01234567890123456789012345678901234567890123456789012345678901234567890123456789

1. Introduction

1.1. Project/Component Working Name:
Modularization of GlassFish using OSGi

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

1.3. Date of This Document:
09 July 2008

2. Project Summary

2.1. Project Description:
OSGi [1] module system will be used to implement the primary features of GlassFish V3 release, such as: modularity, extensibility, and embeddability.

2.2. Risks and Assumptions:
Making the monolithic code modular is the biggest risk, as it involves significant redesign of core functionalities. Existing GlassFish implementation relies heavily on a single class loader hierarchy. OSGi hardly uses such parent delegation model for class loading. Adopting the new class loading scheme can be a challenge. More over, it can lead to some backward incompatibility. We will try to minimize incompatibility, if not completely avoid it.

3. Problem Summary

3.1. Problem Area:
Modularity, extensibility and embeddability are the need of the hour. As Java EE stack matured over time, implementations that include the entire stack were perceived as heavy weight solutions. Not every application uses all the functionalities offered by the stack. Users do not want to pay the runtime cost for features that they don't use. In order to cater to the needs of different classes of applications and their varying need for the stack, we have to come up with a solution that allows us to quickly build customized stack. Making it modular would also allow substitutability of parts of the stack as well giving greater flexibility to user.

Secondly, we would like the implementation to be extensible so that capabilities of the system can be augmented after the system is shipped/installed. This is important for the targeted use case where user starts with a reduced set of requirement, and their requirements grows over time necessacitating installation of newer capabilities in the system. Such extensions does not have to be necessarily provided by GlassFish project team, so there is a need to expose well defined extension points as well.

Thirdly, there is a need for embedding GlassFish in existing runtimes to augment their capabilities for greater adoption of GlassFish.

We propose to use OSGi as the module system in GlassFish to achieve the aforementioned goals.

In addition to these, we shall investigate the use of OSGi by applications deployed in GlassFish, as we think there is definite value addition for applications in this model.

So far, we have been exposing implementation classes to applications. Use of OSGi will allow GlassFish to provide isolated class space for applications.

3.2. Justification:

```
      Please see the "GlassFish V3 Prelude Overview Functional Spec" [1] for
      justification.

4. Technical Description:
     4.1. Details:
      We propose to implement GlassFish as a set of OSGi bundles. A bundle
      is a unit of deployment in OSGi. It is packaged as a Java ARchive (jar)
      with proper manifest entries. The manifest entries define the essential
      details about the module, e.g., name, version, its dependencies and
      capabilities. At the heart (core) of GlassFish, there will be an OSGi
      framework. Although we plan to use Apache Felix [6] as the default OSGi
      framework, we do not want to make any such assumption in our code. So,
      the core implementation only uses OSGi R4 API, which makes it
      possible to run with alternative OSGi runtimes by changing
      configuration. Whether alternative runtimes will be supported or not
      is not an engineering decision.

      GlassFish developers can make use of HK2 component model, which
      substantially eases the development of Service oriented application.
      We shall provide bi-directional mapping between HK2 service and OSGi
      service [7].

      Since GlassFish will be implemented as a set of OSGi bundles,
      distributions of with different capabilities can be made by packaging
      suitable set of bundles. Once a particular distribution is installed,
      bundles can be added, removed or updated to add, remove or update
      system's capabilities.

      It will be possible to embed GlassFish in existing OSGi runtime. A
      typical example of taht would be embedding GlassFish in Eclipse IDE,
      which starts its own OSGi framework called Equinox.

      Now, a note about how traditional Java EE applications would run in
      the new runtime. There are two alternatives that come to my mind:
      A) Java EE applications runs outside of an OSGi bundle context,
      B) Java EE applications runs within an OSGi bundle context.
      Approach #B requires converting user supplied, non-OSGi applications
      to be converted to equivalent OSGi bundles at runtime. This is
      inherently a risky proposal, as the conversion is not always that
      straight forward and can require user intervention. More over, the
      Java EE class loading scheme may not fit well in this model. This
      will affect use of many existing libraries and frameworks that rely on
      Java EE class loading scheme. This severly impacts backward
      compatibility of the server. However, approach #A does not have
      most of these problems. So, we prefer approach #A.

      Having said that, we do want GlassFish to be able to accept OSGi-ed
      applications, irrespective of them being Java EE applications or not.

     4.2. Bug/RFE Number(s):

     4.3. In Scope:
      We will try to make the server compatible with other popular OSGi
      platforms. We will provide necessary maven plugins and build script
      support to help developers build OSGi bundles. We will make various
      Java EE APIs available with proper OSGi metadata.

     4.4. Out of Scope:
      OSGi facilities can't be made available to standard (non-OSGied)
      Java EE applications in this release. We will experiment with running
      OSGi-ed Java EE applications. We will not wrap commonly used libraries
      as OSGi bundles. Instead, we will encourage people to get such
      wrapped bundles from projects like "Apache Felix Commons [4]."
```

```
   4.5. Interfaces:

      4.5.1 Exported Interfaces
          Interface: Felix remote shell
          Stability: External
          Comments: This is a simple command line shell to administer
          the OSGi framework remotely.

          Interface: asadmin GUI and CLI enhancements to administer OSGi
          framework
          Stability: Evolving
          Comments: This will not be done as part of V3 Prelude release.

      4.5.2 Imported interfaces

          Interface: OSGi R4 API
          Stability: Standard
          Exporting Project: Felix (http://felix.apache.org)

   4.6. Doc Impact:
      We expect chnages to administration guide and classloader chapter
      of developers' guide and application development guide.

   4.7. Admin/Config Impact:
      In this release, we will provide administration commands for OSGi
      bundle lifecycle management. In a future release, we will allow user
      to configure the OSGi framework via GlassFish configuration management
      facilities.

   4.8. HA Impact:
      Not Applicable as of now, as V3 Prelude release does not have HA
      features.

   4.9. I18N/L10N Impact:
      Class loading restrictions in an OSGi environment applies to
      resource bundles as well. It requires changes to our code which has
      so far been assuming that all the necessary resource bundles are always
      available to all the implementation modules.

   4.10. Packaging & Delivery:
      To be discussed in packaging proposal[5].

   4.11. Security Impact:
      The OSGi Security Layer is an optional layer that underlies
      the OSGi Service Platform. The layer is based on the Java 2 security
      architecture.

   4.12. Compatibility Impact
      Since the server would be using new class loading scheme, there will
      be some impact on backward compatibility. Details to be discussed soon.

   4.13. Dependencies:
      1. Felix OSGi framework:
         http://felix.apache.org
      2. Maven bundle plugin developed as part of Felix project:
         http://felix.apache.org/site/maven-bundle-plugin-bnd.html
      3. Felix shell service and remote shell

5. Reference Documents:
      1. OSGi Module System:
         http://osgi.org
      2. GlassFish V3 Overview Functional Specification:
         http://wiki.glassfish.java.net/Wiki.jsp?page=V3OverviewFunctionalSpec
      3. HK2 component model:
           https://hk2.dev.java.net/components.html
        4. Apache Felix Commons project:
           http://felix.apache.org/site/apache-felix-commons.html
        5. GlassFish V3 Packaging proposal:

http://wiki.glassfish.java.net/attach/V3FunctionalSpecs/V3PreludePackagingFileLayout.txt
        6. Apache Felix project:
           http://felix.apache.org
        7. Providing HK2 services on OSGi:
           http://wikihome.sfbay.sun.com/asarch/attach/Attachments%2FHK2OSGi.pdf

6. Schedule:
    6.1. Projected Availability:
        Same as GlassFish V3 release dates. Initial implementation will be
        ready by V3 Prelude release.
```