ONTOLOGY-DRIVEN WEB SERVICES COMPOSITION TECHNIQUES

by

RUOYAN ZHANG

(Under the Direction of I. Budak Arpinar)

ABSTRACT

Discovering and assembling individual Web Services into more complex yet new and more useful Web Processes is an important challenge. In this thesis, we present techniques for automatically and semi-automatically composing Web Services into Web Processes by using their ontological descriptions and relationships to other services. In Interface-Matching Automatic Composition technique, the possible compositions are obtained by checking semantic similarities between interfaces of individual services and considering the service qualities. In Human-Assisted Composition the user is allowed to select service classes and instances according to a service template from the ranked lists.

INDEX WORDS: Web Service, Process, Composition and Semantics.

ONTOLOGY-DRIVEN WEB SERVICES COMPOSITION TECHNIQUES

by

RUOYAN ZHANG

Bachelor of Arts, Beijing University, People's Republic of China, 1988

Master of Arts, Beijing University, People's Republic of China, 1995

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

2004

ONTOLOGY-DRIVEN WEB SERVICES COMPOSITION TECHNIQUES

by

RUOYAN ZHANG

Major Professor:	I. Budak Arpinar
Committee:	Hamid R. Arabnia
	Amit P. Sheth

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2004

DEDICATION

This is dedicated to my loving husband Chongshan, Zhang.

ACKNOWLEDGEMENTS

I am very thankful for the members of my thesis committee. I want to express my thanks to Dr. Sheth for his wonderful lecture on Semantic Web and good suggestions for my research work. Thanks to Dr. Arabnia, who impressed me deeply with his sincere concern for his students and his review on my thesis. And finally, it is my advisor, Dr. Arpinar, to whom I owe the most overwhelming debt of gratitude for his support and his invaluable guidance for my thesis research.

I also would like to thank Aleman Meza Boanerges, who started the research on the interactive approach for this project and kindly offered me detailed suggestions for my work. I also like to thank Lin Lin who reviewed part of my work. Thanks to Angela Maduko, Mullai Shanmuhan, and Jorge Cardoso whose talks and works benefited me a lot.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

1.1 Motivation

In recent years, a growing number of Web Services (WSs) have emerged as the Internet develops at a fast rate. The Web is now evolving into a distributed device of computation from a collection of information resources [Fensel02a]. Furthermore, the need for composing existing WSs into more complex services is also increasing, mainly because new and more useful solutions can be achieved. In general, this is a result of complex and increasing user demands and inability of a single WS to achieve a user's goals by itself. For example, a traveler who wants to make a hotel reservation and find a French restaurant less than three miles from the hotel may either utilize some services that s/he knew already or try to find those services by looking it up in a keyword-based search engine (e.g., expedia.com or google.com) or by looking it up in a WSs registry (e.g., a UDDI (Universal Description, Discovery and Integration) registry) at present. Also, composition of discovered services and enabling data-flow among them (e.g., the hotel address is needed as input to a restaurant locator service) are usually done manually, which are highly inconvenient, especially for more complex compositions.

The problem lies with the fundamental abstractions used to model WSs, and methods to compose these services using these abstractions. In more complex examples of scientific data exploration through service compositions, even tens or hundreds of data

collection services can be involved in a composition (e.g., a search involving gene banks). In that case an automatic composition can help in reducing query formulation and execution time enormously.

In general, there are four different dimensions for a service composition (Figure 1.1): (i) degree of user involvement in a composition specification, (ii) if the composition is based on templates, (iii) dynamicity (i.e., adaptation) of the composition, and again, (iv) degree of user involvement in the adaptation of the composition [Cardoso03 & Srivastava03]. In the first dimension, a composition can be defined fully by a user including its control and data-flow besides the individual services making the composite service. In contrast, in an automatic composition user is not involved but system defines control and data-flow. This is very challenging due to difficulty of mapping user needs to a collection of correlated services where their interim outputs can satisfy each other's input requirements and final deliverable meets the user demands. Besides that, in both of user-defined or automatic composition techniques either actual service instances or service templates can be used. In the latter, the individual service instances are searched and integrated automatically at execution time for a given plan [Chandra03]. In a dynamic composition, the composition itself can be adapted mainly because of Quality of Services (QoS) requirements at run-time. Also, a composition may not be defined at design-time but can be assembled dynamically at execution time. Finally, some hybrid methods such as semi-automatic compositions and semi-automatic adaptations are also possible.

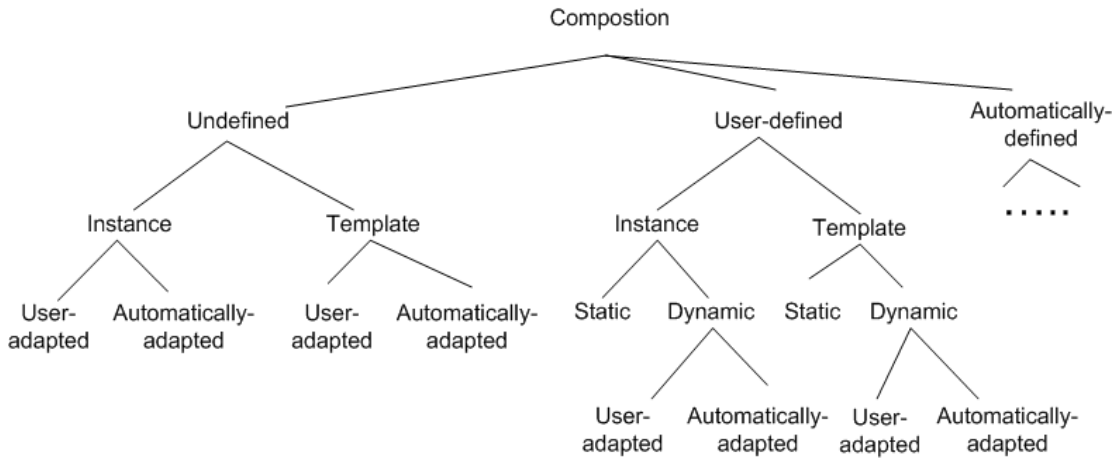

Figure 1.1 A Classification of Web Service Composition Techniques

This work aims for reducing the complexity and time needed to generate, and execute a composition and improve its efficiency by selecting the best possible services available at current time by automatic Interface–Matching Automatic (IMA) composition and Human-Assisted (HA) composition of Web Services. IMA composition has no predefined template and the user is not involved in the composition specification. In contrast, the user is greatly involved in the HA composition specification and adaptation based on predefined the templates.

1.2 Contributions

We developed a collection of Ontology-driven Web Services Composition techniques. In IMA Composition Technique, the possible compositions are obtained by checking semantic similarities between interfaces 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 (or other attributes that clients might concern about) is selected. However, our experiments show that without functionality constraints, IMA technique is more appropriate for the information-retrieval services (i.e., not world-altering services), that always return relatively simple results based on the user-supplied inputs [Zhang03 & Arpinar04]. This is mainly due to the fact that Web services with the same interface could have different functions and the difficulty of mapping of input and output parameters for many services. There are also many earlier efforts in composition of software components using pipe-filter methods [Mao01]. Our contribution is that we developed a shortest-path algorithm to address a more complicated problem in which the services might have multiple I/O (input and output) parameters. This technique can also take Quality of Service (QoS) of WSs into account to find a highest quality (minimum cost) as described later. Furthermore, based on the ontology techniques, we could compose services to finish a task even the services interfaces are not exactly matched syntactically.

The complex services, such as flight booking and hotel reservation services, that the user's request based on not only the inputs and outputs, but also on the quality of service (e.g., service quality rate in DAML-S 0.9), functionality, service name, geographic regions, and possibly user profile, can not be located precisely and composed sequentially merely considering the interfaces. As the number of various services increases, the task of selection of an adequate service quickly becomes tedious. Therefore, new techniques are needed to help users in finding, filtering, and composing these services.

At the beginning of service composition, a user may have a vague picture of the composite service in his/her mind that s/he is seeking. Another possibility is that an available composite template may not satisfy the user, who wants to add or remove some

tasks in the original plan if necessary. Additionally, the service selection and composition are not only determined by the constraints on service description (e.g., interface) yet functionality properties and the service output values to be returned at run-time. For example, consider a trip planning service composition example. A user often prefers a flight service that provides the cheapest flight ticket, which, however, could only be known after comparing the output ticket prices of a set of services. Furthermore, even the type of transportation service in the composition could be determined by the dynamic information of other services, such as parking fee or taxi rate if s/he is considering the trip by flight or rental car. Thus depending on requirements on service descriptions and dynamic information supplied by the services, a customization and dynamic composition with more human involvement may be needed for many applications. An example for this is introduction of XOR-split in the composition for WSs whose behavior can only be determined at run-time (e.g., price selection) and we can filter out undesired services.

The Human-Assisted (HA) Composition Technique is dedicated to applications which require frequent human-intervention for an acceptable composition. The goal of this technique is to build a composition pathway (i.e., map) incrementally in a customized way. Thus in this thesis, we present a complementary technique to IMA, namely Human-Assisted Composition Technique (HA), that guides the users for selecting proper service instances described in DAML-S (0.9), and allows users to create a customized composite service plan. In particular, we intend to address the following issues:

1. Exploiting Semantic Web and Web Service Ontologies to bridge the concept gaps in interface parameters and other parts of descriptions of services. Basically there are four types of semantics for Web Services: (i) data/information semantics, (ii)

functional/operational semantics, (iii) execution semantics, and (iv) QoS semantics [Sheth03]. Our framework mainly uses the former two semantics and intends to help users to select service classes and instances by matching service interfaces and attributes.

2. Ranking and filtering of composable services at certain intervention points by the user. This also considers a semantic user profile and output values for filtering huge numbers of service instances.

3. Automatically adjusting a composite service plan by removing the uninteresting services or adding the services suggested by the system if their I/O matches.

The thesis is organized as follows: Chapter 2 reviews the related work on Web Service description, discovery, composition, especially automatic and interactive composition approaches. Chapter 3 presents Interface-Matching Automatic (IMA) Composition technique for Web Services. Chapter 4 describes Human-Assisted (HA) Composition Technique. Chapter 5 provides the conclusion and future work.

CHAPTER 2

RELATED WORK

This thesis aims to generate a composite service plan out of semantically described existing services. The Web Service composition is related to many efforts. These include the Web Service specification, discovery, composition and execution technologies.

2.1 Web Service Description

The service specification methods help software systems to capture capabilities of WSs. In general, these specification methods are based on either industry-oriented standardization efforts, or academia-oriented WS ontologies. UDDI and Web Services Description Language (WSDL) are current industry standards developed for e-commerce. The 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. Therefore, a UDDI does not provide a semantic search rather it depends on a predefined categorization of WSs through keywords. In complimentary roles, WSDL and Simple Object Access Protocol (SOAP) describe WSs as a set of endpoints (or ports) operating on messages, and a protocol for exchange of these messages between the services respectively. Also, some industry standards have been emerging to represent data and control-flow, and transactional properties among a collection of services. Business Process Modeling Language, XLANG, Web Services Flow Language (WSFL),

and Business Process Execution Language for Web Services (BPEL4WS) can be mentioned in this category.

The semantic approach for WS specifications includes Web Service Modeling Framework (WSMF) in which ontology provides the terminology used by other elements [Fensel02b]. DAML-S (recently evolved to OWL-S) specifies three main components for each service: service-profile, process-model and service-grounding. A service profile is the core element of a DAML-S specification, and it involves semantic descriptions of service interfaces and functions [DAML-S Coalition03]. The process-model provides the information of how the service works, and the service grounding describes how an agent can access the service. Other techniques try to add semantics to existing services by providing mappings between WSDL, UDDI definitions and domain ontologies (e.g., METEOR-S [Siva03 & Patil04]).

In this work, we primarily focus on the collection of inputs, and outputs for composition and leaving pre-condition, and effect (i.e., post-condition) oriented composition as a future work.

2.2 Semantic Web Service Discovery and Composition

Semantic WS discovery related work includes [Cardoso03], which describes how to evaluate a degree of similarity between a service template and an actual service by measuring the syntactic, operational, and semantic similarity. A semantically described service needs to be efficiently discovered. Among the state-of-the-art discovery systems, the project DReggie adds reasoning modules to carry out the semantic matching process for

discovering DAML described Web Services [Chakraborty01]. Unlike other discovery approaches on the basis of WS interfaces, [Klein01] explored ways to search services according to the functionality requirements, and proposed Process Query Language (PQL) to search process models from a process ontology. [McIraith02] presented a method to compose Web services by applying logical inferencing techniques on pre-defined plan templates. This technique focuses on the process-centric description of service as actions that are applicable in states. Semantic representations of state, actions, goals are needed for composing services [Srivastav03].

The main concept behind service composition is not new in computer science. Earlier, software composition techniques aimed to find a good combination of components that responds to the client specific requirements by matching requested properties with provided properties. One approach for finding a suitable composition is to delegate the responsibility for solving certain requirements posed on a component to other components after fulfilling it partially [Sora01]. Similarly, our interface-matching mechanism (IMA) propagates requirements (that are set of a user's expected outputs) to corresponding WSs in an incremental way. [Mao01] proposes a composition path, which is a sequence of operators that compute data, and connectors that provide data transport between operators. The search for possible operators to construct a sequence is based on the shortest-path algorithm on the graph of operator space. However, [Mao01] only considered two kinds of services – operator and connector with one input and one output parameter (which is the simplest case for a service composition) and did not take semantics into the account.

In the instance composition category, SWORD uses a rule-based expert system to determine if a plan of composite service can be built out of existing services [Ponnekanti02]. It mainly focused on the composition of information provider services (i.e., not world-altering services), and (like [Mao01]) it does not address the input and output mismatch problem. In our approach, services possibly have more than one input and output parameters, and their interfaces may not match syntactically. The difficulty of automatic composite services without pre-defined template mainly lies in the lack of explicit way of representation of the goal of a composite service and complementary functionality relationship between services semantically.

2.3 Interactive and Adaptive Composition

There are many efforts in industry to customize processes. Mainly three architectures are proposed to manage inter-organizational business processes, process portal, process vortex and dynamic trading process [Sivashanmugam03 & Sheth99].

At present, academic approaches have been proposed to tackle the problems for personalization and filtering of WSs based on templates. An example is a trip planner, which is declared as a state chart, and the resulting composite services are executed by replacing the roles in the chart by selected individual services [Benatallah02]. Template-based composition techniques are also used in [Narayanan02], and ICARIS project [Tosic01].

The Semantic Web community mainly composes the services based on the goal-oriented inferring and planning. None of the approaches has developed a satisfactory planning solution to the service composition so far [Srivastava03].

In [Sirin2003], users select and filter the services by using a similar matchmaking algorithm. In [Balke2003], the services are selected by using the hard and soft constraint standards in personalized composition.

[Ambite2003] designs a constraint reasoning network to compute any user's input change and produce the corresponding outputs to optimize the schedules for the trip based on the AI constraints reasoning technology. For a relatively fixed template, such as trip planner, generating a predefined constraints network might be feasible in practice. For less widely used services, it is not possible to have such as constraint a priori network. [Gil2003] uses a very similar technique as [Ambite2003] to support users in creating a specification of a pathway and allow users to specify abstract descriptions of steps. Our WSs network which is explained later has a similar feature with the constraints reasoning network except the latter only links services or functions to produce the expected results while in WSs network all services would be connected only if their inputs and outputs are matched.

When the composite service plan is generated, the checking and verifying of the service logic are crucial for execution. [Cheng02] presents an algorithm that checks the validity of the execution of services. Verification and checking of pathway of a composite plan are parts of future work for this thesis.

2.4 METEOR -S Framework

The METEOR-S project by LSDIS at University of Georgia, has studied the use of emerging Web Services and Semantic Web technologies and research, to develop Semantic Web Service and Process specification, semantics-based Web Services discovery, and Process Composition [Patil04b & Cardoso02& Sivashanmugam03b].

MWSAF (METER-S Web Service Annotation Framework) is designed to mark up Web Service descriptions with ontologies and develop algorithms to match and annotate WSDL files with relevant ontologies [Patil04]. The METEOR-S Web Services Discovery Infrastructure (MWSDI) created a scalable infrastructure for semantic publication and discovery of Web Services [Verma04]. A specialized ontology called the Registries Ontology maintains the relationship between all the domain and associates registries to them. Additionally, an algorithm has been developed to find the Web Services with proper interfaces and operational mechanisms for workflow generation [Cardoso02 & Cardoso03].

As part of the METEOR-S project, the MWSCF (METER-S Web Service Composition Framework) platform specifies an activity as a semantic activity template, then weights the overall ranking of services on the two dimensions: semantic matching and QoS criteria matching [Sivashanmugam03].

This thesis work has benefited from the METEOR-S techniques that are mentioned. The similarity algorithm described in Chapter 3 and Chapter 4 is partially based on the algorithm in [Cardoso02]. In HA Composition (Chapter 4), we argue that a semantic activity template [Sivashanmugam03] is a possible way for specification for the HA service template and the generated composite service for execution. If a service has

multiple registries, we plan to utilize MWSDI to discover the appropriate services under other names in HA (see details in Chapter 4).

CHAPTER 3

INTERFACE-MATCHING AUTOMATIC COMPOSITION

With the growing number of Web services, importance of composing existing Web Services into more complex services in order to achieve new and more useful solutions is increasing. However, in order to automatically compose new services, existing services need to be encoded in a machine understandable form. The semantics of a service can be described by annotating it with respect to service ontologies. The goals of automatic composition include reducing the complexity of creating composite services as well as choosing an optimal composition among possible options. This chapter describes the Interface-Matching Automatic (IMA) Composition technique that aims for generation of complex Web Services automatically by capturing user's expected outcomes when a set of inputs are provided; the result is a sequence of services whose combined execution achieves the user goals [Zhang03 & Arpinar04].

3.1 Modeling Semantic Web Services and Queries

A Semantic WS is a unit of composition that can be deployed independently, and may be subject to composition by a third party on the Web. At the same time, its interface, its process specification (i.e., its functionality) and its relations to other services are defined, and advertised in a machine-processable form so it can be automatically discovered, composed, and invoked in new complex WSs. The emerging Semantic Web makes it

possible to specify semantics of a domain such as the terms and concepts of interest, their meanings, relationships between them and the characteristics of the domain through an ontology. In this work, we use DAML-S (0.9) service ontology. A service profile is the core element of a DAML-S specification, and it involves semantic descriptions of service interfaces and functions. Table 3.1 is a part of profile of Wine-Searcher service described in DAML-S. In this work, we primarily focus on the collection of inputs, and outputs for composition.

Table 3.1 Profile-Wine-Searcher in DAML-S

```
<profileHierarchy: Wine-Search rdf: ID="Profile-Wine-Searcher">

<!--  reference to the service specification -->

<service: presentedBy rdf: resource="wine-searcher.owl#wine-searcher" />
<profile: has_process rdf: resource="wine-searcher-Process.owl#Wine-
                                     Searcher ProcessModel" />

<profile: serviceName> Wine-Searcher.com</profile: serviceName>

<profile:textDescription>
       Wine-Searcher helps you save money when buying wines by providing a
       database of regularly updated wine merchants' price list. Our site
       provides the fastest and easiest way to find who is selling a wine, to
       compare prices between wine retailers, and to value wines.
</profile:textDescription>

<!--  specification of quality rating for profile  -->

<profile: qualityRating>

       <profile: qualityRating rdf:ID="wine-search-Rating">
       <profile: ratingName>very good</profile: ratingName>
       <profile: rating rdf:resource="owl-s/1.0/Concepts.owl#GoodRating">

</profile: QualityRating>

<!--  specification of service Input and Output -->

<profile: hasInput  rdf:resource="Service-Concept.owl#wineName" />
<profile: hasInput  rdf:resource="Service-Concept.owl#wineVintage" />
<profile: hasInput  rdf:resource="Service-Concept.owl#merchantLocation" />
<profile: hasOutput rdf:resource="Service-Concept.owl#winePrice" />

</profileHierarchy: Wine-Search Service>
```

A composite service query can be represented in a very similar way as a service description in DAML-S. Like DAML-S template of services, the query profile includes

the description of the composite service and the interface of the expected composite service, in which we define the output parameters, output constraints, input parameters, and the constraints. The output constraint specifies the requirements on the outputs by the user, such as the properties of the output parameter. For example, the user can define the price properties, currency as Franc as shown in the price ontology (Figure 3.1).

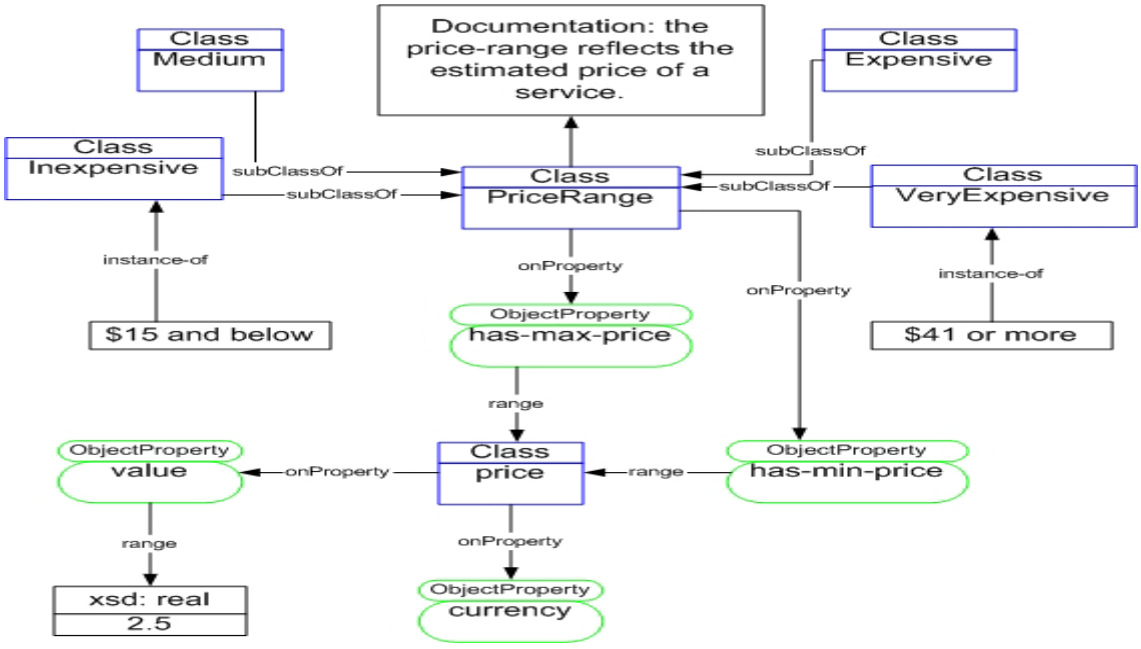

Figure 3.1 Price Ontology[1]

The second part of the query is about the functionality of the composite service (we will investigate this in the future). The user can partially specify how the composite service works and what kind of individual services would be expected to be included (constraints and functionality parts are omitted in the Table 3.2 for brevity). For example, a restaurant owner may want to find matching wines to the meals in a restaurant and learn the prices

[1] http://sf.us.agentcities.net/ontologies/price.jpg

of these wines. S/he specifies a seafood for *Food-Wine-Matching* service and expects the prices and name of matching wines.

Table 3.2 A Composite Service Query

```
<Query: QueryName> Wine Price</profile: serviceName>

<!--  specification of quality rating for query -->

<Query: qualityRating>

       <profile: qualityRating rdf:ID="Query-Rating">
       <profile: ratingName> average </Query: ratingName>
       <profile: rating rdf:resource="owl-q/1.0/Concepts.owl#GoodRating">

</Query:QualityRating>

<!--  specification of service Input and Output -->

<Query: hasInput   rdf:resource="Service-Concept.owl#seafood/Food"/>
<Query: hasOutput  rdf:resource="Service-Concept.owl#wineName/Wine"/>
<Query: hasOutput  rdf:resource="Service-Concept.owl#winePrice/Wine
       <daml:Restriction daml:onProperty rdf:resource=Franc">
       </daml:Restriction>
```

3.2 System Architecture

The system architecture (Figure 3.2) involves three components (i) composer component, (ii) ontology and service storage component and (iii) extraction component. The service storage component hosts the user profile, Web Service network, Web Service ontology, and domain ontologies. User profile records the history of each user's usage of Web Services. The service instances can be ranked based on the frequency (i.e., popularity) of this Web Service usage.

Ontology component includes domain ontologies (in OWL) are specialized for description of parameters of the services. For example, Figure 3.3 depicts the ontology for food and drink that specifies the sub classes and super class relationships of the entities and properties. *Wine* is the subclass of the *Alcohol* that is the subclass of *Drink*. The properties of *Wine* include *wine name, vintage, merchant location and price*, etc. The

user can specify the properties of *Wine* in query and limit the range of the properties as described in query format (Table 3.2).

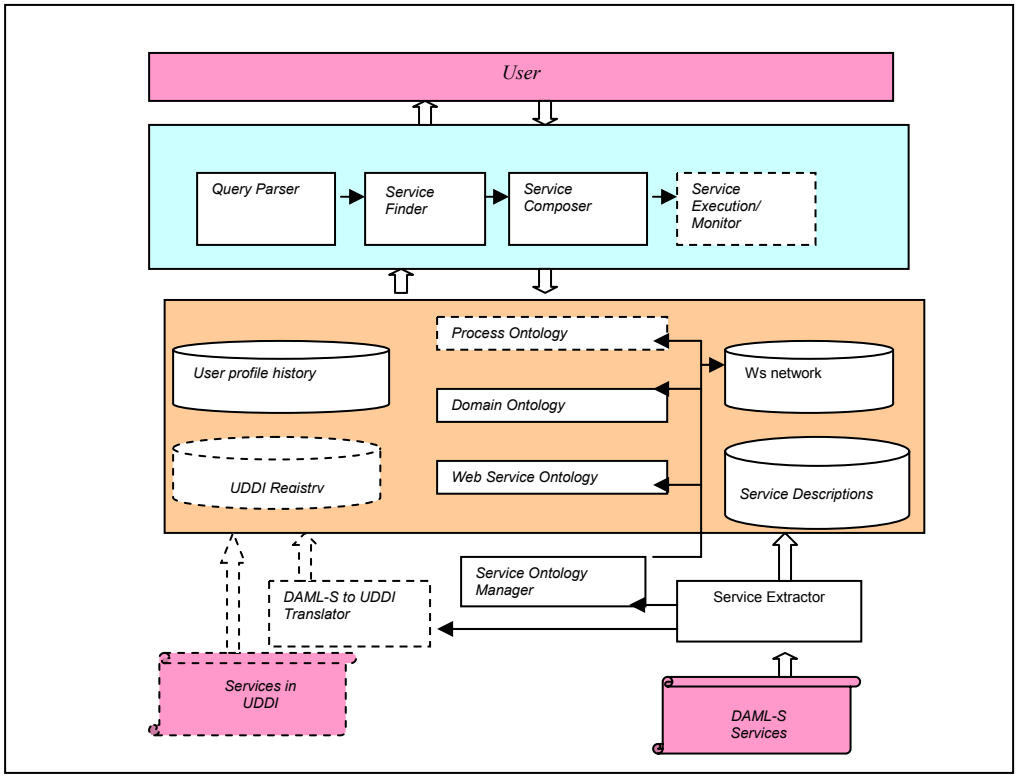

Figure 3.2 System Architecture
(Future components are included in dashed boxes)

Like domain ontologies, a Web Service ontology describes service hierarchy relationship and a subclass of a service inherits the properties and functionality of its super class service and extends with its own attributes (Figure 3.4). The Web Service network is a collection of services that are connected each other if their interfaces are matched semantically and it is built for service automatic navigation to improve the efficiency of composition algorithm (Figure 3.5)

The *Query Parser* parses the query and sends one query object to *Service Finder*, which does a search for a sequence of services by navigating Web service Ontologies and Web Service network and it returns a composite service with the optimal graph to *Service Composer*. A *Service Composer* generates the data flow chart of a composite service and sends the plan to *Service Execution* Monitor which is not implemented yet.

When a service provider sends a registration request, a S*ervice Extractor* extracts service name (including it ID), text description, instance/of relationship, input/output information, etc from the service profile and stores them in a services database. With regards to Web Service ontology, *the Service Ontology Manager* updates the Web Service Network by connecting its parameters to compatible services. This helps in reducing complexity of searching for services in an automatic composition as mentioned earlier. If the services are described in UDDI schema, their profiles would be sent directly into a UDDI registry. The DAML-S to UDDI translator is responsible to translate the information in DAML-S to the UDDI specification (this is also a part of the future work).

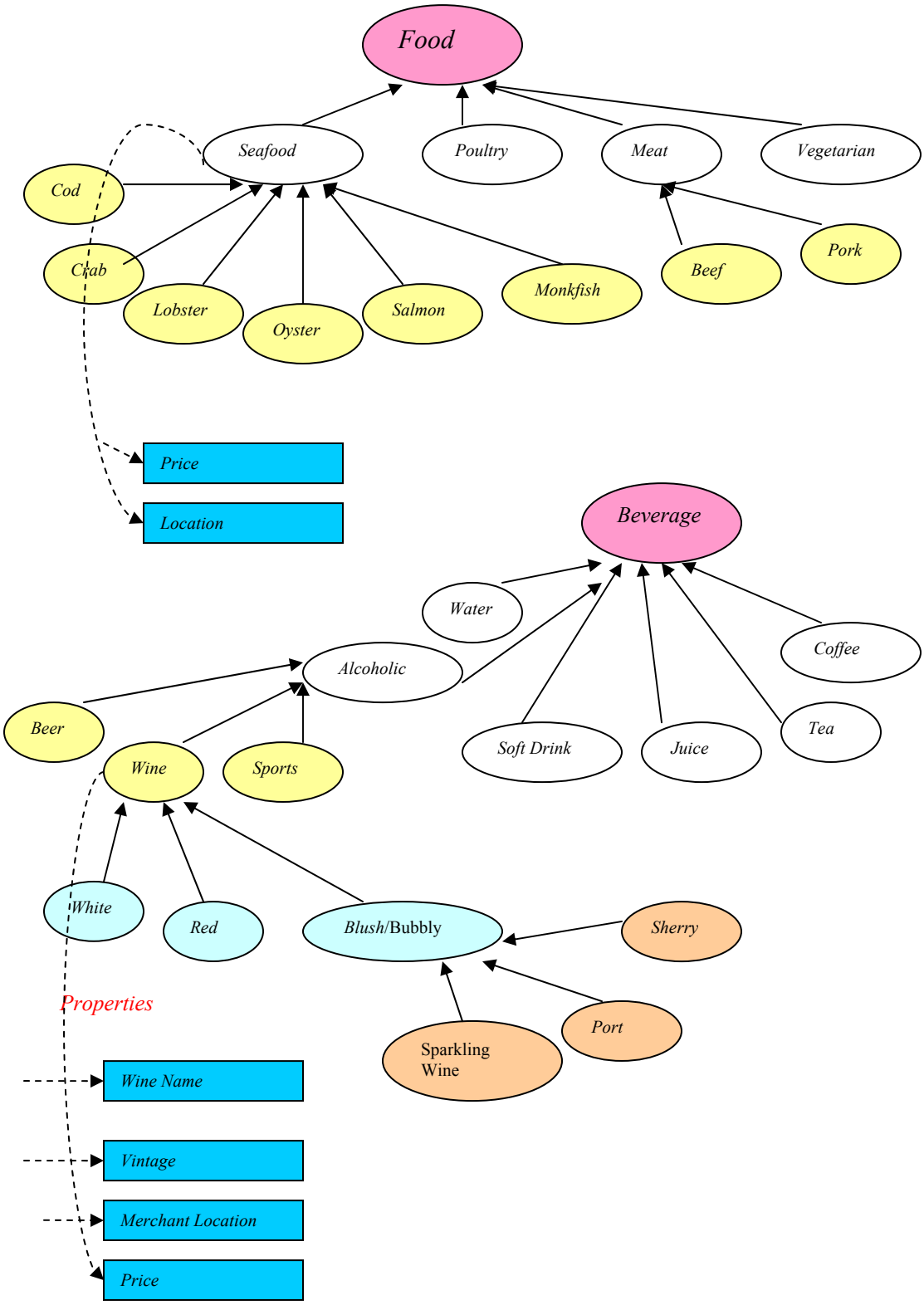

Figure 3.3 Drink and Food Ontology

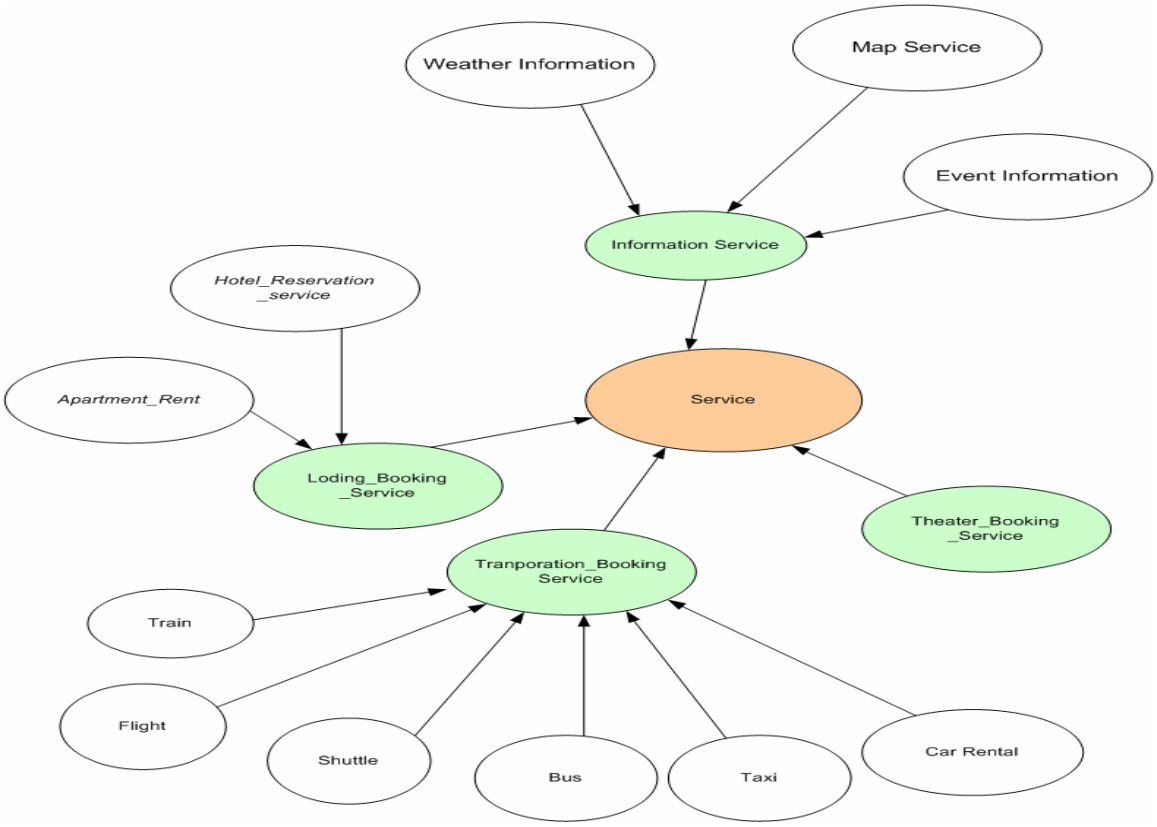

Figure 3.4 Web Services Ontology Example

3.3 Interface-Matching Automatic (IMA) Composition Technique

IMA composition technique 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 given the inputs provided by a user is found, or the system fails to generate such a composition of services.

The goal of this algorithm 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). If service ontologies are complex and the number of services is large this can be a challenging task. The composition starts from the service that needs one or more of the input parameters given 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 user's inputs.

Figure 3.5 shows an extended WS network with new relations by matching input parameters and output parameters. 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 dash-lines represent parameters that are not exact match but they 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 *S1* or *S3* and the other comes from *S4*.

In an example scenario, the user provides input parameter *Si1* and expects the output *So9 as* indicated in the graph. The composition goal is to find a shortest sequence of services from *S1* to *S9*. In this graph the source node *SI* represents the start state and

SF as the ending state, which are added for computing convenience. The weight of every edge is a function of quality rate (or execution time or other QoS that are interesting to the user) and semantic similarity value. In fact, five generic quality criteria for elementary services: (1) execution prices, (2) execution duration, (3) reputation, (4) reliability, and (5) availability [Zeng03]. For execution price and execution duration, Web service providers either directly advertise the execution price and duration of their operations, or they provide means to enquire it. The reliability is the probability that a request is correctly responded and availability of a service is the probability that the service accessible. We assume that the system records and calculates the availability and accessibility after Web services send the notification to the system. Users are also assumed to send their service reputation ranking to the system. In this thesis, we illustrate the automatic composition algorithm using quality rate specified in DAML-S profile. In practice, other Qos standards could be applied. Relative weights of these factors (λ) are defined by the users as follows:

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

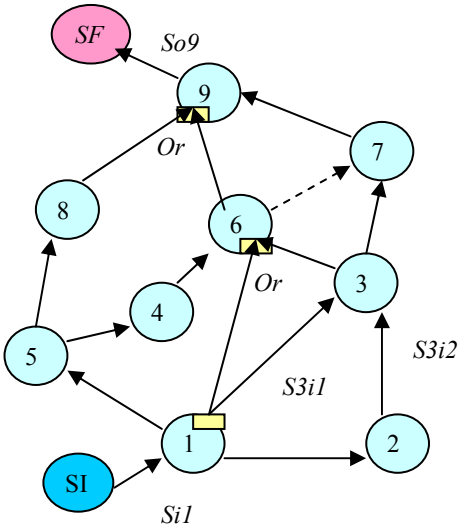

Figure 3.5 A Web Service Network Example

For the time being we consider four cases to check similarity (i.e., matching) of an output and input parameter from the same ontology: (1) if they are same, their similarity is maximal. For example, the output parameter of S1 (in Fig.3.5) exact match with the input parameter of S3 and they have the smallest value 1.0 (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, such as the output of S4 subsumes the expected output parameter - wine price. The similarity value depends on their distance in the ontology. (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. That applies to the relationship between S1 and S4. (4) When two parameters have no subsumption relation or they are from different ontologies, such as S2-S3, 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.

The composition algorithm aims to find the optimal collections of services considering execution time and semantic matching of parameters. We modify Bellman-Ford shortest-path dynamic programming algorithm to find the shortest sequence from initial stage at node *SI* to the termination node *SF*. In a common directed graph, we consider only one incoming edge and one outgoing edge for every node selected in the shortest path. The difference in our graph representation is that some services need more than two incoming edges as input parameters. Therefore, we not only record distance for every node, but also we trace the distance of every path at every node. When all the required input parameters are available, a service can be executed. Therefore, the distance

of every node is determined by the maximum value of distances of all the input parameters. For example, *S3* must have two incoming edges so a distance value of *S3* is determined by the maximum of *S3i1* and *S3i2* because *S3* can be executed after both of these inputs are available. In a different case, when there is more than one incoming edge fitting for one input parameter of a service, such as either edge 3-6 or 1-6 satisfies input of *S6*, we choose the minimum distance of 3-6 and 1-6 as a distance associated with input parameter of *S6*.

Suppose there are N services, each service may have multiple inputs and multiple outputs. Table 3.3 is a simplified algorithm for an optimal sequence of services.

Table 3.3 Automatic Composition Algorithm

```
Do times = 1 to N                  // in worst case, the dynamic algorithm need to update
                                          i distance of each service N time.
  Do i = 1 to N                    // check each service
     Do i_in = 1 to N_in           // N_in is the number of inputs on service i
        Do j = 1 to N              // for input i_in on service I, check each other service
           Do j_out = 1 to N_out   // check each output of service j.
                Check the output j_out on service j and the input I_in on service i , to see
                          whether they are similar(connected), if similar, find their similarity.
           enddo  (for j_out )
        enddo  (for j )
     Find the shortest distance to I_in on service I (it is the minimum of all available paths to I_in
     Enddo (for I_in )
  Find the shortest distance for service I. (if outputs of service I need all input, it is
   maximum distance of all its inputs plus its quality)
  Enddo  ( for I )

If all the required output have obtained, break
Enddo  (for times)
```

3.4 Automatic Composition Examples

Figure 3.6 is one implementation result given Ws network in Figure 3.5. Every box represents a service. Q is the quality rate of a service and D is the distance from start

service to this service. For generality, the input and output parameters are denoted as the integer numbers. The smaller difference of two integers means they are more similar semantically. Now if the input parameter of the query is the input of S1 (number 4) and the expected output is the output of S9 (number 114) and λ is 0.3, the shortest distance from S1 to S9 is 6.0. As we see, S 8 feeds one of the inputs of S9 rather than S6. And S6 select the input from S1 instead of S3.

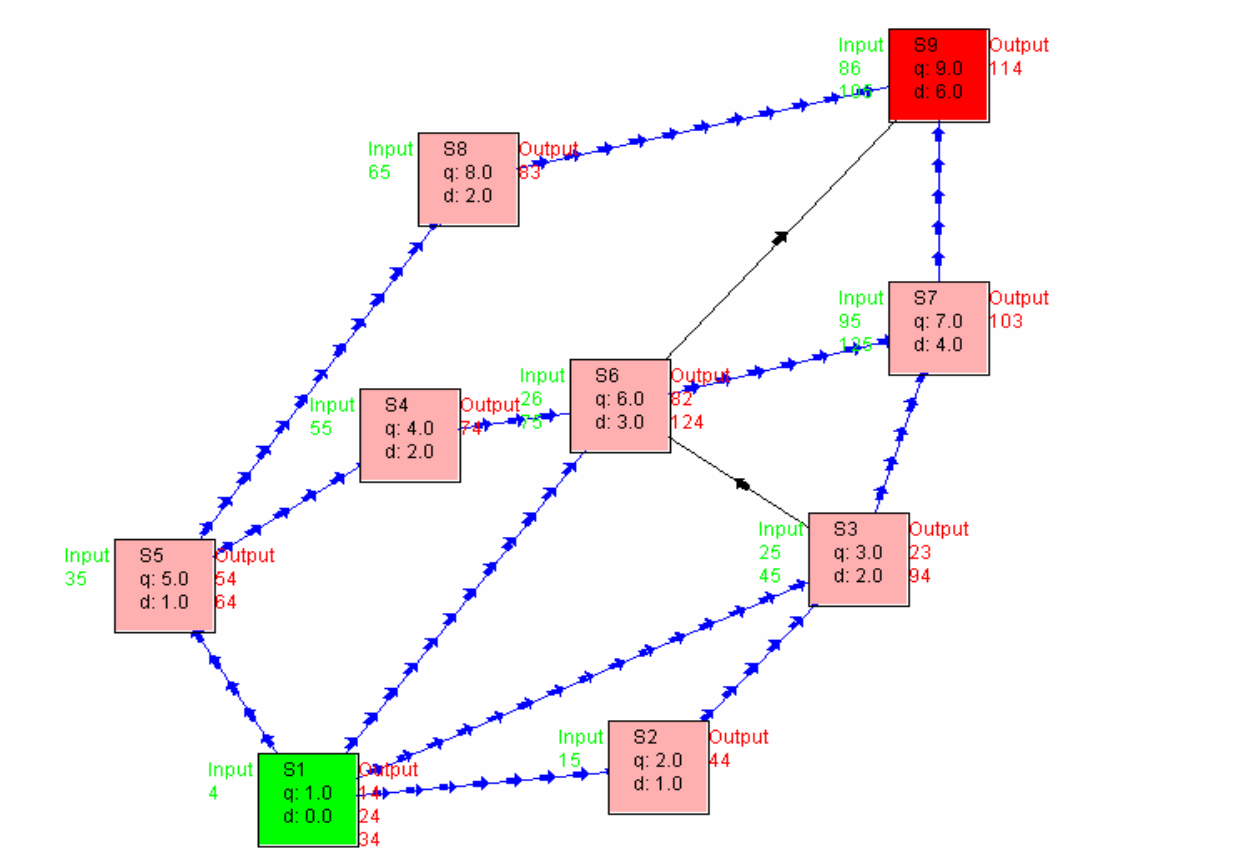

Figure 3.6 An IMA Composition Example (λ =0.3)

If we change λ to 0.7 and quality rate has less weight in matching function while the input and output parameters are held constant, the sequence with shortest distance is

shown in Figure 3.7. The shortest distance from S1 to S9 is 9.0. This time, S6 is the input provider for S9 rather than S8.

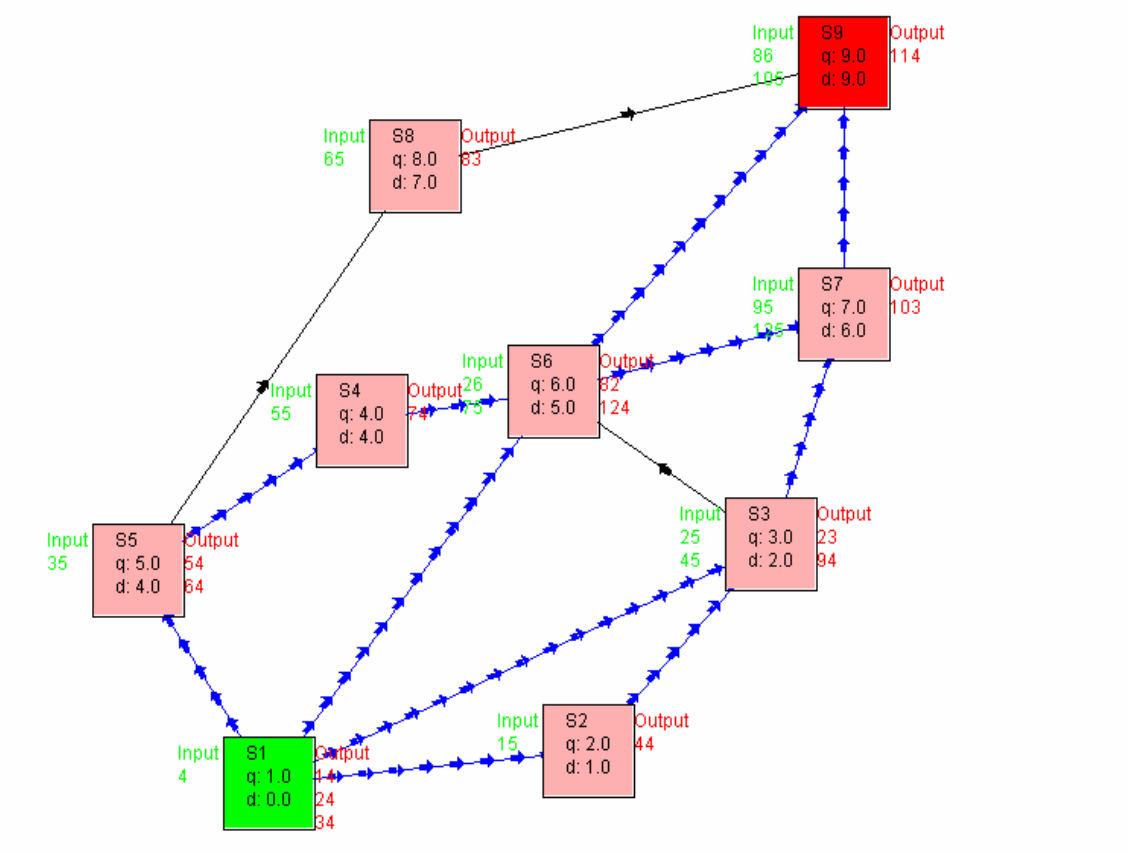

Figure 3.7 An IMA Composition Example (λ =0.7)

Figure 3.8 is a practical example. *Forest, Berry and WineAnswers* are three *Food-Wine-Matching* services that provide the matching wine given food or recipe. *Wine-Searcher, Internet-Wine and K.L.Wine* return the wine prices to the user given the wine name. The converter service can convert the dollar to Franc. *Recipe-Service* always presents the seasonal recipe if the user input the food name, such as beef.

The user inputs the seafood and awaits the matching wine prices in Franc currency as described in Query Format (Table 3.2). All of three services accept such seafood as

inputs. When λ is 0.2 and I/O similarity is low priority, the shortest path is *Forster (A Food-Wine matching service* → *Wine–Search* → *Converter*. The *Berry* matching service has the exact input parameter as the user's input, and *Forster* takes food as input, that is the super class of the seafood. That is regarded as a second best matching in our algorithm.

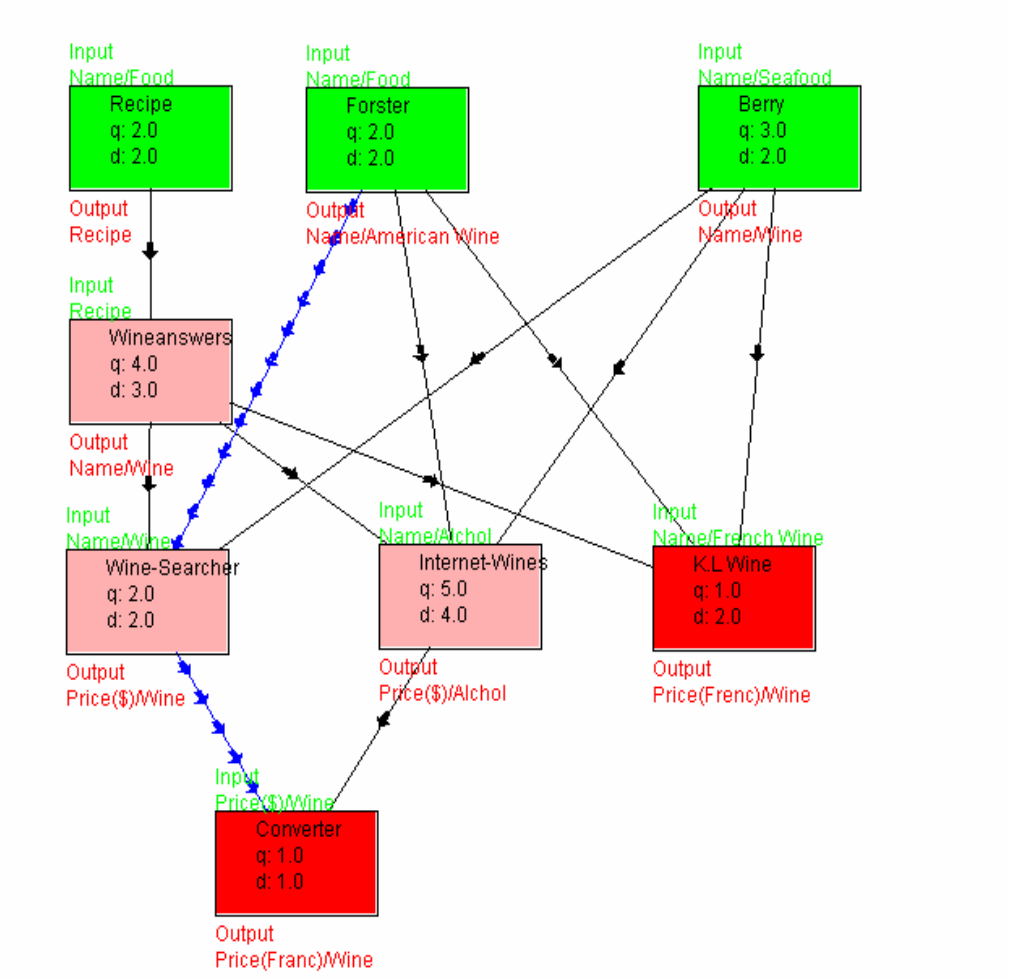

Figure 3.8 Food Wine Matching Service Composition (λ =0.2)

When we increase λ to 0.8 and that means we place more weight to similarity degree. The optimal sequence is shown in Figure 3.9. The shortest path is *Berry (A Food-Wine*

Matching Service) → *Internet-Wine –Search (*for wine prices*)* → *Converter (*from Dollar to Franc*)*. The reason is that *Internet-Wines* has the large value of quality rate (5), but it has good matching degree. If we concern less about quality, *Internet –Wines* stands out as the one service of the sequence. When λ is large and we pay more attention to the quality rate, *Internet-Wines* is not a good choice. *K,L Wine* is another service that can produce the expected result and it is adjacent to the food-matching service. The absence of the *K.L* is due to low the similarity degree. The output of preceding service is *Wine* and *K.L Wine* only accepts *French Wine* that is the subclass of the *Wine* and can't meet the requirement. So the similarity degree is pretty low.

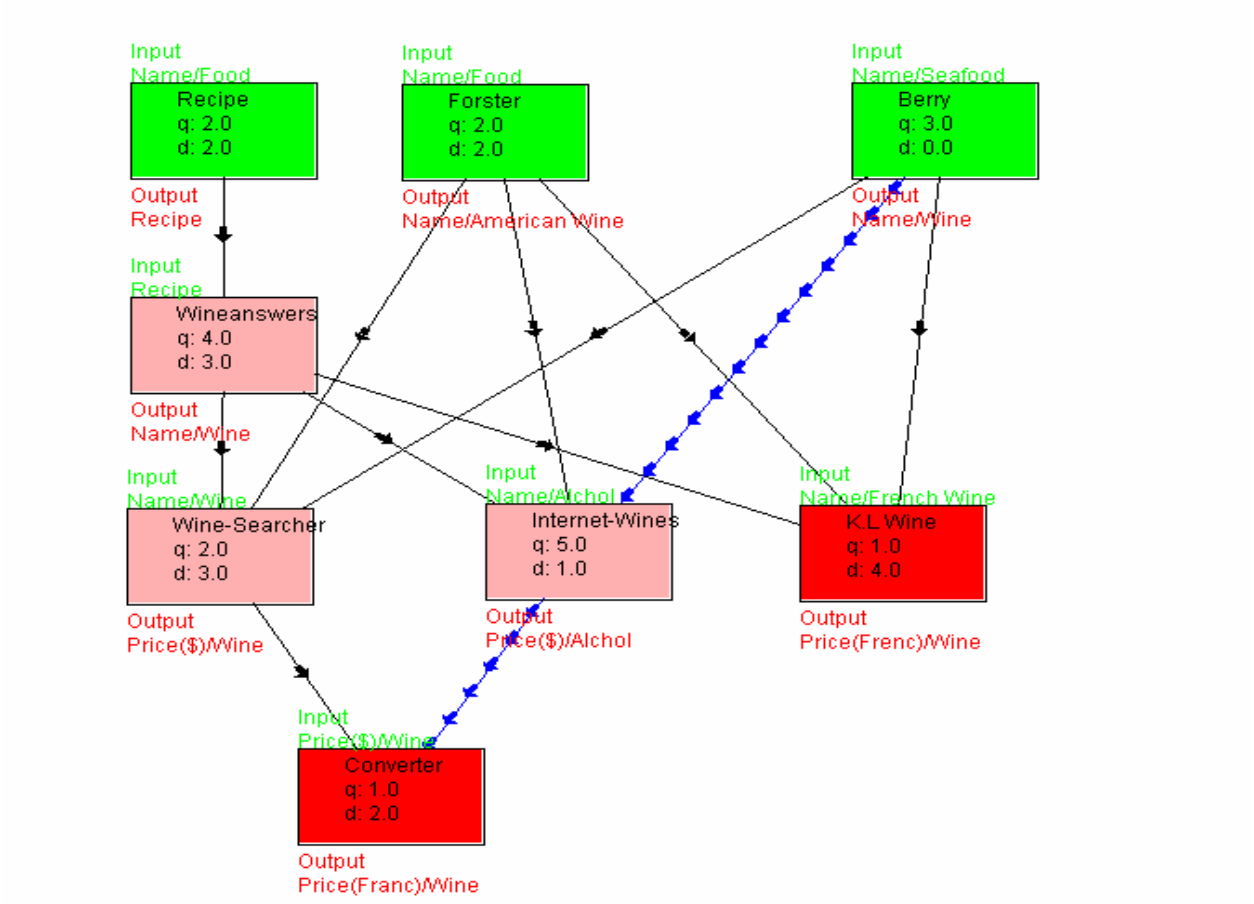

Figure 3.9 Food Wine Matching Service Composition (λ =0.8)

CHAPTER 4

HUMAN-ASSISTED COMPOSITION

In Interface-Matching Automatic (IMA) Composition Technique, the possible compositions are obtained by checking semantic similarities between interfaces of individual services without any predefined template and user's involvement in specification and adaptation. These compositions are ranked and then an optimum composition that can best satisfy a user's needs is selected. The IMA technique is more appropriate for information-retrieval services (e.g., querying wine prices, or matching different foods with drinks), that always return relatively simple results based on the user-supplied inputs. Complex world-altering services, such as flight booking and hotel reservation services cannot be located precisely and composed sequentially only considering the interfaces. This is mainly because that the user's requests are not only based on the inputs and outputs, but also quality of service (e.g., service quality in DAML-S 0.9), cost (e.g., ticket price), geographic region, other service attributes and user profile. Moreover, as the number of various services increases, the task of manually selection of an adequate service quickly becomes a very hard task. Therefore, new techniques are needed to help users in finding, filtering, and composing these services.

In Human-Assisted (HA) Composition Techniques, the system presents an available composite template that includes the basic service classes and defines the control-flow and data flow among the services. The user is allowed to add services under system or

remove some services in the original plan. Additionally, a dynamic service composition can facilitate the composition plans that are not only based on the constraints on service descriptions (e.g., interface) yet functionality properties, and service output values to be returned at run-time, such as prices of flight booking services. Thus depending on requirements on service descriptions and dynamic information supplied by the services, a customization and dynamic composition with more human involvement may be needed for many applications.

The Human-Assisted (HA) composition technique is dedicated to applications which require frequent human-intervention for an acceptable composition. The goal of this technique is to build a composition pathway (i.e., map) incrementally in a customized way.

4.1 Motivating Example

Consider a user, who is planning a round trip to London, U.K. from Atlanta, GA from May 1st to May 15th. Setting up a reasonable and affordable trip is a complex task and finding adequate services can be time-consuming. Initially, the system displays a travel planner for the composition (Figure 4.1).

The original travel planner provides a service composition template, which merely consists of the service classes that need to be replaced by service instances in the later stages of composition design. When the preferred service classes are determined, the user can filter out the service instances through various filters and select the best service instance based on her/his preferences.

The travel planner service template defines the control-flow and data-flow between services as in Figure 4.1. The template includes *Transportation_Booking_Service* and *Lodging_Booking_Service*, *Map_Service and Event_Information_Service.* The user supplies the arrival date, departure date and destination location to the *Event_ _Information_Service* and *Transportation_Booking_Service* that feeds the arrival date and departure date to the *Lodging_Booking_Service* as check-in and check-out time. The *Map_Service* produces a map and driving direction from a source place to a destination, such as a hotel. Eventually the composite service produces the *Flight Booking Confirmation*, *Hotel Booking Confirmation, Map and Driving Direction Information and Event Information* during this trip. Based on this template, the HA Composition tool guides the users to reduce the number of suitable service classes and instances and discover the service instances efficiently according to their preference. This is done through the following procedure:

1. Selection for service classes. (Select the appropriate subclasses of the services).
2. Selection for service instances.
3. Selection for neighboring services.

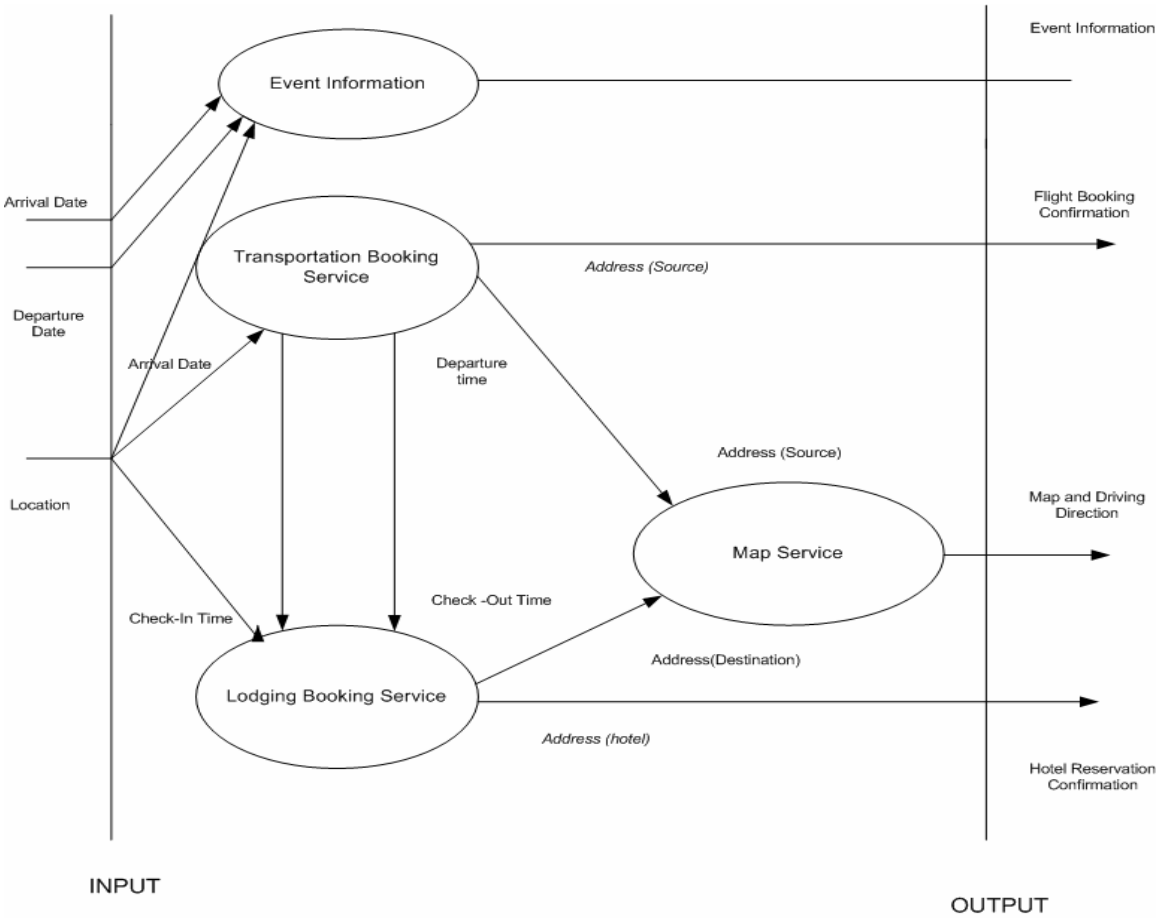

Figure 4.1 Travel Planner

4.2 Selections for Service Classes

Assuming a diversity of services and a vast amount of service instances, the users can't be expected to browse all service descriptions until an adequate match is found. Thus, the tasks of service selection and composition have to be supported by leading to a recommendation of a set of possible service classes and instances organized in a Web Service Ontology. For example, before the selection of Web Service instances, the user can opt for sub-classes of *Transportation_Booking_Service.* In our example,

Transportation_ Booking_ Service subsumes flight, train, shuttle, bus, taxi and limousine booking services. Obviously, *Flight_Booking_ Service* is the most appropriate vehicle for this long-distance trip. The HA composition tool would first list all of the subclass services of each service in hierarchical way as in Figure 4.2 (see top-left corner). For example, a user can click on Service 0 that represents *Transportation_Booking_Service*. Similarly, Service 1 is a *Lodging_Booking_Service*, which subsumes the *Hotel_Reservation* and *Apartment_Rent_Service*. Service 2 *Map_Service* includes *Map_Information* and *Map _Drive_ Information_Services* (See Figure 3.4 for corresponding Web Service Ontology Example). The Web Service classes are organized hierarchically in Web Service Ontology and can be discovered efficiently.

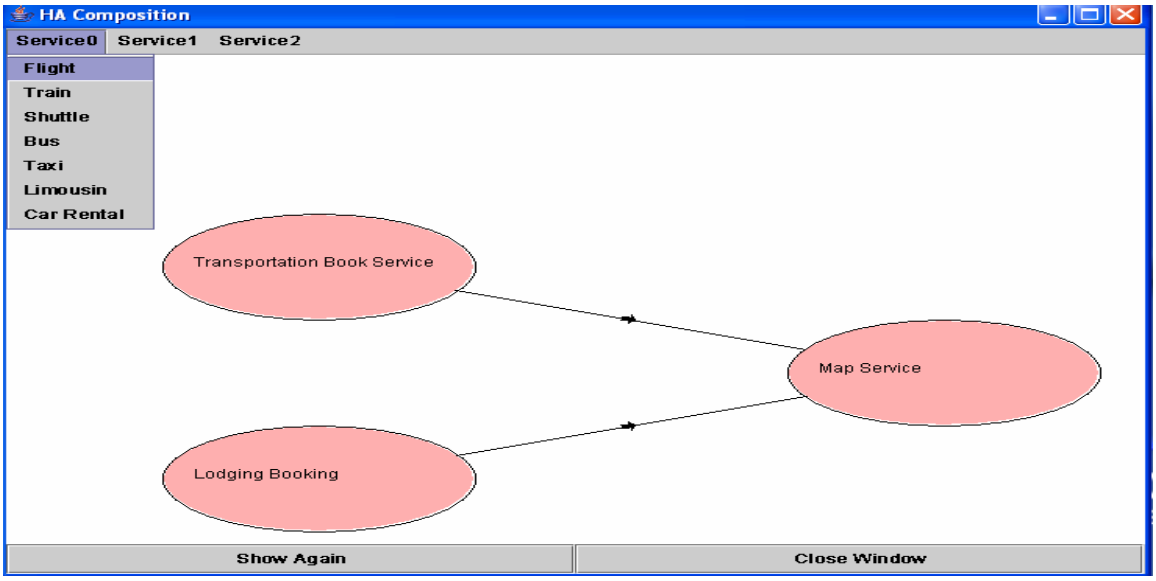

Figure 4.2 Selections for Service Classes

We argue that some service classes can be removed from the composite service plan if the user is not interested in their functions or if they are not necessary parts of the

composite service. For example, the user can remove the *Event_Information_Service* that only needs the destination address provided by the user and has no connections or relationship with other services in the template. For validity of the service plan, only a service or a sequence of services that has no succeeding services existing in the composite plan can be removed. Thus their removal does not affect the execution of other services.

4.3 Selection for Service Instances

After the service class selection terminates, the service classes selected need to be associated with service instances. For this purpose, we use DAML-S Profile and several filters to discover an appropriate service instance. The HA composition tool presents the selection interface (Figure 4.3) to the user when the service instance selection starts.

In the initial state, all of the service instances will be displayed in the box "Select a service", where their name and quality rate, geographic region and other basic information are also presented. For example, hotwire.com has 6 as quality rate and its service geographic region limited to USA (in this work we don't focus on automatic evaluation of quality metrics for Web Services). The HA composition tool allows the user to filter out the appropriate services by any of the following filters or by combining them together.

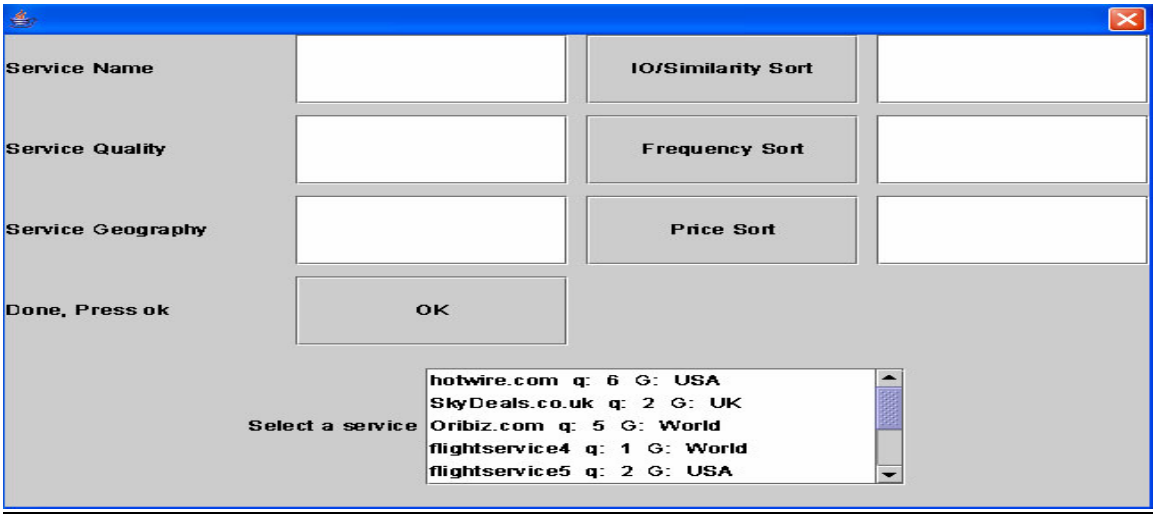

Figure 4.3 Selections for Service Instances

Filter 1: Service Name Filter

When the user supplies the service name in the box "Service Name", the *Service Filter* navigates in the WS file database (including DAML-S Service Descriptions in Figure 3.2) using key-word search and returns the available services with service. For example, if we want to learn the information about "Orbitz.com" and input its name, the Orbitz.com will be shown in the box "Select a service" on the bottom. Otherwise, an error message would be returned if no such service exists in the database.

For the case when a service profile has more than one service name and text descriptions and a service might have multiple registries, we can use the discovery mechanism in METEOR-S Web Services Discovery Infrastructure (MWSDI) to discover the service with multiple names and browse their descriptions in different registries if registries ontologies are deployed in this system.

Filter 2: Service Quality Rate Filter

The services may have different quality ratings defined in DAML-S (see Table 3.1). Therefore, a service provider may want to publish its rating within a specified rating system, to demonstrate the quality of service it provides. Our algorithm exploits this and returns a set of services with quality rate equal or greater than a user specified quality rate; after that the services are shown in the "Select a service" box. For instance, when we supply 2 (good) in the box, that means we are looking for the services with 2 (good) or 1 (excellent quality rate). We assume that these service quality rates share the same quality rate ontology or their specifications should be comparable.

Filter 3: Geographic Region Filter

The number of services with high quality rate might still be extremely high in many cases and some services might be little relevant to the composition even they rank top. For example, consider the travel planner example. The *Hotel_Reservation_Service* whose service region is U.S. is not an appropriate answer to a user who is looking for a hotel in London, U.K. Similarly, the pizza delivery services are only relevant to local clients. Therefore, it is necessary to filter the irrelevant services in terms of geographic region. A region ontology (Figure 4.4) specifies semantic relations among different regions like countries (e.g., U.K. is a member of Europe and Europe is part of World). *The Flight_Booking_ Service* in U.K., Europe or worldwide would all be stratifying answers to the client who requires the service in U.K. However, if the user only prefers services in Europe, the possible results are the services with restriction in Europe or World. In this

way, the number of services satisfying geographic restrictions will decrease in some extent.

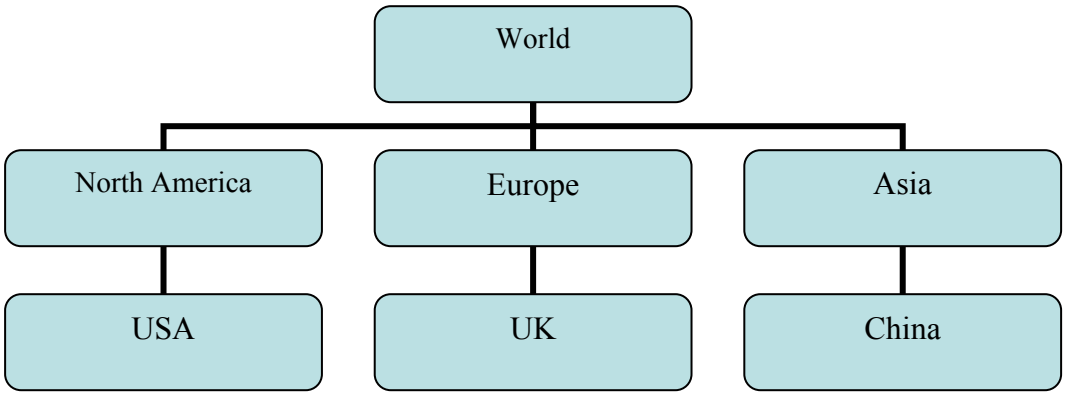

Figure 4.4 Region Ontology

In our example, as the user input "UK" in the box "Service Geography", four services are selected and listed in the box "Select a service" shown in Figure 4.5. *SkyDeals.com* is a service based on U.K and *Orbitz.com, flightservice4* and *WorldCome* are all worldwide services. The service quality and service geography filters can be utilized individually or simultaneously. If they are used together, the results returned include the services that meet both quality and geography requirements by the user.

In addition to the service quality rate and service geographic region, the same ideas and methods could be applied for filtering of service by other properties specified in DAML-S.

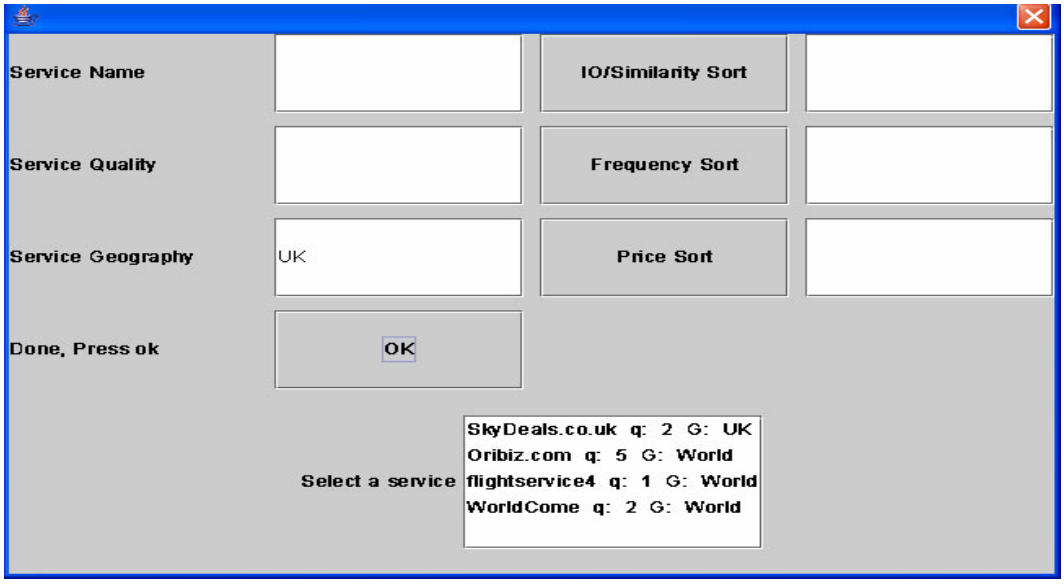

Figure 4.5 Service Geography Filter

Filter 4: IOPE (Input, Output, Precondition and Effect) Similarity Filter

An essential component of the service profile is the specification of what the service provides and the conditions that have to be satisfied for a successful result. The input property specifies the information that the service requires to proceed with the computation. The output properties define the result of the operation of the service. For example, the airport location and hotel location inherit from the concept classes, such as location or address. Also a *US address* can be defined as the input parameter of a *Map_ Service* and it is subsumed by the *Address* class/concept in a domain ontology.

After a user selects one *Flight_Booking_Service* instance and one *Hotel _Reservation_Service* instance, s/he may need to find a *Map_Service* that will match the interfaces of previous two services semantically. The matching step is dedicated to finding correspondence between a service template and a service instance [Cardoso02].

The algorithm calculates a similarity degree, using the individual parameters and constructs the whole profile similarity. In Figure 4.6, every pair of parameters has a similarity degree (which is not displayed for brevity), and the whole profile degree could be obtained by using their individual similarities. For the time being, we consider four cases to check similarity (i.e., matching) of an output and input parameter from the same ontology: (1) If they are same, their similarity is maximal. (2) If an 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. (3) If the output parameter of the former service subsumes the input parameters of the succeeding service. This is the third similarity. (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]. Tverskly's model is based on the idea that common features increase the similarity of two concepts, while feature difference decreases the similarity. The optimal service is the service with highest similarity value considering the I/O similarity (Figure 4.7).

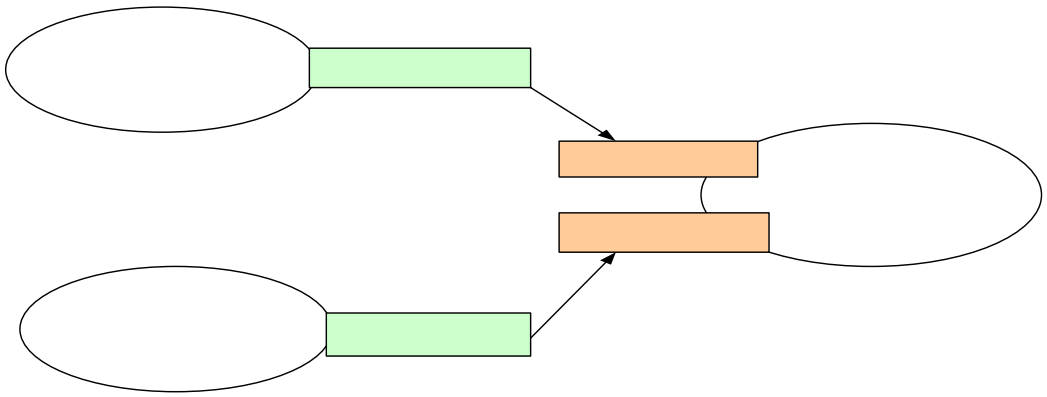

Figure 4.6 I/O Similarity

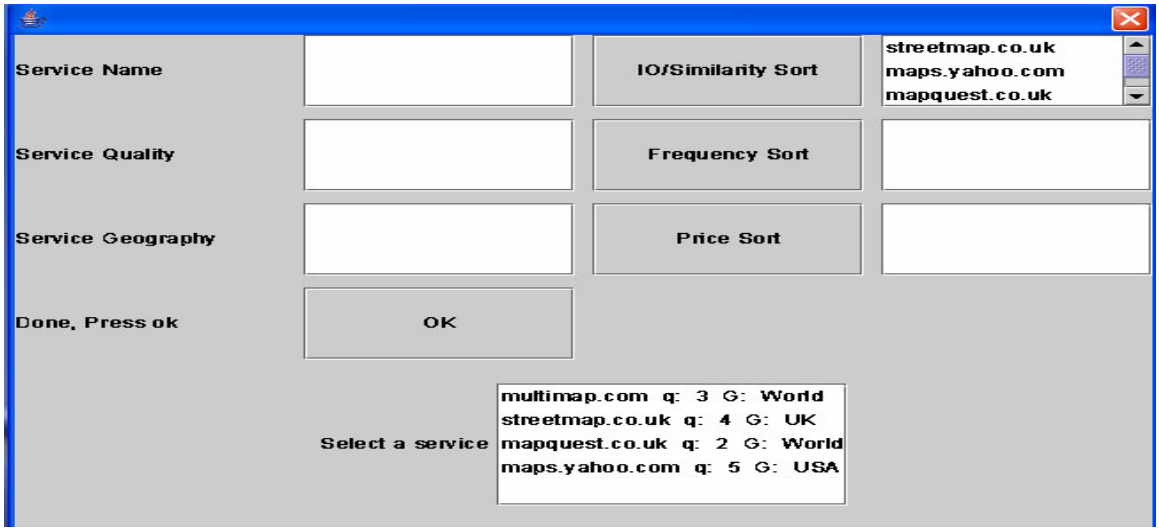

Figure 4.7 I/O Similarity Rank

A user selects the top ranked service such as *Streetmap.co.uk* as the *Map_Service* instance. The tool shows I/O of every service selected (Figure 4.8)

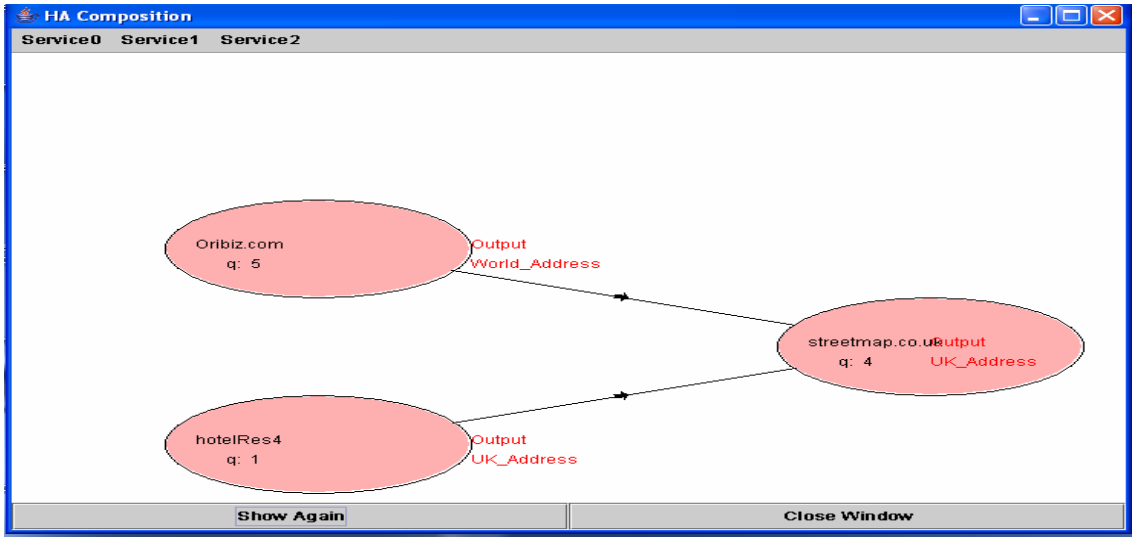

Figure 4.8 Services Selected with Their I/O Matching

Filter 5: Personal Profile History Filter

The personal profile records the history of service instances used involving usage frequency by the user. We assume that the service with highest usage frequency is most likely to be selected in the future. If the user clicks the "Frequency Sort", a set of services in the box "Select a service" will be returned and sorted in descending order based on the usage frequency by the user.

Filter 6: Price Filter

The system also allows the user to select a service based on the output value, such as ticket price. The hotel with cheapest rate can't be located until all the results of services are compared after execution. If there are too many services matching this requirement, the service invocation and comparing might be time consuming. Usually, the user can select some service with better quality rates, good I/O matching or high frequency etc; then sends them to the price filter for dynamic selection at run-time. For example, if the user defines the quality rate of *Hotel_Reservation_Service* is greater than 2 and geographic region is in UK, three *Hotel_Reservation_Service* instances returned as in "Select a Service" box (Figure 4.9). At the design stage, the user has no clue what might be cheapest price rate and wants to select some. The cheapest price service is known only at execution time. If the user supplies 2 in "price filter" box, HA Composition tools would automatically select the first two services as the candidate services and only return the hotel reservation confirmation with the cheapest price. This is equivalent to placing an XOR-split in the composition with two candidate services. The generated composite path is shown in Figure 4.11.

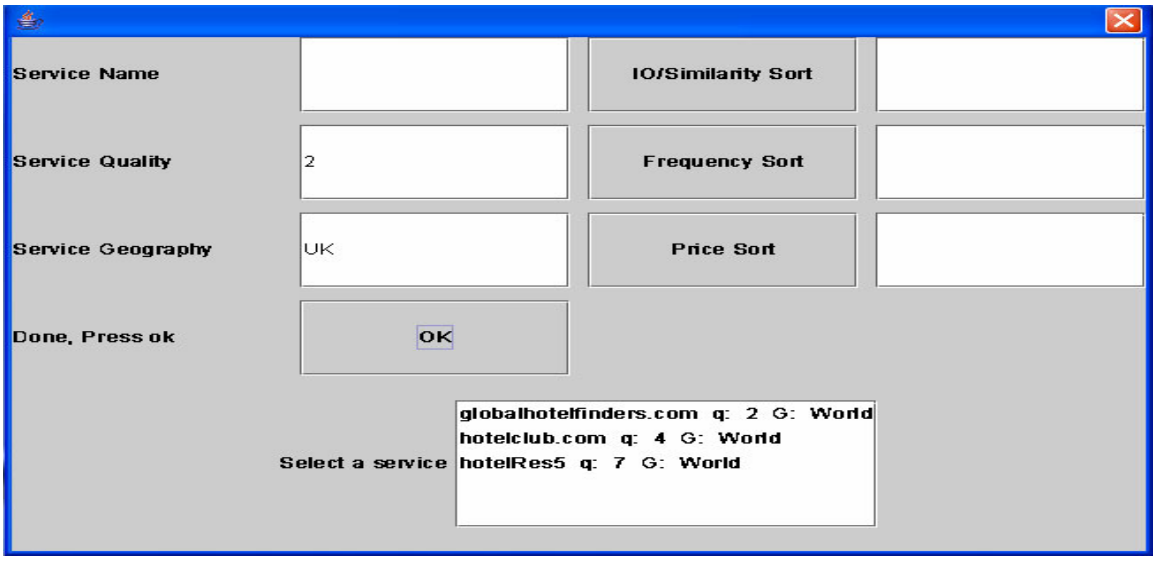

Figure 4.9 Candidate Services for Price Filter

Filter 7: Process Filter

Service model tells "how the service works", that is, it describes what happens when the service is executed. We argue that for non-trivial services (those composed of several steps over time), this description may be used by a service–seeking agent to perform a more in-depth analysis of whether the service meets its needs.

For example, a traveler may want to reserve a hotel as near as possible to the historical downtown area in Charleston, SC. A reservation service that sorts hotels in geographic proximity to the user's location is preferable (e.g., Travelocity.com). In contrast, *Hotel_Reservation_Service (HRS.com)* has no such function. Another case is that a user wants to search a *Flight_ Booking_Service* which provides flight schedule before the credit card is charged, such as expedia.com. S/he may choose to query the sequence of activities of the service to avoid a service like priceline.com, which charges the credit card before providing a schedule. The difference between these two services arises because of the order of their internal steps, although their advertised descriptions

may be the same or very similar. This is what we called “functionality” difference of the service. The system navigates the registered process of services and query if the service has the required function or the sequence of the functions. The process filter would filter out those services that can not satisfy the requirements. However, we leave implementation of process filter as a future work.

4.4 Selections for Neighboring Services

Given the WSs network (Figure 3.2) and one *Flight_Booking_Service*, a *Weather_ Information_ Service* accepts the outputs of *Flight_Booking_Service* as their inputs. Usually it is helpful for individual travelers, but it is not included in the original service template. The user possibly could not compose them in the final composite service without guidance. For simplicity, the system only displays the services whose inputs can be satisfied by the preceding service completely and do not need other services. Without reachability constraints, the alternative services could go into a large snowball and become an unnecessary burden to the user. Another scheme is that we can define the distance of the services related the original service or limit the number of services in this chain extending the process.

For example, when Orbtiz.com is selected as *Flight_Booking_Service*, a list of weather services matches to the Orbitz.com in terms of geographic region and I/O parameters would be presented for selection. If the user then chooses intellicast.com, then the service appear to in GUI as the neighboring service as shown in Figure 4.10.The existing template thus can be extended or modified, that is, the system recommends some

services that are either not defined by the user, or in the original template if the inclusion of such services does not violate the sequence order of the original planner.

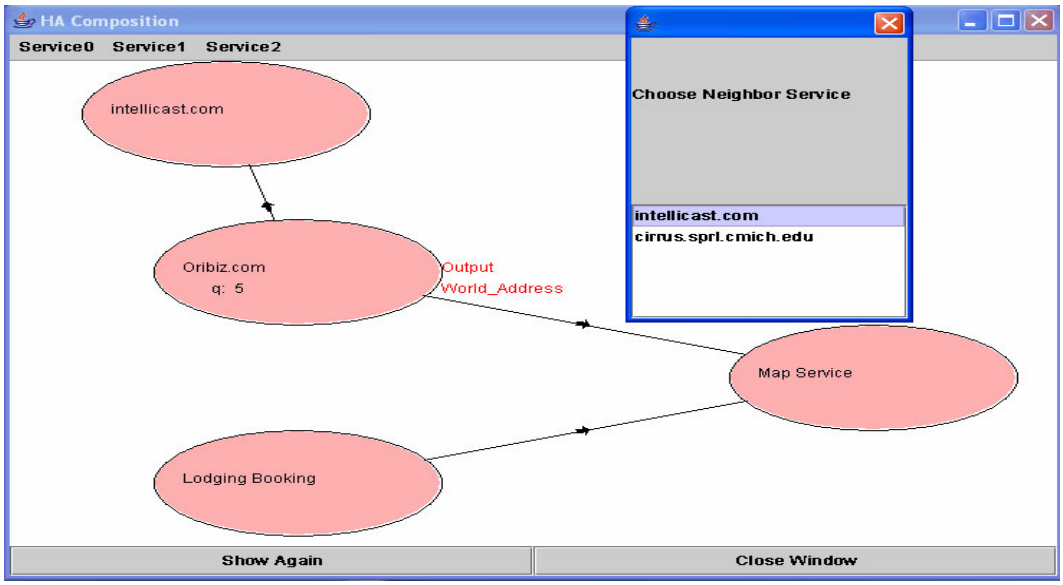

Figure 4.10 Selections for Neighboring Services

The user is guided to select service subclasses and instance for every service class in the original template and add the necessary neighboring services described as above. Eventually, the original template is modified according to the user's requirements as shown in Figure 4.11.

The generated composition is based on the original composite service template which can be described semantically enable as MWSCF in METEOR-S. The Service Composer replaces the service classes with service instances, adds the new neighboring services and removes the unnecessary ones, and then sends the generated composite service to the Service Execution Monitor.

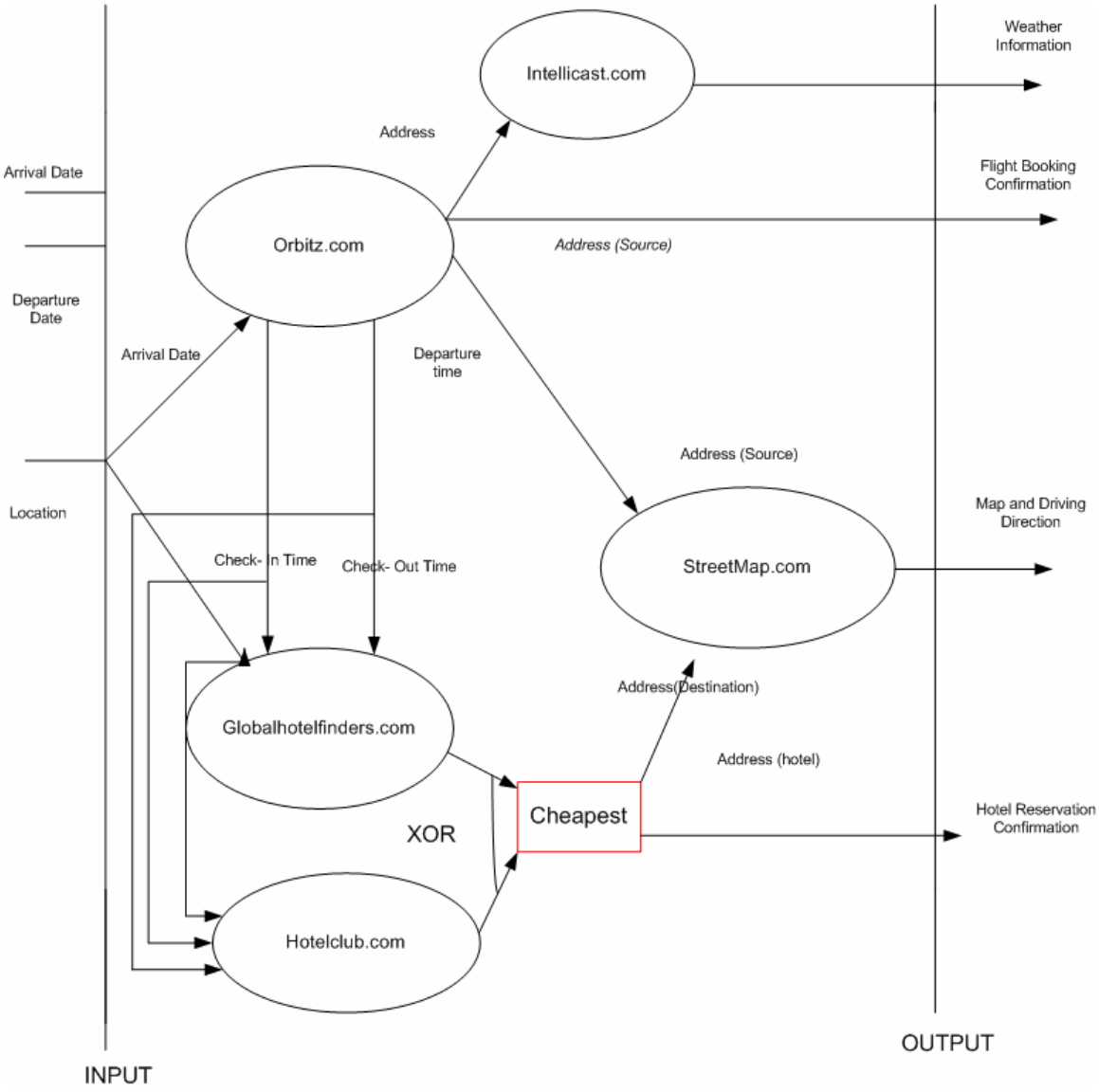

Figure 4.11 Trip Composite Service Graph

CHAPTER 5

CONCLUSION AND FUTURE WORK

Today's search engines and knowledge discovery tools help users to locate relevant documents and assemble relevant knowledge for effective decision-making respectively, and improve their capabilities continuously using semantics. Similarly, users need new tools to help them discover and assemble services into processes for easier and better quality workflow executions given increasing number and complexity of WSs. This thesis illustrates IMA and HA composition techniques for semantic WSs. The main of contribution of this thesis is explicitly ontological descriptions of service descriptions, and finding optimal compositions in a flexible way. This also takes QoS specifications of services to find a composition with minimal cost and highest qualities. We also developed some filters to help users to make better service selection decisions in composition.

However, some interesting technical problems still lie ahead. For example, users may need to compose services based on their internal computations when their profiles may not convey adequate semantics to differentiate them. In this context, modeling Web Service functionalities as pre- and post-conditions and potentially state machines may provide further improvements in Web Service compositions.

REFERENCES

[Ambite2003] J. Ambite, G. Barish, Craig A. Knoblock, M. Muslea, Jean Oh, and S. Minton. Getting from Here to There: Interactive Planning and Agent Execution for Optimizing Travel. *The Fourteenth Innovative Applications of Artificial Intelligence Conference* (IAAI), Edmonton, Alberta, Canada, 2002.

[Arpinar04] I. B. Arpinar, R. Zhang, B. Aleman and A. Maduko. Ontology-Driven Web Services Composition. *IEEE E-Commerce Technology, July 6-9, San Diego, CA.*

[Balke03] W. Balke, M. Wagner. Towards Personalized Selection of Web Services. *WWW 2003, May 20-24, 2003, Budapest, Hungry.*

[Benatallah02] B. Benatallah, M. Dumas, Q. Sheng, and A. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. *IEEE Intl. Conf. on Data Eng., San Jose, 2002.*

[Cardoso02] J. Cardoso (2002). Quality of Service and Semantic Composition of Workflows. *Ph.D. Dissertation. Department of Computer Science, University of Georgia, Athens, GA.*

[Cardoso03] J. Cardoso, and A. Sheth. Semantic e-Workflow Composition, *Journal of Intel. Info. Sys., 2003.*

[Chakraborty01] D. Charkraborty, F. Perich, S. Avancha, and A. Joshi. DReggie: A

smart Service Discovery Technique for E-Commerce Applications. *20th Symposium on Reliable Distributed Systems (SRDS). New Orleans. October, 2001.*

[Chandra03] S. Chandrasekaran, J. Miller, G. Silver, I.B. Arpinar, and A. Sheth. Performance Analysis and Simulation of Composite Web Services. *Electronic Markets: The Intl. Journal of Electronic Commerce and Business Media, 13(2), 2003.*

[Cheng02] Z. Cheng, M. P. Singh and M. A. Vouk. Composition Constraints for Semantic Web Services. *In Proceedings of the International Workshop Real World RDF and Semantic Web Applications 2002, 2002.*

[DAML-S Coalition03] A. Ankolenkar, M. Burstein, et. Al. DAML-S: Web Service Description for the Semantic Web. *The First International Semantic Web Conference, Stanford, 2001.*

[Fensel02a] D. Fensel, C. Bussler. Semantic Web Enabled Web Services. *2nd Annual Diffuse Conference, Brussels, Belgium, January 2002.*

[Fensel02b] D. Fensel, C. Bussler, Y. Ding, and B. Omelayenko. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.

[Gil2003] Y.Gil. Interactive Composition of Computational Pathways. *http://epicenter.usc.edu/cmeportal/docs/1*

[Hoschek02] W. Hoschek. Peer to Peer Grid Databases for Web Services Discovery, *Grid Computing: Making the Global Infrastructure a*

Reality" Ed(s): F. Berman, G. Fox, and T. Hey, Nov. 2002, Wiley.

[Klein01] M. Klein, and A. Bernstein. Searching for Services on the Semantic Web Using Process Ontologies. *International Semantic Web Working Symposium, August 2001.*

[McIlraith01] S. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intel. Sys. March/April 2001.*

[McIlraith02] S. MaIlraith, T. C. Son. Adapting golog for composition of semantic Web services. *In Proc. KRR, 482-493.*

[Narayanan02] S. Narayanan, and S. A. Mcllraith. Simulation, Verification and Automated Composition of Web Services. *11th Intl. WWW Conference, Honolulu, 2002.*

[Palucci02] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. "Semantic Matching of Web Services Capabilities". *The First Intl Semantic Web Conference*, *Sardinia (Italy), June, 2002.*

[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 (to appear).*

[Patil04b] A. Patil. METEOR-S WEB SERVICE ANNOTATION FRAMEWORK. *(The Master Thesis, 2004)*

[Ponnekanti02] S. R. Ponnekanti, and A. Fox. SWORD: A Developer Toolkit for Building Composite Web Services. *11th WWW Conference, Honolulu, 2002.*

[Schlosser02] M. Schlosser, M. Sintek, S. Decker, and W. Neijdl. A Scalable and

Ontology-Based P2P Infrastructure for Semantic Web Services. *2nd IEEE Intl. Conf. on Peer-to-Peer Computing, 2002.*

[Sheth99] A. Sheth, W. M. P. Van Der Aalst, and I. B. Arpinar. Processes Driving the Networked Economy: Process Portals, Process Vortexes, and Dynamically Trading Processes. *IEEE Concurrency Journal, pp. 18-31, July-September 1999.*

[Sheth03] A. Sheth. Semantic Web Process Lifecycle: Role of Semantics in Annotation, Discovery, Composition and Orchestration*, invited talk at WWW 2003 Workshop on E-Services and the Semantic Web, Budapest, Hungary, May 20, 2003.*

[Sirin03] E. Sirin, J. Hendler, and B. Parsia. Semi-automatic composition of web services using semantic descriptions. Web *Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, April 2003.*

[Sivashanmugam 03] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards*, Intl. Conf. on Web Services, Las Vegas NV, June 2003.*

[Sivashanmugam 03b] K. Sivashanmugam. The METEOR-S Framework for Semantic Web Process Composition *(The Master Thesis 2003)*

[Sora01] I. Sora, and F. Matthijs. Automatic Composition of Software Systems from Components with Anonymous Dependencies, *Technical Report CW 314, Leuven, Belgium, May 2001.*

[Srivastava03] B. Srivastava, J. Koehler. Web Service Composition –current

solutions and open problem. *ICAPS 2003 Workshop on Planning for Web Services, Trento, Italy, 10 June, 2003* .

[Tosic01] V. Tosic, D. Mennie, and B. Pagurek. On Dynamic Service Composition and Its Applicability to E-business Software Systems. *Workshop on OO Business Sol. ECOOP, Budapest, Hungary, 2001*.

[Verma04] K.Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar and John Miller, METEOR–S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management (to appear, 2004).*

[Zeng03] L. Zeng, B. Benatallah, M. Dumas. Quality Driven Web Services Composition. WWW2003, May 20-24, 2003, Budapest, Hungary.

[Zhang03] R. Zhang, I. B. Arpinar, and B. Aleman-Meza. Automatic Composition of Semantic Web Services. *Intl. Conf. on Web Services, Las Vegas NV, June 2003.*