Chapter 4

UDDI based Web Service Discovery Mechanism

4.1 Why UDDI based mechanism?

Universal Description Discovery & Integration (UDDI) registry provides a centralized approach in service-oriented architecture and the focus of Universal Description Discovery & Integration (UDDI) is the definition of a set of services supporting the description and discovery of (1) businesses, organizations, and other web services providers, (2) the Web services they make available, and (3) the technical interfaces which may be used to access those services. Based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP, UDDI provides an interoperable, foundational infrastructure for a web services-based software environment for both publicly available services and services only exposed internally within an organization. [2]. In the same centralized approach, if we compare UDDI with other registry ebXML, it is observed that ebXML as always been designed for the management of large amounts of complex information using standardized and extensible metadata; Also it has extensible data-model and is more like repository rather than only registry, whereas in UDDI, protocols and information model is focused and specific and is only registry not repository. One more research influencing factor is that ebXML is a younger technology than UDDI and UDDI seems to have a larger user base than ebXML.

UDDI is a vendor-sponsored registry standard which has emerged taken a dominant role in standardization process of registries. UDDI was the brain child of Ariba, IBM, Intel, Microsoft and SAP. In 2002, the OASIS standards group took over UDDI
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

from UDDI.org. BEA, Cincom, CA, E2Open, Entrust, Fujitsu, HP, IBM, Intel, IONA, Microsoft, Novell, Oracle, SAP, Sun Microsystems and hundreds of other companies have endorsed it. Moreover, no fees or licenses are required to use this technology. It has the following benefits as:

- It is a standardized, transparent mechanism for describing services.
- It describes simple methods for invoking the service.
- It specifies an accessible central registry of services.

The reason that UDDI is acceptable to all the vendors is that it is built on the same SOAP standards that ordinary web services use. This means that a registry can be written in and accessed by any computer language running on any hardware platform running any operating system. Every vendor is able to create tools to interact with these registries.

4.2 The Approach

UDDI contains a number of specifications that describe how a registry stores data and how it can be accessed. Four main specifications of UDDI are as follows:

- The data structure specification describes what kind of data is stored in UDDI. The UDDI data structure is based on XML and described through an XML Schema. This schema is actually published as a separate document available from the UDDI web site.

- The programmer’s API specification contains how an UDDI registry can be accessed. There are two types of API, publishing functions and inquiry functions. The publishing functions are used to create and update existing entries in the registry. The inquiry functions are all read-only and allow the existing entries to be queried programmatically. The API is programming language-independent. This is accomplished by describing the request and response data in terms of an XML document. These request and response structure map the actual content of the registry quite closely. The existing registries offer access via SOAP over HTTP which means that request and response XML data is wrapped into SOAP envelopes. The enquiry functions are available over HTTP, whereas the publishing functions are accessible via HTTPS and require a user ID and
password to be sent along with each request. Each UDDI registry provided ways for a user to obtain a valid user ID and password.

- The replication specification contains descriptions of how registries replicate information among themselves. This information is only needed for those who want to implement their own registry and integrate it with other existing registries.

- Finally, there is the Operator’s specification, which is only for those who are implementing or running a UDDI registry. It defines policies for security and for data management. This specification does not make it compulsory for an operator to follow a certain policy, instead it requires that each operator publish what policies are enabled and enforced.

The following table shows the type of elements that exists in the registry together with some of the API functions that are defined for them.

Table 4.1 UDDI elements and API functions for them

<table>
<thead>
<tr>
<th>Element Type</th>
<th>Find Method</th>
<th>Get Method</th>
<th>Save Method</th>
<th>Delete Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;businessEntity&gt;</td>
<td>find_business()</td>
<td>get_businessDetail()</td>
<td>save_business()</td>
<td>delete_business()</td>
</tr>
<tr>
<td>&lt;businessService&gt;</td>
<td>find_service()</td>
<td>get_serviceDetail()</td>
<td>save_service()</td>
<td>delete_service()</td>
</tr>
<tr>
<td>&lt;bindingTemplate&gt;</td>
<td>find_binding()</td>
<td>get_bindingDetail()</td>
<td>save_binding()</td>
<td>delete_binding()</td>
</tr>
<tr>
<td>&lt;tModel&gt;</td>
<td>find_tModel()</td>
<td>get_tmodelDetail()</td>
<td>save_tModel()</td>
<td>delete_tModel()</td>
</tr>
</tbody>
</table>

Existing web services architecture comprises three roles: Web service Provider, Web service Consumer and Universal Description, Discovery and Integration (UDDI) registry as shown in Figure 4.2.

Figure 4.1: Existing Web service Architecture
The Web service Provider publishes a description of the service in the UDDI registry, as well as details of how to use the service. UDDI registries use an XML-based language, Web Services Description Language (WSDL), to describe a Web service, the location of the service and methods the service exposes. The Web service Consumer uses the UDDI to find an appropriate service that meets its requirements using the information provided with the services, chooses one service manually, and invokes the service. The web service publishing, discovery and binding process is generally done by consumers at design time.

4.3 UDDI based mechanism : Reputation-Enhanced Web Service Discovery with QoS

The existing UDDI registries only support web services discovery based on the functionality of services. As the customers are interested in not only the functionalities of web services, but also their nonfunctional characteristics i.e. quality of service (QoS), that may have huge impact on the result of web service discovery. If there are multiple web services providing the same functionality in UDDI registries, the QoS requirement can be used as a finer search constraint. Ziqiang Xu et al [107] proposed a model of reputation-enhanced web services discovery with QoS to help consumers find the services that best meet their requirements.

![Model of Reputation-enhanced Web Services Discovery with QoS](image-url)

Figure 4.2 : Model of Reputation-enhanced Web Services Discovery with QoS
In this model, the UDDI registry is enhanced with QoS information, and two new roles, discovery agent and reputation manager, are added in our model as shown in Figure 4.2. The UDDI registry stores QoS information of services by using tModels. The discovery agent acts as a broker between a service consumer, a UDDI registry and a reputation manager to discover the web services that satisfy the consumer’s functional, QoS and reputation requirements. The reputation manager collects and processes service ratings from consumers, and provides service reputation scores when requested by the discovery agent.

4.3.1 Publishing QoS Information

When a Web Service Provider publishes a web service, it creates and registers a tModel within a UDDI registry. The QoS information of the Web service is represented in the tModel, which is referenced in the binding template that represents the web service deployment. Each QoS attribute is represented by a keyedReference in the generated tModel. The name of a QoS attribute is specified by the keyValue, and its value is specified by the keyValue. Instead of different units, default units are used for the QoS attributes values in the tModel. For example, the default unit used for price is CAN$ per transaction, for response time is second, for availability is percentage, and for throughput is transaction per second. For example a company publishes its Stock Quote service in a UDDI registry with the following QoS information:

- Service price: CAN $0.01 per transaction
- Average response time: 0.05 second
- Availability: 99.99%
- Throughput: 500 transaction/second

The company creates and registers a tModel that contains the QoS information for this service before it publishes the service with the UDDI registry. With QoS information of web services stored in tModels in a UDDI registry, service consumers can find the services that match their QoS requirements by querying the UDDI registry. The details of this process are discussed in the section 4.3.3.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

<table>
<thead>
<tr>
<th>tModel tModelKey=&quot;somecompany.com:StockQuoteService: PrimaryBinding:QoSInformation&quot;</th>
<th>QoS Information for Stock Quote Service</th>
<th>overviewDoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>overviewURL</td>
<td>http://&lt;URL describing schema of QoS attributes&gt;</td>
<td>overviewURL</td>
</tr>
<tr>
<td>categoryBag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyedReference</td>
<td>tModelKey=&quot;uddi:uddi.org:QoS:Price&quot;</td>
<td>keyName=&quot;Price Per Transaction&quot;</td>
</tr>
<tr>
<td>=keyValue=&quot;0.01&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyedReference</td>
<td>tModelKey=&quot;uddi:uddi.org:QoS:ResponseTime&quot;</td>
<td>keyName=&quot;Average ResponseTime&quot;</td>
</tr>
<tr>
<td>=keyValue=&quot;0.05&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyedReference</td>
<td>tModelKey=&quot;uddi:uddi.org:QoS:Availability&quot;</td>
<td>keyName=&quot;Availability&quot;</td>
</tr>
<tr>
<td>=keyValue=&quot;99.99&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyedReference</td>
<td>tModelKey=&quot;uddi:uddi.org:QoS:Throughput&quot;</td>
<td>keyName=&quot;Throughput&quot;</td>
</tr>
<tr>
<td>=keyValue=&quot;500&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/categoryBag&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;/tModel&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.3 The $tModel$ with the QoS information

4.3.2 Updating QoS Information

Web service Providers need to update the QoS information of their services in the UDDI registry frequently to ensure that the QoS information is accurate and up to date. Only a service provider that publishes a service and its QoS information in a UDDI registry can modify and update the QoS information. A service provider searches the UDDI registry to find the $tModel$ that contains QoS information for the service it published before, updates the QoS information in the $tModel$, and then saves the $tModel$ with the same $tModelKey$ assigned previously.
4.3.3 Discovering web service through Discovery Agent and Reputation Manager

Service Consumer’s request is received by a discovery agent first, it finds the services in the registry that match their requirements and then returns the response to the consumers. A request for web service discovery consists of functional, QoS, and reputation requirement of the web service. The format of request for web service discovery specifying the details of how to specify functional, QoS and reputation requirements is given in Figure 4.4. These types of SOAP messages for discovery requests are not generated manually by the service consumer, instead developers specify QoS and reputation requirements in a Java program that automatically generates required SOAP messages sent to the discovery agent. Customers can specify the following request parameters in the discovery request:

- Maximum number of services to be returned by the discovery agent
- Functional requirements: keywords in service name and description
- Service price: the maximum service price a customer is willing to pay
- Service performance and other QoS requirements such as response time, throughput, and availability.
- Dominant QoS attribute.
- Service reputation requirements.
- Weights for the QoS and reputation requirements
The dominant QoS attribute is the one that consumers consider as the most important and is used in the calculation of the QoS score for each service candidate in the service matching process. A consumer can specify QoS requirements only or both QoS and reputation requirements in the request. The weights for QoS and reputation requirements indicate their importance and they range from zero to one, where zero means no requirement for QoS or reputation while one means it is the only requirement on QoS or reputation. The sum of the weights must to one. Instead of setting the preference to each QoS attribute, a dominant QoS attribute is to be set having highest preference as it is easier for customers to select the most important QoS attribute than to specify separate priority for each of the QoS attributes. This will greatly simplify the calculation of QoS scores in the service ranking process.

As the discovery agent receives the request for service discovery, it finds services in
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

UDDI registry that match the functional requirements specified as keywords in service name and also obtain QoS information of each stored in corresponding tModels. Then it matches the published QoS information with the QoS requirements specified in the request, finds the matched services, ranks the matches by QoS scores and/or reputation scores and returns the result to the customer. The QoS scores of services is calculated by the formula given as below:

\[
QoSScore_i = \begin{cases} 
\frac{DominantQoS_i}{BestDominantQoS} & \text{if dominant QoS attribute is monotonically increasing} \\
\frac{BestDominantQoS}{DominantQoS_i} & \text{if dominant QoS attribute is monotonically decreasing}
\end{cases}
\]

where \(QoSScore_i\) is the QoS score of \(i^{th}\) service,

\(i\) is the position of the service in the list of matched services,

\(DominantQoS_i\) is the value of the dominant QoS attribute of service \(i\),

\(BestDominantQoS\) is the highest/lowest value of the dominant QoS attribute of the matched services when the dominant attribute is monotonically increasing/decreasing.

The adjusted Reputation scores of services is calculated by the formula given as below:

\[
AdjRepuScore_i = \frac{RepuScore_i}{h}
\]

where \(AdjRepuScore_i\) is the adjusted reputation score of service \(i\).
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

\[ i \] is the position of the service in the list of matched services, 
\[ \text{RepuScore}_i \] is the original reputation score of service \( i \), 
\[ h \] is the highest original reputation scores of the matched services.

The final overall scores of services required for ranking is given by the equation below:

\[
\text{OverallScore}_i = \text{QoS} \times \text{QoSWeight} + \text{AdjRepuScore}_i \times \text{RepuWeight}
\]

where \( \text{OverallScore}_i \) is the overall score of service \( i \),
\( i \) is the position of the service in the list of matched services,
\( \text{QoS} \) is the QoS score of service \( i \),
\( \text{QoSWeight} \) is the weight of QoS requirement specified by service consumers,
\( \text{AdjRepuScore}_i \) is the adjusted reputation score of service \( i \),
\( \text{RepuWeight} \) is the weight of reputation requirement specified by consumers.

A reputation manager in this service discovery model is based on the models proposed by Majithia et al. and Wishart et al. A QoS reputation score is calculated based on feedback by service consumers. Service Reputation Manager collects the data based on feedback from the service consumer, processes it and updates the reputation score. After using the web service, the service consumer rates it on a scale of 1 to 10 where, 10 means extreme satisfaction, 5 means average satisfaction and 1 means extreme dissatisfaction. Awarding bonus points to the consumers for their feedback will encourage them to provide valid ratings of the used services which can be used in service discovery to reduce the cost of the discovery.

The service rating storage is based on SuperstringRep, a protocol proposed by Wishart et al. The ratings of services by consumers are stored in the reputation manager’s local database. Each rating record consists of service ID, consumer ID, rating value and a timestamp fields. The service key in the UDDI registry of the service is referred as the service ID, and the service consumer’s IP address is used as the consumer ID.
Following table shows ratings records for services.

**Table 4.2 Ratings record for services along with Timestamp**

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Consumer ID</th>
<th>Rating</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6</td>
<td>117.239.43.139</td>
<td>7</td>
<td>2011-09-03 10:15:34</td>
</tr>
<tr>
<td>8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6</td>
<td>172.50.43.30</td>
<td>8</td>
<td>2011-09-12 11:25:07</td>
</tr>
<tr>
<td>8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6</td>
<td>117.195.125.201</td>
<td>5</td>
<td>2011-09-11 19:20:12</td>
</tr>
<tr>
<td>53164900-f0b0-11d5-bca4-002035223h97</td>
<td>116.23.56.23</td>
<td>7</td>
<td>2011-09-21 12:15:02</td>
</tr>
<tr>
<td>8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6</td>
<td>117.239.43.137</td>
<td>5</td>
<td>2011-09-21 09:20:22</td>
</tr>
<tr>
<td>53164900-f0b0-11d5-bca4-002035223h97</td>
<td>172.50.43.37</td>
<td>6</td>
<td>2011-10-29 09:20:22</td>
</tr>
<tr>
<td>b6cb1cf0-3aaf-11d5-80dc-002035245u62</td>
<td>117.239.43.131</td>
<td>7</td>
<td>2011-09-22 19:10:56</td>
</tr>
<tr>
<td>53164900-f0b0-11d5-bca4-002035223h97</td>
<td>117.239.43.134</td>
<td>6</td>
<td>2011-10-11 12:23:43</td>
</tr>
<tr>
<td>53164900-f0b0-11d5-bca4-002035223h97</td>
<td>116.23.56.26</td>
<td>8</td>
<td>2011-08-12 09:20:22</td>
</tr>
</tbody>
</table>

There are three services in the above table with Service ID “8221cb6e-e8c9-4fe3-9ea8-3c99b1fd2fk6”, “53164900-f0b0-11d5-bca4-002035223h97” and “b6cb1cf0-3aaf-11d5-80dc-002035245u62”, respectively. Each of the three services receives some ratings from consumers. Only one rating for a service per consumer is stored in the table. New ratings from the consumer for the same service replace older ratings. The timestamp is used to determine the latest rating for a particular service rating.

Reputation score for a web service is calculated on the basis of the work by Majithia et al. and the work by Wishart et al. Majithia et al. propose a method to calculate the reputation score as weighted sum of ratings for a service, where a coefficient is the weight attached to a particular context. Wishart et al. propose an aging function that applies a factor to each of the ratings regarding a service. In this model, the reputation score ($U$) of a service is calculated as the weighted average of all ratings the service received from customers, where an inclusion factor is the weight attached to each of the ratings for the service.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

\[ U = \sum_{i=1}^{N} S_i \gamma_i \]

where \( U \) is the reputation score for a service,
\( S_i \) is the \( i \)th service rating,
\( \gamma_i \) is the aging factor for \( i \)th service rating,
\( \lambda \) is the inclusion factor, \( 0 < \lambda < 1 \),
\( d_i \) is the number of the days between the two times \( t_c \) and \( t_i \):
\( t_c \) is the current time when the reputation score is computed,
\( t_i \) is the time of the \( i \)th rating for the service.

The inclusion factor \( \lambda \) is used to adjust the responsiveness of the reputation score to the changes in service activity. When \( \lambda \) is set to 0, all ratings, except the ones that are provided by consumers on the same day as the reputation score is computed, have a weight of 0 and are not be included in the computation. When \( \lambda \) is set to 1, all ratings have equal weight of 1 and used in the computation. A smaller \( \lambda \) means only recent ratings are included and a larger \( \lambda \) means more ratings are included.

Here, service matching, ranking and selection algorithm is based on the matching algorithm proposed by Maximilien and Singh. Simplified flowchart of improved algorithm is given below:
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Figure 4.5 Flowchart for Matching, Ranking and Selecting service
Example:

**SOAP Request for Web Service Discovery**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
  <body>
    <find_service generic="1.0" xmlns="urn:uddi-org:api">
      <functionalRequirement>
        Stock Quote
      </functionalRequirement>
      <qualityRequirement weight=0.4>
        <dominantQoS>availability</dominantQoS>
        <price>0.01</price>
        <responseTime>0.1</responseTime>
        <throughput>400</throughput>
        <availability>99.9</availability>
      </qualityRequirement>
      <reputationRequirement weight=0.6>
        <reputation>8</reputation>
      </reputationRequirement>
      <maxNumberService>1</maxNumberService>
    </find_service>
  </body>
</envelope>
```

**Figure 4.6 Service Discovery Request SOAP Message**

**SOAP Response for Web Service Discovery**

On the request from service consumer, the discovery agent finds two services that match the requirements in the request, ranks the services using their QoS scores and reputation scores, and returns one service to the consumer since the request specifies the maximum number of services to be returned is 1. A SOAP message of service discovery response is shown in following Figure.
Comparative study of mechanisms for discovering the most appropriate web service and proposing an efficient web service discovery mechanism

Summarizing above discussion on Reputation-Enhanced Web Services Discovery with QoS, it can be concluded that a reputation management system provides a mechanism to help a service discovery agent to improve the possibility to find services those match a consumer’s functional, QoS and reputation requirement also provide consistently stable QoS performance. The problem of the accountability of those who provide ratings to the services still remain unsolved. In real world, ratings of a service could be provided by its competitors and trade partners or even the providers itself. Hence, assuming the service ratings are all trustworthy, service consumers could be easily misguided in case of service selection. A third party standard for ensuring the quality of service is needed.

Figure 4.7 Service Discovery Response SOAP Message
4.4 UDDI based mechanism: Web service QoS-Certifier based Web Service Discovery

As we have discussed in earlier section, there is a need of some certifier agency who will certify the claims of quality of services or their ratings before publishing it to the UDDI registry to make it trustworthy. Shuping Ran [83] proposed framework that can serve to the service consumers needing quality of service assurance. There are four roles in this proposed model: Web service supplier, Web service consumer, Web service QoS certifier, and the new UDDI registry. As before, the Web service provider offers Web service by publishing the service into the registry, the Web service consumer needs the Web service offered by the provider, the new UDDI registry is a repository of registered Web services with lookup facilities and the new certifier’s role is to verify service provider’s QoS claims for publishing. The proposed new registry differs from the current UDDI model by having information about the functional description of the web service as well as its associated quality of service registered in the repository. Web service can be discovered by functional description of the desired web service, with the required quality of service attributes as requirement criteria. The new role in this model is the web service QoS certifier that does not exist in the original UDDI model. The certifier verifies the claims of quality of service for a web service before its registration.

Figure 4.8 Web services registration and discovery model with QoS Certifier
As shown in above figure, a web service provider supply service description along with its functional aspect as well as quality of service information related to the proposed web service. The claimed quality of service needs to be certified and registered in the repository. The web service provider first communicates its QoS claim to the web service QoS certifier before publishing in the UDDI registry. The certifier verifies the claims and either certifies or down grade the claim. The report is sent back to the service provider with certification identification information. This information is also registered in the certifier's repository identified by a certification Id. The certifier provides a set of web services for any interested parties to access its repository about QoS claims for verification purposes. After the QoS certification been issued by the certifier, the supplier then registers with the UDDI registry with both functional description of the service and its associated certified quality of service information. The UDDI registry cross checks it with the certifier to ensure the existence of the certification. On successful checking, the registry then registers the service in its repository. In this framework, a new role is introduced– QoS Certifier who verifies the QoS claims from the web service suppliers and its role is very similar to rating agencies in other domains such as the financial sector, service industry etc, but the details regarding its implementation are unexplored yet.

4.5 jUDDI Registry working

jUDDI (pronounced "Judy") is an open source Java implementation of the Universal Description, Discovery, and Integration (UDDI v3) specification for (Web) Services. jUDDI is a pure Java web application and as such can be deployed to any application server or servlet engine that supports version 2.1 or later of the servlet API. jUDDI also requires an external datastore in which to persist the registry data it manages. Typically this is a relational database management system such as MySQL, Oracle or DB2. Support for several open source and commercial database products are included.

jUDDI consist of a core request processor that unmarshalls incoming UDDI requests, invoking the appropriate UDDI function and marshalling UDDI responses (marshalling and unmarshalling is the process of converting XML data to/from Java objects). To invoke a UDDI function, jUDDI employs the services of three configurable sub-
Comparative study of mechanisms for discovering the most appropriate web service and proposing an
efficient web service discovery mechanism

components or modules that handle persistence (the DataStore), authentication (the
Authenticator) and the generation of UUID's (the UUIDGen). jUDDI is bundled and pre-
configured to use default implementations of each of these modules to help the registry
up and running quickly. These sub-components are described briefly as below.
Persistence (jUDDI DataStore)
jUDDI needs a place to store it's registry data so it is understandable that jUDDI is pre-
configured to use JDBC and any one of several different DBMSs to do this. The process
of setting this up is simple. Start by creating a new jUDDI database using the instructions
for the preferred DBMS, in my case I have used MySQL.
To complete the DataStore set up, it is required to configure a JNDI Datasource with a
name of 'jdbc/juddiDB' in the application server, in my case I am using Apache Tomcat
as a application server for deployment. Datasource setup varies on an product-by-product
basis.
Authentication (jUDDI Authenticator)
Authenticating a jUDDI publisher is a two-step process. The first step confirms that the
ID/password combination provided by the user in a get_authToken request is valid. The
default Authenticator implementation simply approves any authentication attempt. It is
expected that a typical jUDDI deployment will use an external authentication mechanism.
The second step confirms that the publisher has been defined to jUDDI. A publisher is
said to be defined when a row identifying the publisher exists in the PUBLISHER table
of the jUDDI datastore.
The PUBLISHER table consists of several columns but only four of them are required
and they are defined as follows:

Table 4.3 Publisher table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUBLISHER_ID</td>
<td>The user ID the publisher uses when authenticating.</td>
</tr>
<tr>
<td>PUBLISHER_NAME</td>
<td>The publisher’s name.</td>
</tr>
<tr>
<td>ADMIN</td>
<td>Indicate if the publisher has administrative privileges. Valid</td>
</tr>
<tr>
<td></td>
<td>values for this column are 'true' or 'false'.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>Indicate if the publishers account is enabled and eligible for use.</td>
</tr>
</tbody>
</table>
The jUDDI web application will be extended to facilitate the Publisher creation process. The value of the ADMIN column in the PUBLISHER table above will be used to determine who has the privilege to create new jUDDI publishers.

**UUID Generation (jUDDI UUIDGen)**

The UDDI specification indicates that each Business, Service, Binding and TModel is to be uniquely identified by a Universally Unique ID (UUID). Additionally, jUDDI also uses the UUID generator to create AuthTokens.