Process-context aware matchmaking for web service composition

Weili Han*, Xingdong Shi, Ronghua Chen

Software School, Fudan University, Shanghai, PR China

Received 12 July 2007; received in revised form 21 November 2007; accepted 26 November 2007

Abstract

Web service composition can help software developer design more powerful and flexible applications according to requirements of enterprise. But during compositing, how to discover suitable web services is a critical problem in design and implementing application-oriented web service technologies. The traditional keyword-based matchmaking approach is difficult to help developer to find suitable service. Current researches find that to attaching semantics to each registered service can help improve the precision of matchmaking. The improvement can help developer find more suitable service for business process. This paper proposes a novel approach of semantics-based matchmaking, which is named process-context aware matchmaking. The process-context aware matchmaking discovers the suitable service during web service composite modeling. During matchmaking, the approach utilizes not only semantics of technical process but also that of business process of a registered service, thus further improving the precision of matchmaking. We integrate the process-context aware matchmaking with business-process-driven web service composition in an integrated development environment based on Eclipse. The performance evaluation shows that performance overhead of this novel approach is acceptable.

Keywords: Web service matchmaking; Process-context aware; Web service composition

*Corresponding author. Tel.: +86 21 51355374 103; fax: +86 21 51355388.
E-mail address: wlhan@fudan.edu.cn (W. Han).
1. Introduction

Web service composition (Shore, 2006; Kuster et al., 2007; Harney and Doshi, 2007; Kazhamiakin et al., 2006; Limthanmaphon and Zhang, 2003; Solanki et al., 2004; Srivastava and Koehler, 2003) will help the integration of business operation and the underlying IT system, thus facilitating business process optimization and innovation in enterprises (Shi et al., 2007). Web service composition provides a novel approach for software development: on the basis of open specifications, a new application system can be built by combining existing web services. However, web service matchmaking is still a critical problem in implementing the composite process, whether the web services are bound in runtime or design-time.

Matchmaking is the process of searching the space of possible matches between generic request and offer descriptions to find most promising ones (Dumas et al., 2004; Kaufer and Klusch, 2006; Kuster et al., 2007; Pastore, 2007; Sioutas et al., 2007). During web service composition, application system need find suitable services which could be installed at different servers. Service matchmaking can help developers and operators of application system to find some suitable services. For example (Kuster et al., 2007), an automotive manufacturer with several locally distributed factories can apply service matchmaking to find suitable suppliers of some parts and relevant delivery services.

The traditional keyword-based web service matchmaking is not accurate enough (Paolucci et al., 2002) due to the lack of sufficient information to help matcher to rank and find suitable services. Lots of unrelated services may be in the result list, while some qualified ones are missing. We propose a novel approach of web service matchmaking, which is process-context aware. Process context can be automatically extracted from a developing composition process, and used in finding the exactly related services. We apply this approach to improve the matchmaking efficiency during service composition in our prototype—a development tool of integrated business-process-driven design for service oriented enterprise application. The performance evaluation shows that performance overhead of this novel approach is acceptable.

The rest of this paper is organized as follows: Section 2 describes the background and motivation; Section 3 introduces and defines business template and process context; Section 4 discusses the implementation issues; Section 5 is a performance test to compare the performance of the proposed approach and that of the traditional keyword-based approach; Section 6 shows an example about how the development tool automatically builds a process context to find a web service; Section 7 surveys related works; Section 8 summarizes this paper and discusses further work.

2. Background and motivation

2.1. Business-process-driven web service composition

Web services provide the basis for the development and execution of business processes that are distributed over the network and are available via standard interfaces and protocols. Service composition is one of the most promising ideas underlying web services: new functions can be defined and implemented by combining and interacting with pre-existing services (Kazhamiakin et al., 2006). Business Process Execution Language (BPEL)
for web service (Andrews et al., 2003) is one of the most important technologies to implement web service composition.

To develop an application on the basis of web service composition, the developer usually requires the help of business expert to draw the demand of an application system. As is shown in Fig. 1 (Shi et al., 2005, 2007), separate the development process into two phases: business environment modeling and business process modeling. During business environment modeling, business experts may draw an abstraction application system (Business Environment Model, BEM) on the basis of their business knowledge. Shi et al. (2005, 2007) apply business template and visual development tool (shown in Fig. 2) to facilitate the development for business expert. In business process modeling, IT experts may concretely design the application system (Business Process Model, BPM) based BPEL according to business environment model.

2.2. Web service discovery and matchmaking

In business process modeling, developer need specify some services for activity in BPEL. These services can be discovered from a Universal Description, Discovery and Integration (UDDI) server. Furthermore, business environment where web services are running is seldom static. Dynamic service binding is an important aspect of service oriented computing (Kuster et al., 2007; Harney and Doshi, 2007). Traditional UDDI server provides a keyword-based service discovery, which is not enough to deal with complex service matchmaking request. Kuster et al. (2007) introduces an integrated approach to
automatic server discovery, matchmaking and composition. In Kuster’s approach, a service matchmaking request includes large mount of semantics (properties) information to help discovery service to find and rank suitable services.

2.3. Motivation

This paper introduces a novel approach to match suitable services according to process context. The contributions presented in the paper are as follows:

(1) Process context can be automatically created according to process model which is the business process model proposed in Shi et al. (2007). This method is different from single keyword-based service discovery, because the process context is more complex than the format of keyword, and can help the developer find more suitable service during service compositing. We will concretely define the process context in Section 3.2.

(2) An ExtUDDI is proposed to support process context aware matchmaking. In the traditional UDDI (UDDI, 2006) specification, UDDI server does not support context based service matchmaking. This paper presents the approach which adds an additional interface to UDDI. jUDDI (jUDDI, 2007) is an open source implementation of UDDI specification, which is developed by apache organization. We add an additional interface find_service_ext to jUDDI, and call the modified jUDDI ExtUDDI (Extented UDDI). Fig. 3 demonstrates the interaction between an ExtUDDI server and its client.

3. Definitions of business template and process context

IT system can improve the efficiency of business in an enterprise. IT system is usually designed by IT professionals and operated by businessmen and IT operators. During IT system designing, business expert usually participate in improving the system design so
that the system meets more of the requirements of enterprise. So the IT system specification should be understood by both business experts and IT professionals.

3.1. Business template for service composition

The concept of business template is introduced to help both business experts and IT professionals to modeling a composite process. A business template describes abstract business process in specific domain, the roles each partner plays, dependencies among these roles (these dependencies can be called rules), as well as other information.

So, a business template can be defined as:

**Definition 1.** Business Template = \{Business Domain, Role Set, Rule Set\}

Fig. 4 shows the XML Schema of business template.

A business template defines one domain, contains one or more roles and zero or more rules.

Each role has one or more properties. Each property can be defined as a triple \langle name, type, description \rangle. The “name” and “description” are used to identify and describe the property. “type” indicates whether this property is an input parameter or output result.

A rule indicates that some property of a role is dependent on some property of another role. Different types of rules can be defined according to different types of dependencies. Currently only one type of rule is defined, namely the “setValue” type. A rule of “setValue” type indicates that the value of the specified property can be one of a series of options. Fig. 5 shows an example of a business template.

![Fig. 4. Structure of a business template.](image-url)
In the example shown in Fig. 5, the business template defines a domain Supply Chain Management (SCM), six roles (Headquarters, Store, Vendor, Bank, Logistics, Warehouse) as well as three rules. The roles and rules will be used in extracting process context in a business process model. Since the example is a business template example for a retail enterprise, the descriptions of the six roles are: Headquarters: headquarters of the retail enterprise, which is in charge of all enterprise-level activities; Store: store of the retail enterprise, e.g., there are six walmart stores in Washington; Vendor: who manufactures product for the retail enterprise, e.g., the Coca-Cola company; Bank: the bank provides financial service for the retail enterprise, e.g., Citibank; Logistics: who provide third-party transportation service for the retail enterprise and the vendor; Warehouse: a large enterprise may have several warehouses, or sometimes called distribution centers, across the country. Usually a store receives goods from the nearest warehouse.

3.2. Process context

In our approach, process context is a core concept. Process context can help discovery service to find more suitable service for composition. Process context can be automatically created by business process when a role need assign a web service.

First, we formally define process context as follows:

**Definition 2.** Process Context = (Criteria, Weight Vector)

Criteria are used to filter out unqualified web services. Criteria are determined by property dependencies on business level, e.g., if a dependency is defined as “the vendor
must provide what the customer want”, and it is known that the customer requires TV set, then the criteria for finding a vendor’s service may be “a vendor who provides TV set”. Criteria can help improve the precision and narrow down the result set when finding services in a service repository.

We formally define Criteria as follows:

**Definition 3.** Criteria = \{property dependencies\}

Weight vector: Weight vector is used to rank all the qualified services so that the user can select a best one among them. Weight vector contains the merits and their weights used in ranking. When finding services for different roles, the weight vectors are also different; when finding services for a same role but in different stage of the process, the weight vectors may also be different from each other.

A weight vector consists of three components: information set, priority set, and Quality of Service (QoS) set.

**Definition 4.** Weight Vector = (Information Set, Priority Set, QoS Set)

(a) Information set

Each web service in a UDDI repository can be attached with semantic information about the input and output of it. This semantic information is related to business domain.

When finding a web service for a role in the process, the information set can be further divided into two subsets: available information set and required information set. Available information set is the set of all the input and output of the role’s preceding services. For example, in the scenario of Section 6, the services from warehouse and vendor will be requested before the service from logistics, so all the input and output of the services from warehouse and vendor together form the available information set when finding the service for logistics. Required information set is the set of all the input of the role’s succeeding services. A service should be given a higher rank if its requirement can be met or it provides what is needed by its successor.

Available information set can be formally defined as:

**Definition 5.1.** AvailableInformationSet = \( \Sigma \) input and output of the role’s preceding services

A service’s score on available information set can be formally defined as:

**Definition 5.2.** Score(AvailableInformationSet) = \(|\{p|p \in \text{AvailableInformationSet} \text{ AND } p \in \text{ParamSet}\}|/|\text{ParamSet}| \) (ParamSet is the set of all the input parameters of all the interfaces in the web service)

Take the scenario in Section 6 as an example, when finding a service for logistics, the available information set is \{product_name, product_quantity, product_remains, product_total_price\}. If a service has five parameters in all, with two of them in the available information set, and the other three not in the set, its score on available information set will be \( 2/5 = 0.4 \).

Required information set can be formally defined as:

**Definition 6.1.** RequiredInformationSet = \( \Sigma \) input of the role’s succeeding services
A service’s score on required information set can be formally defined as:

**Definition 6.2.** Score(RequiredInformationSet) = |{p | p ∈ RequiredInformationSet AND p ∈ ResultSet}| / |ResultSet| (ResultSet is the set of all the output results of all the interfaces in the web service)

Take the scenario in Section 6 as an example, when finding a service for logistics, the required information set is \{from_account, to_account, transfer_amount\}. If a service has four output results in all, with one of them in the required information set, and the other three not, its score on required information set will be 1/4 = 0.25.

(b) **Priority set**

Priority set is a set of domains under which services should be given a higher rank. It can further be divided into two subsets: static priority set and dynamic priority set.

*Static priority set:* in one business process, the static priority sets for finding services for different roles are the same. For example, the static priority set can be the domains of all contractors. Obviously, services from contractors should be given higher ranks.

**Definition 7.** Score(StaticPrioritySet) = 1 if the domain of a service is contained in the static priority set, or 0 if not.

*Dynamic priority set:* in one business process, the dynamic priority sets for finding services for different roles may be different. For example, the dynamic priority set can be the domains of all the already found services for other roles.

**Definition 8.** Score(DynamicPrioritySet) = 1 if the domain of a service is contained in the dynamic priority set, or 0 if not.

(c) **QoS set**

QoS is an important aspect when ranking services. Common merits of QoS are latency, cost, reliability, etc. For example, a QoS set is \{Duration: 30%, Cost: 30%, Reliability: 40%\}, if a service’s quality is \{Duration: 0.8, Cost: 0.5, Reliability: 0.9\}, its score on QoS will be 0.8 \times 30 + 0.5 \times 30% + 0.9 \times 40\% = 0.75. For each service, the score on QoS set can be determined by a third-party evaluation organization. Finally, the ranking is based on the score of each qualified web service. The score for one service is calculated as:

**Definition 9.** Score(Service) = \sum \text{score(Merit i)} \times \text{weight(Merit i)}, where Merit i is one element of the set \{AvailableInformationSet, RequiredInformationSet, StaticPrioritySet, DynamicPrioritySet, QosSet\}

The internal process of ExtUDDI’s find_service_ext interface can be demonstrated as Fig. 6. When ExtUDDI server receives the process context in a service-finding request, it first extracts the criteria from the process context and uses the criteria to find out all qualified services from the services repository, and then extracts the weight vector from the process context and uses the weight vector to rank all the qualified services.
4. Implementation issues

4.1. ExtUDDI server

To implement process-context-based service matchmaking, the UDDI server has to be modified to store the services’ semantic information and give the additional find_service_ext interface for client.

In UDDI specification, the interface find_service is used for keyword-based service finding. jUDDI is an open source implementation of UDDI specification. On the basis of jUDDI, we add an interface find_service_ext to it.

Fig. 7 shows the structure of find_service_ext request’s SOAP message.

The node find_service_ext has three attributes: “plug-in” specifies the server-side plug-in which is used for processing the request, “generic” indicates the version number of UDDI specification, and “xmlns” tells the namespace used in the SOAP message.

There are two sub-nodes under the node find_service_ext: “general_param” contains domain independent parameters such as authentication information, “plugin_specific_param” contains domain dependent parameters which can only be processed by a domain-related plug-in. However, all plug-ins are assumed to support the common parameters shown in Fig. 7: the two attributes “domain” and “role” indicate the business domain and role that a qualified service should play, a sub-node “criteria_set” contains criteria that must be met by a qualified service, and another sub-node “weight_vector” provides the merits and weights used to rank and sort all the qualified services.

Each criterion in the node “criteria_set” has three attributes: “property” tells which property of the role should meet this criterion, while “operation” and “value” together tell...
the requirement for that property. For example, the criteria \{property = “location”, operation = “equal”, value = “Washington”\} tells that a qualified service should stand for a role whose location must be Washington.

The other sub-node “weight_vector” is more complicated than “criteria_set”. The structure of the node “weight_vector” reflects exactly how we define a weight vector in Section 3. Each merit in the weight vector can be identified using a string with namespace, for example, “QoS:reliability” indicates a merit for reliability. Fig. 8 gives a detailed structure of the node “weight_vector”.

UDDI specification has already defined a structure “serviceInfo” to encapsulate the results of a find_service request. The structure of the result of find_service_ext request is not the same as that of find_service, because it returns extra information such as ranking of the service as well as semantics of its input parameters and output results. So a new structure serviceInfoExt must be defined as Fig. 9 to encapsulate the result of find_service_ext request.

4.2. ExtUDDI client API

To access the find_service_ext interface in ExtUDDI server, there should also be support from client-side API. UDDI4J (UDDI4J, 2007) is an open source client-side API for UDDI. The core of UDDI4J is a proxy class UDDIProxy, who provides all the client-side interface of the UDDI specification. We added an additional interface find_service_ext to UDDI4J. The new interface find_service_ext in UDDI4J marshals a process context into XML data, embeds process context into the SOAP message, send the SOAP message to ExtUDDI server, and un-marshals the XML data from ExtUDDI server into Java objects.
5. Performance evaluation

To compare the performance overheads of process-context-based service matchmaking and traditional keyword-based service matchmaking, it would be better to do the test on real services. There are several public service repositories, one of them is Woogle (http://haydn.cs.washington.edu:8080/won/wonServlet) hosted by Washington University. We used a crawler program to get the information of 592 web services in Woogle.

Forty-five of the 592 web services have the keywords “ZIP” in its service description. The 45 services provide “ZIP”-related services: query city name according to ZIP code,
Among the above 45 services, 19 of them provide city name query service. Only 10 provide query of American cities while the rest 9 are for Canada, France, Korea, Japan, Brazil, etc.

We attached semantic information to the 19 services manually, and then started the following test. The testing environment is listed in Table 1.

As we found the first time when a Java program is running, it costs much more time than the next time it runs. We think this is because in the first run, it costs a lot of computer resource to load the program from disk and do some initialization work in memory. So in each test case, we run the test program 1001 times, and record the average performance of the latter 1000 runs while the performance of the first run is discarded.

First, we tested the performance of traditional keyword-based service matchmaking.

**Testcase 1.** We use the find_service interface of UDDI4J to find all services that have the string “%ZIP%US%” or “%US%ZIP%” in its description. UDDI4J treat the two
strings as case-insensitive, and treat the character “%” as a wild-char that can match any number of any characters. The returned services must have the two words “ZIP” and “US” in its description. The test shows an average time of 84.3 ms for each query. Altogether 21 services are returned, but only 8 of them are qualified (providing city query service in US), the rest 2 qualified services are missing because their descriptions do not contain the keywords “US”.

Next, we test the performance of process-context based service matchmaking. We test the effect of each part in the process context by the following test cases.

**Testcase 2.1.** The weight vector contains nothing, i.e., the qualified services are not ranked and sorted before returning. The criteria we use are 
\[
\{ \text{domain} = \"Information\", \text{role} = \"Zipcode2City\", \text{property.country} = \"US\" \}\.
\]
The test shows an average time of 73.3 ms for each query, and the returned services are exactly the 10 qualified services because we have attached correct semantics to them.

**Testcase 2.2.** Based on Testcase 2.1, we added the following information set to weight vector:

\[
\begin{align*}
\text{AvailableInfoSet} &= \{ \text{Information:zipcode, Retail:price, Retail:quantity} \} \\
\text{RequiredInfoSet} &= \{ \text{Information:cityname, Retail:city, Retail:date} \}
\end{align*}
\]

The test shows an average time of 75.5 ms for each query.

**Testcase 2.3.** Based on Testcase 2.2, we continued to add the following priority set to weight vector:

\[
\begin{align*}
\text{StaticPrioritySet} &= \{ \text{www.baccar-inc.com} \} \\
\text{DynamicPrioritySet} &= \{ \text{www.webservicex.net} \}
\end{align*}
\]

The test shows an average time of 85.9 ms for each query.

**Testcase 2.4.** Based on Testcase 2.3, we added the following QoS set to weight vector:

\[
\text{QoSSet} = \{ \text{cost:0.5, reliability:0.5} \}
\]

The test shows an average time of 91.9 ms for each query.

**Fig. 10** summarizes the test.

Compared with the traditional keyword-based approach, the process-context based approach improves the precision with about 10% decrease in performance.

6. **Example of automatic process-context extraction**

**Fig. 11** shows a simplified logic of an SCM system. The store tells the headquarters that it requires some products; the headquarters then query the IT system of warehouse to check stocks, and ask the vendor to provide some amount of that product if the stocks are not enough. A logistics service may be used to transport products from the warehouse to the store, and finally a service from bank may be used for payment to the vendor and logistics.
In our previous research, we have built a development tool for modeling the business requirement. There are six parties involved in this example business: headquarters, store, warehouse, vendor, logistics, and bank. After setting the logistics’ properties of “source” and “destination” to Shanghai and Suzhou (two famous cities in China), respectively, the automatically extracted process context will contain the criteria set as \{WS.domain = SCM, WS.role = Logistics, ’Suzhou’ ∈ WS.source, ’Shanghai’ ∈ WS.destination\}.

In this example, the services from warehouse and vendor will be requested before the service from logistics, and the service from bank will be called after the service from logistics. The semantics for these services are described in Table 2.

When finding services for the role of logistics, the process of building the weight vector is: the static part of weight vector (static priority set, QoS set, weight of each sub-set) are loaded from a configuration file, while the dynamic part (dynamic priority set, information set) are calculated according to the position of the service in the process. For example, as is defined in Section 3, the available information set for logistics’ service will be the input parameters and output results of the services from warehouse and vendor, i.e., \{product_name, product_quantity, product_remains, product_total_price\}, while the
Table 2
Semantics for the services

<table>
<thead>
<tr>
<th>Role</th>
<th>Internet domain</th>
<th>Input parameters</th>
<th>Output results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warehouses</td>
<td>&lt;www.lhok.com.cn&gt;</td>
<td>product_name, product_quantity</td>
<td>product_remains</td>
</tr>
<tr>
<td>Vendor</td>
<td>&lt;www.coca-cola.com&gt;</td>
<td>product_name, product_quantity</td>
<td>product_total_price</td>
</tr>
<tr>
<td>Bank</td>
<td>&lt;www.icbc.com.cn&gt;</td>
<td>from_account, to_account, transfer_amount</td>
<td>transferred_amount</td>
</tr>
</tbody>
</table>

Fig. 12. Computer generated process context.
required information set will be the input parameters of the service from bank, i.e., 
\{from\_account, to\_account, transfer\_ammount\}.

So, the process context used to find the service for the role of logistics is that shown in Fig. 12.

7. Related work

In the field of web services matchmaking, the current commercial solutions are mainly 
based on keyword. UDDI is a standard in web services discovery. There is an interface 
find\_service defined in UDDI specification. This interface can accept a string of keywords 
and return web services whose names or descriptions contain these keywords (UDDI, 
2006). Although most UDDI implementations support wild-chars for find\_service 
interface, the precision cannot be guaranteed.

The researchers have done a lot of work to improve the precision of matchmaking, most 
of which are semantic-based, i.e., to attach semantics to each service and utilize these 
semantics when finding qualified services from a service repository. Kuster et al. (2007) 
proposed an integrated approach to automated service discovery, matchmaking and 
composition. During service matchmaking, Kuster considered single effect and multi-effect 
to rank the qualified services. The approach of Kuster is similar to ours. The difference 
between the Kuster’s and ours is the model of request of service matchmaking. 
Furthermore, the semantics of process context is automatically computed according to 
business process. Fan et al. (2006) proposed web services matchmaking approach on the 
basis of service semantics and QoS, Balke (Balke and Wagner, 2004) proposed an 
approach on the basis of service semantics and user preference (Guo et al., 2006) proposed 
an approach based on service semantics and relationship model (Lu, 2005) proposed the 
idea of sorting web services according to their semantic match degree. Maamar et al. (2004) 
proposed an approach to composite web service based on software agent and context. In 
Maamar’s approach, context is defined as any information relevant to the characterization 
of a situation. Their origination of the above ideas is the same: attach semantics to 
each input parameter and output result when a service is registered in a repository. A 
service discovery request also provides semantics about the service the requester wants. 
The service repository will consider a service as a qualified one if its semantics match 
the semantics in the request, and some formula may be used to calculate the match degree 
which will be used in sorting all the qualified services.

Elgedawy (2003) pointed out that a web service discovery process should be performed 
in two phases. The first phase filters services with different domains and different 
supporting roles. In the second phase, services’ technical contexts are then used to filter the 
obtained results.

Our solution absorbs their ideas, and improves it further by integrating service 
matchmaking with composite service modeling. In this way, we can attach not only the 
technical level but also the business level semantics to a service, thus helping developer and 
operator to find more suitable services.

8. Conclusion and future work

We integrated web services matchmaking with the process of composite service 
modeling, so process information can be used in the matchmaking to further improve the
precision. We present an ExtUDDI to support process-context aware service matchmaking. According to the performance test, this new approach has a performance down about only 10% compared with that of the traditional keyword-based service matchmaking, but the precision will be much higher with the help of pre-attached semantics.

In this paper, we use a simple process-context model to simplify the research of process-context aware service matchmaking. A more complex and standardized semantics technology, such as semantic web OWL-S 2007, can help us express the process context more powerfully in future.

Acknowledgment

This work is supported by IBM Shared University Research (SUR) Program—On Demand Business Integration with Intelligent Web Services and Process Composition.

References


jUDDI Homepage, \(\text{http://ws.apache.org/juddi}\); 19.03.07.


OWL-S Homepage, \(\text{http://www.ai.sri.com/daml/services/owl-s/1.2}\); 27.05.07.


