S. Oundhakar, A. Sheth, and K. Verma, "METEOR-S Web Service Annotation Framework", Proceeding of the World Wide Web Conference, July 2004.

[Peterson81] J. Peterson, Petri Net Theory and the Modeling of Systems, Prentice Hall, Englewood Cliffs, 1981.

[Ponnekanti02] S. R. Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition", The 11th WWW Conference, Honolulu, Hawaii, May 7-11, 2002.

[RDF] Resource Description Framework, 2004, URL: http://www.w3.org/TR/rdf-syntax-grammar/.

[RuleML] The Rule Markup Initiative, URL: http://www.dfki.uni-kl.de/ruleml/.

[Sivashanmugam03] K. Sivashanmugam, "The METEOR-S Framework for Semantic Web Process Composition", Master Thesis, Computer Science, University of Georgia, 2003.

[SOAP] Simple Object Access Protocol, URL: http://www.w3.org/TR/SOAP/

[Stollberg05] Michael Stollberg, Uwe Keller, and Dieter Fensel, "Partner and Service Discovery for Collaboration Establishment with Semantic Web Services", Digital Enterprise Research Institute, University of Innsbruck, Austria, 2005.

[SWRL] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML", Technical report, Version 0.5 of 19, November 2003.

[TAPKB]	TAP knowledge base, URL: http://tap.stanford.edu/tap/tapkb.html
[UDDI]	The UDDI technical white paper, 2000, URL: http://www.uddi.org
[WSDL]	Web Services Description Language, URL: http://www.w3.org/TR/wsdl
[Wu03]	D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau, "Automating DAML-S Web Services Composition Using SHOP2", Proceedings of 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, October 2003.
[Yi04]	X. Yi and K. Kochut, "Process Composition of Web Services with Complex Conversation Protocols: a Colored Petri Nets Based Approach", Proceedings of the Design, Analysis, and Simulation of Distributed Systems Symposium, pp. 141-148, Advanced Simulation Technology Conference 2004.
[Zeng03]	L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, "Quality Driven Web Services Composition", Intl. WWW Conf., 2003.
[Zhang03]	R. Zhang, I. B. Arpinar, and B. Aleman-Meza, "Automatic Composition of Semantic Web Services", International Conference on Web Services, Las Vegas, NV, June 2003.
[Zhang04]	R. Zhang, "Ontology-driven Web Services Composition Techniques", Master Thesis, Computer Science, University of Georgia, 2004.