UDDI and IESR

A report for the Joint Information Systems Committee-funded IESR project

Document details

Authors:	Emma Tonkin
Date:	August, 2005
Version:	1
File Name:	
Level of QA Required (1-3):	
Level of QA Achieved (Y/N):	
Notes:	

UDDI and IESR
Emma Tonkin, UKOLN

1. Introduction

UDDI stands for Universal Description, Discovery and Integration, a SOAP-based application initially developed by UDDI.org for publishing Web Services listings in a UDDI Business Registry (UBR), and for seeking out service listings by name, description, and functionality. Over time, the standard has developed into a new role as manager for services internal to a given business. Version 3 was ratified as an OASIS[1] standard in early 2005, but it is no longer clear how widespread the use of the standard is; industry figures (SAP, IBM...) support the standard, but its complexity has led to a low uptake elsewhere.

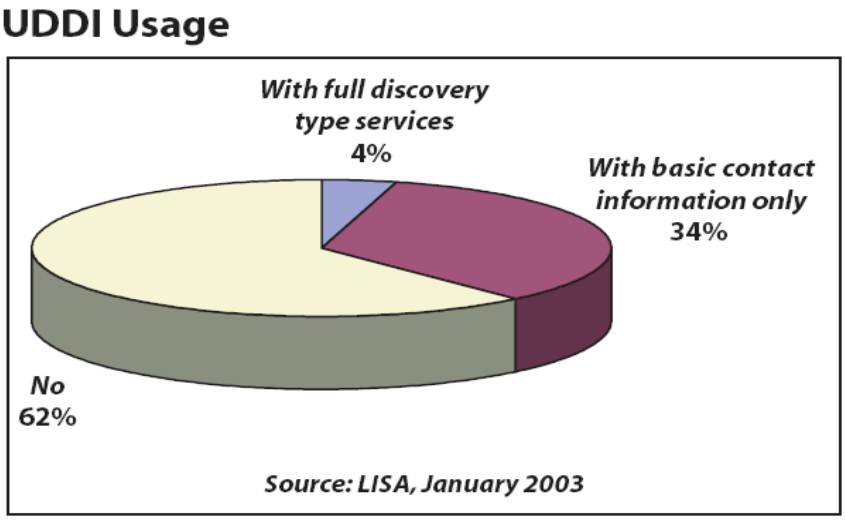


Fig.1: Planned UDDI usage

This complexity has also led to the curious fact that only a few percent of those intending to make use of it at all were intending to use its full functionality [see Fig. 1]. UDDI usage has diversified considerably from that originally envisaged by its creators – it was originally described as to be used in a manner fully analogous to the "Yellow Pages" telephone directory, as a vendor or service discovery index. This gulf between expected and actual usage is partly due to the inevitable confusion between two separate entities – the UDDI standard, and the implemented version, the centralised UDDI Web service-centric business registry.

As a standard, UDDI covers the implementation of business services registries in a business-centric manner. Its heritage as the Web Services registry standard still remains, which is perhaps the origin of the most frequent criticisms against the standard, eg:

- Inflexible and purpose-specific
- Overcomplicated (often directed against the ensemble of Web Service standards; as simpler standards such as REST[2] [Fielding, 2000] become increasingly common, the complexity inherent to the Web Service standards suite attracts this criticism)
- Its data model and discovery methods are optimised for web services
- Difficult metadata model to understand (eg. tModels)

[1]The Organization for the Advancement of Structured Information Standards (OASIS) is a global consortium that drives the development of e-business and web service standards.
[2] REST, Representational State Transfer, is an architectural style for developing networked computer software and accompanying high-level protocols.

However, proponents of the UDDI model point to the following advantages:

- Successfully optimised for web services
- Built specifically to serve the needs of businesses with a service-oriented architecture and artefacts to expose
- A data model sufficiently well constrained to promote standard/compliant usage, and to leave a minimum of developers scratching their heads over the details
- A rich and extensible metadata framework, via tModel and categorisation schemes
- Permits subscription and notification
- Constantly evolving to fit the evolving needs of the various users

It is reasonable to assume that the originally intended usage, the monolithic Web Services Yellow Pages, is essentially dead. The concept of a small number of centrally-controlled root servers does not appear to have caught on, principally for business reasons, with businesses preferring to control their own repositories. In acknowledging this, it is tempting to speculate that UDDI would be well-served to take on either the characteristics of a peer-to-peer system, or of a propagated/caching system such as DNS, a centralised system that nonetheless permits businesses control over their own server/subdomains. As the specification evolves, greater flexibility in the choice of network model becomes available.

The UDDI Business Registry, the Yellow Pages model discussed earlier, worked substantially as follows:

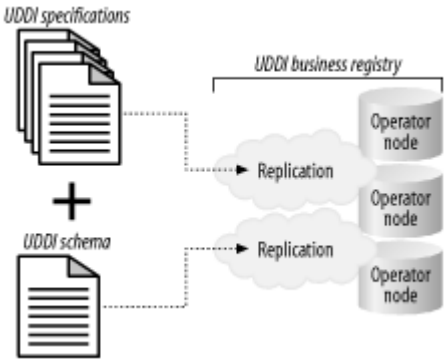


Fig.2: UDDI Business Registry

The UDDI Business Registry, also referred to as 'the public cloud' represents a set of UDDI repositories set up as peers ('nodes' in UDDI parlance) that replicate stored data within that 'cloud' so that each contains the same data. According to the documentation (UDDI Consortium, 2002),

> "A goal of UDDI replication is to ensure that all nodes see all the changes that have originated at individual Operator nodes. An additional goal is that registry inquiries made at any Operator node within the UDDI Service yield results consistent to those made at any other Operator node within the UDDI Service. The response should be complete and sent to the caller as quickly as possible. This consistency is defined as a response comprised of the same businessEntities, businessServices, tModels, bindingTemplates, and publisherAssertions, sorted the same way. The consistency of the results is subject to any replication latencies."

The UDDI standard allows for relatively complex arrangements, such as private nodes run by organisations who do not wish to expose their services publicly, and 'private clouds' replicating within a single business, with no interaction with the public cloud. UDDI has also been used, adapted or proposed for a number of relatively exotic architectures, such as directly peer-to-peer

service discovery for P2P networks (Govoni, D., 2002), or as a basis for grid computing architectures (Zhang et al, 2002).

The UDDI replication feature is designed to ensure the requirements of registries as an entry point – high throughput, low response times, high availability and access to accurate data. It makes use of secure connections, and can be scheduled according to the system administrators' needs.

A public cloud avoids the necessity for a single central registry, meaning that no one network interface causes a bottleneck or point of failure. Nonetheless, the scalability and reliability of replication strategies in various network setup scenarios is, of course, one question that springs to mind, which can only be answered by examining the replication models available in detail. One such study was carried out by Sun, Lin and Kemma (2004). However, less frequently discussed is the possibility of low-availability peers, that is to say, nodes that may be reachable only intermittently, nodes suffering from bottleneck, and the associated cases, such as several new nodes attempting to replicate fully from a small number of parent nodes.

In a fashion analogous to the first incarnation of the Gnutella p2p network, a perfectly flat system built up of many similar nodes typically has low scalability. Therefore, one might choose to look at the solution arrived at by that network; a decentralised system that promotes certain nodes within that system to 'supernodes', fast control nodes with a pivotal role in sustaining the structure of the network. This analogy, the similarity of distributed web service discovery architecture to that of peer-to-peer systems as a field, is not new. Indeed, it is almost inevitable; as UDDI loses its hierarchical structure, it automatically approaches peer-to-peer architecture.

The success of UDDI as a technology was originally adversely affected by the fact that the technology was intended to be used in a very hierarchical, client-server model. There were expected to be relatively few UDDI servers, and a large number of clients; for example, the IBM UDDI directory, the Microsoft directory and perhaps a couple of others. In practice, this niche proved to be illusory, and the return of UDDI has been mostly due to its adoption in other environments. A typical example might involve testing within a single business, with only a limited number of services available for external users; as such, UDDI has begun to rise in popularity as the technology has adapted to a distributed environment.

The same is true of web services in general; within a given e-learning environment, certain services will be available externally, designed for the outside world, whilst others will be designed for internal use only. Several e-learning environments can be connected together by means of *federation* of web services, which is to say that the environments choose to attribute to each other a certain level of trust allowing them by means of agreements, shared services and technologies, to share information and use of services.

When considering the UK academic community, it is worth noting that the scope of the services in question extends beyond simple web services to include z39.50, OAI, CGI scripts, etc, rather than SOAP in isolation..

2. Setting up a UDDI server.

In order to evaluate the features of UDDI an experimental server was installed and configured for testing purposes.

The first and most immediately obvious point to make here is that, despite the widespread hype surrounding UDDI, there really is not a great deal of software available - open source or otherwise. The second observation to make is the existence of a clear UDDI standard - two of them, in fact: versions 2 and 3 are both current. Version 2 clients ought to be able to interoperate seamlessly with a version 3 server, as the v3 server will operate as a v2 server from the perspective of the client. Version 2 features are therefore guaranteed, whilst v3 are not.

The only available free (open source) option was jUDDI (pronounced 'judy'), which describes itself as follows: [3]

- Open Source
- Platform Independent
- Supports JDK 1.3.1 and later
- UDDI version 2.0 compliant implementation
- Use with any relational database that supports ANSI standard SQL (MySQL, DB2, Sybase, JDataStore, HSQLDB, etc.)
- Deployable on any Java application server that supports the Servlet 2.3 specification (Jakarta Tomcat, JOnAS, WebSphere, WebLogic, Borland Enterprise Server, JRun, etc.)
- jUDDI registry supports a clustered deployment configuration.
- Easy integration with existing authentication systems

jUDDI is now (as of June 2005) in version 0.9rc4, after a long period of relative inactivity. Version 0.9rc3 was used throughout this work.

2.1 Installing jUDDI

Requirements:
jUDDI, being a Java-based web application, requires Tomcat (4.1.27 or above). Tomcat requires Java 1.4 or above.

jUDDI is configured to use JDBC, and as such can use any of several databases:

- MySQL
- DB2
- HSQLdb (HypersonicSQL)
- Sybase
- PostgreSQL

[3] http://ws.apache.org/juddi/

- Oracle
- TotalXML
- JDataStore (Borland)

We chose to use MySQL. In order to set up the database appropriately, an SQL script was run (see Appendix 3).

Following this, the jUDDI web application was installed, according to the following instructions:

Unzip the ${HOME}/juddi_0.8.0_bin/build/juddi.war file at the directory ${CATALINA_HOME}/webapps/juddi/
Set the following properties at the file ${CATALINA_HOME}/webapps/juddi/WEB-INF/classes/juddi.properties

(jar: file:/juddi.war!/WEB-INF/classes/juddi.properties)

```
# The UDDI Operator Name
juddi.operatorName = eurodyn.com
# the DataSource JNDI name
juddi.dataSource=java:comp/env/jdbc/juddi
```

Set the following properties at ${CATALINA_HOME}/webapps/juddi/WEB-INF/classes/log4j.properties

```
#
# set the log file to ${HOME}/juddi.log and not the ${PWD}/juddi.log
#
log4j.appender.LOGFILE.File=${user.home}/juddi.log
```

The following lines were then added to ${CATALINA_HOME}/conf/server.xml file[4]:

```
<Context path="/juddi" docBase="juddi" debug="5" reloadable="true"
  crossContext="true">
  <Logger className="org.apache.catalina.logger.FileLogger"
              prefix="localhost_juddiDB_log" suffix=".txt"
              timestamp="true"/>
  <Resource name="jdbc/juddiDB"
                auth="Container"
                type="javax.sql.DataSource"/>
  <ResourceParams name="jdbc/juddiDB">
    <parameter>
      <name>factory</name>
      <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
    </parameter>
    <!-- Maximum number of dB connections in pool. Make sure you
         configure your mysqld max_connections large enough to handle
         all of your db connections. Set to 0 for no limit. -->
    <parameter><name>maxActive</name><value>100</value></parameter>
    <!-- Maximum number of idle dB connections to retain in pool.
         Set to 0 for no limit. -->
    <parameter><name>maxIdle</name><value>30</value></parameter>
    <parameter><name>maxWait</name><value>10000</value></parameter>
    <!-- MySQL dB username and password for dB connections  -->
    <parameter><name>username</name><value>juddi</value></parameter>
    <parameter><name>password</name><value>123456</value></parameter>
```

[4] http://wiki.apache.org/ws/Deploy_jUDDI_on_Tomcat_and_MySQL

```
    <!-- Class name for mm.mysql JDBC driver -->
    <parameter>
      <name>driverClassName</name>
      <value>org.gjt.mm.mysql.Driver</value>
    </parameter>
    <!-- The JDBC connection url for connecting to your MySQL dB.
         The autoReconnect=true argument to the url makes sure that the
         mm.mysql JDBC Driver will automatically reconnect if mysqld closed
the
         connection.  mysqld by default closes idle connections after 8 hours.
    -->
    <parameter>
      <name>url</name>
      <value>jdbc:mysql://host.domain.com:3306/juddi?
autoReconnect=true</value>
    </parameter>
    <parameter>
           <name>validationQuery</name>
           <value>select count(*) from PUBLISHER</value>
    </parameter>
  </ResourceParams>
</Context>
```

jUDDI indicates installation state by self-testing (see; Testing the Installation against Client Applications).

Following installation, a test record was manually inserted into the database, giving a business name, contact details and a service (without T-Model). This was used for testing of client-side applications.

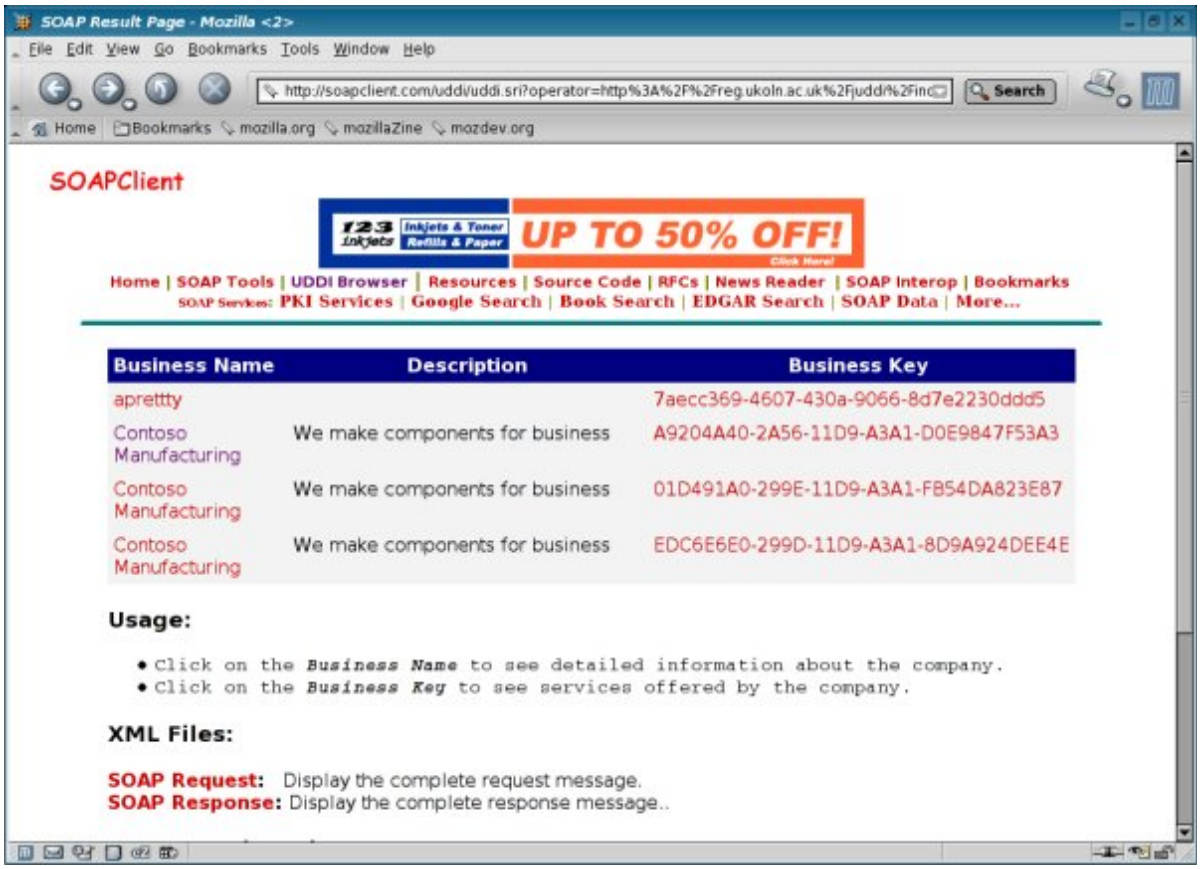
The finished jUDDI installation is available at:

http://reg.ukoln.ac.uk/juddi/inquiry/ for the inquiry interface (port 80)
http://reg.ukoln.ac.uk/juddi/publish/ for the publish interface (port 80)

2.2 Testing the installation against client applications

The first stage in choosing a method of populating the UDDI service was to ensure that the system was operating correctly. It was therefore tested against a certain number of software packages.

- Initially, the service was tested using the check script provided: happyjuddi.jsp .
- The first external test was by means of http://soapclient.com/uddi/ which seemed capable of browsing the system. One can see the results of such a test from the following URI: http://soapclient.com/uddi/uddi.sri?operator=http://reg.ukoln.ac.uk/juddi/inquiry&key=%25a%25&requestname=find_business&maxRows=50
- We then made use of a small script written with the phpuddi library in order to test the browsing capabilities of the system.



2.3 Populating the UDDI server

Firstly, the problem of authentication needs to be addressed. juddi has a simple authentication system based around the juddi_users.xml file. Adding a user is done by editing the file as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<juddi-users>
  <user userid="juddi" password="password" />
</juddi-users>
```

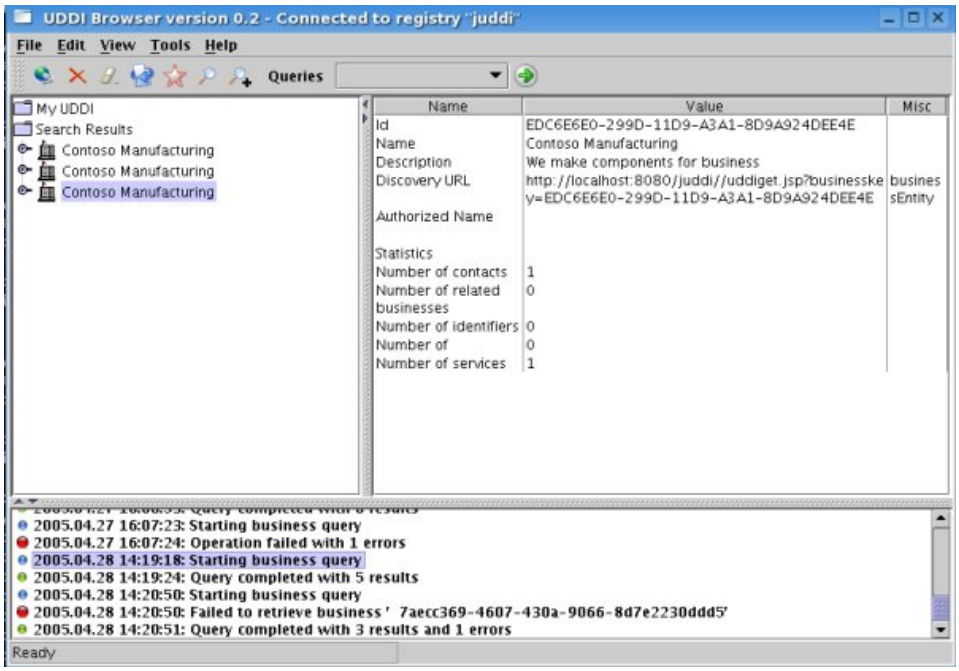
There are three fairly intuitive methods of adding records to the UDDI server:
1) find a piece of software capable of inserting records using the UDDI interface
2) write something making use of a UDDI client-side library, for example the Java library
3) insert records directly into the MySQL database underlying jUDDI

Approach 1) proved to be impractical, since no appropriate tools appeared to exist that could be appropriately used, eg for the batch conversion of XML documents into UDDI records. Approach 2) was both possible and practically achievable, with the additional advantage that it could theoretically be used with all UDDI standard compatible server software, but naturally required a development effort. The third approach was perhaps the simplest for our purposes, since we only required a demonstration system, but was not extensible to any other scenario and was strictly limited in compatibility to the current installation of jUDDI, since there are no guarantees or standards governing underlying database structure.

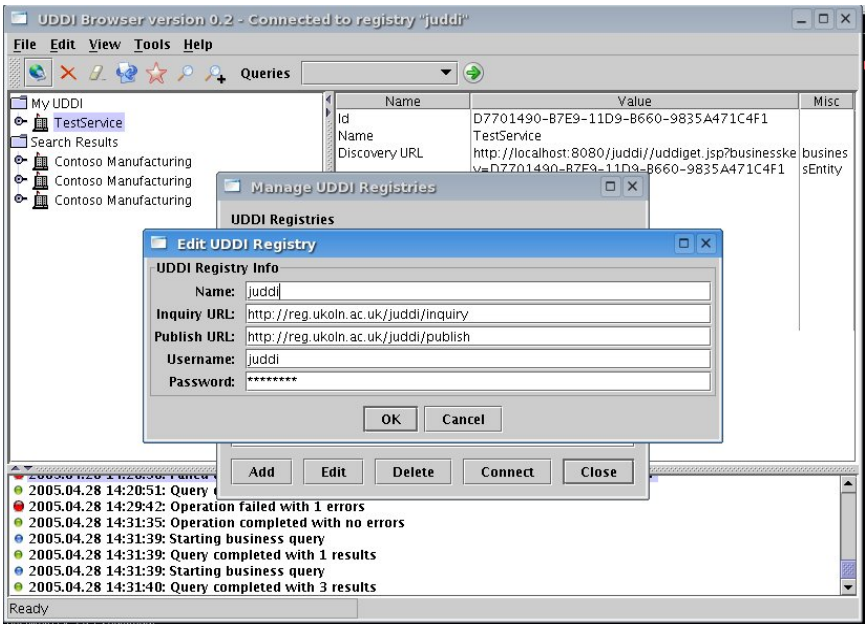
2.4 Making use of available client software

The available software clients tested included:

UDDIbrowser: http://sourceforge.net/projects/uddibrowser/
http://www.uddibrowser.org/



UDDI Browser is an open-source project that provides a friendly user interface allowing users to browse and manipulate content in UDDI registries. It is written in Java using the Swing libraries. Currently the browser supports version 2.0 UDDI registries.
The UDDI Browser supports the entire UDDI API for managing registries, including the whole query API set, and create/update/delete support for all entities in UDDI. It also has a host of features to make the UDDI users and administrators life easier, including query persistence, My UDDI support, and administrator utilities to aid in registry maintenance.
Using uddibrowser, it was possible to add services, businesses and tmodels. In order to do this, one needs to set up a username and password in the browser configuration. However, this is very clearly a manual process and as such is not a solution for our purposes.



2.5 Available APIs

uddi4j http://sourceforge.net/projects/uddi4j/
UDDI4J is a Java class library that provides an API to interact with a UDDI (Universal Description, Discovery and Integration) registry.

SOAP::lite http://sourceforge.net/projects/soaplite/
SOAP::Lite is a collection of Perl modules that provides a simple and lightweight implementation of SOAP, XML-RPC, UDDI and other webservice-related specifications.

phpuddi http://phpuddi.sourceforge.net/
phpUDDI is a set of standalone PHP classes with no external dependencies, supporting the basic UDDI 1.0/2.0 Inquiry APIs. However, it is not able to publish documents and is therefore of little interest in this context.

Of these three, SOAP::lite proved to be the most relevant to our needs. This was due to its relatively large function set, ability to publish, simple API and the generally large number of Perl libraries with which it could be combined, which make additional tasks such as XML processing or database access very simple.

Therefore, I tested browse and publish capabilities (see source code in Appendix II), and found that I could successfully publish single businesses with no problems. Since SOAP::lite is still rather recent and is distributed as untested for publishing purposes, I would nonetheless recommend a look at the uddi4j libraries in the event that this were to become a production system. As uddi4j is an IBM production library, the probability is that it will receive relatively frequent updates or at least support in terms of bug fixes and continued use. That said, it is possible that SOAP::lite is more frequently used than uddi4j and that bugs will therefore be discovered and removed.

2.6. Inserting records directly into MySql for jUDDI

In a sense, this was the easiest of the available methods, principally because it involved no potential difficulties with the UDDI interface and made use only of simple tried and tested technology (eg. perl's DBI, the database interface, which is a very mature set of APIs and very well documented). On the other hand, this conclusion did not reckon with the major difficulty of that approach - the database itself.

jUDDI is not extensively documented. FAQs and other information are available on the Web, but they are relatively sparse, perhaps due to the fact that UDDI remains an infrequently seen technology and is therefore not common enough for many expert contributors to exist. The database itself is entirely undocumented and extremely complex, the only clues to its usage coming from the choice of table and column IDs. Short of taking the time for a complete examination of the jUDDI source code, the only way of figuring out the precise preferred placement of variables within the database was to insert data structures, and examine their representation within the database.

3. Mapping IESR to UDDI

As seen above, a number of approaches are actually available for inserting information into the UDDI directory, two of which are scriptable using Perl, one of which depends on relatively recent and untested APIs and the other depending on the stable database API. The theory at this stage would suggest that we were therefore home and dry, with nothing left to do but writing a little XML-parsing code to extract the information from the various test IESR records and insert it into the UDDI server using either of the above methods. Unfortunately, in the immortal words of Douglas Adams, we could not even be said to be home and vigorously towelling ourselves off.

The remaining issue is the question of tmodels. tModels are UDDI data structures, used to define taxonomies as well as defining a service's technical interface. tModels are complicated, and every other structure in the UDDI data model references them. In fact, they are even able to reference themselves - tModels are self-referential, in that they can categorise themselves by reference to themselves! For classification; the tModel is used as a namespace or taxonomy. Whilst unchecked taxonomies can be used by means of the UDDI keyword system, typically useful in situations involving just a small group of people, the established method for creating a taxonomy is to register a new tModel to document the meaning and intended use of that taxonomy. Thus, each entity typically points at a tModel to define the namespace/taxonomy in which it is classified.

- businessEntity, the information about the parent business, points at a tModel to define its namespace.

- businessService, describing the function of a given service, points to tModel to define its namespace.

- publisherAssertion, describing the relationship between two entities/parties, refers to it to define namespace.

- bindingTemplate, which provides technical information about the entry point to services as well as specifications to do with construction, references interface specifications for the service by pointing at the tModel in question.

TModels exist in order to solve problems relating to natural language and ambiguity, and to facilitate searching by service type information. Service types are typically described simply as a collection of references to tModels. Each tModel is referred to using a globally unique ID, and each referenced tModel must be registered in the UDDI registry. Certain tModels are already available by default in each UDDI server, such as HTTP, homepage, SMTP, fax and FTP, but less basic technologies are not defined by default and *must* be registered before service information can be registered.

It is perhaps worth adding here that in general, the UDDI data model specifies the following:

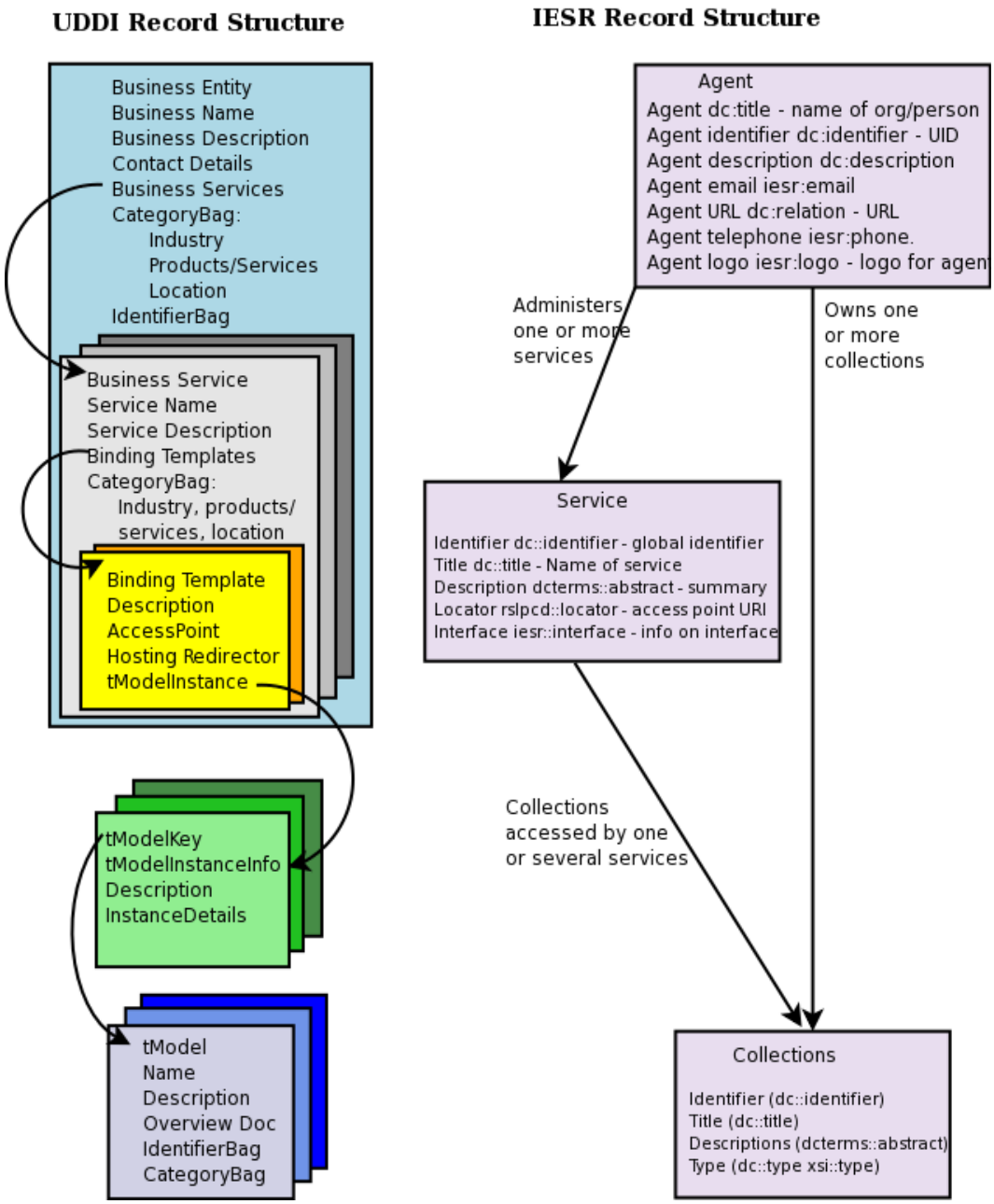
Any one business entity can have several business services - IBM, for example, can provide a world clock web service as well as a service that translates single words from English to Swahili. Any one business service can have one or more binding templates - the world clock service may actually be available through a number of interfaces, or the English-Swahili dictionary might be available through both encrypted and unencrypted channels.

The tModel service is independent of this. It is the basis information upon which the rest of the system is built. Therefore, it is both important and the source of much of the complexity of UDDI. Providing a simple qualitative textual description of a web service can be done very

simply; the difficulty is to make a description available that is sufficiently meaningful to be useful beyond the purely human level, since UDDI is capable of use as a machine-to-machine service registry permitting automated service discovery, and to ensure that the descriptions are sufficiently complete to permit one to interact with the service on that information alone.

Therefore, it has two tasks: application as a taxonomy, in order to provide that meaningful description, and application as a technical fingerprint, which is to say, specific information defining service type, protocols, formats, rules, and so forth. Each standard protocol definition is to be registered as a tModel, and services which are compliant with them must register themselves as such by referring to the tModel in question in the bindingTemplate.

Thus, in order to get IESR data into the UDDI database and have it be useful on a rigorous level (accurate/machine-readable), we require tModels defining each technical detail of each of the services referred to in the IESR information. We then need to refer to these when adding each of the IESR services to the database. From where are these tModels to be specified?



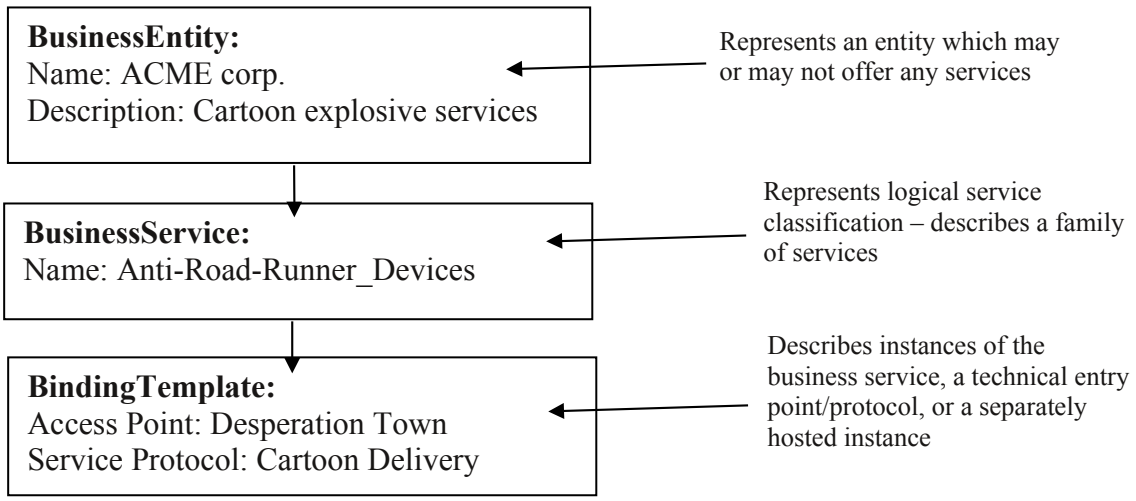
3.1 Choosing a mapping

How should the collections/agent information, as seen in the right column (IESR) be mapped into the UDDI model (left)? Here, one recognises the grounds behind the frequently heard complaint that "the UDDI data model is inflexible". The model permits one to map businesses, services, and relationships between businesses. It does not lend itself easily to collections, agents and services.

One possible solution would involve mapping the relationship between the collection entity and the agent entity as a business to business mapping – as a UDDI relationship. Using business relationships, one can map connections such as publisher assertions – relationships are mapped by assertions provided by both parties, and the specific nature of the relationship is provided by a keyedReference. A built-in canonical tModel (uddi-org:relationships) supports parent-child, peer-to-peer and identity type relationships.

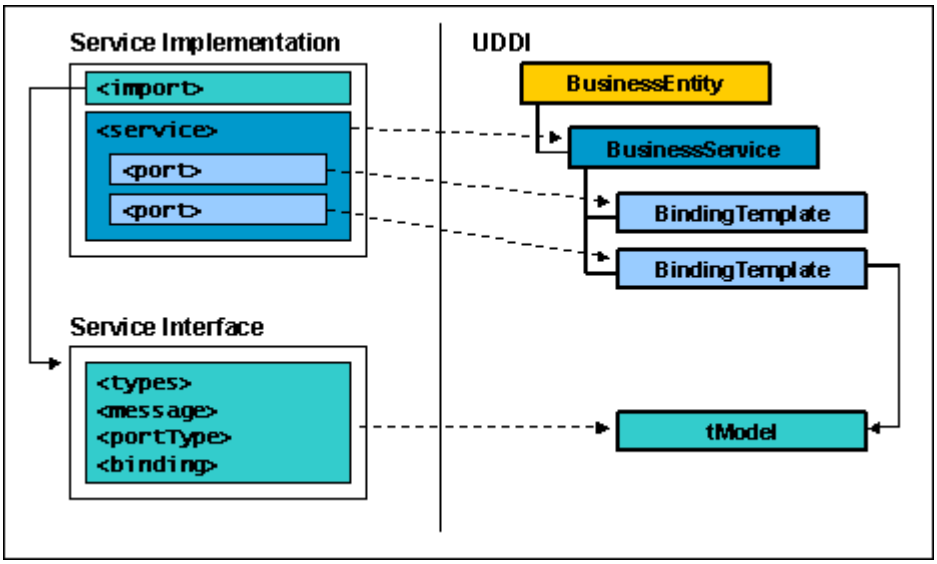
The mapping provided by Matthew Dovey works rather differently, as it treats the business service as the collection, the binding templates available as the services, and so on. The IESR agent is the business entity. On one level, this works well – the business entity may own several business services, so the agent may own several collections – but on the first glance, binding templates do not appear from the UDDI specification to be designed to encompass several services as dissimilar as one might reasonably expect IESR services to be. They generally seem to refer to endpoints, that is to say, URI locations (and associated connection information, eg, protocols) pointing to a place on the network at which a service interface is located. UDDI binding templates do not offer many places for categorisation information, whereas business services include a categoryBag of keyedRefs. One can place the required extra information in tModels referenced by the binding templates, although this appears at first blush to be a relatively complex approach.

To see this, one might take an example of a standard UDDI entry:



From this example, one notes that the BusinessService itself is intended to contain one family of services, each of which do similar things; the group of binding templates contained within one business service simply serves to point one to various ways and means by which one can access that service.

Note that the BusinessService and BindingTemplate approximate to a WSDL description:



(Source: http://www-128.ibm.com/developerworks/webservices/library/ws-wsdl/)

The question of how the collections/agent information should be mapped into the UDDI model is not yet resolved. The way in which one chooses to map IESR collections and services to the UDDI model surely depends on the nature of the IESR collection as an entity. If it contains only one group of services, then it maps well to the BusinessService entity in UDDI; if it contains several, it would seem to map better with the BusinessEntity.

For example, if one wishes to represent an image library in IESR, a collection which contains an OAI repository and a SRU server, the Dovey mapping would return the following:

businessService: Image library
    bindingTemplate: OAI repository for image library
    bindingTemplate: SRU server for image library

whereas a UDDI mapping following the Acme example would return:

businessEntity: Myself
[is declared as parent of:]
businessEntity: Image library
    businessService: OAI repository harvesting
        bindingTemplate: Available on port 80 of machine ____ using OAI
    businessService: OAI repository searching
        bindingTemplate: Available on port ___ of machine ____ using z39.50
        bindingTemplate: Available on port ___ of machine ____ using SRU
        bindingTemplate: Available on port ___ of machine ____ using SRW

Typically in UDDI, a businessService tends to represent a discrete function, with bindingTemplates operating as bookmark pointers to a given address, protocol and access method. However, bindingTemplates can contain a lot of information, so it would be possible to classify service functionality in the bindingTemplate. How simple would it be to search information stored in each mapping? These issues will be discussed along with other possibilities such as adaption of IESR data so that a closer but still analogous mapping could be found.

4. Conclusion

The aim of mapping IESR to UDDI, has proven to be reasonable, and the technologies available are adequate for the task. However, a great deal of work still remains, mostly in defining a set of tModels that are sufficiently extensive to cover the task at hand. Direct injection of information into the jUDDI MySQL backend database has proven to be possible, if relatively complex in implementation, as is making use of one of the available UDDI APIs, whether in Java or in Perl. Whilst the direct MySQL manipulation method is by far the simplest for a UDDI non-specialist to use and understand, it remains a solution useful only because of the narrow constraints of our particular situation; later versions of the jUDDI system may not even retain the same database table model, rendering the mapping previously used useless until revised.

The technical side of the problem is essentially solved. The remaining issues are mostly those to do with the mapping, and a solution to these issues is probably best achieved by discussion with Matthew Dovey and others. Yet this is not the most significant question, as the major issue is to evaluate UDDI itself: UDDI, despite or possibly as a result of its potential, is seemingly overcomplicated.

During this project, the justness of several of the complaints surrounding the UDDI spec has been demonstrated – or the pitfalls that led others to those conclusions have been encountered. This specification gives the impression that it is inflexible and difficult to apply. And yet, UDDI advocates claim that the specification is, once understood, ideally engineered for the problem group that it is designed to solve – which, from all evidence, is remarkably compatible with the requirements of IESR.

As yet, no absolute conclusion can be drawn. Once the mapping questions are solved to general satisfaction, once the demonstration server is up, populated and available, the machine-readability of the information and the resulting repository is available, it will be possible to comment further. The ability of a UDDI implementation to act as a repository for this data is not in doubt; rather, the usability of the result remains questionable, as does the appropriateness of the approach for IESR purposes.

A successful result depends on a standardisation and profiling effort, to ensure that the mapping that is used is well documented and understood, that it fulfils the actual user requirements, and that it satisfies compatibility with the IESR data. In practice, is there sufficient advantage from the use of UDDI to offset those disadvantages?

Bibliography

Fielding, R. (2000). Representational State Transfer (REST). From doctoral thesis, *Architectural Styles and the Design of Network-based Software Architectures*
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Govoni, D. (2002). Web Services over P2P networks. SOA Web services journal. http://webservices.sys-con.com/read/39425.htm

Paolucci, M., Kawamura, T., Payne, T. R. and Sycara, K. (2002) Importing the Semantic Web in UDDI. In *Proceedings of Web Services, E-Business and Semantic Web Workshop, CAiSE 2002.*, pages pp. 225-236, Toronto, Canada. Bussler, C., Hull, R., McIlraith, S., Orlowska, M. E., Pernici, B. and Yang, J., Eds.

Sun, C., Lin, Y., Kemme, B. (2004). Comparison of UDDI Registry Replication Strategies. Proceedings of the IEEE International Conference on Web Services (ICWS'04), http://www.cs.mcgill.ca/~ylin30/paper/ICWS04.pdf

UDDI Consortium (2002). UDDI replication specification. http://uddi.org/pubs/Replication-V2.03-Published-20020719.pdf

Zhang, L, Chung, J., Zhou, Q. (2002). Developing Grid computing applications, Part 1. IBM DeveloperWorks article.

Appendix I: phpuddi code

To be included in UDDI.inc:

```
 var $regarray =
    array('REG' =>
      array('Inquiry'  =>
        array('url'  => "reg.ukoln.ac.uk/juddi/inquiry",
          'port' => 80),
        'Publish' =>
              array('url' => "reg.ukoln.ac.uk/juddi/publish",
          'port' => 80
                    ));
```

Standalone code making use of the definition above:

```
<?php
require_once 'UDDI_Inquiry.inc';

$my_uddi = new UDDI_Inquiry('REG');
$my_uddi->version = 2;
$my_uddi->debug = TRUE;

$input = array(
                'findQualifiers'=>'sortByNameAsc,sortByDateAsc',
                maxRows=>50,
                'name'=>'%o%'
              );

$result = $my_uddi->find_business( $input );
$result = htmlspecialchars( $result );

echo "<pre>$result</pre>"
?>
```

Appendix II: UDDI::lite (SOAP::lite) code

1. Querying the database

```
#!/usr/bin/perl
use UDDI::Lite
  import => 'UDDI::Data',
  import => ['UDDI::Lite' => ':find', ':get'],
  proxy => 'http://reg.ukoln.ac.uk/juddi/inquiry',
;

my @parameters = (
  findQualifiers([findQualifier('sortByNameAsc'),
                  findQualifier('caseSensitiveMatch')]),
  name('%a%'),

);

my $b = find_business(@parameters);
for ($b->businessInfos->businessInfo) {
  print $_->name, "\n";
}
```

2. Adding records to the database

```
#!perl -w

use strict;
use UDDI::Lite
  import => 'UDDI::Data',
  import => 'UDDI::Lite',
  proxy => "http://reg.ukoln.ac.uk/juddi/publish",
;

my $name = 'Sample business ' . $$ . time; # just to make it unique

print "Authorizing...\n";
my $auth = get_authToken({userID => 'juddi', cred => 'password'})-
>authInfo;
my $busent = businessEntity(name($name))->businessKey('');

print "Saving business '$name'...\n";
my $newent = save_business($auth, $busent)->businessEntity;
my $newkey = $newent->businessKey;

print "Created...\n";
print $newkey, "\n";
print $newent->discoveryURLs->discoveryURL, "\n";

print "Deleting '$newkey'...\n";
my $result = delete_business($auth, $newkey)->result;
print $result->errInfo, "\n";
```

To save a service rather than a business (or as well as...) the following syntax is appropriate:

```
my $busent = with businessEntity =>
  name("Contoso Manufacturing"),
  description("We make components for business"),
  businessKey(''),
  businessServices with businessService =>
    name("Buy components"),
    description("Bindings for buying our components"),
    serviceKey(''),
    bindingTemplates with bindingTemplate =>
      description("BASDA invoices over HTTP post"),
      accessPoint('http://www.contoso.com/buy.asp'),
      bindingKey(''),
      tModelInstanceDetails with tModelInstanceInfo =>
        description('some tModel'),
        tModelKey('UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4')
;

my $newent = save_business($auth, $busent);
print $newent->businessEntity->businessKey if ref $newent;
```

Appendix III: jUDDI SQL installation script

Taken from http://wiki.apache.org/ws/Deploy_jUDDI_on_Tomcat_and_MySQL

```
#
# Create the jUDDI database
#
DROP DATABASE IF EXISTS juddi;
CREATE DATABASE juddi;
#
# Sets global privileges to user juddi
#
REPLACE INTO mysql.user SET
  Host = '%', # any hostname (including localhost)
  # alternatively '192.168.0.1/255.255.255.0', '129.%', 'localhost'
  User = 'juddi',
  Password = PASSWORD('123456'),
  Select_priv = 'Y',
  Insert_priv = 'Y',
  Update_priv = 'Y',
  Delete_priv = 'Y',
  Create_priv = 'Y',
  Drop_priv = 'Y',
  Reload_priv = 'Y',
  Shutdown_priv = 'Y',
  Process_priv = 'Y',
  File_priv = 'Y',
  Grant_priv = 'Y',
  References_priv = 'Y',
  Index_priv = 'Y',
  Alter_priv = 'Y'
  # Show_db_priv = 'N',
  # Super_priv = 'N',
  # Create_tmp_table_priv = 'N',
  # Lock_tables_priv = 'N',
  # Execute_priv = 'N',
  # Repl_slave_priv = 'N',
  # Repl_client_priv = 'N',
  # ssl_type = 'X509',
  # ssl_cipher = '', # blob
  # x509_issuer = '', # blob
  # x509_subject = '', # blob
  # max_questions = '0',
  # max_updates = '0',
  # max_connections = '0'
;
FLUSH PRIVILEGES; # required

#
# fix the mysql security gaps
# NOTE: allowed access from root@localhost, root@domain.com
#
DELETE FROM mysql.user WHERE User='';
UPDATE mysql.user SET Password=PASSWORD('123456')
  WHERE user='root';
FLUSH PRIVILEGES; # required

#
# Optionaly Set db provilege and host privileges
# This should be used if the juddi user should not
# have global privileges. In detail mysql access control
# is calculated as follows:
#   global privileges
#   OR (database privileges AND host privileges)
```

```
#   OR table privileges
#   OR column privileges
#
INSERT INTO mysql.db SET
  Host = '%', # if blank intersect with the mysql.host record
  Db = 'juddi%',
  User = 'juddi',
  Select_priv = 'Y', Insert_priv = 'Y',
  Update_priv = 'Y', Delete_priv = 'Y',
  Create_priv = 'Y', Drop_priv = 'Y',
  Grant_priv = 'N', References_priv = 'Y',
  Index_priv = 'Y', Alter_priv = 'Y',
  Create_tmp_table_priv = 'N',
  Lock_tables_priv = 'N'
;
INSERT INTO mysql.host SET
  Host = '%',
  Db = 'juddi%',
  Select_priv = 'Y', Insert_priv = 'Y',
  Update_priv = 'Y', Delete_priv = 'Y',
  Create_priv = 'N', Drop_priv = 'N',
  Grant_priv = 'N', References_priv = 'N',
  Index_priv = 'N', Alter_priv = 'N',
  Create_tmp_table_priv = 'N',
  Lock_tables_priv = 'N'
;

#
# create the jUDDI tables
#
USE juddi;
SOURCE juddi_mysql.ddl;

#
# add a jUDDI publisher
#
# PUBLISHER_ID  The user ID the publisher uses when authenticating.
# IMPORTANT: This should be the same value used to authenticate
# with the external authentication service.
# PUBLISHER_NAME The publisher's name (or in UDDI speak the
# Authorized Name).
# ADMIN Indicate if the publisher has administrative privileges.
# Valid values for this column are 'true' or 'false'. The ADMIN
# value is currently not used.

INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,ADMIN)
  VALUES ('juddi','juddi user','false');
```