JBoss ESB 4.2 Milestone Release 1

Registry Service

Legal Notices

The information contained in this documentation is subject to change without notice.

JBoss Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. JBoss Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Java™ and J2EE is a U.S. trademark of Sun Microsystems, Inc. Microsoft® and Windows NT® are registered trademarks of Microsoft Corporation. Oracle® is a registered U.S. trademark and Oracle9™, Oracle9 Server™ Oracle9 Enterprise Edition™ are trademarks of Oracle Corporation. Unix is used here as a generic term covering all versions of the UNIX® operating system. UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Copyright

Software Version

JBoss ESB 4.2 Milestone Release 1

Restricted Rights Legend

Contents

Table of Contents

About This Guide

What This Guide Contains

The Registry Service contains contain important information on changes to JBoss ESB 4.2 Milestone Release 1 since the last release and information on any outstanding issues.

Audience

This guide is most relevant to engineers who are responsible for administering JBoss ESB 4.2 Milestone Release 1 installations.

Prerequisites

None.

Organization

This guide contains the following chapters:

- **Chapter 1, Configuration:** this chapter contains information on how to configure the registry

- **Chapter 2, Example Configurations:** this chapter illustrates some useful configurations..

- **Chapter 3, Troubleshooting**: information covering how to diagnose and fix common problems.

Documentation Conventions

The following conventions are used in this guide:

Convention	Description
Italic	In paragraph text, italic identifies the titles of documents that are being referenced. When used in conjunction with the Code text described below, italics identify a variable that should be replaced by the user with an actual value.
Bold	Emphasizes items of particular importance.
`Code`	Text that represents programming code.
Function | Function	A path to a function or dialog box within an interface. For example, "Select File | Open." indicates that you should select the Open function from the File menu.
( ) and |	Parentheses enclose optional items in command syntax. The vertical bar separates syntax items in a list of choices. For example, any of the following three items can be entered in this syntax: `persistPolicy (Never | OnTimer | OnUpdate | NoMoreOftenThan)`
Note: **Caution**:	A note highlights important supplemental information. A caution highlights procedures or information that is necessary to avoid damage to equipment, damage to software, loss of data, or invalid test results.

Table 1 Formatting Conventions

Additional Documentation

In addition to this guide, the following guides are available in the JBoss ESB 4.2 Milestone Release 1 documentation set:

1. **JBoss ESB 4.2 Milestone Release 1** *Trailblazer Guide*: Provides guidance for using the trailblazer example.

2. **JBoss ESB 4.2 Milestone Release 1** *Getting Started Guide*: Provides a quick start reference to configuring and using the ESB.

3. **JBoss ESB 4.2 Milestone Release 1** *Programmers Guide*: How to use JBossESB.

4. **JBoss ESB 4.2 Milestone Release 1** *Release Notes*: Information on the differences between this release and previous releases.

5. **JBoss ESB 4.2 Milestone Release 1** *Administration Guide*: How to manage the ESB.

Contacting Us

Questions or comments about JBoss ESB 4.2 Milestone Release 1 should be directed to our support team.

Configuring the Registry

Introduction

The JBossESB Registry architecture allows for many ways to configure the ESB to use either a Registry or Repository. By default the we use a JAXR implementation (Scout) and a UDDI (jUDDI), in an embedded way.

The following properties can be used to configure the JBossESB Registry. In the jbossesb-properties.xml there is section called 'registry':

```
<properties name="registry">
     <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
     <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
     <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="org.apache.juddi.registry.local.InquiryService#inquire"/>
     <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="org.apache.juddi.registry.local.PublishService#publish"/>
     <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
     <property name="org.jboss.soa.esb.registry.password"
value="password"/>
     <!-- the following parameter is scout specific to set the type of
communication between scout and the UDDI (embedded, rmi, soap) -->
     <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.LocalTransport"/>
    </properties>
```

In short, the properties are

1. org.jboss.soa.esb.registry.implementationClass, a class that implements the jbossesb Registry interface. We have provided one implementation (JAXRRegistry interface).

2. org.jboss.soa.esb.registry.factoryClass, the class name of the JAXR ConnectionFactory implementation.

3. org.jboss.soa.esb.registry.queryManagerURI, the URI used by JAXR to query.

4. org.jboss.soa.esb.registry.lifeCycleManagerURI, the URI used by JAXR to edit.

5. org.jboss.soa.esb.registry.user, the user used for edits.

6. org.jboss.soa.esb.registry.password, the password to go along with the user.

7. org.jboss.soa.esb.scout.proxy.transportClass, the name of the class used by scout to do the transport from scout to the UDDI.

The components involved

The registry can be configured in many ways. Figure 1 shows a blue print of all the registry components. From the top down we can see that the JBossESB funnels all interaction with the registry through the Registry Interface. By default it then calls into a JAXR implementation of this interface. The JAXR API needs an implementation, which by default is Scout. The Scout JAXR implementation calls into a jUDDI registry. However there are many other configuration options.

Figure 1. Blue print of the Registry component architecture.

The Registry Implementation Class

Property: org.jboss.soa.esb.registry.implementationClass

By default we use the JAXR API. The JAXR API is a convenient API since it allows us to connect any kind of XML based registry or repository. However, if for example you want to use Systinet's Java API you can do that by writing your own SystinetRegistryImplentation class and referencing it in this property.

Using JAXR

Propery: `org.jboss.soa.esb.registry.factoryClass`

If you decided to use JAXR then you will have to pick which JAXR implementation to use. This property is used to configure that class. By default we use Scout and therefore it is set to the scout factory `'org.apache.ws.scout.registry.ConnectionFactoryImpl'`. The next step is to tell the JAXR implementation the location of the registry or repository for querying and updating, which is done by setting the `org.jboss.soa.esb.registry.queryManagerURI`, and `org.jboss.soa.esb.registry.lifeCycleManagerURI` respectively, along with the username (`org.jboss.soa.esb.registry.user`) and password (`org.jboss.soa.esb.registry.password`) for the UDDI.

Using Scout and jUDDI

Property: `org.jboss.soa.esb.scout.proxy.transportClass`

When using Scout and jUDDI there is an additonal parameter that one can set. This is the transport class that should be used for communication between Scout and jUDDI. Thus far there are 4 implementations of this class which are based on SOAP, SAAJ, RMI and Local (embedded java). Note that when you change the transport, you will also have to change the query and lifecycle URIs. For example:

SOAP
```
queryManagerURI         http://localhost:8080/juddi/inquiry
lifeCycleManagerURI     http://localhost:8080/juddi/publish
transportClass          org.apache.ws.scout.transport.AxisTransport
```
RMI
```
queryManagerURI
jnp://localhost:1099/InquiryService?org.apache.juddi.registry.rmi.Inqui
ry#inquire
lifeCycleManagerURI
jnp://localhost:1099/PublishService?org.apache.juddi.registry.rmi.Publi
sh#publish
transportClass          org.apache.ws.scout.transport.RMITransport
```
Local
```
queryManagerURI
      org.apache.juddi.registry.local.InquiryService#inquire
lifeCycleManagerURI
      org.apache.juddi.registry.local.PublishService#publish
transportClass          org.apache.ws.scout.transport.LocalTransport
```

For jUDDI we have two requirements that need to be fulfilled:

1. access to the juddi database. You will need to create a schema in your database, and add the jbossesb publisher. The product/install/jUDDI-registry directory contains db create scripts for you favority database.

2. juddi.properties. The configuration of jUDDI itself. If you do not use a datasource you need to take special care to set the following properties:

```
juddi.isUseDataSource=false
juddi.jdbcDriver=com.mysql.jdbc.Driver
```

```
juddi.jdbcUrl=jdbc:mysql://localhost/juddi
juddi.jdbcUsername=juddi
juddi.jdbcPassword=juddi
```

if you do use a datasource you need something like

```
juddi.isUseDataSource=true
juddi.dataSource=java:comp/env/jdbc/juddiDB
```

Configuration Examples

Introduction

As mentioned before, by default the JBossESB is configured to use the JAXR API using Scout as its implementation and jUDDI as the registry. Here are some example how you can use deploy this combo.

Embedded

All ESB components (with components we really mean JVMs in this case) can embed the registry and they all can connect to the same database (or different once if that makes sense).
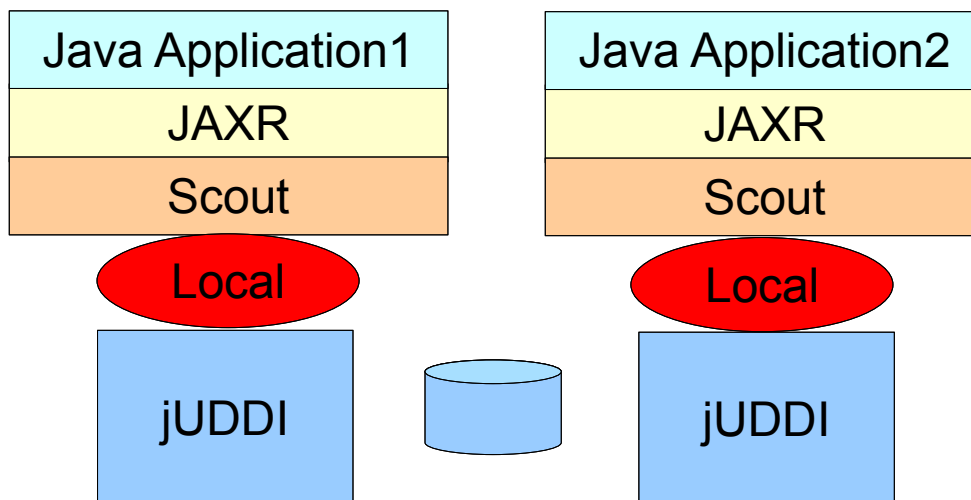

Figure 2. Embedded jUDDI.

Properties example:

```
<properties name="registry">
     <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
     <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
     <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="org.apache.juddi.registry.local.InquiryService#inquire"/>
     <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="org.apache.juddi.registry.local.PublishService#publish"/>
     <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
```

```
        <property name="org.jboss.soa.esb.registry.password"
value="password"/>
        <!-- the following parameter is scout specific to set the type of
communication between scout and the UDDI (embedded, rmi, soap) -->
        <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.LocalTransport"/>
    </properties>
```

RMI using the juddi.war or jbossesb.sar

Deploy a version of the jUDDI that brings up an RMI service. The JBossESB comes
with a juddi.war in the product/install/jUDDI-registry directory. This war brings up
the regular webservices but also an RMI service. Along with the juddi.war you need
to deploy a datasource which points to your jUDDI database. An example file is
supplied for mysql. Note that the jbossesb.sar also registers a rmi service. So you'd
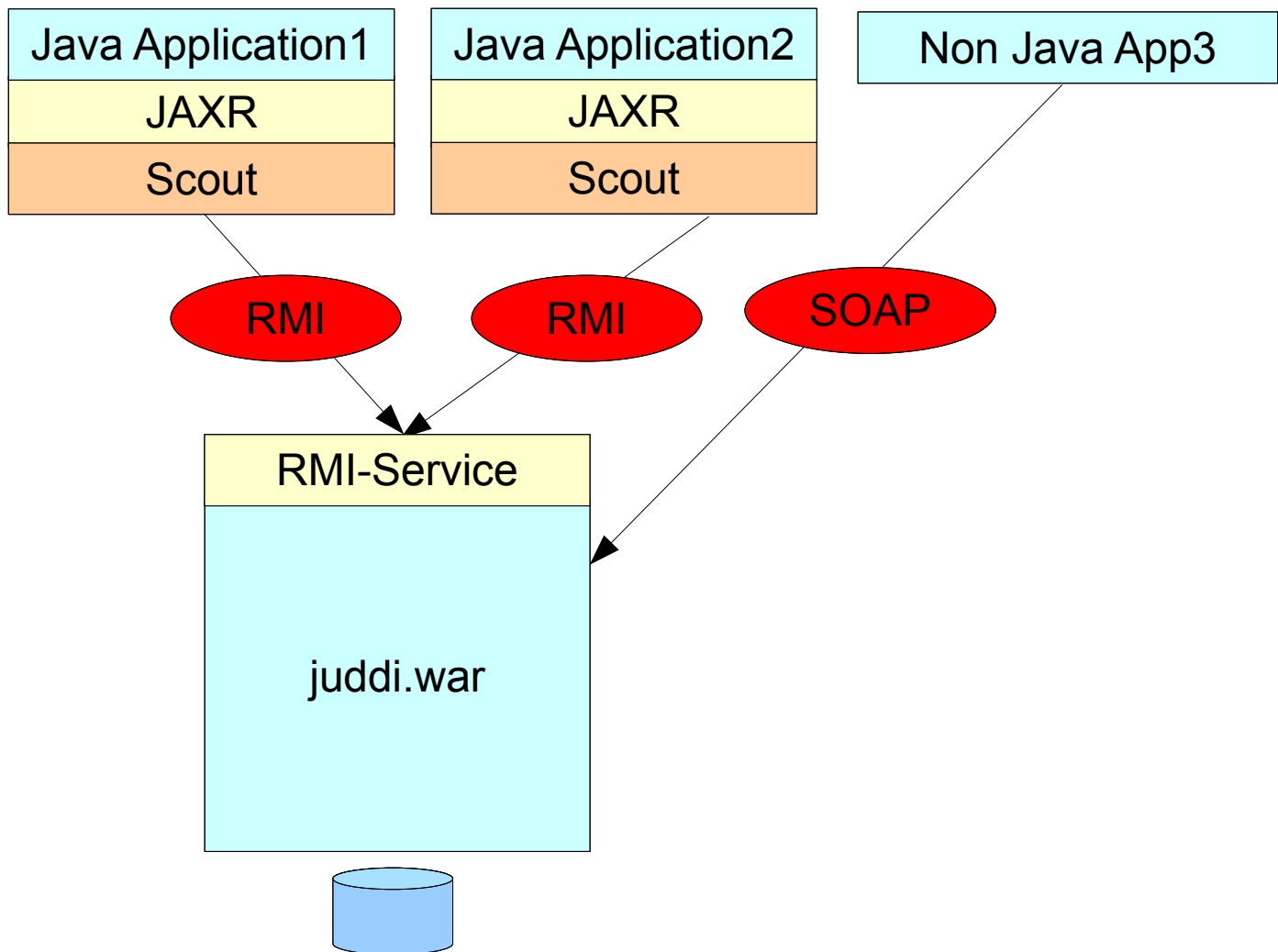only need to deploy the juddi.war if you need webservice access.

Figure 3. RMI using the juddi.war

Properties example:

```
<properties name="registry">
        <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
        <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
        <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="jnp://localhost:1099/InquiryService?org.apache.juddi.registry.rm
i.Inquiry#inquire"/>
        <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="jnp://localhost:1099/PublishService?org.apache.juddi.registry.rm
i.Publish#publish"/>
        <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
        <property name="org.jboss.soa.esb.registry.password"
value="password"/>
        <!-- the following parameter is scout specific to set the type of
communication between scout and the UDDI (embedded, rmi, soap) -->
        <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.RMITransport"/>
```

The juddi.war is configured to bring up a RMI Service, which is triggered by the following setting in the web.xml

```
<!--  uncomment if you want to enable making calls in juddi with rmi
-->
  <servlet>
        <servlet-name>RegisterServicesWithJNDI</servlet-name>
        <servlet-
class>org.apache.juddi.registry.rmi.RegistrationService</servlet-class>
        <load-on-startup>1</load-on-startup>
  </servlet>
```

Make sure to include -for example- the following JNDI settings in your juddi.properties:

```
# JNDI settings (used by RMITransport)
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.provider.url=jnp://localhost:1099
java.naming.factory.url.pkgs=org.jboss.naming
```

Note: the RMI clients need to have the scout-client.jar in their classpath.

RMI using your own JNDI Registration of the RMI Service

If you don't want to deploy the juddi.war you can setup one of the ESB components that runs in the the same JVM as jUDDI to register the RMI service.While the other applications need to be configured to use RMI.
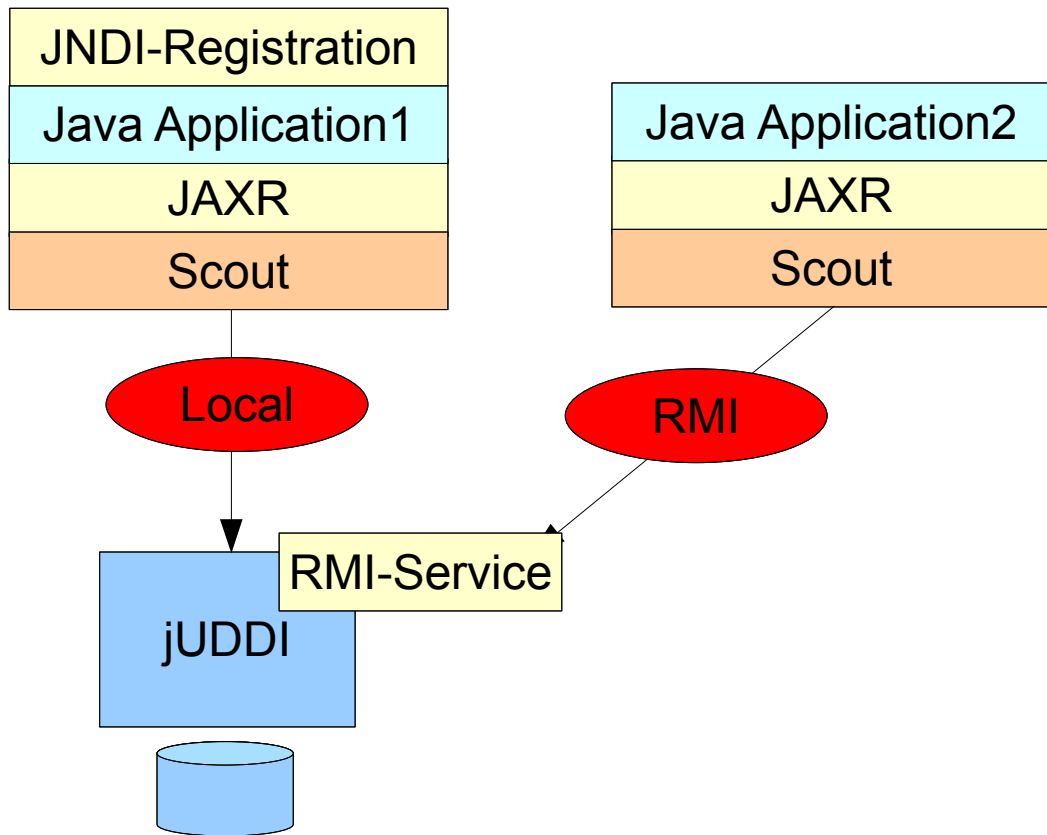
Figure 4. RMI using your own JNDI registration

Properties example: For application 1 you need need the Local settings:

```
<properties name="registry">
      <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
      <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
      <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="org.apache.juddi.registry.local.InquiryService#inquire"/>
      <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="org.apache.juddi.registry.local.PublishService#publish"/>
      <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
      <property name="org.jboss.soa.esb.registry.password"
value="password"/>
      <!-- the following parameter is scout specific to set the type of
communication between scout and the UDDI (embedded, rmi, soap) -->
      <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.LocalTransport"/>
    </properties>
```

while for application2 you need the RMI settings:

```
<properties name="registry">
      <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
      <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
      <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="jnp://localhost:1099/InquiryService?org.apache.juddi.registry.rm
i.Inquiry#inquire"/>
      <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="jnp://localhost:1099/PublishService?org.apache.juddi.registry.rm
i.Publish#publish"/>
      <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
      <property name="org.jboss.soa.esb.registry.password"
value="password"/>
      <!-- the following parameter is scout specific to set the type of
communication between scout and the UDDI (embedded, rmi, soap) -->
      <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.RMITransport"/>
```

> Where the hostname of the queryManagerURI and lifeCycleManagerURI need to
> point to the hostname on which jUDDI is running (which would be where
> application1 is running). Obviously application1 needs to have access to a naming
> service. To do the registration process you need to do something like:

```
//Getting the JNDI setting from the config
String factoryInitial = Config.getStringProperty(
Properties env = new Properties();
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_INITIAL,factoryInit
ial);
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_PROVIDER_URL, providerURL);
env.setProperty(RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_URL_PKGS,
factoryURLPkgs);
log.info("Creating Initial Context using: \n"
 + RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_INITIAL + "=" + factoryInitial +
"\n"
 + RegistryEngine.PROPNAME_JAVA_NAMING_PROVIDER_URL   + "=" + providerURL +
"\n"
 + RegistryEngine.PROPNAME_JAVA_NAMING_FACTORY_URL_PKGS + "=" + factoryURLPkgs
+ "\n");
InitialContext context = new InitialContext(env);
Inquiry inquiry = new InquiryService();
log.info("Setting " + INQUIRY_SERVICE + ", " + inquiry.getClass().getName());
mInquery = inquiry;
context.bind(INQUIRY_SERVICE, inquiry);
Publish publish = new PublishService();
log.info("Setting " + PUBLISH_SERVICE + ", " + publish.getClass().getName());
mPublish = publish;
context.bind(PUBLISH_SERVICE, publish);
```

2.4 SOAP

> Finally, you can make the communication between Scout and jUDDI SOAP based.
> Again you need to deploy the juddi.war and configure the datasource. You probably
> want to shutdown the RMI service by commenting out the
> `RegisterServicesWithJNDI` servlet in the web.xml.
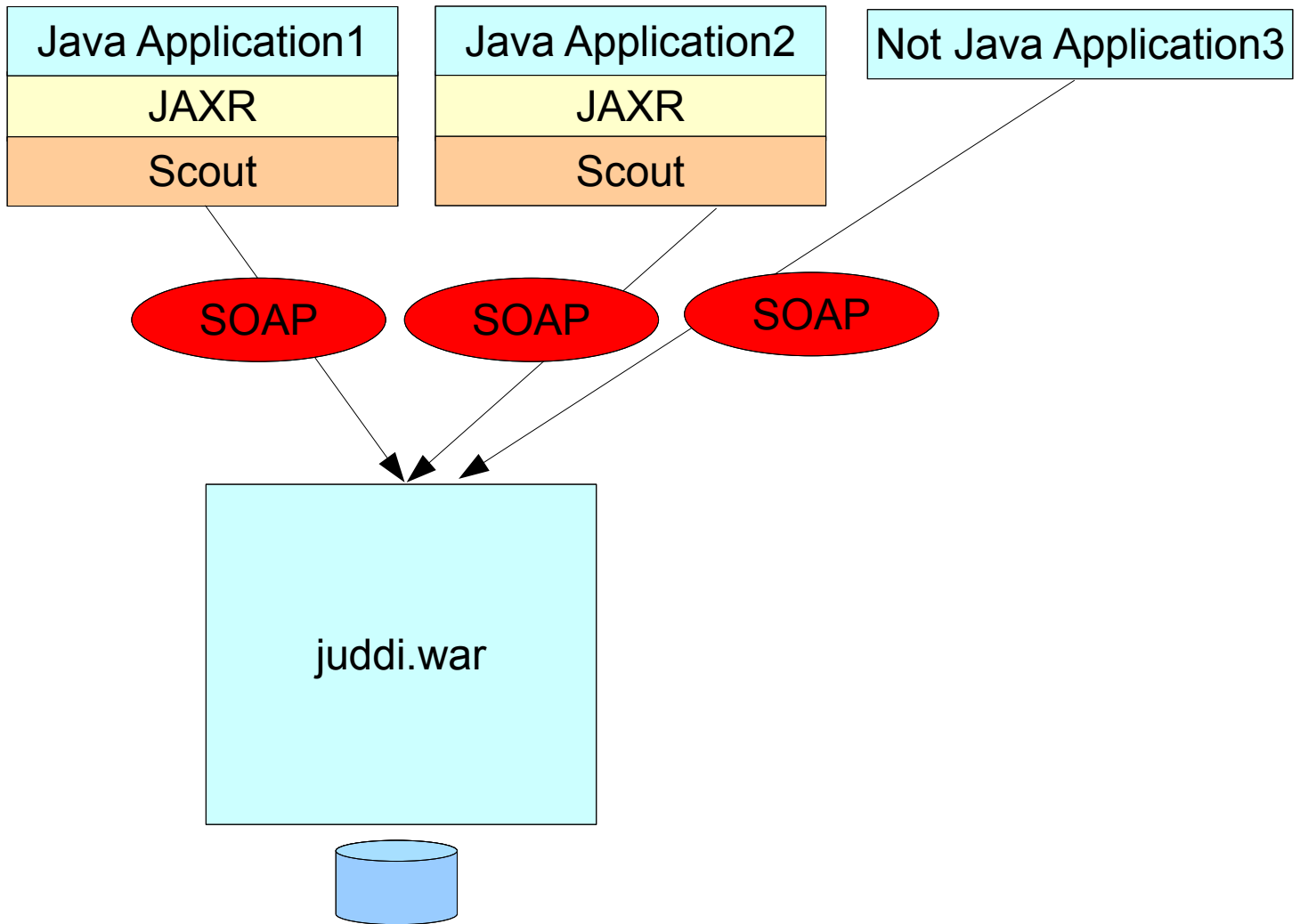
Figure 5. SOAP.

Properties example:

```
<properties name="registry">
      <property name="org.jboss.soa.esb.registry.implementationClass"
value="org.jboss.internal.soa.esb.services.registry.JAXRRegistryImpl"/>
      <property name="org.jboss.soa.esb.registry.factoryClass"
value="org.apache.ws.scout.registry.ConnectionFactoryImpl"/>
      <property name="org.jboss.soa.esb.registry.queryManagerURI"
value="http://localhost:8080/juddi/inquiry"/>
```

```
        <property name="org.jboss.soa.esb.registry.lifeCycleManagerURI"
value="http://localhost:8080/juddi/publish"/>
        <property name="org.jboss.soa.esb.registry.user"
value="jbossesb"/>
        <property name="org.jboss.soa.esb.registry.password"
value="password"/>
        <!-- the following parameter is scout specific to set the type of
communication between scout and the UDDI (embedded, rmi, soap) -->
        <property name="org.jboss.soa.esb.scout.proxy.transportClass"
value="org.apache.ws.scout.transport.AxisTransport"/>
```

SAAJ

JBoss 4.0.x comes with scout and juddi. If you run in clustered mode ('all'). It brings up the jUDDI registry to which you can communicate using SAAJ. This is an untested feature.

Troubleshooting

Scout and jUDDI pitfalls

● Make sure to put our version of the jaxr-api-1.0.jar, scout-0.7rc2-embedded.jar and the juddi-embedded.jar first. Other versions of these libraries are present in the JbossAS libraries and they are, for the time being, incompatible. This should get resolved in future release of the Application Server.

● If you use RMI you need the juddi-client.jar.

● Make sure the jbossesb-properties.xml file is on the classpath and read or else the registry will try to instantiate classes with the name of 'null'.

● Make sure you have a juddi.properties file on your classpath for jUDDI to configure itself.

● Make sure to read the README in the product/install/jUDDI-registry directory, to prepopulate your own juddi database.

More Information

● For more information see the wiki http://labs.jboss.com/wiki/JudyEvaluation

● JBossESB user forum: http://www.jboss.com/index.html?module=bb&op=viewforum&f=246

● When you deploy the juddi.war that ship with our esb, you enable access through soap, which means that you can use any uddi brower. You can use http://uddibrowser.org