A Formal Model of UDDI: UML Metamodel

Emilian PĂŞCĂLĂU, Adrian GIURCĂ, Gerd WAGNER

Department of Internet Technology
Institute of Informatics
Brandenburg Technical University at Cottbus, Germany
{pascalau, giurca, wagnerg}@tu-cottbus.de

Abstract. This paper presents the UML metamodel of UDDI v3.02 (release Oct 19, 2004). This release seems to be a final one since the main promoters (i.e. IBM, Microsoft, HP) stopped their activities in the UDDI technical committee. Since all major SOA vendors (BEA, IBM, Microsoft, Oracle) offers UDDI support a metamodel helping to understanding the standard and to easy deploy models to systems and tools is necessary.
Keywords: Web Services, WSDL, UDDI
Math. Subject Classification 2000:68N19, 68U35, 68P05

1 Motivation

A new emerging software paradigm of nowadays is SOA. As stated in Marks et. all [10] SOA is a conceptual business architecture, where business functionality or application logic is made available through services with exposed interfaces, and are invoked by services. Examples of Web services technologies are SOAP [9], Universal Description, Discovery and Integration (UDDI [15]), WSDL [5], electronic business XML (ebXML [6]), Security Assertion Markup Language (SAML [17]), WS-Security [16], and Business Process Execution Language (BPEL [12]). Checking web services literature (for example Alonso et. all [1] and Graham et. all [8]) we could not find an UDDI UML Metamodel.

The UDDI takes advantage of WorldWide Web Consortium (W3C[1]) and Internet Engineering Task Force (IETF[2]) standards such as Extensible Markup Language (XML [3]), and HTTP [7] and Domain Name System (DNS [11]) protocols. Additionally, cross platform programming features are addressed by adopting early versions of the proposed Simple Object Access Protocol (SOAP) known as XML Protocol messaging specifications found at the W3C Web site. The UDDI protocol is the building block that will enable businesses to quickly, easily and dynamically find and transact with one another using their preferred applications.

On the WEB before January 2006 there existed some public UDDI registries. IBM, Microsoft, SAP, Systinet offered public registries. IBM discontinued it's public UDDI Registry. It is not available neither for inquiry neither for

[1] WorldWide Web Consortium, `http://www.w3.org`
[2] Internet Engineering Task Force, `http://www.ietf.org/`

publish. Microsoft still has also a test registry and also a production registry, but these are available only for inquiry. However, all these companies and some others offers UDDI in their commercial SOA support.

Today SAP still has a public UDDI test registry that can be utilized for testing purposes[3]

This paper puts into light the basic UDDI UML Metamodel. The UDDI Metamodel presented here is based upon the UDDI v3.02 schema and standard. For the ease in understanding the UDDI XML vocabulary and standard, a graphical representation would be of a great help. The proper graphical representation of such problems is achieved by using UML. MDA techniques can be useful to deploy UDDI Metamodel in order to build different servers and tools.

2 UDDI

According with OASIS Executive Overview [14] the Universal Description, Discovery, and Integration (UDDI) protocol is a key member of the group of interrelated standards that comprise the Web services stack. It defines a standard for publishing and discovering the network-based software components of a service-oriented architecture (SOA).

The main roles of UDDI are the *Registry* role and *Service Discovery* role.

As is in object programming where the concept of an object or component registry is an essential element in the framework, a *web service registry* has the same role. The Web service architecture it is about the interaction between three primary roles: *service provider*, *service registry*, *service requester*.

The *Service Discovery* deal with tasks such as: (i) Discovering business partner, (ii) Establish potential relationship between service requester and service provider, (iii) Describes the way how the requester found out about services provided by a service provider and (iv) Describes the way in which a requester can obtain the WSDL service description from a service provider.

The IT market shows us a number of other registry technologies such as MDS[4], WS-RF[5], CCLRC[6] but none of them has a large number of users by comparison with UDDI/ebXML.

3 Basic UDDI UML metamodel

The UDDI information model is composed of the following entity types:

[3] SAP UDDI Test Registry: `http://udditest.sap.com/`, SAP UDDI Inquiry API: `http://udditest.sap.com/uddi/api/inquiry/`, SAP UDDI Publish API: `https://udditest.sap.com/uddi/api/publish/`

[4] Globus Monitoring and Discovery System: `http://www-unix.globus.org/toolkit/docs/4.0/info/key-index.html`

[5] Web Services Resource Framework: `http://www.globus.org/wsrf/`

[6] CCLRC Scientific Metadata Model and Data Portal: `http://epubs.cclrc.ac.uk/bitstream/485/csmdm.version-2.pdf`

1. *BusinessEntity*: describes a business or other organization (may also be an affiliate or subdivision) that typically provides Web services.
2. *BusinessService*: describes a collection of related Web services offered by an organization described by a BusinessEntity
3. *BindingTemplate*: describes the technical information necessary to use a particular Web service. It can be implemented by multiple businessServices.
4. *tModel*: describes a "technical model" representing a reusable concept, such as a Web service type, a protocol used by Web services, or a category system. tModels are in fact meant as a general way to speciffy information about something, via services, a business, or anything else.
5. *PublisherAssertion*: describes, in the view of one BusinessEntity, the relationship that the BusinessEntity has with another BusinessEntity.

Carlson [4] created in 2001 an UDDI metamodel based on UDDI schemas compliant with an XML Schema Candidate Recommendation(from October 24, 2000). This schema is deprecated and does not exist anymore on the UDDI official page.

We underline again that the metamodel of the present paper is based upon UDDI v3.02 standard and XML schema according with [15].

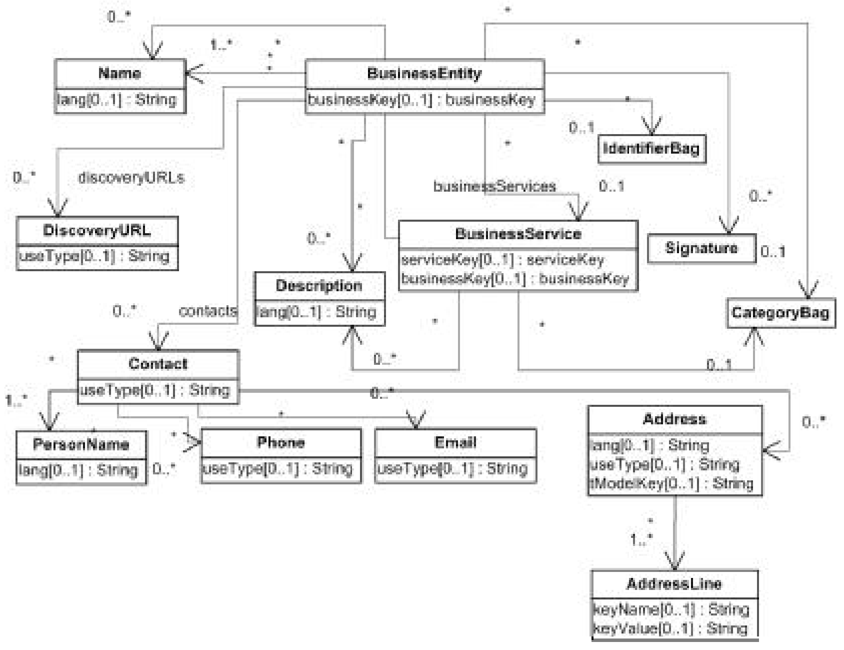

Fig. 1. UDDI-BusinessEntity

The Figure 1 presents the BusinessEntity UML metamodel.

A BusinessEntity contains descriptive information about a business or organization.

A *BusinessEntity* has two properties. The *businessKey* property uniquely identifies a businessEntity. When a BusinessEntity is published within a UDDI registry, the businessKey **must** be omitted if the publisher wants the registry to generate a key. When a BusinessEntity is retrieved from a UDDI registry, the businessKey **must** be present. *BusinessEntity* refers to a *Name*.

A *Name* specifies the name of the business. Additional names can be used to specify different names for the business, in different languages with the help of the `lang` property (using guidelines from RFC 3066, but the name has to have no more than 26.).

Also *BusinessEntity* instance refers to a list of optional *DiscoveryURL*, *Description*, *Contact*, *BusinessService*, *CategoryBag*, *IdentifierBag*, and *Signature* instances.

DiscoveryURL is used to specify locations where an user can go for further information about a BusinessEntity (a company). *DiscoveryURL* has optional property `useType` that encodes the name of the convention that the referenced document follows. There are reserved values like: "businessEntity", "homepage".

Description encodes the description of the business. One can specify a different descriptions in different languages by using the optional `lang` property.

A *Contact* contains information about the person or a job role within the BusinessEntity so that someone who finds the information can make human contact for any purpose. A *Contact* provides an optional *useType* property that help in defining a department i.e. "technical questions", "technical contact", "establish account", "sales contact", etc. A *Contact* refers to a *PersonName*, that encodes the name of the person or name of the job role supporting the contact.

PersonName provides an optional property `lang` that encodes the language signify the contextual language (if any) in which a given name is expressed in. i.e. real person name is John in English and you can use the lang="RO" and the name will be Ion. A *Contact* refers also to optional *Phone*, *Email*, and *Address*. *Phone* and *Email* both provide optional property `useType` which is used for descriptive purpose.

Address represents the contacts postal address, in a form suitable for addressing an envelope. *Address* provides some optional properties: `lang`, `useType`, `tModelKey`, and refers to at least one *AddressLine*. `lang` property encodes the language in which the address is expressed. `useType` encodes the type of the address. i.e. "headquarters", "sales office", "billing department", etc. `tModelKey` is a tModel reference that specifies that *keyName*, *keyValue* pairs given by subsequent addressLine elements, if *AddressLine* are present at all, are to be interpreted by the address structure associated with the tModel that is referenced. i.e. having a tModel that describes a postal address; the postal address could have predefined Street with a value associated i.e. 20 etc. In this case the *AddressLine keyName* will be Street, and *keyValue* will be 20.

AddressLine encodes a part of the actual address. *AddressLine* provides two optional properties, *keyName* and *keyValue*. Both properties **must** be present in each address line if a tModelKey is specified in the address structure. When no *tModelKey* is provided for the address structure, the *keyName* and *keyValue* properties have no defined meaning.

A businessEntity can refer to none or more Contacts.

A *Signature* is an identification signature. *Signature* is an XML digital signature, and is in accordance with XML-Signature specification.

The *CategoryBag* and *IdentifierBag* are going to be presented in the Section 3.1 because by their use UDDI catalogoues information that is registered.

There exists differences between the metamodel presented here and the one introduced in [4]. For example in the present paper a BusinessEntity does not have Name and Description as properties, but they are referenced. This is because in the UDDI v3 schema these are block elements, and they also have properties. Also there are optional properties and not mandatory as they were presented in [4].

The *BusinessService* describes the service that is associated with the *BusinessEntity*. A *BusinessEntity* can refer more than one BusinessService. The metamodel of *BusinessService* is depicted in Figure 2.

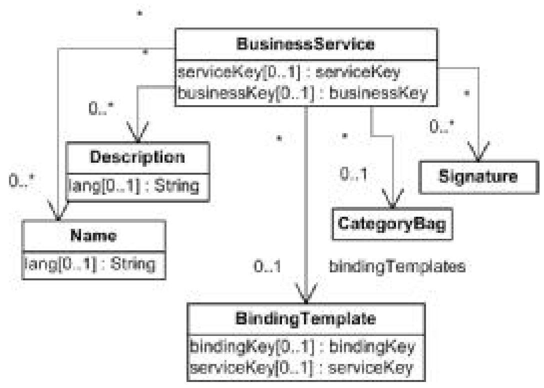

Fig. 2. UDDI-BusinessService

BusinessService optional refers to *Name*, *Description*, *Signature*, *CategoryBag*, and *BindingTemplate*.

The *Name*, *Description*, *Signature* and *CategoryBag* have the same properties as stated when describing *BusinessEntity*.

A *BusinessService* can refer more than one *BindingTemplate*. A *BindingTemplate* defines how a service can be invoked and also what it does. There can be multiple ways of invoking each BusinessService. Each invoke is encoded in a *BindingTemplate*. Figure 3 depicts the *BindingTemplate* UML metamodel.

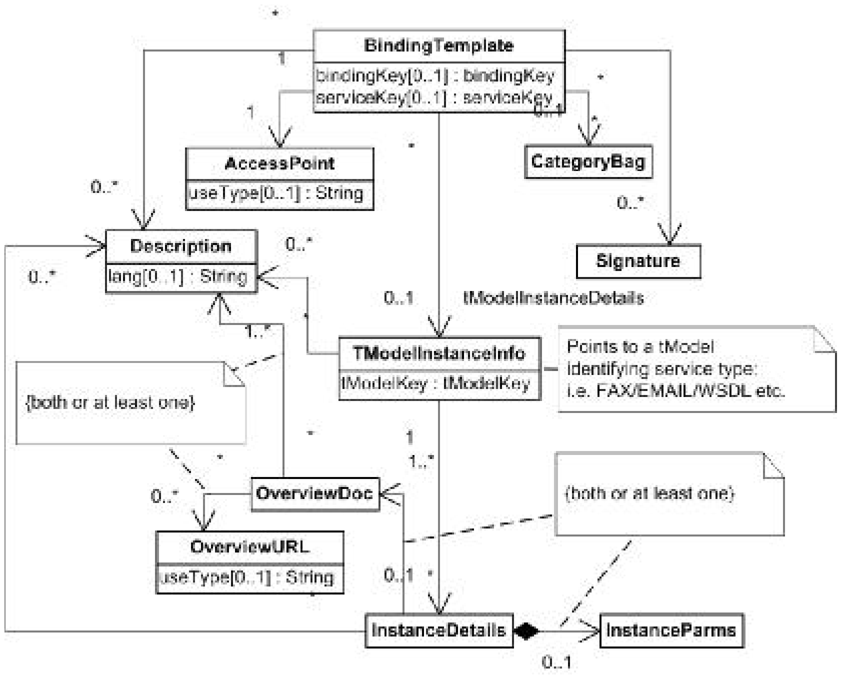

Fig. 3. UDDI-bindingTemplate

BindingTemplate has two optional properties: *bindingKey* and *serviceKey*. *bindingKey* uniquely identifies a *BindingTemplate*. When a BindingTemplate is published within a UDDI registry, the *bindingKey* **must** be omitted if the publisher wants the registry to generate a key. When a *bindingTemplate* is retrieved from a UDDI registry, the *bindingKey* must be present. *serviceKey* uniquely identifies the *BusinessService* that contains the *BindingTemplate*.

BindingTemplate refers to a list that has already been discussed: *Description*, *Signature*, or will be discussed later: *CategoryBag*.

AccessPoint is used to convey the network address suitable for invoking the Web service being described. This is typically a URL but may be an electronic mail address, or even a telephone number. *AccessPoint* has an `useType` optional property with the following possible values:

1. *endPoint*: The actual service endpoint, i.e. the network address at which the Web service can be invoked
2. *BindingTemplate*: *AccessPoint* contains a *bindingKey* that points to a different *BindingTemplate* entry
3. *hostingRedirector*: *AccessPoint* can only be determined by querying another UDDI registry.

4. *wsdlDeployment*: The remotely hosted WSDL document that already contains the necessary binding information, including the actual service endpoint.

A BindingTemplate may refer a *TModelInstanceInfo*. The *tModelKey* required property of TModelInstanceInfo references a UDDI *tModel* that encodes a specification with which the Web service represented by the containing *bindingTemplate* complies.

An instance of *InstanceDetails* may consist of *Description InstanceParms* and *OverviewDoc*. *InstanceParms* is used to encodes settings or parameters related to the proper use of a TModelInstanceInfo, the suggested format by the UDDI standard is a namespace-qualified XML document. At least one *OverviewDoc* or *InstanceParms* **must** be provided within the InstanceDetails.

The *OverviewDoc* used to house URI references (i.e. *OverviewURL*) to remote descriptive information or instructions related to the use of a particular *tModel*. *OverviewDoc* contains a *Description* of the document and the *OverviewURL* of the document.

OverviewURL holds an URL referring to an overview document that covers the way a particular *tModel* is used as a component of an overall Web service description. The content from that URL is stored outside of the registry, typically at the publisher site. It provides an optional property `useType` encodes the type of the document, found at the URL. i.e. "text".

A *tModel* can be viewed as a service vocabulary. The unique key of a *tModel* i.e. *tModelKey* accommodates namespaces. A *tModel* helps the user to find information about the compliance with specifications, concepts. To illustrate let's take the following example: `49-355-690000`, `49-355-690001`, `1-56743-878-x`. There is no way to know what these numbers are. A context or a namespace is needed to establish that `49-355-690000` is a telephone number, `49-355-690001` is FAX number and `1-56743-878-x` is an ISBN number. So there will be three tModels, one describing each of those numbers.

Figure 4 describes the *tModel* information model. The content model of a *tModel* was already discussed. Recall that *CategoryBag* and *IdentifierBag* will be discussed in the next section of the paper.

3.1 Categorization and identification

UDDI supports flexible capabilities for categorization and identification. It offers built-ins and user defined value sets. When information is classified it is important to agree on that. The classifications that are agreed upon are called taxonomies.

As stated also in [2] built-in taxonomies includes NAICS[7] industry taxonomy; UNSPSC[8] project and service taxonomy; and ISO-3166-2 geographic

[7] North American Industry Classification System

[8] United Nations Standard Products and Services Code

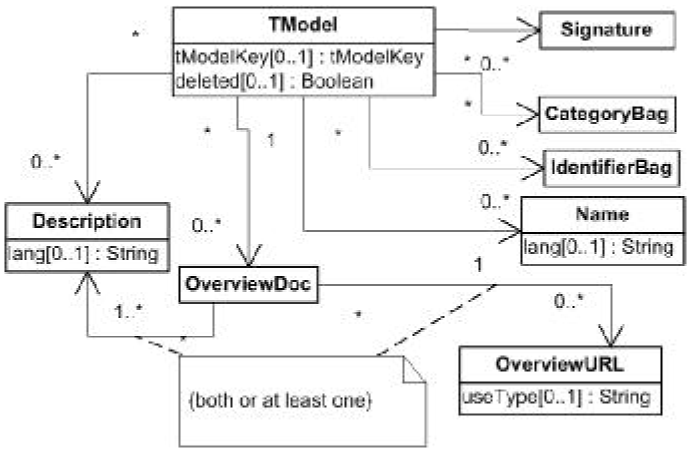

Fig. 4. UDDI-tModel

taxonomy. Use of the built-in taxonomies is "internally" checked by the UDDI registry - therefore attempts to save invalid code are rejected.

There is now specific way to identify a BusinessEntity as belonging to some specific business branch. UDDI provides two ways of identifying business by means of *identifierBag* and *categoryBag*.

Figure 5 depicts the UML metamodel of IdentifierBag. *IdentifierBag* refers

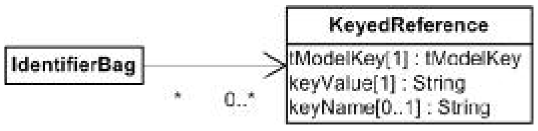

Fig. 5. UDDI-identifierBag

to many *KeyedReference*.

Any *KeyedReference* instance encodes an identifier of a specific identifier system. *KeyedReference* has two required propertiess: *tModelKey* and *keyValue* and an optional one *keyName*. *keyName* is used to provide a descriptive name for the business identification and *keyValue* contains the actual identifier within the system referred by *tModelKey* value. *KeyedReference* might be used to build "search terms" in applications using the registry.

Below is an example of an *identifierBag*. Your Company is doing business in Germany, and for this it has a tax ID number.

```
<identifierBag>
    <keyedReference
        tModelKey="DE-tax-code"
        keyName="taxnumber"
        keyValue="000000" />
        ....
</identifierBag>
```

IdentifierBag instances allows *BusinessEntity* structures to be identified according to different identifier systems described in the registry via *tModelKey* values.

CategoryBag classifies *BusinessEntity* instances according to already published categorization systems. For example, a *BusinessEntity* might contain UNSPSC product and service categorizations that describe its product and service offering, and ISO 3166 geographical regions that describe the geographical area where these products and services are offered.

Figure 6 depicts the UML metamodel of *CategoryBag*.

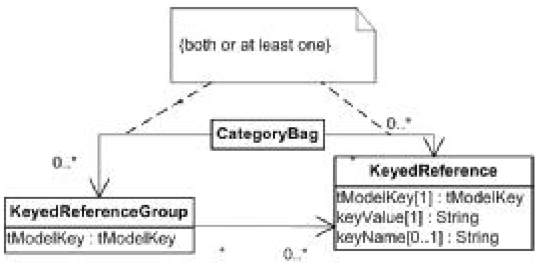

Fig. 6. UDDI-categoryBag

CategoryBag may consist of *KeyedReference* and a *KeyedReferenceGroup*. At least one *KeyedReference* or one *KeyedReferenceGroup* **must** be provided within the *CategoryBag*.

KeyedReferenceGroup refers a *KeyedReference* list that logically belong together, and has a required *tModelKey* property.

For a better understanding see the example below that says that Vodafone offers for example two types of services: Mobile Phone Network communication and the numbers are identified by 3, and Fixed Phone Network communication and the numbers are identified by 4. So both Mobile and Phone service types belong logically to the category of servicetypes. Another example would be that latitude and longitude belong logically to coordinates category.

The *tModelKey* of a *KeyedReferenceGroup* would be the tModelKey of the tModel that encodes the group to which belong all the referenced *KeyedReference*.

```
<categoryBag>
  <keyedReferenceGroup tModelKey="uddi:vodafone.com:servicetypes">
    <keyedReference tModelKey="uddi:vodafone.com:servicetypes:mobile"
          keyName="Mobile"
          keyValue="3" />
    <keyedReference tModelKey="uddi:vodafone.com:servicetypes:phone"
          keyName="Phone"
```

```
        keyValue="4" />
    </keyedReferenceGroup>
  </categoryBag>
```

4 UDDI programmatically

This section presents a comparison (see Table 1) of three open source tools implementing UDDI specification. The comparison criteria are:

1. *License.* The licence under the product is published.
2. *UDDI version support.*
3. *Development Level.* The stage of the product according with the official home page.
4. *Implementation Platform* i.e. java, C# .NET ...
5. *Minimal Deployment Resources* i.e. the minimal set of software packages necessary for the product deployment.
6. *Actual stage* i.e. If the application is still under development
7. *Feature set* Features that are implemented
8. *High level API* If is provided user friendly interface for publish and inquiry.
9. *Registration of WSDL descriptions* If is possible to register WSDL descriptions.
10. *Authentication support* i.e. Signature or Token Authentication.

We propose as production solution the JUDDI implementation. It is the most widely used and it also very easy to install. Our proposal includes JBoss AS[9] and of course JUDDI. JUDDI need also data base support. You can use the JBoss HSQLDB support, by using the creation scripts that come with JUDDI distribution. But you can also use other data base support. The distribution comes with creation SQL scripts for a list of data base systems.

Be aware that you will also need a data source file deployed into JBoss. This is not specified into JUDDI specification. An example of a data source file is bellow:

```
<datasources>
  <local-tx-datasource>
    <jndi-name>juddiDB</jndi-name>
    <connection-url>jdbc:hsqldb:${jboss.server.data.dir}${/}hypersonic${/}
      localDB</connection-url>
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
    <check-valid-connection-sql>select count(*) from PUBLISHER</check-valid-connection-sql>
    <max-pool-size>30</max-pool-size>
    <min-pool-size>5</min-pool-size>
  </local-tx-datasource>
</datasources>
```

For more information about installation you can browse JBoss ESB[10] web page and Apache JUDDI page. Grimoires has very good documentation pages

[9] http://labs.jboss.com/jbossas/downloads

[10] JBoss ESB - JBoss SOA infrastructure: http://labs.jboss.com/jbossesb/

Table 1. UDDI Registries Open Source Implementations

	JUDDI	**Grimoires**	**OpenUDDI Server**
General			
License	Apache License 2.0	Modified BSD licence	Apache License 2.0
Home site	`http://ws.apache.org/juddi/index.html`	`http://grimoires.org`	`http://softwareborsen.dk/projekter/softwarecenter/serviceorienteret-infrastruktur/openuddi-server`
UDDI version support (UDDI vX SOAP API)	v2	v2	v3
Development Level	Production	Production	Beta
Implementation Platform	Java (as a Web Service)	Java (as a Web Service)	Java (as a Web Service) based on Novell Nsure UDDI Server
Minimal Deployment Resources	Tomcat (includes SOAP stack) + SQL database	Tomcat+Axis+SQL database	Tomcat (includes Axis SOAP stack) + SQL database
Actual stage	Inactive	Inactive	Active
Feature set	UDDI v2 feature set	UDDI v2 feature set + WSDL 1.1 +Metadata	UDDI v3 not fully implemented
High level API	UDDI4J-like API	Yes, WSDL API, Metadata API	UDDI4J-like API
Semantic related features			
Registration of WSDL descriptions	No	Yes(WSDL 1.1)	No
Security Features			
Authentication support	authentication tokens via username/password	Signature based authentication	authentication tokens via username/password

that you can use, but you will also have to install and configure Apache Axis, since Grimoires does not have a SOAP stack as JUDDI. OpenUDDI Server is a UDDI v3 implementation but is beta version and also suffers from lack of documentation, so our proposal for an UDDI with no problem start is JUDDI.

5 Conclusions

For those who want to offer Web Services, it is good to have UDDI Support. Using UDDI you offer a way to find a Web Services without having to explicitly publish each Web Service that is offered. Also Web Services are about to change. Each change will broke the business process. Having UDDI these kind of problems are resolved because if a broke occurred in the process the application can always go back to the registry and recover. Also because UDDI was developed especially for the Web Services it is most suited to be used.

References

[1] **G. Alonso, F. Casati, H. Kuno, V. Machiraju**:- Web Services Concepts, Architectures and Applications, Springer, 2004, ISBN: 3-540-44008-9

[2] **T. Bellwood, P. Brittenham, A. Hately, S. Field**:- Publication and Discovery of Web Services, TR IBM Library, 2003, `ftp://ftp.software.ibm.com/software/websphere/webservices/publicationdiscoverywebservices.pdf`

[3] **T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan**:- Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation 16 August 2006, `http://www.w3.org/TR/xml11/`

[4] **D. Carlson**:- Modeling the UDDI Schema with UML. `http://xml.coverpages.org/carlson-ModelingUDDI.pdf`, 2001

[5] **R. Chinnici, J.-J. Moreau, A. Ryman, S. Weerawarana** (Eds.):- Web Services Description Language (WSDL), Version 2.0, Part 1: Core Language, W3C Recommendation, 26 June 2007, `http://www.w3.org/TR/wsdl20/`

[6] **F. Najmi, C. Mattocks**:- ebXML Registry overview, OASIS ebXML Registry 3.0 Webinar 2005 Slides. `http://ebxmlrr.sourceforge.net/presentations/ebXML%20Registry%20webinar5.pdf`

[7] **R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee**:- Hypertext Transfer Protocol – HTTP/1.1, 1999, `http://www.ietf.org/rfc/rfc2616.txt`

[8] **S. Graham, S. Simeonov, T. Boubez, D. Davis, G. Daniels, Y. Nakamura, R. Neyama**:- Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI. Sams Publishing, 2001, ISBN:0-672-32181-5

[9] **M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, Y. Lafon**:- SOAP Version 1.2, Part 1: Messaging Framework (Second Edition), W3C Recommendation, 27 April 2007, `http://www.w3.org/TR/soap12-part1/`

[10] **E. A. Marks, M. Bell**:- Service-Oriented Architecture: A Planning and Implementation Guide for Business and Technology, John Wiley and Sons, 2006, ISBN: 0470036141

[11] **P. Mockapetris**:- Domain Names - Implementation and Specification, 1987, `http://www.ietf.org/rfc/rfc1035.txt`

[12] ***:- OASIS, Web Services Business Process Execution Language Version 2.0, OASIS Standard 11 April 2007, `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`

[13] ***:- OASIS, UDDI Technical White Paper, 2000, `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`

[14] ***:- OASIS, UDDI Executive Overview: Enabling Service-Oriented Architecture. UDDI.org white papers, 2004, `http://uddi.xml.org/files/uddi-exec-wp.pdf`

[15] ***:- OASIS, UDDI Version 3.0.2, 2004 `http://uddi.org/pubs/uddi-v3.0.2-20041019.htm`

[16] ***:- OASIS, Web Services Security, 2006, `http://docs.oasis-open.org/wss/v1.1/`

[17] **N. Ragouzis, J. Hughes, R. Philpott, E. Maler, P. Madsen, T. Scavo**:- Security Assertion Markup Language (SAML) V2.0, Technical Overview, OASIS Draft, February 2007, `http://www.oasis-open.org/committees/download.php/22553/sstc-saml-tech-overview-2%200-draft-13.pdf`