一种基于语义的服务组装框架①

李伟平 褚伟杰 高福亮 刘 利 童 缙 (北京大学 软件与微电子学院 北京 102600)

A Framework for Ontology-Based Service Discovery and Composition

Weiping Li, Weijie Chu, Fuliang Gao, Li Liu, Frank Tung

(School of Software and Microelectronics, Peking University, Beijing 102600)

Abstract: Currently a large number of web services as well as other kinds of services such as EJBs, COM, and even Java Classes are made available to the general public. Facilitating the SOA based system development by leveraging such kinds of services becomes a challenge. A framework for service repository, ontology based service discovery and service composition is put forward. The service repository can maintain the web services, EJBs, and Java Classes with the functions such as service registration, publishing, discovery, matching, versioning, and monitoring. The details of service description are analyzed. A domain ontology for Procurement, Selling, and Inventory is also given. Based on the domain ontology and the service repository, the semantic enhanced service composition algorithm is discussed.

Key words: SOA; svice repository; srvice composition; ontology; OWL-S

When building SOA based application systems how to manage the increasing number of both publicly available services and services only exposed internally within an organization becomes a challenge. The Universal Description, Discovery and Integration (UDDI) and the IBM WebSphere Service Registry and Repository (WSRR)are the two popular solutions. UDDI specifications define a registry service for Web services and for other electronic and non-electronic services. A UDDI registry service is a Web service that manages information about service providers, service implementations, and service metadata. Service providers can use UDDI to advertise the services they offer. Service consumers can use UDDI to discover services that suit their requirements and to obtain the service metadata needed to consume those services. Although the newly updated Version 3.0 claims to support the SOA [1], it suffers from the lower searching accuracy and lacks the support for other kinds of services such as EJBs, COMs, and Java Classes. The WSRR from IBM provides an alternative for managing services [2]. WSRR is the master metadata repository for service interaction endpoint descriptions. As the integration point for service metadata, it establishes a central point for finding and managing service metadata. Once service metadata is placed in Registry and Repository, visibility is controlled, versions are managed, proposed changes are analyzed and communicated, and usage is monitored.

Aiming to enhance the efficiency and accuracy of such kind of service registry and discovery, we propose a semantic enhanced service repository in this paper. Accordingly a framework for service repository,ontology

① Supported by the National High-Tech Research and Development Plan of China(863 Program) under Grant No.2007AA04Z150; the National Natural Science Foundation of China under Grant No.60704027

based service discovery and service composition is brought forward.

The remainder of the paper is organized as follows. Section 1 describes the basic idea as well as the framework for ontology based service composition. Section 2 presents a domain ontology model. Section 3 illustrates the rationale and the functions of the service repository. Section 4 introduces the semantic encapsulation method as well as a semantic enhanced service discovery algorithm. Section 5 concludes the paper with a brief description of future work.

1 The Framework for Ontology-Based Service Composition

To leverage both the external and internal deployed services requires a way of registry, discovery, and composition of them. A framework for service repository, ontology based service discovery and service composition is defined(see Fig.1) The service repository provides the functions such as service registration, publishing, discovery, matching, versioning, and monitoring. And both web services and other services such as EJBs, and Plain Old Java Objects (POJOs) are supported. The services, both external and internal to an organization, can be registered in this repository. The repository hereby acts as the portal of all these existing services. The semantic service enhancement encapsulate the service with Ontology Web Language for Services language (OWL-S), which add more detailed information of services with service profile, service model, and service grounding. With the semantic service enhancement, the service discovery can be expected to be more efficient compared with the situation in UDDI. The domain ontology defines the basic concepts for a specific vertical application, which can facilitate the service description and hence discovery and composition. There are two kinds of user interface available in this framework. The details of this framework are discussed in the later sections.

2 The Domain Ontology

The domain ontology model is required in order to describe the services with semantics, which provides the necessary knowledge about a certain service related to a specific domain. In this paper the Web Ontology Language(OWL)is used for modeling the domain ontology.OWL is a language for defining and instantiating Ontologies[3].Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. OWL is intended to provide a language that can be used to describe the classes and relations between them that are inherent in Web documents and applications.

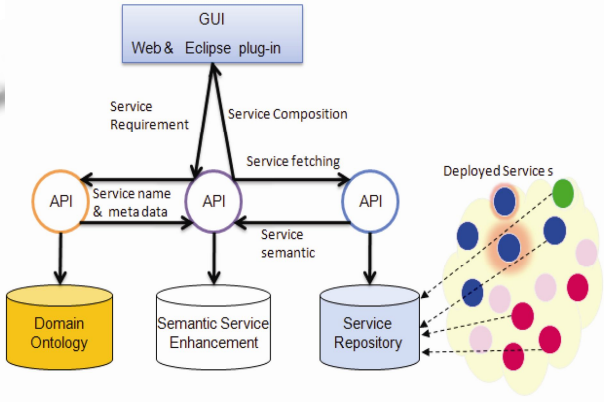

Fig.1 Ontology-based service composition framework

The use of the OWL language is to formalize a domain by defining classes and properties of those classes, define individuals and assert properties about them, and reason about these classes and individuals to the degree permitted by the formal semantics of the OWL language[4].

From section 1 we know that the domain ontology build the solid foundation for the service semantic enhancement and service discovery and composition. Based on some mature methods about creating the domain ontology, one can build a Supply-Marketing-Inventory ontology, which includes some basic concepts in Supply-Marketing-Inventory, such as enterprise, provider, contract, goods, and customer. Following the methods of Natalya F. Noy and Deborah L. McGuinness[5], we use the following seven steps to build our ontology:

Step1.Determine the domain and scope of the ontology.

Step2. Consider reusing existing ontologies.

Step3. Enumerate important terms in the ontology.

Step4. Define the classes and the class hierarchy.

Step5. Define the properties of classes-slots.

Step6. Define the facets of the slots.

Step7. Create instances.

Fig.2 shows the main concepts in this ontology. There are three components in the domain of Procurement-Selling-Inventory. Next, we will go into the details of one of the important concepts, i.e., the selling concept.

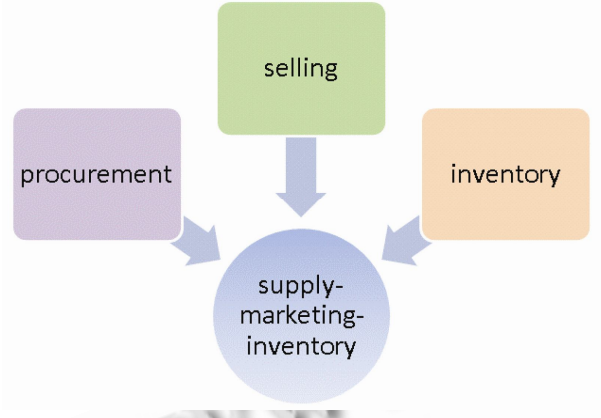

Fig.2 Concept sets of supply-marketing-inventory

Fig.3 shows the relationship between different classes under the domain ontology of selling, which is generated by the Protégé-OWL editor 3.2.1[6], which enables users to build ontologies in the W3C's OWL. Some resources such as Word Net are referenced when building the ontology[7].

Domain Ontology about selling includes 5 concepts (or 5 classes). They are goods, client, enterprise, contract, and status_contract respectively. The class of Goods has two subclasses, consumable (for consumption) and goods_industry (for industry). The class of client, which has relationship with enterprise, has two subclasses, common (means that common client) and VIP (means that very important client). The class of Contract, which is the bridge between client and enterprise, has two subclasses, purchase_contract (about purchasing, not used for selling) and sell_contract (contract about selling). The class of status_contract, which describe the status of the contract, has 5 different statuses.

In OWL all classes may have two kinds of properties, i.e. Datatype properties and Object properties. Fig. 4 describes the properties of class contract.

Every contract has such Datatype properties as ContractID, Begin_Time, End_Time, and Quantity. If two classes need to be connected the Object properties are necessary. For example, class contract must record enterprise, client and status, so it must have at least four Object properties, hasEnterprise, hasClient, hasStatus, and hasGoods, which are related to enterprise, client, status_contract, and goods respectively.

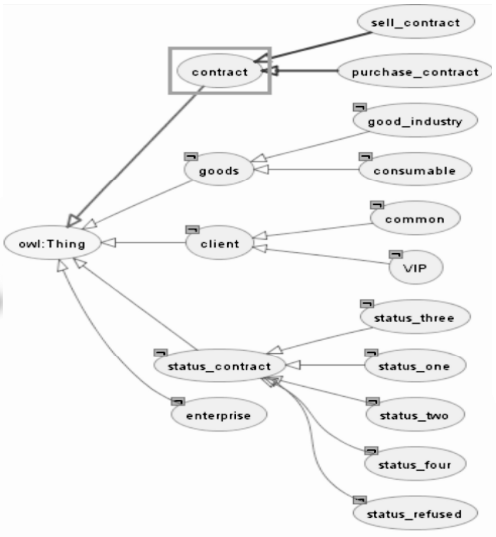

Fig.3 Relationship of different classes at the domain of selling

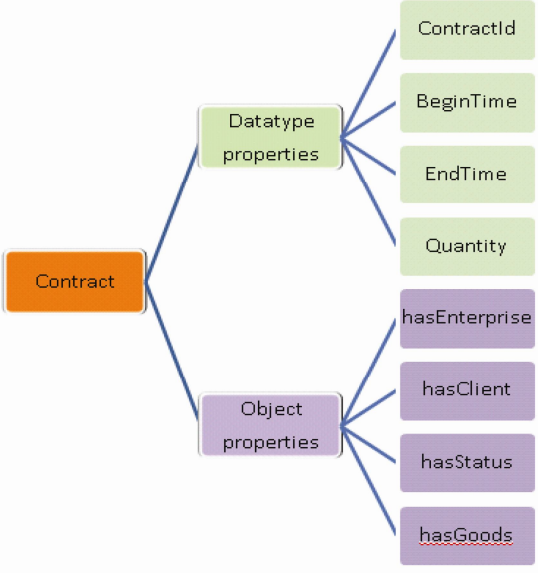

Fig.4 All properties of contract

3 The Service Repository

To leverage both the external and internal deployed services a service repository is built with the capabilities of managing services with functions such as service registration, publishing, discovery, matching, versioning,

and monitoring. Besides web services, other services such as EJBs, and POJOs are supported. The repository hereby acts as the navigation point of all the deployed services either inside or outside an enterprise.

There are two objective of our service repository: One is to provide the capabilities to store, manage and version service information and their artifacts. The artifacts include interfaces, contracts, SLAs, dependencies, etc. This function is from holistic view other than technical details of an enterprise's services. The other is to define a set of XML Schema for service descriptions and generates WSDL accordingly, when needed.

3.1 Principles of services descriptions metadata

In our services repository, we define XSD schemas for services descriptions. The current industry standards such as WSDL describe a service as a collection of operational interfaces and their type specification together with deployment information. These specifica- tions are limited in their ability to express the capabilities and requirements of the services themselves. The service repository is the master metadata repository for service descriptions. The concept of "service" here includes traditional Web services that implement WSDL interfaces with SOAP/HTTP bindings as well as a broad range of SOA services that can be described using WSDL or XSD, but might use a range of protocols and be implemented according to a variety of programming models.

The schema's definition is referred to the Web Services Invocation Framework (WSIF)[8].WSIF enables developers to interact with abstract representa- tions of Web services through their WSDL descriptions instead of working directly with the Simple Object Access Protocol (SOAP) APIs, which is the usual programming model. With WSIF, developers can work with the same programming model regardless of how the Web service is implemented and accessed. In this paper we do not take advantage of the WSIF's programming model. Rather, we use the description capability of services of WSIF.

The schemas we defined here have four parts:

(1) The first part is the declarations of interfaces of a service, which describes the operations provided by the service. The interface is described by <interface> element. The interface includes one or more operations with input and output parameters. The parameters are defined in the <types> section.

```
<<?xml version="1.0" ?>
<description      targetNamespace="http://ss.pku/helloworld/"
                  xmlns:tns="http://ss.pku/helloworld/"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  xmlns:xsd1=:"http://ss.pku
/ServiceRepository/xsd"
                  xmlns:ejb="http://ss.pku
/ServiceRepository/xsd/ejb/"
                  xmlns="http://ss.pku/helloworld/">

<!-- types declarations -->
<types>
   <xsd:schema targetNamespace="http://ss.pku
/ServiceRepository/xsd"
     xmlns ="http://ss.pku /ServiceRepository/xsd">
      <xsd:element name="GetHelloStringRequest"
type="xsd:string"/>
      <xsd:element name="GetHelloStringResponse"
type="xsd:string"/>
   </xsd:schema >
</types>

<!—service interface    declaration -->
<interface name="HelloWorld">
   <operation name="getHelloString">
      <input messageLabel="In" element="xsd1:
GetHelloStringRequest" />
      <output messageLabel="Out" element="xsd1:
GetHelloStringResponse"/>
   </operation>
</ interface>

<!-- binding declaration -->
<binding name="HelloWorldEJBBinding"
interface="tns:HelloWorld">
   <ejb:binding/>
   <ejb:operation ref="tns:getHelloString"
                  methodName="getHelloString"
                  homeInterface="pku.ss.HelloWorld"
                  ejb-link-name="
HelloWorldEJB.jar#getHelloString"
                  session-type=" stateless"
                  ejb-version="EJB2"
                  name="pku.ss.HelloWorldHome"
              uri= "corbaname:
iiop://localhost:2089:#ejb/pku.ss.HelloWorld/>
   </ejb:operation>
</binding>

<!-- service declaration -->
<service name="HelloStringService"
interface="tns:HelloWorld">
   <endpoint name="HelloWorldEndpoint"
              binding =    "HelloWorldEJBBinding"
              address="http://ss.pku/helloworld/"/>
</service>
</description>
```

Fig.5　An EJB binding service

(2) The second part describes the "binding" of a Service. Services use <binding> element to describe the access mechanism that service's consumers have to use to

call the service, including Web service, stateless session EJB, and Java class.

(3) The third part describes the physical location (address, URL) where the service is available. The URL was defined in <address>element.

(4) The fourth part describes nonfunctional attributes and attributes for service-level agreements of the services. For example: how long a service usually runs, who is allowed to call it, how much a service call costs, and so on.

Fig.5 shows an example that describes an EJB into a service with our schema.

3.2 The functions of a service repository

The functions of service repository include:

(1) Service Registry: The registration information includes attributes such as name, version, and description of the service and links to the auxiliary documents imported by the service definition document.

(2) Service discovery. In the services repository, one can search a target service and its artifacts based on the key word. This discovery method is very simple and is used only inside this framework, whereas the service discovery and composition method discussed in next part is the advanced one and will be used by the application developers.

(3) Service validation. The service repository checks whether the services are available in the repository.

(4) Service versioning. The service repository provides versioning functions for all service artifacts, regardless of their type.

(5) Service Repository APIs: Besides using a Web-based console application to manage services artifacts, applications can interact with repository using its Application Programming Interfaces (APIs) for CRUD operations.

4 Semantic Encapsulation of Services and Ontology-based Service Composition

The traditional way of service discovery such as the UDDI discovery is keyword-based matching, which is usually considered poor in performance. To enhance the efficiency and accuracy of service discovery, we propose the idea to append additional semantic information for the services in the service repository, which describe the services' details that can be used to discover appropriate service.

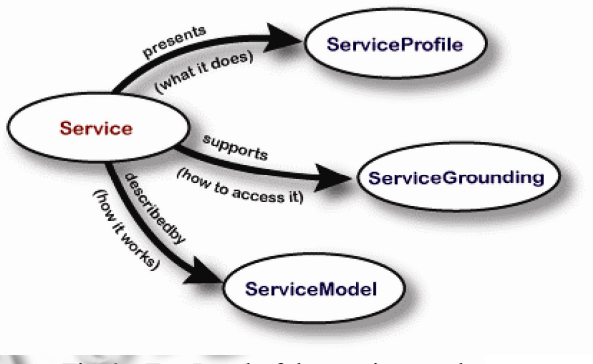

Fig.6 Top Level of the service ontology

The W3C proposes an ontology language named OWL-S for semantic description of web services. The OWL-S' top level is showed in Fig.6 [9]. An OWL-S web service advertisement has three parts: service profile tells "What the service does", service model tells "how the service works", and service grounding tells "how to access the service". It also has four attributes: Inputs, Outputs, Preconditions and Effects, which are usually used for service discovery and composition.

4.1 An example for semantic encapsulation of services

In this part a simple example of encapsulate a POJO into service is given. The concepts in the Domain Ontology are used for the semantic encapsulation of services. Fig.7 shows a java class named Generate SaleOrder. Let's suppose an enterprise provide this service. The customer signs in, selects the goods he wants and then the service will generate an Order for him.

```
public class generateSaleOrderImpl {
        private String Sale_Order_Id;
        public String generateSaleOrder(String goodsId,int
        quantity) {
                return Sale_Order_Id;
        }
}
```

Fig.7 A java class

Fig. 8 shows a part of the whole file. You can see the service has two input "goodsId" and "quantity". It also has an output "SaleOrder". The concept used by the IOs is defined in the domain ontology before. The rest part of this file defines the service's preconditions, effects and

other information. Other services may be more complex but they all follow the similar format.

```
<!--
URI1: http://www.w3.org/2001/XMLSchema#anyURI
URI2: http://www.owl-ontologies.com/Ontology1230965933.owl
URI3: http://www.example.org/owls/generateSaleOrder.owl
-->
<rdf:RDF>
  <owl:Ontology rdf:about=""> …  </owl:Ontology>
  <process:Output rdf:ID="SaleOrder">
    <process:parameterType rdf:datatype="&URI1">
    &URI2#sell_contract</process:parameterType>
  </process:Output>
  <rdf:Description rdf:about="&URI3#generateSaleOrderProfile">
    <profile:hasResult>
      <process:Result rdf:ID="result">
        <process:hasResultVar>
          <process:ResultVar rdf:ID="ResultVar">
            <process:parameterType rdf:datatype="&URI1">
            &URI2#status_one</process:parameterType>
          </process:ResultVar>
        </process:hasResultVar>
      </process:Result>
    </profile:hasResult>
    <profile:hasPrecondition
rdf:resource="http://www.daml.org/services/owl-s/1.2/generic/Expressi
on.owl#AlwaysTrue"/>
    <profile:hasOutput rdf:resource="#SaleOrder"/>
    <profile:hasInput>
      <process:Input rdf:ID="quantity">
        <process:parameterType rdf:datatype="&URI1"

>http://www.w3.org/2001/XMLSchema#int</process:parameterType>
      </process:Input>
    </profile:hasInput>
    <profile:hasInput>
      <process:Input rdf:ID="goodsId">
        <process:parameterType rdf:datatype="&URI1">
          &URI2#goods</process:parameterType>
      </process:Input>
    </profile:hasInput>
  </rdf:Description>
  <process:Result rdf:ID="Result_5"/>
  <rdf:Description rdf:about="&URI3#in0">
    <rdfs:label
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >goodsId</rdfs:label>
    <process:parameterType rdf:datatype="&URI1">
    &URI2#goods</process:parameterType>
  </rdf:Description>
  <process:Result rdf:ID="Result"/>
  <rdf:Description
rdf:about="&URI3.owl#generateSaleOrderAtomicProcessGrounding">
    <grounding:wsdlOutput>
      …..
    </grounding:wsdlOutput>
  </rdf:Description>
  <rdf:Description>
    <grounding:owlsParameter rdf:resource="#goodsId"/>
  </rdf:Description>
  <rdf:Description>
    <grounding:owlsParameter rdf:resource="#quantity"/>
  </rdf:Description>
  <rdf:Description rdf:about="&URI3#generateSaleOrderProcess">
    <process:hasOutput rdf:resource="#SaleOrder"/>
  </rdf:Description>
</rdf:RDF>
```

Fig.8　An OWL-S service description

4.2 The service discovery algorithm

After the semantic encapsulation is finished, the semantic service can be used for service discovery and composition.　We experimented with two algorithms in our early work. One is proposed in Ref.[10]. It is just an algorithm for service discovery. The other one called Service Aggregation Matchmaking (SAM) can do the composition work while discovering services[11].

The first algorithm compares the IOs (inputs and outputs) given by client with the IOs of all services stored in the service repository and computes the minimal distance between their concepts in the taxonomy tree of domain ontology. It uses four match degrees: exact, plug in, subsume and fail to distinguish the match results. Matched services will be sorted by the degree and returned to the client.

The second algorithm SAM is a more complex one. At first SAM builds a process model tree for each OWL-S service's service model in service repository. An example is shown in Fig. 9. Then the algorithm finds out useful trees for the client's request and uses them to produce a dependency graph representing the dependencies among atomic processes and their IOs as shown in Fig. 10. The dependency graph will be analyzed to recognize the useful nodes and remove the useless nodes. The process nodes remain in the graph at last can be used to produce the matching results which will be return to the client.

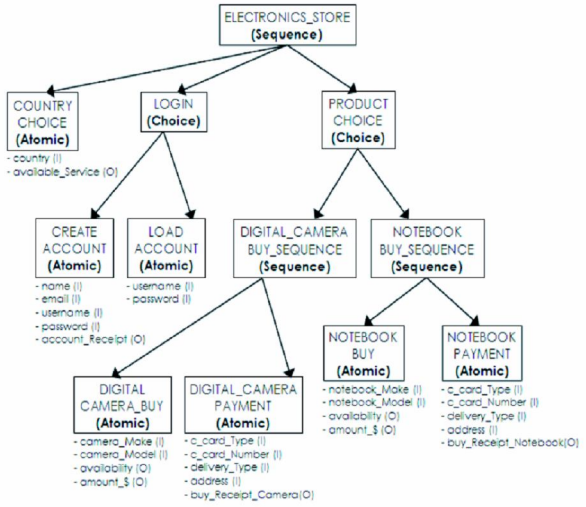

Fig.9　Process model of an electronics store service[11]

Both of the two algorithms still have some deficiencies. The time complexity may become very high if there are a lot of services in the repository. They also need the client to provide a lot of IOs of the anticipant service, which may confuse the client sometimes. To

overcome the deficiencies, we are trying to find out a new algorithm for service discovery and composition in our future work.

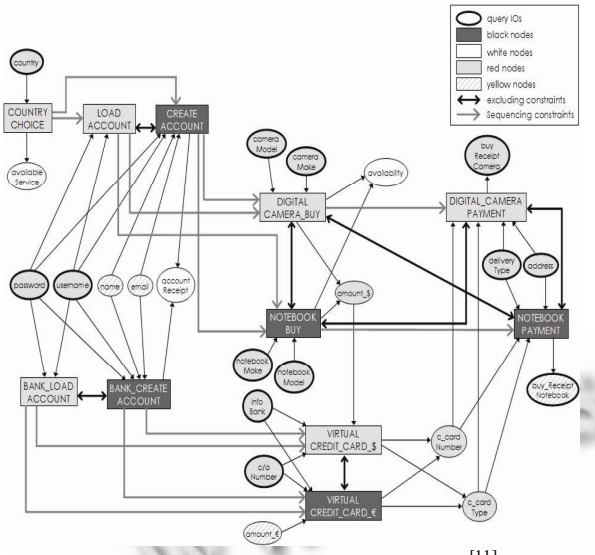

Fig.10 Colored dependency graph [11]

5 Conclusion

A framework for ontology based service composition is proposed. The three main parts, i.e. the Domain Ontology, the Service Repository, and the Semantic service enhancement are thoroughly discussed. The SOA based system development will benefit from this framework. With the service repository the large amount of existing services deployed external and internal of an enterprises can be managed. With the semantic service enhancement one can easily find a service more accurately and more efficiently. Doman Ontology provides the concepts and their properties of a certain domain, which can facilitate the semantic description of a service with OWL-S and definitely can accelerate the service discovery and service composition. Currently we are working on the toolset for this framework and on the service discovery and service composition algorithms.

References

1 http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm.

2 Chris Dudley, Laurent Rieu, et al. WebSphere Service Registry and Repository Handbook, March 2007, IBM.

3 Smith MK, et al. OWL Web Ontology Language Guide,http://www.w3.org/TR/2004/REC-owl-guide-20040210/.

4 http://www.w3.org/TR/2004/REC-owl-guide－20040210/ #Usage.

5 Noy NF, Deborah L. McGuinness, Ontology Development 101: A Guide to Creating Your First Ontology, http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.htm.

6 http://protege.stanford.edu/.

7 http://wordnet.princeton.edu/.

8 http://ws.apache.org/wsif/.

9 http://www.w3.org/Submission/OWL-S/,OWL-S: Semantic Markup for Web Services.

10 Paolucci M, Kawamura T, Payne T, Sycara K. Semantic Matching of Web Services Capabilities. First Interna- tional Semantic Web Conference on The Semantic Web, LNCS 2342, Springer-Verlag, 2002:333－347.

11 Brogi A, Corfini S, Popescu R. Composition-oriented Service Discovery. Gschwind T, ed. Proceedings of the Software Composition. Edinburgh: Springer-Verlag, 2005:15－30.