

UK Engineering Task Force

Evaluation of UDDI for UK e-Science

Matthew Dovey (OeSC), Ivaylo Kostadinov (OeSC), Jonathan Giddy (WeSC), Patrick Green (NeSC), Dave Berry (NeSC), Dharmesh Chohan (CCLRC), Xiao Wang (CCLRC)

Overview

The ETF Service Registries Workpackage was to establish a pilot services Registry for the UK e-Science Programme which was:

- geographically distributed;
- load-balanced;
- fully redundant/replicated

For this purpose the group evaluated the use of UDDI, with UDDI nodes deployed at four locations – Daresbury, NeSC, OeSC and WeSC. The evaluation work took place between June 2004 and December 2004.

Activities

1. Investigation of suitable UDDI software

Initially it was intended to base the prototype service on UDDI version 3.0. Version 3.0 was the current implementation of UDDI and offered a number of improvements over UDDI deemed essential for supporting e-Science. The two key aspects were the replication API and subscription API which would allow proper load balancing and redundancy, and support for signing of entities to ensure their authenticity. The replication API and subscription API were deemed important so that the UDDI registry could be load balanced both for performance (e.g. requests would be routed to a nearby registry to avoid network latencies) and for reliability (downtime or inaccessibility of nodes would be compensated for by the other available nodes). It was noted, however, that even with these APIs, UDDI did not offer a fully redundant editing platform (whilst a record can be discovered at replicated nodes, it can only be edited at the “home” node).

The subscription API also offered other UDDI topologies, whereby a local private UDDI registry might only replicate a subset of services from the UK e-Science registry based on local policies and/or contain additional information or services not replicated to the UK e-Science registry. However, this aspect was not addressed in the evaluation.

However, although there were a number of UDDI registry implementations from IBM, Microsoft, Systinet, Novell and Apache only the Systinet project offered UDDI

version 3.0 support at the time we began the evaluation.¹ We did have non-committal responses from the other providers as to the availability of version 3.0 support, which forced us to stick with UDDI version 2.0 for the prototype, and seek alternative approaches to replication.

At the time of investigation there were only two Open Source implementations of UDDI: Novell had recently released their implementation, Novell NSure UDDI, into the Open Source community; jUDDI which was one of the earliest UDDI implementations had been taken into the Apache fold.

It was decided to implement the prototype service using Apache jUDDI. The Novell NSure UDDI implementation at that time was still dependent on a Novell LDAP based backend.

2. Pilot Service Implementations

jUDDI was installed at the four locations Daresbury, Oxford, Cardiff and Edinburgh. We used a mixture of Postgres and MySQL as the backend database. The current node information is as follows:

Daresbury: Two nodes using jUDDI 0.9 running on MySQL

inquiryURLs = <http://trent.dl.ac.uk:9000/juddi/inquiry>
<http://grid12.esc.rl.ac.uk:8080/juddi/inquiry>
publishURLs = <http://trent.dl.ac.uk:9000/juddi/publish>
<http://grid12.esc.rl.ac.uk:8080/juddi/publish>

OeSC: jUDDI 0.9 running on Postgres

inquiryURL = <http://oescedge.oucs.ox.ac.uk/juddi/inquiry>
publishURL = <http://oescedge.oucs.ox.ac.uk/juddi/publish>

NeSC: jUDDI 0.9 running on MySQL

inquiryURL =
publishURL =

WeSC: jUDDI 0.9 running on Postgres

inquiryURL = <http://uddi.wesc.ac.uk:8334/juddi/inquiry>
publishURL = <http://uddi.wesc.ac.uk:8334/juddi/publish>

Installation was relatively painless – some sites had some initial difficulties but these were quickly resolved. The only major issue was that jUDDI was still under development with frequent releases. As a result it was necessary to update the software running on the nodes fairly frequently.

OeSC extended the base jUDDI package to support certificate based authentication in addition to the normal username/password token authentication mechanism.

¹ As of the summer of 2005, both Systinet and Computer Associates offer full version 3.0 implementations of UDDI.

A major limitation of the jUDDI release was the lack of any log statistics of use. NeSC developed an additional component to analyse the logs produced by jUDDI². WeSC made some modifications to this to enable it to work with Postgres.

3. UDDI Clients

In order to encourage use of the UDDI registry, the group investigated available UDDI clients. Most UDDI clients tended to be fairly rudimentary and integrated with particular development or webservice environments. We adopted two Open Source UDDI clients – UDDIBrowser, an independent development and a web based client developed by CCLRC. We also briefly looked at a Cocoon based client, which was not available in source code, but which was in discussions about inclusion into the Apache jUDDI project. However, the discussions were at too premature a stage for inclusion at that time.³

NeSC did some additional development work to address issues in the UDDIBrowser client – in particular this client was somewhat weak in terms of its ability to publish and edit entries in an UDDI registry. The NeSC code for adding and editing business details and contact details has been included in the standard release.

The following table gives a comparison between the two clients:

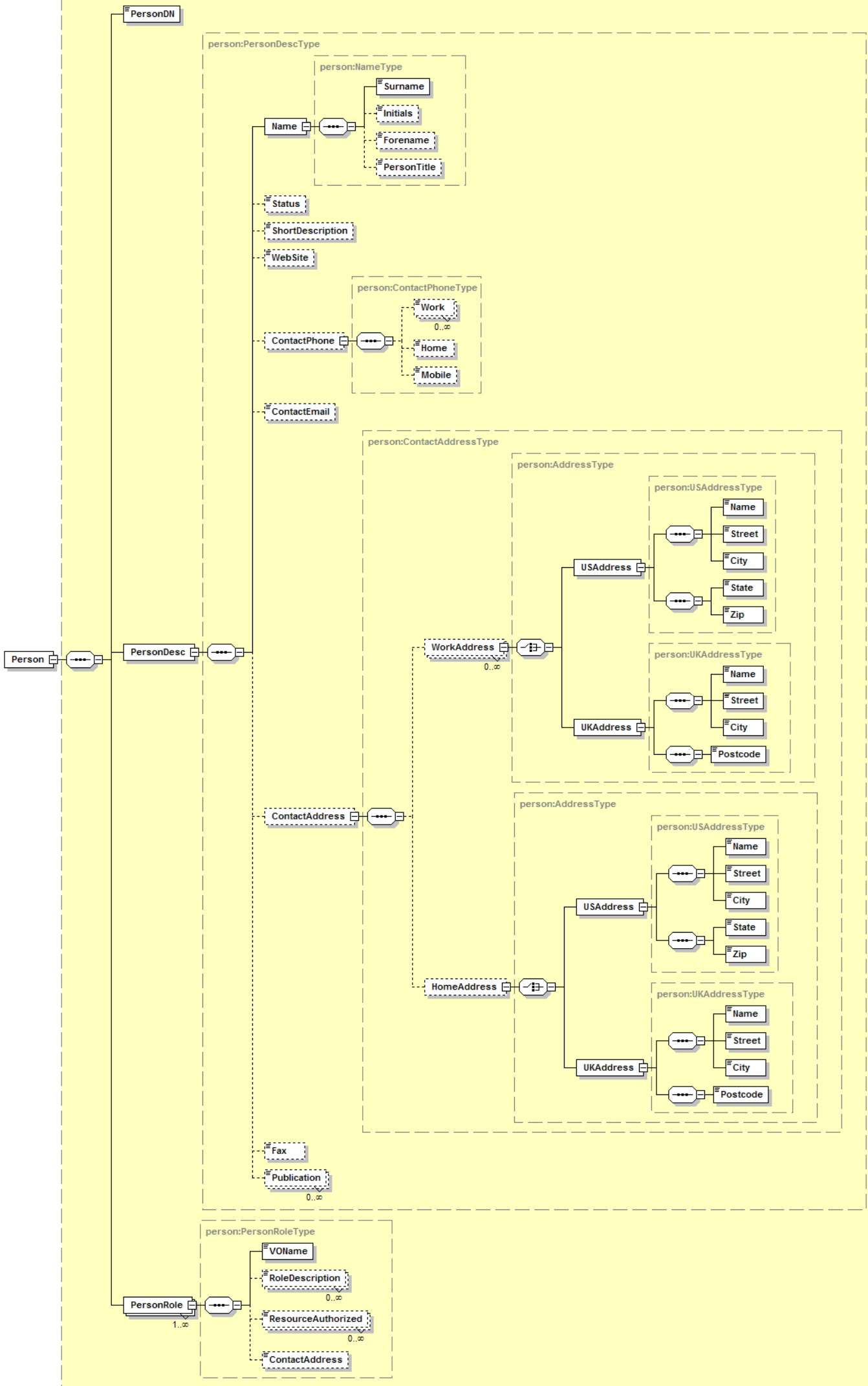
| Items | UDDI Browser | CCLRC UDDI Client Tool |
|---|---|---|
| 1. URL | http://www.uddibrowser.org | http://kermit.dl.ac.uk:8080/uddi |
| 2. Usage | To download and install | Only Web Browser |
| 3. Function: | | |
| Query: | | |
| By Business Name | √ | √ |
| By Service Name | √ | √ |
| By tModel Name | √ | √ |
| All Business | √ | √ |
| All Service | √ | √ |
| All tModel | √ | √ |
| Advanced query (e.g. case sensitive, exact name match, sort by name, sort by date...) | √ | √ |
| Direct query by Business Key | √ | √ |
| Direct query by | √ | √ |

² <http://homepages.nesc.ac.uk/~patrick/juddistats/>

³ This client was eventually include in the jUDDI cvs in December 2004

| | | |
|------------------------------|---|---|
| Service Key | | |
| Direct query by Binding Key | × | √ |
| Direct query by tModel Key | √ | √ |
| Direct query by discoveryURL | √ | × |
| Relationship | √ | √ |
| | | |
| Publish: | | |
| | | |
| Contactor | √ | √ (extension based on person schema) |
| Business | √ | √ |
| Service | √ | √ |
| tModel | √ | √ |
| Relationship | × | √ |
| Publisher Assertion | √ | × |
| Identifier | √ | √ |
| Category | √ | √ |
| Binding Template | √ | √ |
| | | |
| User: | | |
| | | |
| Login | × | √ |
| Register | × | √ |
| Choose UDDI Registry online | √ | × |

In addition the CCLRC client extends the UDDI registry to allow for additional contact information. The schema is shown on the following page.



4. Potential UDDI Users

The group identified a number of projects to use the prototype service and encouraged them to use it and feed back into the process. The projects were: RAVE (WeSC); GECEM (WeSC); DIPSO (WeSC); GridPP (OeSC); Mouse Atlas (NeSC); BRIDGES (NeSC); AstroGrid (NeSC); ATLAS (NeSC); Data Portal (CLRC); ehptx (CLRC); eccp1 (CLRC); SAKAI (CLRC). Based on project feedback, an initial “cookbook” for using UDDI was developed.

One particular issue which did emerge was that of modelling multiple instances as bindingTemplates rather than businessServices. A project had adopted the approach of multiple service instances represented by multiple bindingTemplates within one businessService based on an article which can be found in Dr. Dobb's Journal, February 2004, "UDDI & Dynamic Web Service Discovery" by Peter Lacey at Systinet. To quote the article, "Because a businessService can contain multiple bindingTemplates, it is possible to instantiate the same web service multiple times on a network. This allows client applications to continue functioning even when one or more service instances fail." An alternative approach is that services that implement the same service portType, but are distinct in the sense of running at different locations, are represented by different businessServices. However, if a service implements two bindings to the same portType (e.g. SOAP/HTTP and pure HTTP bindings), then that is one businessService with two bindingTemplates. This allows us to, for example, query every known service exactly once, by detecting when multiple bindingTemplates actually refer to the same "backend". As an example, say the web service is an interface to Fishbase, a database of fish species, and there are several replica databases across the world, each with a web service interface. With this later architecture, we can sensibly say "check that each FishBase service has the entry for Fabbolium Richardus" without accessing the same database backend multiple times through different bindings.

This interpretation gives the most flexibility, and should be preferred. In essence, there is one businessService for each instance of each implementation of each portType, and there is one bindingTemplate for each instance of each binding. Using any bindingTemplate from a single businessService eventually gets you to the same instance of an implementation of a portType.

5. UDDI “Cookbook”

The UDDI Technical Committee has created two documents describing how to publish and discover web services using UDDI. Both describe the records that should be stored in the UDDI registry in order to enable a client to discover a web service. Version 1 described a simple method for publishing basic service information. The latest version “Using WSDL in a UDDI Registry, Version 2” can be found at <http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm>. Version 2 increases the amount of information stored, allowing searches on more detailed criteria (e.g. to select from two services implementing the same portType, one with a SOAP binding and one with a HTTP binding).

To investigate the use of these, we developed code to publish and discover web services. The code is written in Java and uses the UDDI4J library available from

<http://uddi4j.sourceforge.net/>. There are other libraries available for UDDI such as Novell's UDDIJ, as well as a canonical API generated by the WSDL (as detailed in the technical note here:) Code written against such API's will look similar, and none of the example use any specific features of UDDI4J which would not be available in other generic UDDI implementations.

The code implements the publication and discovery protocols of the Version 2 document. Currently the code only supports web services using SOAP over HTTP. Full source is contained in Appendix A.

Finding a web service

Create a `UDDIInquirer` instance, and set the inquiry URL.

```
UDDIInquirer inquirer = new UDDIInquirer();
inquirer.setInquiryURL(inquiryURL);
```

Find binding `tModels` that have the required namespace and binding name (and use SOAP over HTTP).

```
TModelList tModelList = inquirer.findSOAPBindings(
    "http://giddyjp.cf.ac.uk/2004/10/foo", "FooBinding");
```

Convert the returned `TModelList` to a `TModelBag`.

```
TModelBag tModelBag = new TModelBag();
Vector tModelInfos =
    tModelList.getTModelInfos().getTModelInfoVector();
for (int i = 0; i < tModelInfos.size(); ++i)
{
    TModelInfo tModelInfo = (TModelInfo)tModelInfos.elementAt(i);
    tModelBag.add(new TModelKey(tModelInfo.getTModelKey()));
}
```

Find a number of suitable access points. This function returns a vector of `org.uddi4j.datatype.binding.AccessPoint` objects.

```
Vector accessPoints = inquirer.findAccessPoints(tModelBag, 10);
```

The file [InquireExample.java](#) provides a complete example.

The file [UDDIInquirer.java](#) implements the `UDDIInquirer` helper class.

Publishing a web service

Create a `UDDIPublisher` instance, set the inquiry and publish URLs, and obtain an `AuthToken`.

```
UDDIPublisher publisher = new UDDIPublisher();
publisher.setInquiryURL(inquiryURL);
publisher.setPublishURL(publishURL);
AuthToken authToken = publisher.get_authToken(userid, password);
```

Publish a tModel for the WSDL portType, providing a URL for the WSDL document, the portType namespace, the portType name, and a human-readable description of the portType.

```
String portTypeRef = publisher.publishPortType(authToken,
    "http://bouscat.cs.cf.ac.uk:8080/test.wsdl",
    "http://giddyjp.cf.ac.uk/2004/10/foo", "FooPortType",
    "Foo Test Service Port Type");
```

Publish a tModel for the WSDL binding, providing a URL for the WSDL document, the binding namespace, the binding name, a human-readable description of the binding, and the portType reference returned by `publishPortType`. Note that this code only supports bindings for SOAP over HTTP.

```
String bindingRef = publisher.publishSOAPBinding(authToken,
    "http://bouscat.cs.cf.ac.uk:8080/test.wsdl",
    "http://giddyjp.cf.ac.uk/2004/10/foo", "FooBinding",
    "Foo Test Service Binding", portTypeRef);
```

Both these methods return a reference to an existing tModel if possible, otherwise they create a new tModel.

Publish a business name. The values in this do not come from the WSDL document, but need to be created to represent the service provider. You can reuse this business name for multiple types of services you provide. The parameters are a business name and a description of the business.

```
String businessRef = publisher.publishBusiness(authToken,
    "Jon's Foo Business",
    "There's no business like foo business");
```

Publish a service, providing the business reference returned by `publishBusiness`, a human readable service name, the service namespace, the service name, and a human readable description of the service.

```
String serviceRef = publisher.publishService(authToken,
    businessRef, "Foo Test Service",
    "http://giddyjp.cf.ac.uk/2004/10/foo", "FooService",
    "A service to foo");
```

Create a `org.uddi4j.datatype.binding.AccessPoint` instance to represent the protocol-specific endpoint. For SOAP over HTTP, this can be created with the SOAP endpoint address and the string "http".

```
AccessPoint accessPoint = new AccessPoint(
    "http://bouscat.cs.cf.ac.uk:8080/test", // SOAP address
    "http"); // URL type
```

Create a binding template, providing the service reference returned by `publishService`, the WSDL port element name, the previously created `AccessPoint`, the binding reference returned by `publishSOAPBinding`, and the portType reference returned by `publishPortType`.

```
String templateRef = publisher.publishBindingTemplate(authToken,
    serviceRef, "FooPort", accessPoint, bindingRef,
    portTypeRef);
```

The file [PublishExample.java](#) provides a complete example. (In production code, the username and password should be obtained in a more secure manner).

The file [UDDIPublisher.java](#) implements the `UDDIPublisher` helper class.

6. Registry Database Replication

Replication proved to be a major obstacle. The Apache jUDDI project would not commit to any timescale for implementation of the version 3.0 replication API (although would be interested in any developers taking such work on!). Also it was deemed that the group did not have the resources to develop this component of jUDDI.

OeSC investigated the available replication methods were naturally narrowed down to implementations available for the two databases in use (Postgres and MySQL), although the option of running all nodes on a single database platform was also considered as both MySQL and PostgreSQL have native support of single master / multiple slaves model of replication. However, a single master/multiple slave approach would mean that a redundant system for editing would not be possible.

The following requirements were identified:

- Peer to peer / distributed replication
- Stable / production quality
- Driver / software actively supported and having versions for the latest Postgres versions
- Continuing development and support for future database distributions
- Easy to add on to existing DB installation

The following solutions were considered:

Usogres

Type: Full, Read Only, Pre-Process

Location: usogres.good-day.net/

Description

Real-time backup utility

Replicates data via pre-postmaster

Supports only one main server and backup server

eRserver

Type: Partial, Read Only, Transactional

Location: <http://www.erserver.com/>

Description:

Supports snapshots (bundles changes)

Uses SPI, Perl and PG_Perl interface

SyncIDs are used to keep track of the slave data updates

Uses a replication table on master to capture updates via trigger then synchronization can be manual via command line or by number of snapshots

Only one slave and no fail over support

PostgreSQL Replicator

Type: Partial, Peer-to-Peer, Async

Location: pgreplicator.sourceforge.net

Description:

Robust update conflict detection and resolution mechanism

Creates a set of Replication Schema Tables (RST) to store the replication rules

Uses SP to dynamically capture triggers and auxiliary tables, and act as an interface between the DBA and the replication engine

TCL replication daemon running on each system in the distribution allows database synchronization to be started from any site at any time.

Uses triggers written in PL/TCL. The TCL daemon uses PostgreSQL connectivity API and TCL/DP libraries for communication over TCP/IP

Postgres-R

Type: Embedded, Peer-to-Peer, Sync

Location: gborg.postgresql.org

Description:

Uses a "total order" group communication system (multicasting updates)

Shadow copies are used to enforce isolation

Propagates tuple changes to decrease processing on remote or query strings if too many tuples changed

Implemented on PostgreSQL 6.4.2

4 branches of code (partial and recovery)

Only replicates one database

DBBalancer

Type: peer-to-peer

Location: <http://sourceforge.net/projects/dbbalancer/>

Description:

Load balancing multithreaded PostgreSQL connection pool.

Has a replication mode to keep in sync the load balanced backend servers. Needs the ACE framework libraries.

Development status is "alpha"

Not updated since 2001

This list is not comprehensive. Other solutions have subsequently come to light which were not fully investigated.⁴

Most of these solutions, judging from documentation, reports, mailing lists, and hands-on evaluation seem to be in their early stages of development. A more precise definition would be that they were in their early stages of development couple of years ago but have not been updated or maintained since then. They were mostly developed for PostgreSQL version 6.x and 7.0; the current PostgreSQL version being

⁴ Mammoth Replicator (commercial) (<http://www.commandprompt.com/>)
PGCluster (opensource) (<http://pgfoundry.org/projects/pgcluster>)

8.0. Some of them involve a major source code change or running a different binary for PostgreSQL; most administrators would not be willing to downgrade/substitute their running production database system, or to have couple of simultaneous database copies running. Others seem to implement the replication functionality in separate binaries that are to be run on a single master system or in parallel to every running copy of the database.

The best choice to fulfil the requirements seemed to be DBBalancer. The fact that it has been included in several major Linux distributions was encouraging. Unfortunately it was necessary to modify the source code in multiple source files to get it to compile. When compiled, installed and configured it displayed a very unstable behaviour - crashing immediately or soon after in connected to the database.

It seems that there is ongoing effort in the PostgreSQL community to port one or more of these replication solutions to the latest database distributions and to support them. The time spans and deadlines of these initiatives are however not clearly defined.

7. UDDI Workshop

In conjunction with the September Face to Face meeting of the OASIS UDDI Technical Committee hosted in Oxford, a workshop was held to discuss issues arising from UDDI within the UK e-Science community. This was attended by the UDDI Technical Committee and representative from myGrid, UDDIe, and the ETF. The Workshop identified the following limitations with the current versions. Some of this will form the basis of UDDI version 4.

Access Control List Issues

More granular ACLs are needed in order to allow others to edit metadata for services. In addition to basic authenticity afforded by signing of UDDI entities available in version 3.0 (but identified as an issue for project using UDDI version 2.0), a more extensive provenance is needed in terms of signing and edit audit. One possible solution might be an extension to OperationalInfo

Better replication of ACLs would be needed in order for proper peer to peer replication supporting redundancy of editing as well as discovery (rather than the current master/salve custody model)

Currently there is no ability to limit or control discovery of entities based on authentication.

Some of the above would feed into the discussion of ACLs in the next version of UDDI.

Quality of Service Issues

The commonest extension to UDDI was in order to express availability, reservation and leasing information. It was not clear whether this should be a function of UDDI, or whether UDDI would be able to indicate another service where this information could be found. Querying a large number of such services would however be

unscalable. This might be covered by work in the next version of UDDI about adding properties. However, this also has relevance to the stale, short lived data issue described below.

Metadata/Searching

The template model of UDDI queries are generally considered a weakness, especially when addressing WSDL 2.0 inheritance and exposure of suitable WS-ResourceProperties (i.e. non dynamic). A number of e-Science extensions attempt to address this. The general_keywords tModel offers one solution although has its limits especially in regard to range based searching. The next version of UDDI is considering support for OWL based topologies, semantic based searching and range based searching which would address these issues.

Stale Data

The current UDDI models assumes very static long term entries in the registry. However, the e-Science needs are for shorter term, short lived data. Whilst truly dynamic data (e.g. current CPU load) might not be appropriate, there is a need for managing entities and data elements with short life spans such as applying a lifetime or keepalive property. Stale data is an item for deliberation in the next version of UDDI.

Authentication

There is a need to support alternative authentication mechanisms such as certificates and shibboleth. This should be possible in the current versions of UDDI and would form the basis of a technical note.

Contacts

There is a need for more complex contact information. The next version of UDDI will support contacts stored in external directories such as LDAP

Workflow

The relationship between UDDI and workflow was mentioned. SAP are working on a technical note on this issue (in conjunction with the BPEL Technical Committee).

Conclusions

In general UDDI could be used for supporting UK e-Science but there are a number of issues to be addressed:

1. The current Open Source implementations are still in a state of development and do not support version 3.0 – in particular this means that they do not support replication or signing (for authenticity purposes) of entries in the registry
2. Neither the version 3.0 replication specification, nor replication at the database level (for Postgres or MySQL) support peer to peer replication for editing as opposed to a master node having custody for editing purposes.
3. UDDI has poor support for granular access controls or replication of access controls for peer to peer editing
4. UDDI has no mechanisms for tracking provenance of edits
5. UDDI has poor support for short lived data

6. UDDI has a poor searching model for rich semantic data and range searches
7. UDDI has poor support for external contact directories

Of these, concerning (1) - effort is proceeding strongly in the Apache jUDDI project, and any e-Science funded development should contribute to this effort. Whilst UDDI 2.0 can be used in the interim, there is no mechanism for checking the authenticity of any published service, and any replication would rely on database level replication. However, once UDDI version 3.0 implementations are readily available, more exotic replication topologies whereby a private UDDI registry may replicate some services to and from a UK e-Science registry can be further investigated.

It is not however possible to address issues (2) – (7) using either UDDI version 2.0 or version 3.0 without developing proprietary extensions. However these issues are currently being addressed in the next version of the UDDI specification. Any UK e-Science project developing interim proprietary extensions should work closely with the OASIS UDDI Technical Committee. However, the lack of UDDI version 3.0 implementations, despite the fact that UDDI version 3 was published over 12 months ago should be noted, as the lead time between the publication of the next version of UDDI (which is still undetermined) and usable implementations may be long.

Due to the issues mentioned in establishing a working and replicated UDDI registry, performance bench-mark testing and usage trace analyses were not performed within this evaluation, but remain a potential area for future research.

Appendix A

Full source code from UDDI cookbook.

InquireExample.java

```
import org.uddi4j.UDDIException;
import org.uddi4j.datatype.binding.AccessPoint;
import org.uddi4j.response.TModelInfo;
import org.uddi4j.response.TModelList;
import org.uddi4j.transport.TransportException;
import org.uddi4j.transport.TransportFactory;
import org.uddi4j.util.TModelBag;
import org.uddi4j.util.TModelKey;

import uk.ac.cf.giddyjp.UDDIInquirer;

import java.util.Vector;
import java.io.*;

public class InquireExample
{
    static String inquiryURL = "http://uddi.wesc.ac.uk:8334/juddi/inquiry";

    public static void main (String args[])
    throws java.net.MalformedURLException, TransportException, UDDIException
    {
        InquireExample app = new InquireExample();
        app.run();
        System.exit(0);
    }

    public void run()
    throws java.net.MalformedURLException, TransportException, UDDIException
```

```

    {
        // Apache SOAP
        System.setProperty(TransportFactory.PROPERTY_NAME,
            "org.uddi4j.transport.ApacheSOAPTransport");
        // Apache Axis
        //System.setProperty(TransportFactory.PROPERTY_NAME,
        //    "org.uddi4j.transport.ApacheAxisTransport");

        UDDIInquirer inquirer = new UDDIInquirer();
        inquirer.setInquiryURL(inquiryURL);

        TModelList tModelList = inquirer.findSOAPBindings(
            "http://giddyjp.cf.ac.uk/2004/10/foo", "FooBinding");

        TModelBag tModelBag = new TModelBag();
        Vector tModelInfos =
tModelList.getTModelInfos().getTModelInfoVector();
        for (int i = 0; i < tModelInfos.size(); ++i)
        {
            TModelInfo tModelInfo = (TModelInfo)tModelInfos.elementAt(i);
            tModelBag.add(new TModelKey(tModelInfo.getTModelKey()));
        }

        Vector accessPoints = inquirer.findAccessPoints(tModelBag, 10);

        for (int i = 0; i < accessPoints.size(); ++i)
        {
            AccessPoint accessPoint =
(AccessPoint)accessPoints.elementAt(i);
            System.out.println(accessPoint.getText());
        }
    }
}

```

UDDIInquirer.java

```

package uk.ac.cf.giddyjp;

import org.uddi4j.UDDIException;
import org.uddi4j.client.UDDIProxy;
import org.uddi4j.datatype.binding.AccessPoint;
import org.uddi4j.datatype.binding.BindingTemplate;
import org.uddi4j.response.BindingDetail;
import org.uddi4j.response.ServiceInfo;
import org.uddi4j.response.ServiceInfos;
import org.uddi4j.response.ServiceList;
import org.uddi4j.response.TModelList;
import org.uddi4j.transport.TransportException;
import org.uddi4j.util.CategoryBag;
import org.uddi4j.util.FindQualifier;
import org.uddi4j.util.FindQualifiers;
import org.uddi4j.util.KeyedReference;
import org.uddi4j.util.TModelBag;

import java.util.Vector;

public class UDDIInquirer extends UDDIProxy
{
    public TModelList findSOAPBindings(
        String targetNamespace, // WSDL targetNamespace
        String bindingName) // WSDL binding element name
    throws TransportException, UDDIException
    {
        CategoryBag categoryBag = new CategoryBag();
        KeyedReference kr;

        kr = new org.uddi4j.util.KeyedReference("WSDL type", "binding",
            "uuid:6e090afa-33e5-36eb-81b7-1ca18373f457");
    }
}

```

```
categoryBag.add(kr);

kr = new org.uddi4j.util.KeyedReference("binding namespace",
    targetNamespace,
    "uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824");
categoryBag.add(kr);

kr = new org.uddi4j.util.KeyedReference("SOAP protocol",
    "uuid:aa254698-93de-3870-8df3-a5c075d64a0e",
    "uuid:4dc74177-7806-34d9-aecd-33c57dc3a865");
categoryBag.add(kr);

kr = new org.uddi4j.util.KeyedReference("HTTP transport",
    "uuid:68de9e80-ad09-469d-8a37-088422bfbc36",
    "uuid:e5c43936-86e4-37bf-8196-1d04b35c0099");
categoryBag.add(kr);

kr = new org.uddi4j.util.KeyedReference("uddi-org:types",
    "wsdlSpec",
    "uuid:clacf26d-9672-4404-9d70-39b756e62ab4");
categoryBag.add(kr);

TModelList tModelList = find_tModel(bindingName, categoryBag,
    null, null, 1);

return tModelList;
}

public Vector findAccessPoints(
    TModelBag tModelBag, // bag of tModels
    int max) // maximum number of returned
entries
throws TransportException, UDDIException
{
    Vector accessPoints = new Vector();

    FindQualifiers findQualifiers = new FindQualifiers();
    findQualifiers.add(new FindQualifier(FindQualifier.orAllKeys));

    ServiceList serviceList = find_service(null, null, null,
        tModelBag, findQualifiers, max);
    Vector serviceInfos
        = serviceList.getServiceInfos().getServiceInfoVector();

    for (int i = 0; i < serviceInfos.size(); ++i)
    {
        ServiceInfo serviceInfo =
        (ServiceInfo)serviceInfos.elementAt(i);
        String serviceKey = serviceInfo.getServiceKey();
        BindingDetail bindingDetail = find_binding(findQualifiers,
            serviceKey, tModelBag, 1);
        Vector bindingTemplates =
        bindingDetail.getBindingTemplateVector();
        for (int j = 0; j < bindingTemplates.size(); ++j)
        {
            BindingTemplate bindingTemplate
            =
            (BindingTemplate)bindingTemplates.elementAt(j);
            AccessPoint accessPoint = bindingTemplate.getAccessPoint();
            accessPoints.add(accessPoint);
        }
    }
    return accessPoints;
}
}
```

PublishExample.java

```
import org.uddi4j.UDDIException;
import org.uddi4j.datatype.binding.AccessPoint;
import org.uddi4j.response.AuthToken;
import org.uddi4j.transport.TransportException;
import org.uddi4j.transport.TransportFactory;

import uk.ac.cf.giddyjp.UDDIPublisher;

import java.util.Vector;
import java.io.*;

public class PublishExample
{
    static String inquiryURL = "http://uddi.wesc.ac.uk:8334/juddi/inquiry";
    static String publishURL = "http://uddi.wesc.ac.uk:8334/juddi/publish";
    static String userid = "userid";
    static String password = "password";

    public static void main (String args[])
    throws java.net.MalformedURLException, TransportException, UDDIException
    {
        PublishExample app = new PublishExample();
        app.run();
        System.exit(0);
    }

    public void run()
    throws java.net.MalformedURLException, TransportException, UDDIException
    {
        // Apache SOAP
        System.setProperty(TransportFactory.PROPERTY_NAME,
            "org.uddi4j.transport.ApacheSOAPTransport");
        // Apache Axis
        //System.setProperty(TransportFactory.PROPERTY_NAME,
        //    "org.uddi4j.transport.ApacheAxisTransport");

        UDDIPublisher publisher = new UDDIPublisher();
        publisher.setInquiryURL(inquiryURL);
        publisher.setPublishURL(publishURL);
        AuthToken authToken = publisher.get_authToken(userid, password);

        String portTypeRef = publisher.publishPortType(authToken,
            "http://bouscat.cs.cf.ac.uk:8080/test.wsdl",
            "http://giddyjp.cf.ac.uk/2004/10/foo", "FooPortType",
            "Foo Test Service Port Type");

        System.out.println("PortRef " + portTypeRef);

        String bindingRef = publisher.publishSOAPBinding(authToken,
            "http://bouscat.cs.cf.ac.uk:8080/test.wsdl",
            "http://giddyjp.cf.ac.uk/2004/10/foo", "FooBinding",
            "Foo Test Service Binding", portTypeRef);

        System.out.println("Binding " + bindingRef);

        String businessRef = publisher.publishBusiness(authToken,
            "Jon's Foo Business",
            "There's no business like foo business");

        System.out.println("Business " + businessRef);

        String serviceRef = publisher.publishService(authToken,
            businessRef, "Foo Test Service",
            "http://giddyjp.cf.ac.uk/2004/10/foo", "FooService",
            "A service to foo");
    }
}
```

```

        System.out.println("Service " + serviceRef);

        AccessPoint accessPoint = new AccessPoint(
            "http://bouscat.cs.cf.ac.uk:8080/test",    // SOAP
address          "http");                          // URL type

        String templateRef = publisher.publishBindingTemplate(authToken,
            serviceRef, "FooPort", accessPoint, bindingRef,
            portTypeRef);

        System.out.println("Template " + templateRef);
    }
}

```

UDDIPublisher.java

```

package uk.ac.cf.giddyjp;

import org.uddi4j.UDDIException;
import org.uddi4j.client.UDDIProxy;
import org.uddi4j.datatype.OverviewDoc;
import org.uddi4j.datatype.OverviewURL;
import org.uddi4j.datatype.binding.AccessPoint;
import org.uddi4j.datatype.binding.BindingTemplate;
import org.uddi4j.datatype.binding.BindingTemplates;
import org.uddi4j.datatype.binding.InstanceDetails;
import org.uddi4j.datatype.binding.InstanceParms;
import org.uddi4j.datatype.binding.TModelInstanceDetails;
import org.uddi4j.datatype.binding.TModelInstanceInfo;
import org.uddi4j.datatype.business.BusinessEntity;
import org.uddi4j.datatype.service.BusinessService;
import org.uddi4j.datatype.service.BusinessServices;
import org.uddi4j.datatype.tmodel.TModel;
import org.uddi4j.response.AuthToken;
import org.uddi4j.response.BindingDetail;
import org.uddi4j.response.BusinessDetail;
import org.uddi4j.response.BusinessInfo;
import org.uddi4j.response.BusinessInfos;
import org.uddi4j.response.BusinessList;
import org.uddi4j.response.ServiceDetail;
import org.uddi4j.response.TModelDetail;
import org.uddi4j.response.TModelInfo;
import org.uddi4j.response.TModelInfos;
import org.uddi4j.response.TModelList;
import org.uddi4j.transport.TransportException;
import org.uddi4j.transport.TransportFactory;
import org.uddi4j.util.CategoryBag;
import org.uddi4j.util.KeyedReference;

import java.util.Vector;

public class UDDIPublisher extends UDDIProxy
{
    private String publishTModel(AuthToken authToken,
        String wsdlURL, String name, String description,
        CategoryBag categoryBag)
        throws TransportException, UDDIException
    {
        String tModelKey;

        TModelList tModelList = find_tModel(name, categoryBag,
            null, null, 1);

        TModelInfos tModelInfos = tModelList.getTModelInfos();
        if (tModelInfos.size() > 0)

```

```

    {
        TModelInfo tModelInfo = tModelInfos.get(0);
        tModelKey = tModelInfo.getTModelKey();
    }
    else
    {
        TModel tModel = new TModel();
        tModel.setName(name);
        tModel.setDefaultDescriptionString(description);
        tModel.setCategoryBag(categoryBag);

        OverviewDoc overviewDoc = new OverviewDoc();
        overviewDoc.setOverviewURL(wsdlURL);
        tModel.setOverviewDoc(overviewDoc);

        Vector tModels = new Vector();
        tModels.add(tModel);

        TModelDetail tModelDetail = save_tModel(
            authToken.getAuthInfoString(), tModels);

        tModels = tModelDetail.getTModelVector();
        tModel = (TModel)tModels.elementAt(0);
        tModelKey = tModel.getTModelKey();
    }
    return tModelKey;
}

public String publishPortType(
    AuthToken authToken,
    String wsdlURL,           // URL for WSDL document
    String targetNamespace, // WSDL targetNamespace
    String portTypeName,    // WSDL portType element name
    String description)     // human readable description
throws TransportException, UDDIException
{
    CategoryBag categoryBag = new CategoryBag();
    KeyedReference kr;

    kr = new org.uddi4j.util.KeyedReference("WSDL type", "portType",
        "uuid:6e090afa-33e5-36eb-81b7-1ca18373f457");
    categoryBag.add(kr);

    kr = new org.uddi4j.util.KeyedReference("portType namespace",
        targetNamespace,
        "uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824");
    categoryBag.add(kr);

    return publishTModel(authToken, wsdlURL, portTypeName,
        description, categoryBag);
}

public String publishSOAPBinding(
    AuthToken authToken,
    String wsdlURL,           // URL for WSDL document
    String targetNamespace, // WSDL targetNamespace
    String bindingName,     // WSDL binding element name
    String description,     // human readable description
    String portTypeRef)     // UUID returned from
publishPortType
throws TransportException, UDDIException
{
    CategoryBag categoryBag = new CategoryBag();
    KeyedReference kr;

    kr = new org.uddi4j.util.KeyedReference("WSDL type", "binding",
        "uuid:6e090afa-33e5-36eb-81b7-1ca18373f457");
    categoryBag.add(kr);
}

```

```

        kr = new org.uddi4j.util.KeyedReference("binding namespace",
            targetNamespace,
            "uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824");
        categoryBag.add(kr);

        kr = new org.uddi4j.util.KeyedReference("portType reference",
            portTypeRef, "uuid:082b0851-25d8-303c-b332-
f24a6d53e38e");
        categoryBag.add(kr);

        kr = new org.uddi4j.util.KeyedReference("SOAP protocol",
            "uuid:aa254698-93de-3870-8df3-a5c075d64a0e",
            "uuid:4dc74177-7806-34d9-aecd-33c57dc3a865");
        categoryBag.add(kr);

        kr = new org.uddi4j.util.KeyedReference("HTTP transport",
            "uuid:68de9e80-ad09-469d-8a37-088422bfbc36",
            "uuid:e5c43936-86e4-37bf-8196-1d04b35c0099");
        categoryBag.add(kr);

        kr = new org.uddi4j.util.KeyedReference("uddi-org:types",
            "wsdlSpec",
            "uuid:clacf26d-9672-4404-9d70-39b756e62ab4");
        categoryBag.add(kr);

        return publishTModel(authToken, wsdlURL, bindingName,
            description, categoryBag);
    }

    public String publishBusiness(
        AuthToken authToken,
        String businessName,    // human readable business name
        String description)    // human readable description
    throws TransportException, UDDIException
    {
        String businessKey;

        Vector names = new Vector();
        names.add(new org.uddi4j.datatype.Name(businessName, "en"));

        BusinessList businessList = find_business(names, null, null,
            null, null, null, 1);

        BusinessInfos businessInfos = businessList.getBusinessInfos();
        if (businessInfos.size() > 0)
        {
            BusinessInfo businessInfo = businessInfos.get(0);
            businessKey = businessInfo.getBusinessKey();
        }
        else
        {
            BusinessEntity business = new BusinessEntity();
            business.setDefaultNameString(businessName, "en");
            business.setDefaultDescriptionString(description);

            Vector businesses = new Vector();
            businesses.add(business);

            BusinessDetail businessDetail = save_business(
                authToken.getAuthInfoString(), businesses);

            businesses = businessDetail.getBusinessEntityVector();
            business = (BusinessEntity)(businesses.elementAt(0));
            businessKey = business.getBusinessKey();
        }
        return businessKey;
    }
}

```

```
public String publishService(
    AuthToken authToken,
    String businessRef,    // UUID returned from
publishBusiness
    String serviceLabel,  // human readable service name
    String targetNamespace, // WSDL targetNamespace
    String serviceName,   // WSDL service element name
    String description)   // human readable description
throws TransportException, UDDIException
{
    BusinessService businessService = new BusinessService();
    businessService.setDefaultName(
        new org.uddi4j.datatype.Name(serviceLabel));
    businessService.setDefaultDescriptionString(description);
    businessService.setBusinessKey(businessRef);

    CategoryBag categoryBag = new CategoryBag();
    KeyedReference kr;

    kr = new org.uddi4j.util.KeyedReference("WSDL type", "service",
        "uuid:6e090afa-33e5-36eb-81b7-1ca18373f457");
    categoryBag.add(kr);

    kr = new org.uddi4j.util.KeyedReference("service namespace",
        targetNamespace,
        "uuid:d01987d1-ab2e-3013-9be2-2a66eb99d824");
    categoryBag.add(kr);

    kr = new org.uddi4j.util.KeyedReference("service local name",
        serviceName, "uuid:2ec65201-9109-3919-9bec-
c9dbefcaccf6");
    categoryBag.add(kr);

    businessService.setCategoryBag(categoryBag);

    Vector businessServices = new Vector();
    businessServices.add(businessService);

    ServiceDetail serviceDetail = save_service(
        authToken.getAuthInfoString(), businessServices);

    businessServices = serviceDetail.getBusinessServiceVector();
    businessService = (BusinessService)(businessServices.elementAt(0));
    String serviceKey = businessService.getServiceKey();
    return serviceKey;
}

public String publishBindingTemplate(
    AuthToken authToken,
    String serviceRef,    // UUID returned from publishService
    String portName,     // WSDL port element name
    AccessPoint accessPoint, // WSDL address location
    String bindingRef,   // UUID returned from publishBinding
    String portTypeRef) // UUID returned from
publishPortType
throws TransportException, UDDIException
{
    BindingTemplate bindingTemplate = new BindingTemplate();
    bindingTemplate.setServiceKey(serviceRef);
    bindingTemplate.setAccessPoint(accessPoint);

    TModelInstanceDetails tModelInstanceDetails = new
TModelInstanceDetails();
    TModelInstanceInfo tModelInstanceInfo;

    tModelInstanceInfo = new TModelInstanceInfo(bindingRef);
    tModelInstanceInfo.setDefaultDescriptionString("WSDL binding");
}
```

```
InstanceDetails instanceDetails = new InstanceDetails();
InstanceParms instanceParms = new InstanceParms(portName);
instanceDetails.setInstanceParms(instanceParms);
tModelInstanceInfo.setInstanceDetails(instanceDetails);
tModelInstanceDetails.add(tModelInstanceInfo);

tModelInstanceInfo = new TModelInstanceInfo(portTypeRef);
tModelInstanceInfo.setDefaultDescriptionString("WSDL portType");
tModelInstanceDetails.add(tModelInstanceInfo);

bindingTemplate.setTModelInstanceDetails(tModelInstanceDetails);

Vector bindingTemplates = new Vector();
bindingTemplates.add(bindingTemplate);

BindingDetail bindingDetail = save_binding(
    authToken.getAuthInfoString(), bindingTemplates);

bindingTemplates = bindingDetail.getBindingTemplateVector();
bindingTemplate = (BindingTemplate)(bindingTemplates.elementAt(0));
String bindingKey = bindingTemplate.getBindingKey();
return bindingKey;
}
}
```

make1.bat

```
set JDKBIN=%JAVA_HOME%\bin

rem UDDI4J from http://www-124.ibm.com/developerworks/oss/uddi4j/
set UDDILIB=E:\uddi\lib
rem Apache SOAP from http://ws.apache.org/soap/
set SOAPLIB=E:\uddi\soap-2_3_1\lib
rem MYCLASS = location of directory heirarchy containing helper classes
set MYCLASS=E:\uddi

set CLASSPATH=%UDDILIB%\uddi4j.jar;%SOAPLIB%\soap.jar;%MYCLASS%;

%JDKBIN%\javac -classpath "%CLASSPATH%" %1 %2 %3 %4 %5 %6 %7 %8 %9
```

run1.bat

```
set JDKBIN=%JAVA_HOME%\bin

rem UDDI4J from http://www-124.ibm.com/developerworks/oss/uddi4j/
set UDDILIB=E:\uddi\lib
rem Apache SOAP from http://ws.apache.org/soap/
set SOAPLIB=E:\uddi\soap-2_3_1\lib
rem ...requires JavaMail from http://java.sun.com/products/javamail/
set MAILLIB=E:\uddi\javamail-1.3.2\lib
rem ...and JAF from http://java.sun.com/products/javabeans/glasgow/jaf.html
set JACTLIB=E:\uddi\jaf-1.0.2
rem MYCLASS = location of directory heirarchy containing helper classes
set MYCLASS=E:\uddi

set
CLASSPATH=%UDDILIB%\uddi4j.jar;%SOAPLIB%\soap.jar;%MAILLIB%\mailapi.jar;%JACTLIB%\activation.jar;%MYCLASS%;

%JDKBIN%\java -Dorg.uddi4j.logEnabled=true -classpath %CLASSPATH% %1 %2 %3
%4 %5 %6 %7 %8 %9
```