Discovering E-Services Using UDDI in SELF-SERV

Quan Z. Sheng, Boualem Benatallah, Rayan Stephan, Eileen Oi-Yan Mak, Yan Q. Zhu
School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia
Tel: +61 2 9385 7998 Fax: +61 2 9385 5533
{qsheng, boualem, rayans, eileenm, yqzhu }@cse.unsw.edu.au

Abstract

One of the key needs for businesses today over the Internet is the ability to dynamically discover services that they need, and compose them in an interoperable manner to carry out their business processes. The Universal Description, Discovery and Integration (UDDI) is a new technology that offers a standard way for businesses to build a registry, discover each other and describe how to interact over the Internet. In this paper, we discuss how UDDI is integrated into a Web service composition framework.

Keywords: e-services, advertisement and discovery, UDDI

1. Introduction

As e-business increasingly move towards an environment of machine-to-machine interoperations, efficient discovery and composition of Web services become essential. By Web service (also called e-service)[1], we mean a semantically well defined abstraction that allows users to access functionalities offered by Web applications. A typical example of a Web service is booking an airline ticket through an HTML-based interface. There seems to be a consensus that the future of e-business collaboration will be through Web services.

Web service composition is a very active area of research and development [4]. For instance, HP has developed a platform, called eFlow [2, 3], for specifying, enacting, and monitoring composite Web services. SELF-SERV [1] project focuses on declarative Web services composition using statecharts [12], where the resulting services can be executed in a decentralised way within a dynamic environment. However, all these service composition frameworks assume that Web services can be easily collected using some certain service discovery tools.

UDDI [6] is a new technology announced by Microsoft, IBM and Ariba. It provides a set of specifications that define a way to publish and discover information about Web services. It offers a standard way for businesses to build a registry, discover each other, and describe how to interact over the Internet.

In our research project SELF-SERV (compoSing wEb accessibLe inFormation & buSiness sERVices), we implemented the *Service Discovery Engine* using UDDI[2]. In this paper, we show in details on how UDDI is integrated into SELF-SERV platform for Web services advertisement and discovery.

The rest of the paper is organised as follows: in Section 2, we introduce the basic concepts of UDDI. In Section 3, we briefly introduce Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP) and then describe how UDDI, WSDL and SOAP work together for Web service advertisement and discovery. In Section 4, we discuss the implementation of the SELF-SERV Service Discovery Engine. Finally, conclusion is given in Section 5.

2. Universal Description, Discovery and Integration (UDDI)

UDDI is an industry effort started in September 2000 by Ariba, IBM, Microsoft and other 33 companies. It is a specification that defines a service registry of available Web services. It allows a company to publish a

[1] In the remainder, we use the terms e-service, Web service and service interchangeably.

Correspondence to: Quan Z. Sheng, School of Computer Science and Engineering, UNSW, Sydney, NSW 2052, Australia.

[2] In the former implementation, we just used discovery engine in AgFlow [10] project.

description of available services to the registry, thus announcing itself as a service provider. Service requesters can send requests as SOAP messages to the service registry to discover a service provider for obtaining services. Upon finding a potential business partner, the service requester can easily integrate with the service provider.

The UDDI information model, defined through an XML schema, identifies four core types of information as shown in Figure 1. They are *businessEntity* (i.e., service provider), *businessService* (i.e., service), *bindingTemplate* (i.e., access information) and *tModel* (i.e., service type definition).

BusinessEntity describes information about business (e.g., names, description, services offered and contact data etc.). A businessEntity contains a collection of businessService elements, one for each Web service that the business wishes to publish. Each businessService provides more details on the service offered by the service provider. A businessService contains a collection of bindingTemplate elements. A bindingTemplate

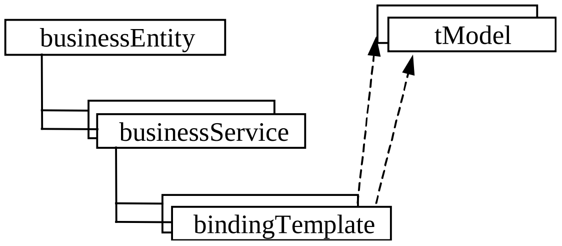


Figure 1. UDDI data structure.

describes the access information (e.g., the endpoint address) and describes how the businessService uses various technical specifications. In other words, a bindingTemplate provides the details of how and where the service is accessed. A technical specification is modelled as a tModel. tModel provides the ability to describe compliance with a specification, a concept, or a shared design. When a particular specification is registered with the UDDI as a tModel, it is assigned a unique key, which is then used in the description of service instances to indicate compliance with the specification. More details about UDDI specifications can be found in [9].

In summary, UDDI provides two main functionalities:

- A mechanism for service providers to register themselves and their services with the UDDI registry.
- A mechanism for service subscribers to search for the available services from the UDDI registry.

It should be noted that the UDDI specification provides a platform-independent way of describing and discovering services. UDDI itself is not a service description language. Services need to be described in service description languages like WSDL. There is also a need for a remote invocation mechanism like SOAP. In the next section, we first briefly introduce WSDL and SOAP, and then describe how UDDI, WSDL and SOAP complement each other for advertising and discovering Web services.

3. Web Service Advertisement and Discovery

3.1. Web Services Description Language (WSDL)

WSDL [7], proposed by IBM and Microsoft, is a general purpose XML language for describing the interface, protocol bindings and the deployment details of Web services. It has been submitted to the W3C for consideration as a recommendation.

A WSDL document describes how to invoke a service. It provides information on the data being exchanged, the sequence of messages for an operation, the location of the service and the description of bindings (e.g., SOAP or HTTP).

The *import* element in WSDL allows the separation of the service description into two parts: *service interface definition* and *service implementation definition* (see Figure 2). This enables each part to be defined separately and independently, and reused by other parts.

The information that is common to a certain category of business services is included in the service

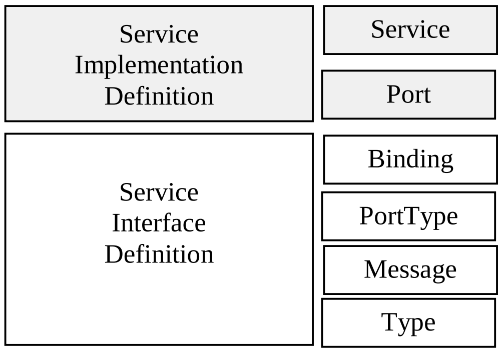

Service Implementation Definition
Service
Port
Service Interface Definition
Binding
PortType
Message
Type

Figure 2. Service description of WSDL.

interface definitions, which are then registered as UDDI tModels. The service interface contains WSDL elements that comprise the reusable portion of the service description. The WSDL:*portType* element defines the operations of the Web service. The input and output parameters of an operation of the service are defined in the WSDL:*message* element. The WSDL:*type* element contains the complex data types (defined by service provider) used within message. The WSDL:*binding* element describes the protocol, data format, security and other attributes for a particular service interface (WSDL:*portType*).

The service implementation definition is a WSDL document that describes how a particular service interface is implemented by a given service provider. A Web service is modelled as a WSDL:*service* element which contains a collection (usually one) of WSDL:*port* element. A port associates an endpoint (e.g., a network address or URL) with a WSDL:*binding* element from a service interface definition. A concrete example of Web service represented in WSDL can be found in [11].

WSDL documents of a Web service can be hand coded or automatically created by some generation tools (e.g., IBM Web Services ToolKit 2.4 [8]). In SELF-SERV platform, we used WSDL Generator of IBM Web Services ToolKit 2.4 (WSTK2.4) for WSDL generation. We will give the details about this implementation in Section 4.

3.2. Simple Object Access Protocol (SOAP)

XML messaging is the most fundamental part of a Web service. The current industry standard for XML messaging is SOAP [5].

SOAP is a simple and lightweight XML-based mechanism for exchanging structured data between network applications. It is composed of three parts: an *envelope* that defines a framework for describing what is in a message and how to process it, a set of *encoding rules* for expressing instances of application-defined data types, and a *convention* for representing remote procedure calls and responses. SOAP can be used in combination with or re-enveloped by a variety of network protocols such as HTTP, SMTP etc.

SOAP is an easy way to invoke a remote procedure call (RPC) on another machine in a Request/Response fashion. The client program forms a request that consists of a valid SOAP XML document and sends it over HTTP to a server. The server picks up the SOAP request, parses and validates it, invokes the requested method with the supplied parameters, forms a valid SOAP XML document and sends it back over HTTP to the client.

3.3. Integrating UDDI with WSDL and SOAP

From Figure 3 we can see that there are three roles involved in Web service applications: *service provider*, *service registry* and *service requestor*. The interactions between three roles are *publish*, *find* and *invoke*. Web services are implemented and published by service providers. They are discovered and invoked by service

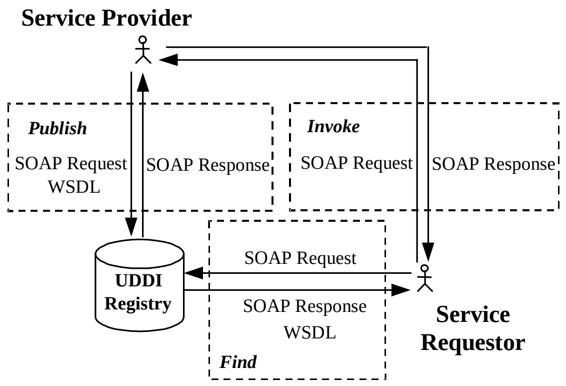


Figure 3. Integration of UDDI with WSDL and SOAP

requestors. Information about a Web service is kept within a service registry.

Figure 3 shows how UDDI, WSDL and SOAP operationally complement each other in service advertisement and discovery.

Service registration, discovery and invocation are implemented by SOAP calls. First, service provider sends a UDDI SOAP request, together with the service descriptions (i.e., WSDL) of the Web services, to UDDI registry operators (e.g., Microsoft UDDI test operator). The acknowledgment of successful registration also returns a SOAP message. After a service is registered in the UDDI registry, service requestor can discover this service from the registry. Requestor sends the UDDI SOAP request (e.g., business name, service type etc.) to the UDDI registry. The registry locates the service and returns its description (WSDL). Then requestor invokes the service using the binding details in the service description to locate, contact and invoke the service.

4. Service Discovery Engine in SELF-SERV

In this section, we discuss how to integrate UDDI into SELF-SERV platform. First, we briefly overview the architecture and the Service Discovery Engine of the SELF-SERV, and then we introduce the implementation of the Service Discovery Engine of SELF-SERV.

4.1. Service Discovery Engine

SELF-SERV, is a framework for dynamic and peer-to-peer provisioning of Web services [1]. In SELF-SERV, Web services are declaratively composed, and the resulting composite services are executed in a decentralised way within a dynamic environment.

Figure 4 shows the architecture of the SELF-SERV prototype, composed of an *interface*, a *service manager* and a *pool of services*. The service manager consists of three modules, namely the *Service Discovery Engine*, the *Service Editor* and the *Service Deployer*.

SELF-SERV distinguishes three types of services: *elementary services*, *composite services*, and *service communities*. An elementary service is an individual Internet–accessible application (e.g., a Java program) that does not reply on another Web service to fullfill user requests. A composite service aggregates multiple Web services, which are referred to as its components. It is represented using a declarative language based on statecharts [12]. The concept of service community is a solution to the problem of composing a potentially large number of dynamic Web services. A community describes the capabilities of a desired service without referring to any actual Web service providers.

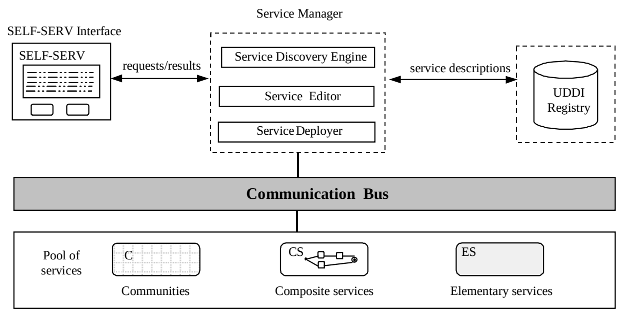


Figure 4. Architecture of the SELF-SERV prototype.

Any service wishing to participate in the SELF-SERV platform needs to register with the Service Discovery Engine. The Service Editor provides a visual interface to create new services by composing existing Web services using statecharts. Before this procedure takes place, the service composer must find the relevant services among the advertised Web services in the Service Discovery Engine at design time.

In this paper, we focus on the implementation of Service Discovery Engine. For other modules of SELF-SERV, see [1] for more details.

4.2. Implementation of the Service Discovery Engine

We implemented the Service Discovery Engine using UDDI technology. In our implementation, we massively benefited from the release of the IBM WSTK2.4 [8]. WSTK is a showcase package for Web services-related emerging technologies. It is available for free on the IBM alphaWorks website.

IBM WSTK provides several components and tools for the new emerging technologies of Web services (e.g., XML, UDDI, SOAP etc.). Some of them are listed as follows:

- A lightweight application server called “embedded WebSphere”. The embedded WebSphere includes all the implementation of the UDDI. In addition, WSTK has its own UDDI registry for private use. This is ideal for those who just want to build a private UDDI environment for research prototype like our SELF-SERV.
- A WSDL generation Tool for assisting in encapsulating legacy code as Web services (i.e., the tool creates WSDL documents and SOAP deployment descriptors from legacy codes such as Java Beans, Java classes, EJBs and COM objects.)
- The UDDI Java API (UDDI4J) which allows applications to perform the Save, Delete, Find and Get operations against a UDDI registry.
- The Web Services Toolkit Configuration Tool that can be used to set up the toolkit (e.g., configure hostname and port information etc.) .

The Service Discovery Engine was developed in Java on top of Windows NT 4.0 platform. There are two fundamental parts in the Service Discovery Engine: publishing and discovering Web services. However, before a Web service can be published with UDDI registry, the WSDL descriptions of the Web service should be created (see Section 3.1).

Creating WSDL descriptions for Web services

The application code for Web services can be developed using any language (e.g., Java, C++ etc.). The WSDL description of the service is used to wrap that code. This provides a great convenience for businesses because they can migrate their legacy applications (e.g., Java class, EJB, COM etc.) of the service applications to Web services without implementing the transformation logics.

The WSDL descriptions can be created manually or using WSDL Generation Tool (e.g., WSDL Generator of WSTK 2.4). In our implementation, we created WSDL descriptions for Web services by using the WSTK WSDL Generator.

The WSDL Generator creates three documents for each service. Two documents are with WSDL extensions and another one is named “*deploymentdescription.xml*”. One of the WSDL documents includes the term “*interface*” in the filename, which contains the abstract *portType* definition of the Web service interface. The other WSDL document contains the concrete implementation information about the Web service. The deploymentdescription.xml document is the Apache SOAP deployment descriptor used to link to the application code being exposed.

Deploying and publishing Web services

Before publishing a Web service using UDDI, the Web service should be deployed first. This procedure includes the deployment of the WSDL definitions to a location where they may be retrieved using public URLs. After that, the deployment descriptor document is linked to the code functionality to the Web services engine running inside the Application Server.

Upon successful deployment of the service, it is easy to publish it using just a few lines of Java code using APIs provided by WSTK2.4. Figure 5 shows the Java code fragment for the whole procedures of the advertisement, including creating the UDDI service provider entry, service registration and deploying the service to the SOAP engine.

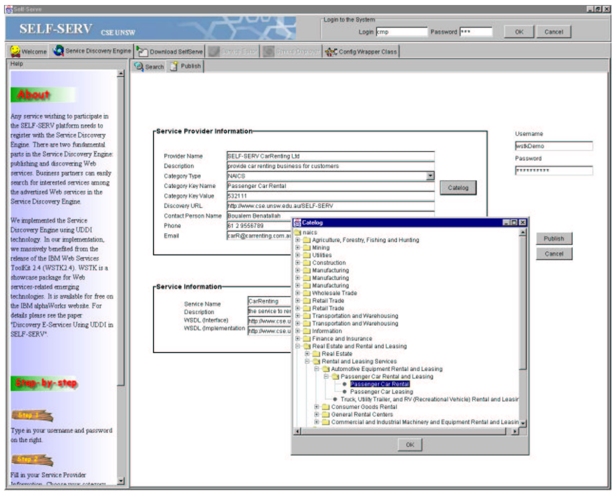
```
1.  public class publishService{
2.  String serviceDefinitionWSDL=”http://www.cse.unsw.
3.  edu.au/SELFSERV/wsdl/carRental.wsdl”;
4.  String serviceInterfaceWSDL=”http://www.cse.unsw.
5.  edu.au/SELF-SERV/wsdl/carRental-interface.wsdl”;
6.  String soapRouter=”http://www.cse.unsw.edu.au
7.  /soap/servlet/rpcrouter”;
8.  String serviceDeployment=”http://www.cse.unsw.edu.
9.  au/SELF-SERV/wsdl/deploymentDescriptor.xml”;
10. String inquiryURL=”http://www.cse.unsw.edu.au/
11. services/uddi/inquiryapi”;
12. String publishURL=”https://www.cse.unsw.edu.au/
13. services/uddi/publishapi”;
14. ServiceRegistryProxy srp = new ServiceRegistryProxy(
15. inquiryURL, publishURL, username, password);
16. CategoryList spCatList = new CategoryList();
17. spCatList.addCategory(TModelKeyTable.UNSPSC,
18. ”84121”, TModelKeyTable.UNSPSC_TMODEL_KEY);
19. ServiceProvider sp=new ServiceProvider(“CarRental”,
20. “SELF-SERV service provider example”, spCatList);
21. sp=srp.publish(sp);
22. ServiceInterface si=new ServiceInterface(
23. serviceInterfaceWSDL);
24. si=srp.publish(si);
25. CategoryList sdCatList=new CategoryList();
26. sdCatList.addCategory(TmodelKeyTable.UNSPSC,
27. “84121”, TmodelKeyTable.UNSPSC_TMODEL_KEY);
28. SOAPServiceDefinition sd=new SOAPServiceDefinition
29. (serviceDefinitionWSDL, sdCatList, serviceDeployment,
30. soapRouter);
31. sd.createServiceManagerProxy().deployService(sd);
32. srp.publish(sp, sd);
33. }
```

Figure 5. Java code fragment for publishing a service.

The lines 2 to 9 define the locations of the service description documents (WSDL). These documents are generated by WSTK WSDL Generator. A proxy object is created, and then pointed to the SELF-SERV UDDI registry (see lines 10 to 15). Notice that the protocol in the *publishURL* is https[3]. The UDDI standard defines that anyone should be able to query a registry, but only applications and users with the proper access can modify registry information.

Lines 16 to 21 show the Java fragment for publishing a service provider. A proper category is chosen for this service provider (lines 16 to 18) before the publication. Note, if this service provider has already registered with the UDDI registry, this procedure should be skipped.

Lines 22 to 24 show the Java fragment for publishing the service interface, while deploying and publishing the service itself is showed in the lines 25 to 32. Figure 6 shows the interface of the Service Discovery Engine of SELF-SERV.



Login cmp Password ***

Any service wishing to participate in the SELF-SERV platform needs to register with the Service Discovery Engine. There are two fundamental parts in the Service Discovery Engine: publishing and discovering Web services. Business partners can easily search for interested services among the advertised Web services in the Service Discovery Engine.

We implemented the Service Discovery Engine using UDDI technology. In our implementation, we massively benefited from the release of the IBM Web Services ToolKit 2.4 (WSTK2.4). WSTK is a showcase package for Web services-related emerging technologies. It is available for free on the IBM alphaWorks website. For details please see the paper "Discovery E-Services Using UDDI in SELF-SERV".

Type in your username and password on the right.

Fill in your Service Provider Information. Choose your category

Service Provider Information

Provider Name	SELF-SERV CarRenting Ltd
Description	provide car renting business for customers
Category Type	NAICS
Category Key Name	Passenger Car Rental
Category Key Value	532111
Discovery URL	http://www.cse.unsw.edu.au/SELF-SERV
Contact Person Name	Boualem Benatallah
Phone	61 2 9556789
Email	carR@carrenting.com.au

Username wstkDemo
Password **********

Service Information

Service Name	CarRenting
Description	the service to re
WSDL (Interface)	http://www.cse.u
WSDL (Implementation	http://www.cse.u

Catelog

```
naics
  Agriculture, Forestry, Fishing and Hunting
  Mining
  Utilities
  Construction
  Manufacturing
  Manufacturing
  Manufacturing
  Wholesale Trade
  Retail Trade
  Retail Trade
  Transportation and Warehousing
  Transportation and Warehousing
  Information
  Finance and Insurance
  Real Estate and Rental and Leasing
    Real Estate
    Rental and Leasing Services
      Automotive Equipment Rental and Leasing
        Passenger Car Rental and Leasing
          Passenger Car Rental
          Passenger Car Leasing
        Truck, Utility Trailer, and RV (Recreational Vehicle) Rental and Leasir
      Consumer Goods Rental
      General Rental Centers
      Commercial and Industrial Machinery and Equipment Rental and Leasin
```

Figure 6. Interface of the Service Discovery Engine.

Discovering and invoking a Web service

WSTK also provides a client API that allows the access to a service’s WSDL document and dynamically create a proxy for invoking that service.

Figure 7 shows the Java code fragment for finding

```
1.  String service=”CarRental”;
2.  String opertaion=”CarRenting”;
3.  ServiceRegistryProxy srp=new ServiceRegistryProxy();
4.  ServiceDefinition[] sdList=srp.findServices(service, false);
5.  ServiceDefinition sd=sdList[0];
6.  String portName=”CarRentingPort”;
7.  WSDLDocument wsdl=sd.getServiceImplementation().
8.  GetWSDLDocument();
9.  ServiceProxy proxy=ServiceProxyFactory.getServiceProxy
10. (service, portName, wsdl);
11. Object[] oArgs=new Object[1];
12. ServiceRequest request=new ServiceRequest(operation,
13. OArgs);
14. Response res=proxy.invoke(request);
15. Object ret=res.getReturnValue().getValue();
16. }
```

Figure 7. Java code fragment for discovering and invoking a Web service.

[3] https means that SSL(Secure Socket Layer) is used in the online transactions.

and invoking a Web service in the UDDI registry by name. In the example given in Figure 7, the Web service named “carRental” is selected (see lines 1 to 5). But note that more advanced search parameters can be applied (e.g., the collections of service identifiers, business categories, URLs and tModels). A *ServiceDefinition* object is returned that can be used to find specific information (e.g., service provider, service name, operations and their input and output parameters etc.). Such information can then be used in SELF-SERV to compose new Web services using statecharts.

Once the service description is found, the service can then be invoked using the information contained in the service description. Lines 6 to 15 shows how to invoke the service “carRental”.

5. Conclusion

In this paper, we have presented the design and implementation of the Service Discovery Engine in SELF-SERV framework, using UDDI technology. Web services can register with the UDDI registry and can be discovered at the design time by service composer, to compose new Web services using SELF-SERV platform. We believe this is a very useful contribution because we not only discussed the integration of UDDI technology into a Web service composition framework, but also demonstrated how UDDI, WSDL and SOAP complement operationally each other for Web service advertisement and discovery.

The work presented in this paper is part of a larger, long-term research effort aiming at developing a framework for dynamic and peer-to-peer provisioning of Web services. The future work will focus on security management, self-adaptability of composite Web services.

Reference

1. B. Benatallah, M. Dumas, Q. Sheng, and A. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In Proc. of the 18th Int. Conference of Data Engineering (ICDE), San Jose, USA, February 2002. IEEE Computer Society.
2. F. Casati and M-C. Shan. Models and Languages for Describing and Discovering E-Services. In Proc. of ACM SIGMOD Int. Conference on Management of Data, Santa Barbara, 2001.
3. F. Casati, S. Ilnicki, L.-J. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and Dynamic Service Composition in eFlow. In Proc. of the Int. Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000, Springer Verlag.
4. H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In Proc. of the Int. Conference on Advanced Information Systems Engineering (CAiSE), Stockholm, Sweden, June 2000, Springer Verlag.
5. Simple Object Access Protocol (SOAP). http://www.w3.org/TR/SOAP
6. Universal Description, Discovery and Integration (UDDI). http://www.uddi.org
7. Web Service Discription Language (WSDL). http://www.w3.org/wsdl
8. IBM WSTK Toolkit. http://alphaworks.ibm.com/tech/webservicestoolkit.
9. UDDI Data Structure Reference V1.0. http://www.uddi.org/pubs/DataStructure-V1.00-Open-20000930.html
10. L. Zeng, B. Benatallah, A. Ngu, and P. Nguyen. AgFlow: Agent-based Cross-Enterprise Workflow Management System. In Proc. of 27th Int. Conference on Very Large Data Base (VLDB), Roma, Italy, 2001.
11. Web service description WSDL example. http://www.cse.unsw.edu.au/~qsheng/wsdl.htm.
12. D. Harel and A. Naamad. The STATEMATE semantics of statecharts. ACM Transactions on Software Engineering and Methodology, 5(4):293-333, October 1996.