Web Service Composition with Case-Based Reasoning

Benchaphon Limthanmaphon and Yanchun Zhang

Department of Mathematic and Computing
University of Southern Queensland
Toowoomba, Australia

{Limthan,zhang}@usq.edu.au

Abstract

To run a smart E-Business or provide efficient Web service, a web services composition model is needed. Web services composition refers to the process of collaborating the heterogeneous web services. This paper presents a model of web services composition by using Case-Based Reasoning (CBR) techniques. CBR is applied in the process of service discovery, which is the crucial composition process. Our service composition model integrates the two behaviours of proactive and reactive service compositions. We will address dynamic composition and collaboration among services. The similarity feature of CBR is used for efficient service discovery.

Keywords: Web Service, Case-Based Reasoning, Web Service discovery.

1 Introduction

E-Commerce or Internet Commerce has become the casual medium of our daily activities because of its availability and facilitation. However the process of E-commerce service is more complex. Collaboration among cyber businesses is required and this requires the use of Web Services.

Web Services or E-Services are modular Internet-based applications that communicate with one another over the Internet (Machiraju, Dekhil, Griss, Wurster 2000). Web Service composition is a key challenge to manage collaboration among Web Services. Service composition refers to the technique of composing arbitrarily complex services from relatively simpler services available over the Internet (Chakraborty, Joshi 2001).

Consider the example of a business trip. The service provider may integrate existing services such as airline ticket, hotel reservation and/or car rental services. To integrate or compose these services is not a difficult task but to achieve efficient and better service is a challenge. For instance, if a client has a limited budget, a compromise cost may be needed for each service. The professional service provider has to provide each suitable service. It is important to have an intelligent and efficient mechanism to control the service composition.

This paper proposes a model for service composition. We apply Case-Based Reasoning (CBR) techniques for service discovery. It can be used to allocate the pre-assembled closest services depending on user requirements. We also describe the relationships among the sub-services.

The rest of this paper is organised as follows: Section 2 presents an overview of service composition and related works. A web services platform is presented in section 3. The relationship among sub-services and the CBR framework for service discovery is presented in section 4. Section 5 presents our service composition model. Finally, the conclusion and future research direction are presented in section 6.

2 Web Service Composition

2.1 Overview

Service Composition refers to the process of creating customised services from existing services by a process of dynamic discovery, integration and execution of those services in a deliberate order to satisfy user requirements (Chakraborty, Perich, Joshi, Finin, and Yesha 2002). Chakraborty and Joshi (Chakraborty, Joshi 2001) classified the service composition from two points of view: the first is proactive and reactive composition; the other is mandatory and optional service composition.

Proactive composition means offline or pre-compiled composition. Services that compose in a proactive manner are usually stable and used at a very high rate over the Internet. *Reactive Composition* means creating a compound service on the fly. It requires a component manager to take the responsibility of collaborating with the different sub-services to provide the composite service to the client. The interaction cannot be predefined and varied according to the dynamic situation. It is better to exploit the present state of services and provide certain runtime optimisations based on real-time parameters like bandwidth, and cost of execution of the different sub-services.

Mandatory composite services refer to the rule that all the sub-services must participate for the proper execution. These types of services will be dependent on the successful execution of other sub-services to produce a satisfactory result. *Optional composite services* are the opposite of mandatory composite services. They do not necessarily need the participation of certain sub-services for the successful execution of a query.

Service composition is required to improve the quality of E-business. It provides two important benefits. First, more automation, it reduces the amount of human interaction needed to exchange or transfer information.

Second, more flexibility, the sub-services can be composed depending on the client's requirement. Satisfying the client is the heart of doing business or providing services.

2.2 Related Works

There have been a few works done in service composition. For example, the dynamic service composition called software hot-swapping at the Carleton University, Canada (Mennie and Pagurek 2000), successor to the research in the field of dynamic software component upgrading at runtime, has been developed. They have identified two different ways of carrying out heterogeneous service composition. First, a composite service interface - this method of composite interface formation identifies a good way of exposing the necessary available services to a client. The advantage of this technique is the speed that a composite service can be created and exported as an interface. A new sub-service does not need to be constructed dynamically. Only the interface of that sub-service needs to be known to construct the composite interface. Second, the standalone composite service is suitable when the performance of the composite service is more critical. A standalone composite service is created by dynamically assembling the available services by way of pipes and filters. All the service components remain independent.

eFlow (Casati, Ilnicki, Jin, Krishnamoorthy, and Shan 2000) from HP laboratories is an e-commerce services composition. A composite service is modelled by a graph, which defines the order of execution among the nodes or different processes. The graph modelling a composite service consists of service nodes, event nodes or decision nodes. Service nodes represent simple or composite services. Event or decision nodes specify alternative rules that control the execution flow. Event nodes enable services and the process to receive various kinds of event notification. eFlow engine offers the facility of being able to plug in new service offerings and enables adaptiveness with several features such as dynamic service discovery, multi service nodes and generic nodes.

The main objective of the Ninja (UC Berkeley) service composition at the University of California Berkeley is to enable diverse clients with varying resources and network accessibility to access the same unchanged network service. This addressed the problem of changing a network service to suit the needs of various clients with respect to bandwidth and resource capabilities, which were not a very good design choice. Their service composition platform addresses this by allowing the service to remain as it is, and by dynamically plugging in required services to cater to the needs of various clients.

Self-Serv (Benatallah, Dumas, Fauvet and Paik 2001) is a framework through which web services can be composed and the resulting composite services executed in a decentralised dynamic environment. The providers of the services participating in a composition collaborate in a peer-to-peer fashion to ensure that the control-flow dependencies expressed by the schema of the composite service are respected. A subset of statecharts has been adopted to express the control-flow perspective of composite service. States can be simple or compound. The data-exchange perspective is implicitly handled by variables: input and output, the parameters of services and events. Other web service composition developments are also summarised in their paper (Benatallah, Dumas, Fauvet and Rabhi 2001)

3 Web Services Platform

Web services require a method to define the data types used in web services messages. XML provides a meta-language to express complex interactions between clients and services or between components of a composite service transported over HTTP (Hypertext Transfer Protocol). SOAP (Simple Object Access Protocol) is an XML-based mechanism for exchanging structured data among network applications while WSDL (Web Services Description Language) defines an XML-based grammar for describing network services (WSDL 2000). WSDL describes what functionality a service provides, where the service is located, and how to invoke the service while UDDI (Universal Description, Discovery and Integration.) is used for listing what services are available.

3.1 Universal Description Discovery and Integration (UDDI)

UDDI specification provides a platform-independent way of describing services, discovering businesses, and integrating business services. In short, it provides a registry of businesses and web services. A client is able to find web services through a UDDI interface. UDDI describes businesses by their physical attributes such as name, address, and the services that they provide. Abstractly, a business can register three types of information into a UDDI registry:

- White pages present the basic contact information and identifiers about a business entity, including business name, address, contact information, and known identifiers such as tax IDs. This information allows others to discover the service based on its business identification.
- Yellow pages present information that describes a web service using different categorizations, including industrial categorizations based on standard taxonomies.
- Green pages present technical information about services that are exposed by the business. Green pages include references or pointers to specifications for Web Services.

In summary, a UDDI registry has two kinds of clients: businesses that want to publish a service and clients who want to obtain services. Two APIs are described by this UDDI specification: the inquiry API and the publishing API. The *inquiry API* locates information about a business, the business offers, the specifications of services, and information about what to do when failure occurs. The inquiry API does not require authenticated access and is subsequently accessed using HTTP. The

publishing API is used to create, store, or update information in a UDDI registry. Every function in this API requires authenticated access to a UDDI registry.

UDDI data structures, as shown in figure 1 (UDDI 2002), are passed as input and output parameters for major API messages.

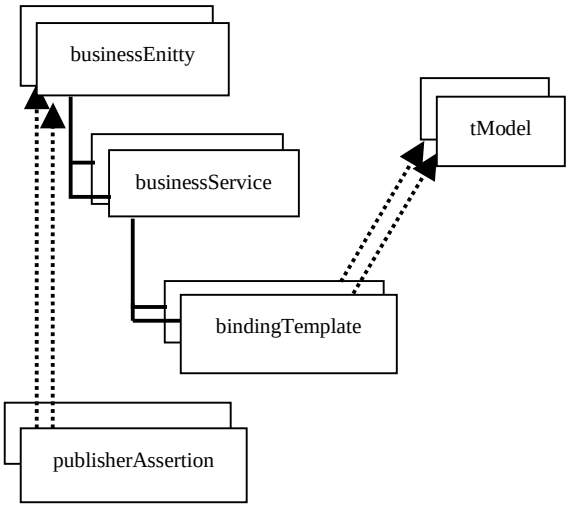

Figure 1.Five core data structure types in UDDI

- *businessEntity* or business information: Information about a business that provides support for "yellow pages" so that searches can be performed to locate businesses.

- *businessService* or service information: A collection of published Web services. It is used to describe a set of services provided by the business. Each businessService structure is the logical child of a single businessEntity. businessService contains specific instances of bindingTemplate.

- *bindingTemplate* or binding information: Technical information about a service entry point and construction specification, but not the details of the service's specifications. A bindingTemplate contains an optional text description of the web service, the URL of its access point, and a reference to one or more tModel structures. Each bindingTemplate structure has a single logical businessService parent.

- *tModels* or service type definition. tModels present technical specification for service type, such as interface definitions, wire or message protocols, interchange or message formats and sequencing rules. tModels are a core data structure in the UDDI specification and represent the most detailed information that a UDDI registry can provide about any specification.

- *PublisherAssertion* : Information about a relationship between two business parties, asserted by one of both. The PublisherAssertion exposes a set of APIs for inquiry and publication. The *inquiry APIs* search and browse information in a UDDI registry while the *publication APIs* are for publishers to put their information.

The two UDDI data structures that are particularly relevant to the use of WSDL in the context of a UDDI registry are the *tModel* and the *businessService.* Services are represented in UDDI by the *businessService* data structure while the details of how and where the service is accessed are provided by one or more nested *bindingTemplate* structures. In other words, to invoke a web service is typically performed based on cached bindingTemplate structures. The steps of how to invoke a UDDI register are described as follows:

- First, the programmer, granted to write a program that uses a remote web service, uses the UDDI business registry to locate the businessEntity information, either via a web interface or other tools that use the Inquiry API.
- Second, the programmer either requests a full businessEntity structure or looks into more detail about a businessService. The businessEntity structures contain all the information about web services from which the programmer is able to select a particular bindingTemplate and save it for later use.
- Third, the programmer prepares a program based on the knowledge of the requirement for the web service. This information may be obtained via the tModel key contained in the binding-Template structure for a service.
- Finally, at run time, the program invokes the web service as prepared using the cached bindingTemplate information.

3.2 Web Services Definition Language (WSDL)

WSDL provides a way for service providers to describe the basic format of web service requests over different protocols. The WSDL specification defines seven elements of network services:

- *Types* provides data type definitions used to describe the messages exchanged.

- *Message* represents an abstract definition of the data being transmitted or communicated.

- *Operation* is an abstract description of an action supported by the service. Each operation refers to an input message and output message.

- *Port Type* is a set of operations supported by one or more endpoints.

- *Binding* specifies concrete protocol and data format specification for the operations and messages defined by a particular port type.

- *Port* specifies an address for a binding or the URL where the web service is listening.

- *Service* is a collection of related ports.

Typically, information common to a certain category of business services such as message formats, port types, and protocol bindings, are included in the reusable portion, while information pertaining to a particular service endpoint is included in the service implementation definition portion. Within the context of UDDI, only the reusable portion of the WSDL service

information is involved. The details of registering and referencing WSDL in a UDDI registry are described in (WSDL-UDDI, 2002).

4 Web Service Composition Design

We use the example of a business trip that requires a combination of airline, hotel and car rental services for designing the composition service. For instance, a client wants to travel from Australia to attend a conference in Beijing and stay in a hotel near the conference site. The budget for the trip is $A 2000. He/she needs to find a cheap air ticket and look for reasonable accommodation. By searching from the Internet, he/she obtains the lists of airfares to, and hotels in, Beijing. Then he/she has to find out which combinations of airline and hotel charges total less than the budget of $A 2000. It will not be difficult if he/she can pay $A1500 for the airline and $A 500 for the hotel, or pay $A 1200 for the airline and $A 800 for the hotel. Complex collaboration of these two services occurs when these two services have to be negotiated simultaneously for mutual agreement. Moreover, each service has its own constraints. For instance, the hotel that offers the best price may not accommodate foreigners. The lowest price airline may not have a seat available on the required date. These problems need the intelligent and efficient algorithms of service composition.

In the rest of this section we describe the relationship between web services, and present the idea of applying CBR for service discovery.

4.1 Web Services Relationship

Let service Si and service Sj be sub-services of the composite service S while service Si provides a different type of service from service Sj. The relationship R between sub-services Si and Sj can be identified as follows:

R1: Independent Relationship: (+ denotes this relationship.)

$$Si + Sj = Sj + Si$$

Each sub-service is freely independent of the other. The order of execution of these two sub-services does not affect the composition service, which means that the result is the same in either case.

R2: Prerequisite Relationship (→)

$$Si \rightarrow Sj$$

The prerequisite relationship means that one service has to finish before the other starts. Service Si has to finish before service Sj starts. For example, the booking service has to be done before the payment service.

R3: Parallel-Prerequisite Relationship (⇒)

$$Si \Rightarrow Sj$$

Here service Si processes or executes at the same time as service Sj but service Sj has to wait for the result from service Si before completing its process. This relationship differs from R2 in respect of the time which the service processes must start.

R4: Parallel-Dependency Relationship (⇔)

$$Si \Leftrightarrow Sj$$

Here service Si and service Sj process or execute in parallel (simultaneously) but the results of each service need to be compromised with the other. This kind of relationship needs negotiation and deadlock-free mechanisms.

R5: Substitute Relationship (//)

$$Si \ // \ Sj$$

Service Si can be substituted by service Sj. The service Si and Sj seem to provide the same service but they have some different attributes. For example, the delivery service, Si can be an air courier service delivery while Sj is a truck service delivery. The attributes that make these two services different and can be substituted are cost, time, place, method and so on.

R6: Overlapping Relationship (⊗)

$$Si \otimes Sj$$

The example of the business trip includes airline and hotel services. Hotel service may include car-rental service from the airport to the hotel (cost $). The airline may provide free car service pick up from the airport to the hotel (cost 0). In this case the car rental service is repeated. To compose this overlapping relationship, the overlapping part from the service that costs more needs to be excluded.

4.2 CBR for Service Discovery

4.2.1 Case-Based Reasoning Overview

CBR is a problem solving approach, one of the AI machine learning techniques (Aamodt and Plaze 1994). The basic idea of CBR is to solve new problems by comparing them to old problems, which have already been solved in the past (Leake 1996). The existing problems and their solutions are stored in a database of cases, called a case-base. When a new problem is presented, the CBR system tries to retrieve the most similar existing problems from the case-base. The idea is that, if two problems look similar, then the solutions to these problems are also possibly similar. The concept of case similarity measure plays an important role in performing these processes.

The similarity measure is a function that evaluates the similarity between a given query and cases in the case base. It measures the differences of each attribute or dimension between a query and existing case. The details of the similarity measure rules and processes are presented in our previous work (Limthanmaphon, Zhang and Zhang 2002).

4.2.2 Service Case-Base

The service case-base stores the collection of service cases. A service case is a pre-assembly composite service. The component of a service case consists of service name and service description.

S: (N, D)

S : refers to a composite service.

N : is the service name and

D : is the service description. Service description consists of a set of sub-services, relationships and constraints.

Service description can be defined as:

D: {(Si, Sj, R1, Cl_1), (Si, Sk, R2, Cl_2),..,(Sm, Sn, Rn, Cl_n)}

Si, Sj, Sk, … Sm, Sn: refer to sub-services of the composite service S.

R1, R2, …, Rn: refer to the relationships between pairs of sub-services.

Cl_i: refers to the list of constraints for the sub-services. Constraints are the services' input parameters.

For example, the business trip service consists of the airline and hotel services. The relationship between these services can vary depending on the constraints from the query.

Ex1. The business trip consists of Airline and hotel services. They are independent and there is no other requirement.

S: (Business Trip, {Airline, Hotel, +, 0})

Ex2. The business trip consists of two services. Each service starts at the same time but the hotel service has to wait until the air ticket is available or confirmed by the airline service.

S: (Business Trip, {Airline, Hotel,⇒, air ticket})

Ex3. The business trip consists of two services. The hotel service will start when the airline service finishes execution. There is no constraint between the services.

S: (Business Trip, {Airline, Hotel, →, 0})

Ex4. The hotel service provides the car-rental service if the client stays up to three days.

S: (Hotel, {Car-rental, , ,3 days free})

Ex5. The hotel service provides breakfast service.

S: (Hotel, {Breakfast, , , })

Ex6. The breakfast service includes the choice of an Oriental Breakfast.

S: (Breakfast, { , , , Oriental})

The service relationship constructed from these service cases as shown in figure 2.

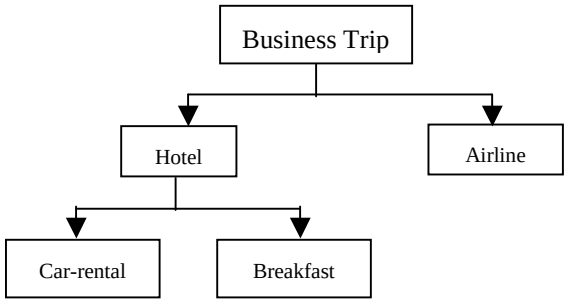

Figure 2. Business trip composition service Structure

4.2.3 Service Case Retrieval

Retrieval is the process of finding and getting an appropriate stored case, which is close to the requirements. The goal of retrieval is to retrieve cases, which have the potential to be most useful. Case retrieval requires a combination of searching and matching. Similarity measurement is used as a tool to find the closest case which matches the query. We summarise the processes of retrieval in the following algorithm:

(1) Initial problem. Query service qS is instantiated as a new problem. For example, a business trip is a query service.

(2) Look up the service case base to find cases, which have a service name, the same as the query service qS. For example the Ex1, Ex2, and Ex3 in 4.2.2 are the matched cases.

(3) Consider and extract the query service qS into the query sub-services qSS.

(4) Compare the query sub-service qSS with the service description of each case in the step (2). From this example, airline and hotel are the sub services from cases Ex1, Ex2 and Ex3.

(5) Consider and determine the constraints of each query sub-service qSS. The constraints of each query sub-service can be defined as qSSiC.

(6) Compare the constraints of each query sub-service qSSiC with the constraint list Cl from each case in step (4). For example, if air ticket is a constraint of the sub-service airline, then case Ex2 is retrieved and the parallel-prerequisite relationship (from the example) is suggested as the relationship solution for the composition service.

Note that the query service extraction is recursive (repeat from step (2)) until the service does not have any sub-services.

In summary, composite service is constructed depending on the relationship and constraints. The request analyst (explained in section 5.) will compare the client's query with the tree structure and measure the similarity between the query and service cases. The highest similarity measure value is the most satisfactory service. The model of the service composition is explained in section 5.

5 Web Services Composition Model

There are three main components in our model: request analyst, outsource agents and services composer. The composition process executes in two modes: proactive and reactive as shown in figure 3.

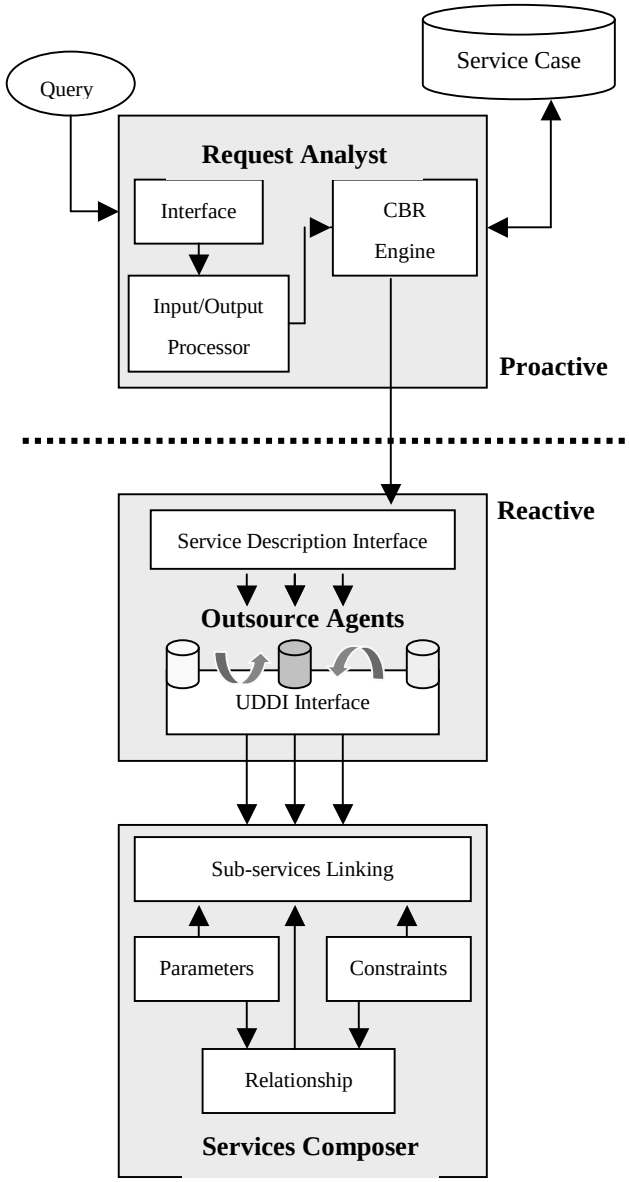

Figure 3. Web Services Composition Model

Request Analyst: determines the client's query through its interface. The interface passes the query to the Input / Output processor. The Input / Output processor extracts the query into parameters and constraints and passes them to the CBR Engine. CBR Engine then retrieves the most similar pre-assembly service case from the service case-base. The result of this state will be a number of sub-service components. These sub-service descriptions will be sent to the Outsource Agents.

Outsource Agents: will use the service descriptions that have been passed from the CBR Engine through the service description interface. Outsource Agents then contact the UDDI service directory to discover the required services.

Services Composer: will compose all the sub-services according to the parameters, constraints and the relationship.

5.1 Proactive Composition

The pre-assembly composition services are stored as service cases in the service case-base. When a client requests a service, the request analyst will extract the query through its input/output processor and pass the query to the CBR Engine. The CBR Engine then retrieves the pre-assembly composition service from the service case-base. For instance, the composite service of the business trip as shown in figure 2 will be provided for the business trip query. The result of the list of sub-services can have many forms of composition as shown in Ex1, Ex2 and Ex3 in section 4.2.2.

The Interface Unit: accepts the query service (qS) from the client and then extracts the query into a number of query sub-services (qSS) and parameters (C). For example from section 4, the query is the conference trip. Query sub-services are: 1. Travel from Australia to Beijing, China, and 2. Hotel service locates in Beijing. Parameters are budget ($A2000) and conference dates.

The Input/Output Processor: analyses the token queries (input parameters (C) and output services (qS or qSS)) and then passes them to the CBR Engine. The following table shows an example of the analysis results.

Input	Output
Travel & Hotel Services	Conference Trip
Place (Australia - Beijing, China), Date, Budget (cost)	Travel Service
Place (Beijing), Date, Budget (cost)	Hotel Service

The query service at this stage can be formed as

qS : (N, {qSS1, qSS2,… qSSn})

$qSS_1 \in \{ (sN_1, \{S_{11}, S_{21}, R_{1(12)}, C_1\}), \ldots (sN_1, \{S_{1j}, S_{2k}, R_{2(j\,k)}, C_2\}),$
$(sN_1, \{S_{11}, S_{21}, R_{1(12)}, C_1\}), \ldots (sN_1, \{S_{1j}, S_{2k}, R_{m(j\,k)}, C_m\}),\}$

$qSS_2 \in \{ (sN_2, \{S_{11}, S_{21}, R_{1(12)}, C'_1\}), \ldots (sN_1, \{S_{1j}, S_{2k}, R_{2(j\,k)}, C'_2\}),$
$(sN_2, \{S_{11}, S_{21}, R_{1(12)}, C'_1\}), \ldots (sN_1, \{S_{1j}, S_{2k}, R_{m(j\,k)}, C'_m\}),\}$

:

$qSS_n \in \{ (sN_n, \{S_{11}, S_{21}, R_{1(12)}, C'^{n}_1\}), \ldots (sN_1, \{S_{1j}, S_{2k}, R_{2(j\,k)}, C'^{n}_2\}),$
$(sN_n, \{S_{11}, S_{21}, R_{1(12)}, C'^{n}_1\}), \ldots (sN_1, \{S_{1j}, S_{2k}, R_{m(j\,k)}, C'^{n}_m\}),\}$

which mean the query service has a service name and many sub-services. Each sub-service has its service name and several service descriptions. A service description describes the relationship of sub-services of its service and the parameter or constraints. The service extraction process will repeat until the service has no more sub-services. Ex6 in section 4.2.2 is an example of the service, which cannot be extracted any further.

The CBR Engine gets the information from the input/out put processor then tries to match and retrieve the most similar services from the service case-base. The retrieval process has been described in section 4.2.3. The CBR engine then passes the result of the list of sub-services, their parameters or constraints, and their relationship to the outsource agents, which is the next phase of service composition.

5.2 Reactive Composition

This phase of service composition deals with the discovery and integration of the sub-services of a composite request. When the CBR Engine passes the list of sub-services, parameters and their relationships, the service description interface will contact the UDDI Interface to allocate those services, which have similar service specifications to the required sub-services.

Outsource agents provide the mechanism for comparing the required service description and the *businessService* from the UDDI registry. For instance, if the query requires the car-rental service in the hotel service, the outsource agents need to allocate a hotel service which has a relationship with car-rental service. The relationship between car-rental and hotel service can take any of the forms explained in section 4.1. The outsource agents will allocate all the satisfied services and pass these services to the service composer.

5.2.1 UDDI Invocation and Discovery

UDDI or service directory is used for the process of service discovery. Since UDDI service registries are described in XML-based format, we use JAXR (Java API for XML Registries) to parse the XML-based service descriptions, find out a service matching the query by comparing the XML attributes of each service and hence enable a wider range of service selection. We also employ CBR methodologies for matching and comparing to find the most suitable service requirement. We summarise the processes of UDDI invocation and comparing in the following algorithm:

(1) Each service name and its parameters, which are provided by the CBR Engine from the previous phase, are the search key words or queries. According to the UDDI specification, these queries can match to three core data structures: businessEntity, businessService, and service type of tModel.

(2) The UDDI interface module is created for looking up the UDDI registry. This interface module will contact the publisherAssertion structure to locate the appropriate business partner that is advertising the web service.

(3) When a set of required business entities is provided from step (2), the full businessEntity structure of each business entity is retrieved.

(4) Since a businessEntity consists of business-Services (as shown in figure 1), each business-Service needs to be extracted for comparison purpose.

(5) Each businessService consists of business key, service key, description and bindingTemplates, which can be used to compare with the queries as defined in step (1). The similarity measure rules (Limthanmaphon, Zhang and Zhang 2002) will be applied to find the closest required service.

Every sub-service of the composite service will repeat these steps of service discovery and matching. The result of a set of the most suitable services will be passed to the service composer.

5.2.2 Relational Service Composition

The service composer then integrates all the allocated services by considering the relationship and constraints of each service. 'Parameters' refer to the input or output parameters of each service while 'constraints' refer to the mandatory variables (mv_i). We use figure 4 to describe the relationship of services composition.

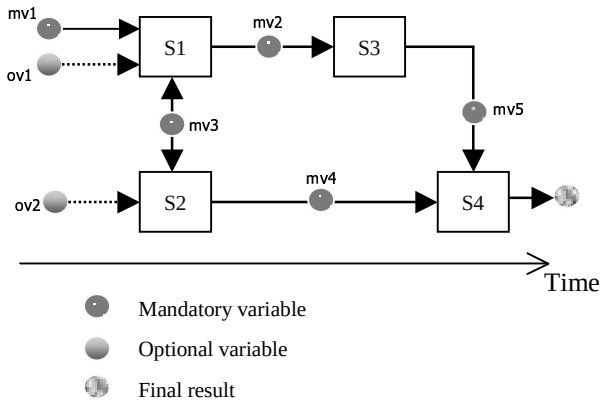

Figure 4. Services Composition Relationship

Note that the optional variable (ov_i) is disregarded to form the service composition relationship. The relationship descriptions between each pair of services can be defined as:

R1 = {(S1, S2), ⇔, mv3, R0}: relationship R1 is a parallel-dependency relationship between services S1 and S2 while mv3 is a constraint. There is no other relationship (R0) before R1.

R2 = {(S1, S3), →, mv2, R1}: relationship R2 is a pre-requisite relationship between services S1 and S3 while mv2 is a constraint. The relationship R1 invokes before relationship R2.

R3 = {(S2, S4), →, mv4, R1}: relationship R3 is a pre-requisite relationship between services S2 and S4 while mv4 is a constraint. The relationship R1 invokes before relationship R3.

R4 = {(S3, S4), →, mv5, R2}: relationship R4 is a pre-requisite relationship between services S3 and S4 while mv5 is a constraint. The relationship R2 invokes before relationship R4.

The sub-service linking role will link all the services according to the services' relationship descriptions, which finally provide a composition service as a result.

6 Conclusion

We present a model of web services composition using the Case-Based Reasoning (CBR) technique. CBR is applied in the process of service discovery, which is the crucial composition process. Our service composition model integrates the two behaviours of proactive and reactive service compositions. The proactive composition phase provides the pre-assembly of the composition service, which includes the sub-services, parameters and relationships among sub-services, while the reactive composition phase deal with the processes of service discovery and integration of the existing sub-services. There are three main components in our model: request analyst, outsource agent, and services composers. We give a brief overview of service invocation and discovery. We also present the various forms of relationships between sub-services.

There are three advantages in our model: first, proactive composition can reduce the cost of composition. Second, we design for collaboration among the services to satisfy the client. Dynamic composition is flexible. Third, the similarity feature of CBR is used for efficient service discovery, which will have the resulting benefit of efficient service composition. On-going work includes the prototyping to cater for more general applications other than travelling, and to illustrate the viability of the proposed model.

7 References

AAMODT, A., and PLAZE, E. (1994): *Case-Based Reasoning: Foundational Issue, Methodological Variations, and System Approaches*. In AI Communications, Vol.7, IOS Press.

BENATALLAH, B., DUMAS, M., FAUVET M.C. and PAIK H.Y. (2001): *Self-Coordinate and Self-Traced Composite Services with Dynamic Provider Selection*. Technical report, UNSW-CSE-TR-0108, School of Computer Science & Engineering, University of New South Wales, Australia.

BENATALLAH, B., DUMAS, M., FAUVET M.C. and RABHI F.A. (2001): *Towards Patterns of Web Services Composition*. Technical report, UNSW-CSE-TR-0111, School of Computer Science & Engineering, University of New South Wales, Australia.

CASATI, F., ILNICKI, S., JIN, L., KRISHNAMOORTHY, V. and SHAN, M. (2000): *Adaptive and dynamic service composition in eflow*. Technical Report, HPL-200039, Software Technology Laboratory, Palo Alto, USA.

CHAKRABORTY, D. and JOSHI, A. (2001): *Dynamic Service Composition: State-of-the-Art and Research Directions*, Technical Report TR-CS-01-19, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, USA.

CHAKRABORTY, D., PERICH, F., JOSHI, A., FININ, T. and YESHA, Y. (2002): *A Reactive Service Composition Architecture for Pervasive Computing Environments*. In 7th Personal Wireless Communications Conference (PWC 2002), Singapore.

LEAKE, D. (1996) *CBR in Context: The Present and Future. In Case-Based Reasoning, Experiences, Lessons, & Future Directions*. Ed. D. Leake, Menlo Park, MIT Press.

LIMTHANMAPHON, B., ZHANG Z. and ZHANG Y. (2002): *Adaptive Case-Based Reasoning Systems for E-Commerce*. International Conference Intelligent Information Technology ICIIT-02, Beijing, China

MACHIRAJU, V., DEKHIL, M., GRISS, M. and WURSTER, K. (2000): *E-services Management Requirements*. HP Laboratories Palo Alto, HPL-2000-60.

MENNIE, D. and PAGUREK, B. (2000): *An Architecture to Support Dynamic Composition of Service Components*, Systems and Computing Engineering, Carleton University, Canada.

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

UC Berkeley Computer Science Division. http://ninja.cs.berkey.edu.

UDDI, Universal Description, Discovery and Integration, http://www.uddi.org

UDDI (2002): UDDI Version 2.03 *Data Structure Reference, UDDI Published Specification*, 19 July 2002. http://www.uddi.org/pubs/DataStructure-V2.03-Published-200207.pdf

WSDL, Web Services Description Language (WSDL) 1.1, March 15, 2000. http://www.w3.org/TR/wsdl.

WSDL-UDDI (2002): *Using WSDL in a UDDI Registry, Version 1.07 UDDI Best Practice*, May 21, 2002, http://www.uddi.org/pubs/wsdlbestpractices.pdf.