// Delete comments before final submission.
// make sure that the response fits within 80 columns
01234567890123456789012345678901234567890123456789012345678901234567890123456789

1. Introduction

    1.1. Project/Component Working Name:

        JBI Integration with appserver.
        (Belongs to umbrella arc case WSARC/2006/586)

    1.2. Name(s) and e-mail address of Document Author(s)/Supplier:

        Binod PG: binod.pg@sun.com
        Keith Babo : Keith.babo@sun.com

    1.3. Date of This Document:

        8th December 2006.

2. Project Summary

    2.1. Project Description:

        For growing the community around jbi component developers and for
        providing better integrated middle offering to the customers,
        JBI will be integrated with appserver by default.

        Administration integration, Runtime Integration and EE (clustering)
        integration of the JBI runtime and the components are the main
        areas targetted for this release of appserver.

    2.2. Risks and Assumptions:

        The project depends on JBI team meeting the appserver
        schedule and integration requirements.

3. Problem Summary

    3.1. Problem Area:

        o JBI runtime is well integrated only to Java EE SDK.

    3.2. Justification:

        o It is important to make sure that Sun's middleware software
          products are well integrated together.

4. Technical Description:

     4.1. Details:

        Basic JBI runtime, SOAP binding component and WSDL shared library

wil be integrated to appserver by default.

4.1.1. Installation and Packaging.

The JBI will be installed directly by the appserver installer. Most of the JBI files will be installed in AS_HOME/jbi directory. This directory will contain all common jar files and system components of the JBI. This will also include the lifecycle module that start the JBI framework in the appserver JVM.

In Java EE 5 SDK, when JBI gets installed on top of appserver as an addon, it uses DOMAIN_ROOT/jbi as the location to save all the jbi application bits (engines, bindings, shared libraries etc).

As part of the tighter integration in Appserver 9.1, all the default components like SOAP BC, WSDL shared library, Java EE service engine etc will be considered as system components and will be installed within AS_HOME/jbi itself. Thus DOMAIN_ROOT/jbi will be almost empty on a fresh installation of appserver except the directories and the JBI registry.

Any user application (service assembly, service engine etc) will be created in the DOMAIN_ROOT/applications/composite-applications directory. In case of standalone instances or cluster instances, INSTANCE_ROOT/applications/composite-applications will be used as the application directory.

4.1.2. Startup.

JBI runtime will be integrated to appserver runtime as a lifecycle module. This will be a user defined lifecycle module so that it can be turned on/off by user. By default it will be switched on.
The actual lifecycle module implementation will reside in a jar file in the AS_HOME/jbi directory.

When lifecycle module starts during appserver startup, the JBI framework will inspect the contents of JBI registry in INSTANCE_ROOT/jbi and check for any installed user components or service assemblies. If there is none, only a minimal JBI core service will be started to listen for user events.

On the first user-action on the JBI framework (eg. deployment of service assembly) , the rest of JBI framework and system components will be initialized.

4.1.3. Administration integration.

JBI administration GUI will be a node in the default appserver admin GUI. The basic GUI code will be checked in to appserver cvs repository itself. The jbi admin GUI code will interact with the jbi common client library. The jbi common client library is the basic interface used by all jbi tools to contact with the jbi administration runtime.

jbi admin cli commands will be developed based on the CLI framework cookbook document. The CLI descriptor xml file will be modified to include the jbi admin cli commands. The asadmin scripts will be modified to include the jbi commands archive. See interfaces and reference documents section for more details.

Administration operations in the GUI and CLI will support all appserver targets (server, domain and cluster).

4.1.4 Transactions.

JBI components like BPEL engine will be capable of handling transactions. JSR 208 allows JTA transaction usage in the components. It also allows to pass the transaction object between components. JBI components will use javax.transaction.Transaction and javax.transaction.TransactionManager interfaces for transaction operations like enlistment, delistment, suspend, resume, commit, rollback etc.

Java EE Service Engine (JSE) will be enhanced to handle transactions. In case of inbound communication, JSE will execute the EJBs and servlets under the same transaction passed by JBI.

In case of outbound communication from a Java EE application, the current transaction of the Java EE application will be passed to JBI. Both these will enable stronger integration of Java EE with JBI.

A new interface will be exposed by appserver to obtain XAResource

objects from JBI components for recovery. During recovery, application
server will use this interface to retrieve the XAResource objects
from JBI components.

```
public class RecoveryResourceRegistry {
     public static RecoveryResourceRegistry getInstance();
     public void
addRecoveryResourceListener(com.sun.enterprise.resource.
     RecoveryResourceListener);
     public java.util.Set getListeners();
}
```

This is a singleton registry class, that will be used by jbi lifecycle
to register a listener (RecoveryResourceListener) class.

```
public interface RecoveryResourceListener{
     javax.transaction.xa.XAResource[] getXAResources();
     void recoveryStarted();
     void recoveryCompleted();
}
```

Appserver will use the RecoveryResourceListener instance from the jbi
to obtain the XAResource objects for recovery. It also gives two
recovery lifecycle operations that can potentially be used for
some house keeping operations.

4.1.4 EE and Clustering.

Each server instance, whether it is standalone, DAS (Domain
Adminisistration Server) or clustered will contain a JBI runtime.
DAS will contain Facade mbeans that will be communicating to the
cluster instances or standalone instances based on the target
information.

Basic JBI runtime will not have any special capability to handle
clustering. The components on top of it will be aware of EE clustering.
So, it will be the component's responsibility to work in a clustered
manner. For example, a BPEL process doing a correlation will need the
messages to routed to the same instance. However this will be done
in a way specific to the component. For example, BPEL engine
deployment to a cluster will use a group ID to identify the cluster.
This can be the cluster name.

Load balancing to the EE cluster will be handled in a component specific way. For example, the SOAP BC, will use HTTP load balancer.
and the JMS BC will use inbound JMS load balancing.

4.1.5 Java EE Service Units.

At present when a user compose a composite application, he includes
all other service units (BPEL app, SOAP BC deployment, XSLT engine)
in the service assembly. However service units developed as Java EE
applications will not be included and instead they are deployed separately from the service assembly and they work with the NMR (Normalized Message Router of JBI) transparently.

By allowing the Java EE applications to be included in the service assembly, it is possible for packaging composite application as one
archive, including the Java EE application. This will provide a better
experience while using netbeans.

Also, when Java EE applications are part of service assembly, the Java
EE service engine receives the lifecycle events (deploy/undeploy/start
/stop etc) from the NMR. This will enable the service engine to do necessary house keeping (eg. removing the WSDL object cache, while undeploying the service assembly)

This would also enable better management/monitoring capabilities for
the Java EE service units by the JBI tools.

Description of the packaging

The contents of the service assembly(StockPlanCompositeApp.zip) would
look like the following. StockWSApplication.jar is a Java EE app.
.
./META-INF
./META-INF/MANIFEST.MF
./META-INF/jbi.xml
./StockPlanner@BPELSEDeployment.jar
./com.sun.httpsoapbc-1.0-2@BCDeployment.jar
./StockWSApplication.jar

The content of ./META-INF/jbi.xml of the service assembly would

contain the follwing information service unit information for
Java EE application as follows.

```
<service-unit>
   <identification>
     <name>StockWSJavaEEUnit</name>
     <description>Represents this Service Unit</description>
   </identification>
   <target>
     <artifacts-zip>StockWSApplication.jar</artifacts-zip>
     <component-name>JavaEEServiceEngine</component-name>
   </target>
</service-unit>
```

The StockWSApplication.jar is the Java EE application. Note that, it
would also be possible to bundle a ear or war as service unit as well.

The Java EE application, in it's META-INF directory will contain
another jbi.xml as per JSR 208 spec. Java EE Service Engine will not
use any information from this jbi.xml

Deployment

The name of the Java EE archive will be prepended with the name of
the service assembly. A Java EE archive that is deployed as part
of service assembly cannot be managed from appserver's admin GUI
or CLI. The property element in the j2ee-application (and other
modules) will be leveraged for this purpose. The Java EE Service
engine will set a property name/value pair
"externally-managed"/true to indicate that it is an externally
managed application.

4.2. Bug/RFE Number(s):

4.3. In Scope:

1) Installation.
2) Administration Integration.
3) Runtime Integration.
4) Transaction Support.
5) Package Java EE applications in Service Assembly.
6) Clustering support in JCAPS components.

4.4. Out of Scope:

1) Failover of composite applications.

4.5. Interfaces:

// http://www.opensolaris.org/os/community/arc/policies/interface-taxonomy/

// describes the permitted interface taxonomy.

4.5.1 Exported Interfaces

1. Interface:
   JBI interfaces defined by JSR 208 specification.
   Stability:
   Committed
   Comments:
   This will be exposed by JBI integrated with the appserver.
2. Interface:
   API to obtain XAResource objects from JBI.
   Stability:
   Contracted Project Private.
   Comments:
   If a similar functionality is added to JTA specification, then
   this interface can be removed. Also, this interface will be
   a project-private interface used only by JBI.
3. Interface:
   JBI Administration CLI and GUI.
   Stability:
   Uncommitted.
   Comments:
   This is explained in section 4.7 of this one pager.

4.5.2 Imported interfaces

1. Interface:
   JBI interfaces(JBI API, jbi.xml dtd etc) as specified by JSR 208.
   Stability:
   Committed.
   Exporting Project:
   JSR 208 specification.
   Comments:
   This is used by Java EE Service Engine.
2. Interface:
   JBI common client API
   (http://www.glassfishwiki.org/jbiwiki/Wiki.jsp?page=Glassfish9.1CommonClient)
   Stability:
   Contracted Project Private.
   Exporting Project:
   JBI.
   Comments:

This is used by Administration GUI and CLI of appserver.
3. Interface:
CLI framework.
http://sac.sfbay.sun.com/WSARC/2005/166/
Stability:
Contracted Project Private.
Comments:

4.6. Doc Impact:

Most of the parts of appserver documentation will have an impact. Administration Guide, Developer's Guide and CLI man pages will need to include the JBI specific details or pointers to JBI documentation.

4.7. Admin/Config Impact:
// How will this change impact the administration of the product?
// Identify changes to GUIs, CLI, agents, plugins...

Template will be modified to include jbi lifecycle module.
The jbi administration will be an integral part of appserver administration. This is explained in the following docs.

jbi CLI commands.
http://www.glassfishwiki.org/jbiwiki/Wiki.jsp?page=Glassfish9.1JBICLIProjectPlan
http://www.glassfishwiki.org/jbiwiki/Wiki.jsp?page=TableListingJBICommandSet

jbi admin GUI.
http://www.glassfishwiki.org/jbiwiki/attach/Glassfish9.1WebConsolePEProjectPlan/projplan-jbiadmin-jsf-pe.pdf

4.8. HA Impact:

Explained in section 4.1.4 of the document.

4.9. I18N/L10N Impact:

4.10. Packaging & Delivery:
// What packages, clusters or metaclusters does this proposal
// impact? What is its impact on install/upgrade?

Main directories of jbi within appserver file layout are
1. AS_HOME/jbi : This directory will contain all the installation wide data like famework jar file, lifecycle module, system components etc.
2. DOMAIN_ROOT/jbi: This directory will contain all the data specific to a domain.

3. INSTANCE_ROOT/jbi: This directory will contain all the data specific to each instance.

More information on the file layout can be found at:
http://www.glassfishwiki.org/jbiwiki/Wiki.jsp?page=FileSystemLayout

Since JBI will be integrated to appserver, the JBI files will be added to appserver packages (SUNWasu etc).

4.11. Security Impact:

The server.policy will be modified to give permissions for all jbi libraries and composite applications.

More about this issue is available at:
http://www.glassfishwiki.org/jbiwiki/Wiki.jsp?page=JBISecurity

4.12. Compatibility Impact

Appserver will need to pass JBI (JSR 208) TCK (Technology compatibitity kit
from compatibility team).

4.13. Dependencies:

This project depends on the JBI schdule.

5. Reference Documents:

JBI documents.
http://www.glassfishwiki.org/jbiwiki/Wiki.jsp?page=GlassfishJbiIntegration

JSR 208 spec.
http://jcp.org/en/jsr/detail?id=208

Appserver 9.1 ARC case.
http://sac.sfbay.sun.com/WSARC/2006/586

6. Schedule:
6.1. Projected Availability:

This will be available when applserver 9.1 is released.