The GlassFish Community
Delivering a Java EE Application Server

Eduardo Pelegri-Llopart
Yutaka Yoshida
Alexis Moussine-Pouchkine
Sun Microsystems, Inc.

http://blogs.sun.com/theaquarium

Last Updated September 2007

At JavaOne 2005, Sun announced *Project GlassFish,* an initiative to open source its Application Server and the Java EE Reference Implementation (see inset: *Project GlassFish*). This was the first step of open sourcing all the Java platform, but it also had other effects. Project GlassFish accelerated the adoption of Java EE 5, added a new enterprise-quality App Server to the options available to the Open Source community, and has lead to a transformation of how Sun's AppServer is developed, tested, made available, and evolved, in the process creating a much better product.

A year and a half after the initial launch, the *GlassFish community* has already delivered its first final release and is on its way to its second. In this article we will provide an overview of all the aspects of the GlassFish Community and the GlassFish AppServer.

Project GlassFish

Sun launched *Project GlassFish* in June 2005, during JavaOne, to Open Source the Reference Implementation for Java EE 5 and Sun's Application Server.

The first release followed less than a year after, at JavaOne 2006 (May 2006).

The second release was released in September 2007.

The focus is now on GlassFish v3, the modular and lightweight application server (see HK2).

What is GlassFish

GlassFish is a *Community* and an *Application Server*. The community's main deliverables are a Java EE 5 compatible Application Server, the *GlassFish AppServer*, and the Reference Implementation for the Java Persistence API, *TopLink Essentials*.

The GlassFish Community also delivers many other useful components including reusable modules for JCP technologies included in Java EE 5, like *JAXB*, *JAX-WS*, *JAXP*, *StAX*, *JSP* and *JSF*. GlassFish also includes several popular Web.Next projects like *jMaki, Phobos* and *DynaFaces,* many tools, like the Continuous Integration tool *Hudson,* and useful infrastructure like *Grizzly* the NIO-based server framework. The GlassFish community also maintains a *Maven repository* for all these components.

Java Platforms and Application Servers

Sun released Java in 1995 and the next year the Java Servlet API was presented during the first JavaOne(tm) conference. In 1999, Java Server Pages and Enterprise JavaBeans were combined with Java Servlets into the first *enterprise* Java platform: J2EE 1.2 - the version number was chosen to match the accompanying J2SE. Addition releases followed in 2001 (J2EE 1.3) and 2003 (J2EE 1.4) and then, in a name change, Java EE 5 in 2006.

The Java EE specifications have been adopted widely by both vendors and enterprises and many software vendors, Sun included, have implemented them in their Application Servers. For a number of years, Sun used to distribute both a *Reference Implementation* (see inset: *What is a Reference Implementations*) and a *Commercial Application Server,* marketed under different brands.

Application Servers at Sun

Sun has used several brands for its Application Servers. The most recent releases are:

- *iPlanet AS 6.0* – Originally from Netscape (Netscape AS) J2EE 1.2 compatible
- *SunOne AppServer 7.0* – First release including J2EE 1.3 RI
- *Sun Java System AppServer 8.0* – J2EE 1.4 compatible
- *Sun Java System AppServer 9.0* – Java EE 5 compatible. GlassFish v1.
- *Sun Java System AppServer 9.1* – Java EE 5 compatible. GlassFish v2.

What is a Reference Implementation

Specifications created through the Java Community Process (JCP) are the outcome of an Expert Group (EG) coordinated through an EG Lead.

The Expert Group delivers:

- A Specification Document
- A Technology Compatibility Kit (TCK)
- A Reference Implementation (RI)

The RI implements the Specification and passes the TCK.

The quality of the RI can vary from a *proof of concept* to a *production-quality* implementation. GlassFish is production quality, Java EE 5 compliant, Application Server.

Strictly speaking, the Java EE 5 RI is a specific *snapshot* of the GlassFish code base that has been declared as such.

These two offerings had different properties: the RI was focused on development and teaching the specifications and was free for development but could not be used in deployments, while the Commercial product was for revenue and was focused on Deployers. The two offerings have converged over time, starting with the 7.0 release where the commercial product (then under the *Sun One* brand) included for the first time the functionality from the J2EE 1.3 SDK and continuing through the 8.x and 9.x releases under the *Sun Java System* brand (see inset: *Java AppServers at Sun*).

By J2EE 1.4, the RI was identical to the SJS AS 8.0 PE, was free for development and deployments, and was included in wide distribution releases like the J2EE 1.4 SDK.

The SJS AS 8.x products were still available in 3 different versions: a *Platform Edition* (PE) which was identical to the RI, a *Standard Edition* (SE) that included some enterprise-quality features like Clustering and Failover, and an *Enterprise Edition* (EE) version that was targeted to the high-availability (99.999% availability) market. Starting with the 9.x releases, all these features are being incorporated into a single version which is being developed transparently by the GlassFish community under an Open Source license. The user is free to configure GlassFish using profiles: developer, cluster, and enterprise.

Open Source Licenses

The Open Source Initiative (OSI) recognizes a diverse set of licenses. Licenses often have a lineage from which they inherit some of their properties.

The ASL license is in the lineage of BSD and the MIT license. ASL is quite flexible in usage.

GPL, LGPL, GPLv2 and GPL+CPE are all GNU licenses. GNU licenses emphasize preserving the "freedom" of the code base. The GPL licenses are project based.

The CDDL license is a derivative from MPL, the Mozilla Public License. Unlike the GPL licenses, these licenses are file-based.

A good place to learn about open source licenses is the OSI foundation (*opensource.org*).

The GlassFish AppServer

Sun announced Project GlassFish in June 2005, during JavaOne 2005. Less than a year after that, at the next JavaOne in May 2006, the first release was made available to the public and the Java EE 5 specification went final . This AppServer was made available at Sun's download sites under the *Sun Java System 9.0 PE* name and at the community site under the *GlassFish v1* name but both are identical except for the choice of installer.

GlassFish v1 focused on implementing the Java EE 5 specification and some of the enterprise-level features were not included in the first release. To reflect this, the AppServer was labeled *PE* in the Sun distribution. GlassFish v2 adds back all these features and drops the *PE* label altogether because the features have been fully integrated: the same executable can be installed in a *developer,* an *enterprise* or a *cluster* profile, depending on the intended use.

GlassFish v2 is from Sun's sites under the *Sun Java System AS 9.1* name. It was released in September 2007.

The Java Platforms

The Java EE specification is an *umbrella specification* that builds on the corresponding Java SE specification (e.g. Java EE 5 requires Java SE 5) and includes many other JCP specs; the GlassFish community reflects this and includes a number of subprojects that are the official reference implementation for most of these specifications. Using separate subprojects has several benefits, including making it easier to reuse the implementation by other groups and projects. For example, the JAXB implementation is used in many projects, most of them not related to Sun.

The Java standard APIs are developed in Expert Groups using the *Java Community Process* and many Expert Group leaders are members of the GlassFish community. The GlassFish community implements these specs and also provides feedback during the review process and submits requests towards future releases.

The work on Java EE 6 has started with JSR 316 which is an umbrella project grouping together specifications such as EJB 3.1, JPA 2.0, etc... GlassFish v3 will implement Java EE 6.

Open Source and GlassFish

One of the first widely adopted server-side Java projects was Tomcat. Tomcat was started at Apache by a group that included Sun and the JServ developers and became the Reference Implementation for the early versions of the Java Servlets and Java Server Pages specifications (the RI for the latest specs is GlassFish). Tomcat was critical to the early adoption of server-side Java, was available under an Open Source license and contributed to the popularity of Open Source software within enterprise organizations.

There are many different licenses (see inset: *Open Source Licenses*) and we will not try to characterize them in detail in this article. Different licenses have different properties that attract different communities: what one community perceives as a benefit in one license may be perceived as a drawback by another; for example, the Apache community uses ASL licenses while the GNU community uses GPL licenses.

When GlassFish was launched, it used only the Common Development and Distribution License (CDDL) but, for the open sourcing of Java SE and Java ME in November 2006, Sun reevaluated the requirements of the different communities and decided to use the GPLv2 license for Java ME and GPLv2 with the ClassPath Exception (GPLv2+CPE) for Java SE. Using a GPL license provided more synergy with the GNU ClassPath and Kaffe communities, and CPE enables linking against non-GPL artifacts. At that moment, the GlassFish community **added** GPLv2+CPE to CDDL and became dual-licensed. Dual license means that either license can be used, and thus we can maintain the benefits that CDDL provides to the GlassFish commercial partners and add synergies with the GNU, Java SE and Java ME communities. In any case, there is no binary license in the GlassFish distribution.

GlassFish and Java EE 5 Adoption

The presence of an Open Sourced, production quality, implementation of Java EE 5 has accelerated the adoption of the Java EE 5 platform by creating demand and providing supply.

Resource Injection

J2EE 1.4 requires a fair amount of template code. The following is typical for EJB References:

```
Context initial = new InitialContext();
Context myEnv = (Context)initial....
Object objref = myEnv.lookup("...");
ConverterHome home = (ConverterHome)
PortableRemoteObject.narrow(...);
Converter currentConverter = home.create();
```

In Java EE 5, through convention and resource injection, we can just say:

```
@EJB Converter currencyConverter;
```

WS Endpoints in Java EE 5

The following code snippet shows a WS end point:

```
import javax.jws.WebService;
@WebService
public class MySimpleWS {
    public String sayHello(String s) {
      return "Hello" +s;
    }
}
```

This can be converted into a (transactional) Stateless Session Bean as follows:

```
import javax.jws.WebService;
import javax.ejb.Stateless;
@WebService
@Stateless
public class MySimpleWS {
    public String sayHello(String s) {
        return "Hello" +s;
    }
}
```

As of this writing, there have been over 3.5 million downloads of GlassFish and these developers and deployers (see insert: *Going to Production*) have been exposed to the benefits of Java EE 5 and has increased demand for its support. The GlassFish AppServer has satisfied some of that demand but it has also helped other Java vendors as GlassFish's production-quality components can be used to complete a Java EE 5 compliant implementation more quickly and/or more cost-effectively.

Vendors using GlassFish include TmaxSoft, Oracle, BEA, JBoss, Jetty and Geronimo. TmaxSoft is a leading Java EE licensee in Korea and their latest relese JEUS 6 uses JAXB, JAX-WS, Fast Infoset, JSP, JSF and TopLink Essentials Through its participation in the GlassFish community, TmaxSoft had early access and ensured that a number of components were well suited to their needs. This meant that TmaxSoft's JEUS 6 became the first Java EE 5 certified AppServer after Sun's own AppServer.

Oracle has not yet released a full Java EE 5 AppServer but they are a main contributor to the TopLink Essentials project in the GlassFish community and Oracle is including this implementation in their products. Of the vendors mentioned above, BEA uses JAXB and JAX-WS, JBoss uses JAXB and JSF, Jetty uses JSP and Grizzly and Geronimo uses JAXB.

Overview of JavaEE 5

The main theme of Java EE 5 is *Ease of Development* and a main tool to accomplish this is the use of *annotations* from Java SE 5 to enable programming based on POJOs – *Plain Old Java Objects.* Annotations are used in many ways including indicating properties of methods and classes (see inset: *Endpoints in Java EE 5*), resource injection (see inset: *Resource Injection*), and as portable descriptions of behavior (see inset: *Annotations Are the Default*).

The main Java EE 5 specifications are:

- JAX-WS 2.0 & JAXB 2.0
- EJB 3.0 & Java Persistence API
- JSF 1.2 & JSP 2.1
- StAX

A number of other specifications (like JAXP and Servlet) have also had smaller, maintenance, releases. The overall effect of all these changes in the developer experience has been very substantial.

Java EE 5 and GlassFish have had a very symbiotic relationship: the strength of the Java EE 5 specification has increased the value of GlassFish and the availability of GlassFish has validated Java EE 5. We expect this relationship to continue in future versions of the Java EE platform with an even more effective feedback loop in the evolution of the specifications themselves, as the early availability of the implementations enable the

Annotations are the Default

This is a typical JAXB 2.0 generated code.

```
@XmlAccessorType(FIELD)
@XmlType(name="",propOrder={"x","y"})
@XmlRootElement(name="point")
public class Point {
  protected float x, y;
  public float getX() {return x; }
  public setX(float value){ this.x = value; }
  public float getY() {return y; }
  public void setY(float value) { this.y = value;}
}
```

community to try the specifications earlier.

We are just starting to learn how to use annotations. For example, the JAXB 2.0 specification is structured in two parts. One part describes how to map between XML Schema into annotations; the other describes the semantics of those annotations. This means it is possible to change the generated code and it will still *run portably in any conformant AppServer.* In particular, we take the code described in the insert and add code that will perform side-effects on setters and getters, and it will be portable.

GlassFish Releases

There are three versions of GlassFish at different phases of development: v1, v2 and v3. With the caveat that these are the early days of the community and we are still creating the process, so far, the cycle for a GlassFish release is as follows:

- *Concept Creation* – Collecting key features, rough time-frame, prototyping
- *Active Development* – Implementation leading to usable Milestones and a *final release*.
- *Maintenance* – Bug fixing of final releases, creating Update Releases.

GlassFish v1 is in maintenance mode, GlassFish v2 was very recently released and thus has entered a maintenance mode. GlassFish v3 is currently moving from a concept creation phase to active development. It will most likely be released in phases given it's modular approach.

GlassFish v1

GlassFish v1 is in maintenance mode. The final *v1* release was on May 2006, right before JavaOne 2006 and was followed by a bug-fixing *v1 UR1* in October 2006 and a second, smaller, *v1 UR1p1* release in December 2006. There are no more maintenance releases scheduled at this time, as the development has moved to v2.

A distinguished release of GlassFish v1 was named as the official Java EE 5 RI.

GlassFish v1 is also distributed by Sun under the name *Sun Java System AS 9.0 PE* and it is included in a number of downloads distributed by Sun, including the *Java EE SDK*, the *Java Application Platform SDK* and the *NetBeans 5.5 Tools Bundle*. The only difference between GlassFish v1 and SJS AS 9.0 PE is the installer. Sun provides different levels of commercial support for final versions of GlassFish v1 (and SJS AS 9.0 PE), and the community provides best-effort, but good, support.

GlassFish v2

GlassFish v2 was released in September 2007. The main and most important feature of this release is clustering (grouping, load-balancing, data replication).

GlassFish v2 includes all the enterprise-quality features from the SJS AS 8.2 SE/EE releases and will be tested against all the products in the *Java Enterprise System* set of middleware offerings. GlassFish v2 supports the concept of profiles and the same executable can be configured in a *developer*, *enterprise* or *cluster* profile. The enterprise profile can also be configured to use HADB (High Availability Data Base) for very high (99.999%) availability.

GlassFish v2 is also distributed by Sun under the *Sun Java System AS 9.1* label and included in a number of bundles (such as Java ES 5.1). Community and Commercial support are provided.

GlassFish v3

GlassFish v3 was presented at JavaOne 2007 and drew quite a bit of interest. Its architecture is modular by default, its kernel is extremely small (under 100Kb which makes it suitable for desktop and even mobile use), and its startup time is under a second. Containers supported by the modular kernel are multiple and so far Java Web, PHP (via Caucho's Quercus), jRuby on Rails, Phobos JavaScript are integrated. More will follow and you are encouraged to write your own.

A preview version of GlassFish v3 is available from http://glassfish.java.net and documentation is available from http://wiki.glassfish.java.net and http://hk2.dev.java.net.

Active development on GlassFish v3 is starting right now. Planning is still ongoing but the modular nature of the development will likely enable phased releases up to a full Highly-Available Java EE 6 product.

GlassFish v2 Features

This section describes the features in GlassFish v2 as that release should be in beta by the time of publication. There are many features and will just provide a brief overview of each of them; more information is available online (see inset: *Useful Links*).

The Web Tier is the workhorse of most Enterprise Applications and GlassFish provides very rich Web Tier support.

Web Tier Java Specifications

GlassFish includes support for the latest JCP specs: JSP 2.1, JSF 1.2, Servlet 2.5 and JSTL 1.2.

Java Server Faces (JSF, the next-generation standards-based MVC framework) was added to the platform in Java EE 5 and provides a component model for the Java presentation layer. JSF can be used with JavaServer Pages (JSP) or with other technologies like *facelets.* JSF 1.2 has a number of improvements over the earlier releases, most notably the *unified expression language* that is now shared between *JSF* and JSP. JSF 1.2 also has several improvements targeted at the request cycle, including improved AJAX support.

The main change in JSP 2.1 has been the unified EL; Servlet 2.5 and JSTL 1.2 have also received some changes but not as deeper.

GlassFish also has a number of substantial improvements beyond the spec changes. The JSF implementation has had substantial improvements on the performance and it a few key bugs have been addressed. The JSF implementation is also used by JBoss and a few other groups. The Servlet container still has its origins in Tomcat but it is now maintained separately for stability and performance.

Another noteworthy change in GlassFish is Jasper, the JSP compiler, which can now take advantage of the compiler APIs (JSR-199) in Java SE 6 to avoid file IO and compile much faster (informally 10x faster). Jasper can also be configured to use the Eclipse JDT compiler, although it is not as fast as when using JSR-199. The JSF implementation has also been improved substantially although we do not have benchmark results handy at the moment.

GlassFish recently also started using the high quality JSF components from Project Woodstock, which should be of interest to many developers on their own right. The components have been open sourced at http://woodstock.dev.java.net

Performance Improvements

There are many areas where the performance of GlassFish v2 has improved. A partial list includes JSF, JSP and Grizzly in the web tier, JAXB and JAX-WS in Web Services, CORBA and EJB.

The net result of all these efforts are simply a world record in SPECjAppServer 2004 numbers. In July 2007, Sun announced a the #1 result on a T2000 machine with 883.66 JOPS@Standard. This is 60% faster than GlassFish V1/SJSAS 9.0, 10% faster than BEA WebLogic and 30% faster than IBM WebSphere 6.1! Another benchmark result (813.73 JOPS@Standard using the PostgreSQL database) shows a 3x price/perf ration versus an Oracle on HP score.

Clearly you **no longer have to choose between Open Source and Enterprise Features**: you can have both.

Disclaimers: SPEC and the benchmark name SPECjAppServer 2004 are registered trademarks of the Standard Performance Evaluation Corporation. Sun Fire T2000 (1 chips, 8 cores) 1.4ghz 883.66 SPECjAppServer2004 JOPS@Standard. Competitive benchmark results stated above reflect results published on www.spec.org as of 07/10/06. For the latest SPECjAppServer 2004 benchmark results, visit http://www.spec.org/.

A separate effort has been made to improve start-up performance. The startup and shutdown architecture has been simplified substantially and all start-up dependencies have been analyzed to minimized unneeded service initializations. The result is substantial reductions of start-up and shutdown times of both simple instances or entire clusters.

Compatibility With Popular Frameworks

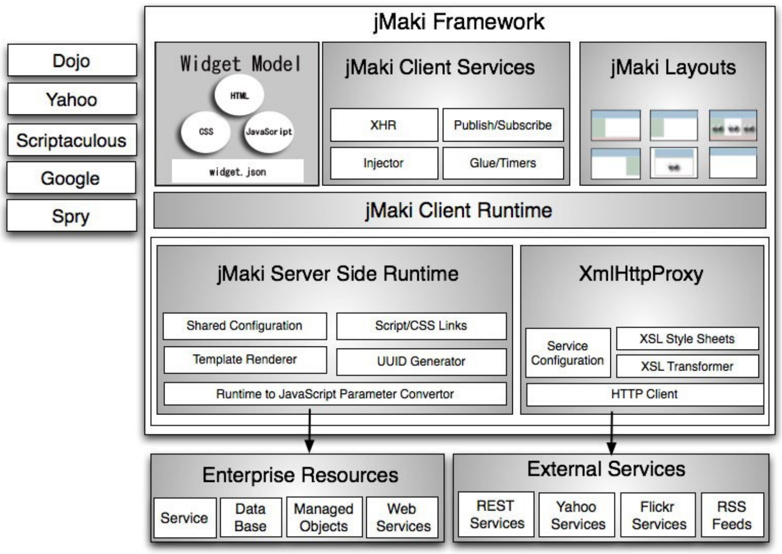
One of the core strengths of the Java community over the years has been its diversity and this diversity is particularly visible in the Web Tier in the form of many frameworks. These frameworks improve substantially the productivity of the developers and one of the explicit goals in the GlassFish Community has been to ensure that these frameworks work well, *out of the box*, with the different GlassFish releases. A similar approach has been taken with popular applications.

The list of frameworks is large and always growing; a small subset includes *AppFuse*, *DWR*, *Facelets*, *IBATIS*, *JBoss Seam*, *Shale*, *Spring*, *Struts*, *Tapestry*, *WebWork*, *Wicket*... etc.

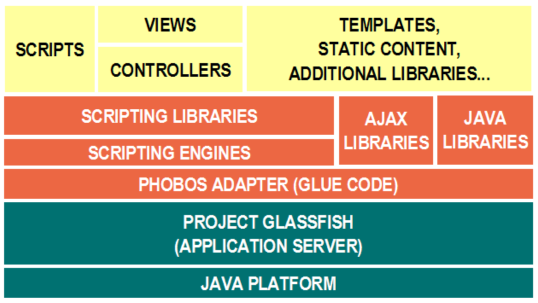
AJAX Support

The GlassFish Community is pursuing a number of projects in the area of AJAX. In most of these cases, the projects are not GlassFish-specific, a few may depend on extensions specific to GlassFish.

jMaki (*ajax.dev.java.net*) is a framework that provides a lightweight model for creating JavaScript centric AJAX-enabled web applications using Java, PHP, and Phobos. The model is based on JavaScript, HTML and CSS and can be modified easily. jMaki works with a number of AJAX widget libraries, including *Dojo*, *Yahoo*, *Scriptaculous*, *Google* and *Spry* and it is very easy to add new ones. jMaki has shallow dependencies on the server model and it is easy to adopt and incorporate.



Phobos (phobos.dev.java.net) is a lightweight, scripting-friendly, web application environment running on the Java platform, aimed at addressing emerging developer requirements. The initial focus of Phobos is on the use of *JavaScript* but the design supports the use of other dynamic languages as well. Phobos is a more ambitious project than jMaki and it is at an earlier stage of development but it is showing interesting synergies. Using JavaScript on the server and the client side simplifies transfering content through JSON as well as reusing some code. The Java scripting engine *Rhino* is also very solid and extensions such as E4X and bytecode compilation are also available.


SCRIPTS
VIEWS
CONTROLLERS
TEMPLATES, STATIC CONTENT, ADDITIONAL LIBRARIES...
SCRIPTING LIBRARIES
SCRIPTING ENGINES
AJAX LIBRARIES
JAVA LIBRARIES
PHOBOS ADAPTER (GLUE CODE)
PROJECT GLASSFISH (APPLICATION SERVER)
JAVA PLATFORM

Phobos Architecture

GlassFish also have several JSF-specific projects related to AJAX, including *Woodstock,* a collection of JSF components that include AJAX components, and *Dynamic Faces* which explores high-quality integration of AJAX into JSF. AJAX is also supported through projects like *DWR* and *AJAX4JSF*.

Grizzly

The lower layers of the Web Tier in GlassFish are implemented through the Grizzly Framework (*grizzly.dev.java.net*). The framework is written in Java taking advantage of the NIO APIs (scalable network and file I/O) to provide scalability and is also highly customizable, and fairly general. The role of the Grizzly HTTP connector is similar to that of the Java-based Coyote connector used in Tomcat and that of the C-based Apache Portable Runtime (APR). Informal initial tests have shown very good scalability.

Grizzly has generated a fair amount of interest because it makes it much easier to take advantage of NIO; for example, both AsyncWeb and Jetty have prototyped integrations with it. Grizzly is in the process of being generalized further to provide additional functionalities and better address the needs of other groups, inside and outside the GlassFish community.

Grizzly can be used separately from GlassFish, and that is how it is for instance used in the NetBeans plugins for Phobos and Ruby. Grizzly's flexibility can be used to provide efficient support for “long-lasting HTTP connections”, also known as *Comet,* which can be used in applications like Chat, online calendaring or document sharing, and continuously updated content.

Alternatives in the Presentation Layer

A common theme in the GlassFish community is to embrace diversity. There are many reasons for this, from “competition fosters quality” to “one size does not fit all” to “innovation happens everywhere”. This applies specially to the presentation layer where GlassFish enables the typical frameworks based on the Java platform as well as several newer alternatives. In some cases these alternatives layer directly on the GlassFish AppServer; in some, they live alongside it.

As indicated, server-side *JavaScript* is supported via the Phobos project. In this case, the JavaScript server-side code is executed on GlassFish using the Rhino interpreter.

Ruby-based applications, most notably Rails, can be executed in two different approaches. In one, Rails executes on jRuby, on top of the Java Platform. In the other Rails executes on the *native* Ruby interpreter, which communicates with GlassFish via the CGI interface. Rails on GlassFish is a particularly attractive arrangement and is being explored aggressively, so stay tuned for future developments.

PHP can also be used with GlassFish and the *Quercus* PHP 5 implementation developed by Caucho in Java. The Caucho and GlassFish communities are working together to make Quercus work on GlassFish.

Web Services – Metro

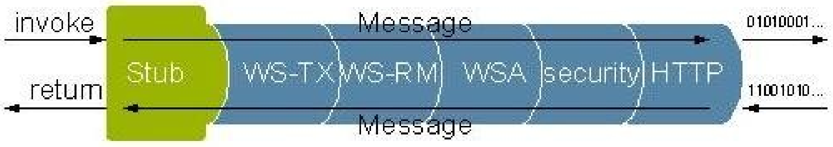
The Web Services stack in GlassFish is called Metro. GlassFish v1 delivered a big improvement in Ease of Development through the new JAX-WS 2.0 API (jax-ws.dev.java.net). GlassFish v2 refines that slightly into JAX-WS 2.1 but the major change is in the implementation which is the result of several years of evolution and several redesigns. The Metro implementation is very sophisticated, fully featured and has high performance as shown in our benchmarks with 30%-100% performance improvement over the AXIS 2 stack (see image below) we believe it is industry-leading technology. JAX-WS 2.0 is also available in Sun's Java SE 6 RI and it can be upgraded to JAX-WS 2.1.

SJS AS 8.2 had an implementation of JAX-RPC 1.1. The initial implementation of JAX-WS 2.0, used in GlassFish v1, was loosely based on that but it has gone through *two* sets of changes since then two address two major challenges. The first challenge is driven by the new specification: unlike JAX-RPC 1.1, JAX-WS 2.0 delegates all data binding decisions to JAXB 2.0 and also carefully enables non-HTTP protocols. On top of this, Sun recognized a while ago the performance cost of XML textual encodings and has been participating in different standard bodies involved in efficient XML encodings (see inset: *Fast Infoset*).

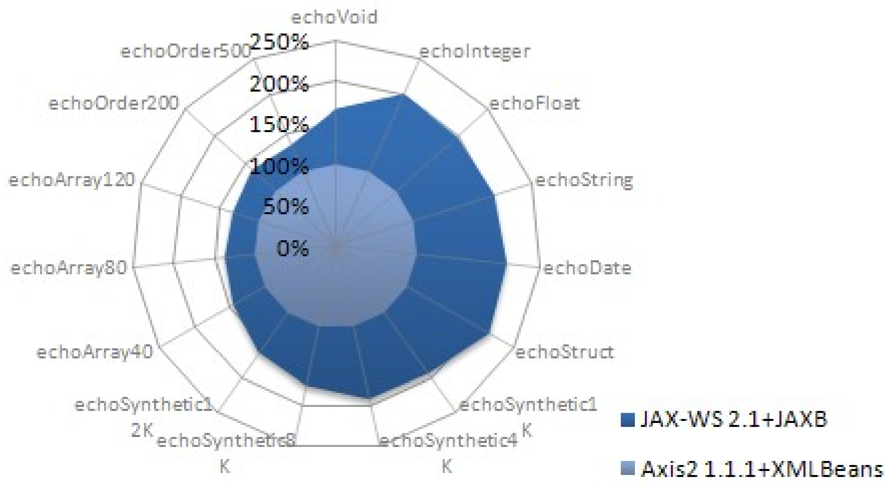
The result was a rearchitecture of the JAX-WS implementation that clearly separated Transport, Encoding, and Data Binding. This new Metro architecture has been integrated into GlassFish v2 and supports:

- Multiple Transports – HTTP, JMS and TCP/IP have been implemented and SMTP has been designed.
- Multiple Encodings – Textual XML, MTOM, and Fast Infoset are supported
- Data Binding – Data Binding is handled by JAXB and the handover is done very carefully to avoid copying data around.

The architecture is designed to support multiple, stackable, processing entities, each implementing one of the WS-* specifications, without incurring data copies. The structure is as shown below.



Performance is a moving target and the JAX-WS 2.0 implementation doubles the performance of the (once highly performant and competitive) older JAX-RPC 1.1 stack. The performance results of recent Metro (JAX-WS 2.1) stack compared to Axis 2 show the intent of the GlassFish team to provide the best stack in the industry.



Throughput is not the only important performance metric; in some contexts scalability is even more important and the Business Integration group wanted to be able to process thousands of transactions, some of them fairly long-lived. The implementation does this through an execution model where tasks share a pool of execution threads so very few threads are needed to efficiently execute a very large number of transactions.

WSIT and Project Tango

One of the appeals of Web Services is as an interoperability vehicle between the Java and the Microsoft platforms. *Project Tango* is the initiative within the GlassFish Community that is delivering that interoperability, and WSIT (Web Services Interoperability Technologies) is the formal name (wsit.dev.java.net). This is a subset of Metro (metro.dev.java.net).

A detailed description of the Project Tango feature in Metro is available in this complete overview article: http://wsit.dev.java.net/docs/tango-overview.pdf

WSIT takes advantage of the new WS architecture in GlassFish v2 and the different WS-* standards are processed by elements in the WS pipeline. WSIT works transparently; the developers can use the standard JAX-WS development model and declaratively add WS-* attributes using NetBeans (a NB's plugin is included). The standards are to ensure interoperability with other vendors, most notably Microsoft. The interoperability has been tested in a number of WS Plugfests.

The following standards are currently supported:

- Boostrapping: WS-MetadataExchange, WS-MetadataExchange WSDL, WS-Transfer
- Reliable Messaging: WS-ReliableMessaging, WS-ReliableMessaging Policy
- Atomic Transactions: WS-Coordination and WS-Atomic Transaction
- Security: WS-Security, WS-SecurityPolicy, WS-Trust, WS-SecureConversation
- Policy: WS-Policy, WS-PolicyAttachment

Fast Infoset

XML documents can be quite verbose which hinders their applicability in some areas, not just because of size but also due to transmission cost. An encoding like Gzip reduces size but it does so at the expense of encoding time, and the Fast Infoset ANSI/ISO standard provides a better overall trade-off.

As of this writing there are 4 different commercial implementations of Fast Infoset but other standards may also gain traction (for instance the W3C EXI Working Group is also looking into this area), so the GlassFish Web Services stack is designed to support multiple encodings. It currently supports the MTOM and Fast Infoset encodings – in addition to textual encodings – but it will also support other standards as they appear.

Fast Infoset can be used with WSIT. The Windows Communication Foundation (WCF) also provides encoding isolation and some vendors exploit it for FI support

XML Processing

GlassFish includes support for the JAXP and the StAX APIs for XML processing.

The *Streaming API for XML* (StAX) is a new XML parsing API that was lead by BEA and it is a new addition to the Java Platform; it is included in the Java EE 5 and Java SE 6 platforms. StAX can be used as an alternative to SAX and DOM and has different performance and API characteristics. DOM is an in-memory, document-centered API. The result of parsing a document is a collection of objects than can then be navigated in any order, but this flexibility is has substantial performance implications both in memory consumption and in parsing performance. SAX is an event-driven API where callbacks can be invoked as new elements in the XML document are parsed. SAX has better performance characteristics than DOM but it leads to an inversion of control where state must be updated in response to events.

A pull parser has the performance characteristics of SAX but the main code remains in control as it *pulls* the events when / if it needs them. Recent GlassFish development include working with the Woodstox community (a highly optimized StAX implementation from Codehaus.org).

The latest *JAXP specification* is JAXP 1.4. JAXP 1.4 is a *maintenance* release from JAXP 1.3; JAXP 1.4 includes a number of improvements and cleanups but its major change is that it incorporates StAX (by reference). JAXP 1.4 is included in Java SE 6 while J2SE 5.0 only includes JAXP 1.3. GlassFish can be used with J2SE 5 or Java SE 6.

The JAXP 1.4 implementation delivers both JAXP 1.4 and StAX, but the StAX implementation is also available separately. All implementations are done within the GlassFish community and are delivered through the different release vehicles.

XML Data Binding

XML Data Binding in Java EE is done through the JAXB API; the version required by Java EE 5 is JAXB 2.0 (jaxb.dev.java.net) which, like in the case of JAX-WS, provides a substantial improvement over the corresponding J2EE 1.4 specification. This improvement has been recognized by the Java community and we are seeing a very fast switch to the new specification.

The JAXB 2.0 implementation is very high quality, fast, and fully featured. The JAXB developers have also been very receptive towards other communities and the result has been that many groups are using the JAXB implementation, including JBoss, Apache Axis, Apache XFire, TmaxSoft, ServiceMix, and ActiveSOAP.

Newer than Java EE 5

The Java EE 5 specification requires JAXB 2.0 and JAX-WS 2.0 and those are the specifications implemented by GlassFish v1 and Java SE 6. Experience with these implementations in multiple use cases, including Web Services Interoperability with Microsoft's Windows Communication Framework, has lead to compatible refinements to these specifications in the form of JAXB 2.1 and JAX-WS 2.1.x. These are the specifications implemented by GlassFish v2.

GlassFish Integration Technologies

TopLink Essentials and JPA

One of the big functional improvements in the Java EE 5 specification is the addition of the Java Persistence API (JPA) and the new version of the Enterprise JavaBeans Specification, EJB 3.0. These APIs evolve from the earlier EJB 2.1 but take advantage of new Java language features like Java annotations and incorporate the experiences from communities like Hibernate, Oracle TopLink and JDO.

The two APIs can be used separately. JPA provides a simple programming model for entity persistence and it is available separately of Java EE 5 and can be used with J2EE 1.4 and even Java SE; it also includes an API for plugging different persistence providers. The rest of the EJB specification has gone through extensive cleanup and simplification and uses JPA. Both specifications have been very well received and are gaining wide adoption.

TopLink Essentials (TLE) is the JPA implementation in the GlassFish Community. It was started with a big contribution from Oracle Corporation, building on their long experience with the *Top Link* products, and since then it has been a joint project with Sun. The initial implementation of TLE like GlassFish v1, was driven by spec compliance, but TLE is intended to be a fully featured implementation and Oracle's plan is to continuously enhance its features drawing on the capabilities in TopLink and contributions from the wider community. TopLink Essentials is used in a number of Oracle commercial products, including their AppServer, in Sun's distributions of GlassFish and in NetBeans, and in several other containers including Apache Tomcat, TmaxSoft's JEUS, Spring 2.0 and EasyBeans.

The pluggability feature also means that it is quite easy to take other JPA implementations and plug them into the GlassFish AppServer. This has been successfully done for Apache OpenJPA and Hibernate and is very popular.

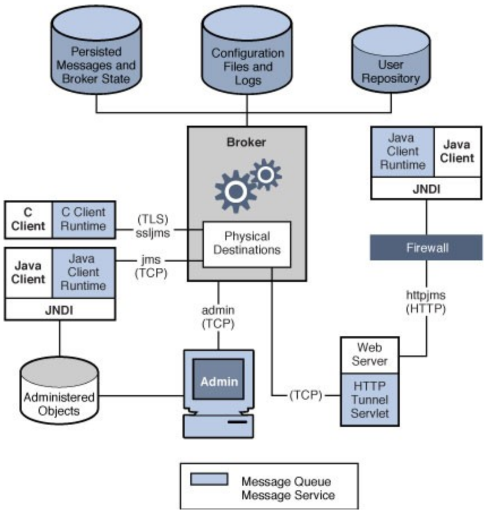
CORBA

GlassFish includes a fully featured CORBA implementation. This implementation has been improved consistently for a number of years, for instance in GlassFish v2 it now generates stubs and skeletons dynamically and it is exploiting the NIO framework. GlassFish v2 will also include a number of key performance improvements.

OpenMQ Messaging

GlassFish includes a production-quality Message Queue implementation (mq.dev.java.net) that provides:

- Message buffering between enterprise system components
- Scalable distribution of message servers (broker clustering)
- Integrated SOAP / HTTP messaging
- Java and C Client API
- Scalable JCA 1.5 compliant Resource Adapter
- Enterprise Administration Features
- Extensive JMX support



GlassFish also includes the *Generic Resource Adaptor for JMS* which can be used with a number of different Messaging Solutions, including: Tibco, WebSphere MQ 6, Active MQ and JBoss Messaging.

Java Business Integration

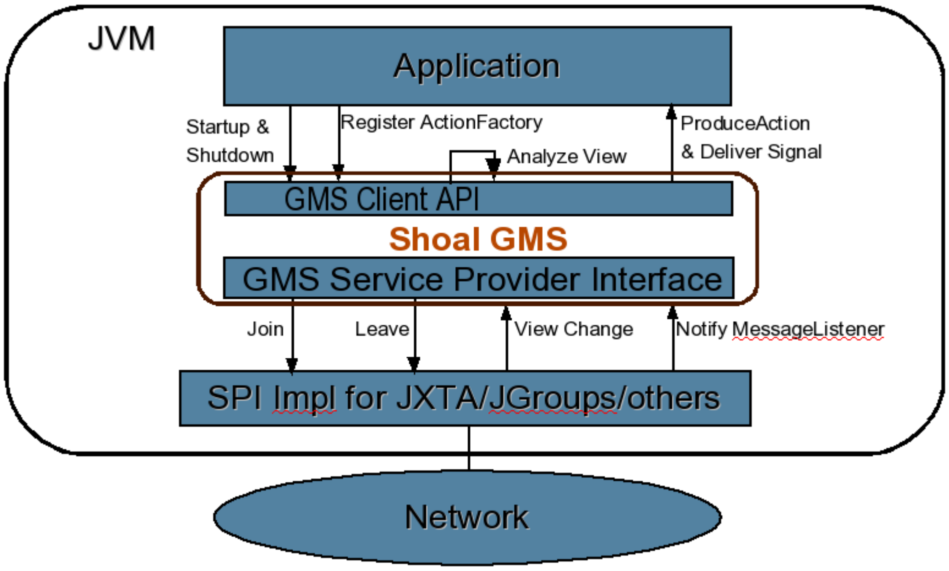
GlassFish v2 also includes support for the Java Business Integration (JBI) API. This JCP specification (JSR-208) defines the core of a service oriented integration bus and component architecture for integration. JBI standardizes the common message routing architecture, plug-in interfaces for service engines and bindings, and a mechanism to combine multiple services into a single executable and auditable unit of work.

The implementation included in GlassFish v2 is that from *Project OpenESB* and this implementation is used in other enterprise-level products at Sun. OpenESB integrates into GlassFish through the *JavaEE Service Engine*, and this same interface can be used to use other implementations like *ServiceMix*. There is also first-class tooling provided by the Enterprise Pack of NetBeans (Service Assembly editor, graphical WSDL and XSLT editors, and an Intelligent Event Processor.)

Clustering and State Replication

GlassFish v2 includes clustering support with dynamic clustering and memory replication. *Dynamic clustering* is provided through *Project Shoal* (shoal.dev.java.net) a generic framework that takes advantage of JXTA to enable an application to dynamically become a member of a predefined cluster, and as such, is subscribed to cluster events, such as:

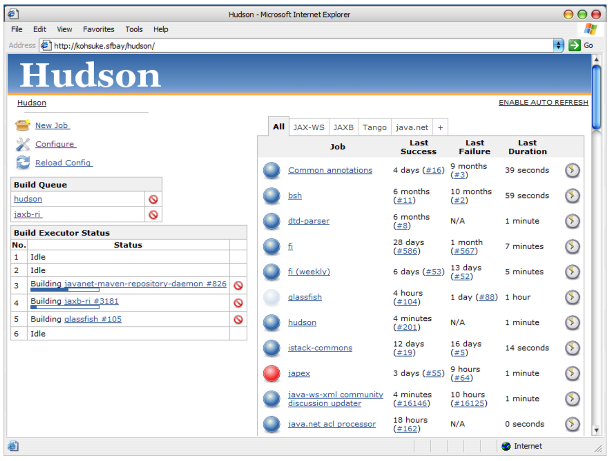
- Member join, planned shutdown, failures
- Recovery member selection
- Automated delegated recovery initiation



GlassFish v2 also provides *In-Memory Replication* for a light-weight, low-cost solution for HTTP Session persistence and Stateful (EJB) Session Bean persistence. This is sufficient for most applications but, for those requiring 99.999% availability, it is still possible to use the High Availability Data Base (HADB) persistence solution.

Tools

GlassFish has been collaborating with the NetBeans community through the development and implementation of the Java EE 5 specification and NetBeans was the first IDE to support GlassFish. Since the launch, IntelliJ, JBuilder and MyEclipse have also added support for GlassFish. More recently a plugin for Eclipse 3.3 (Europa) was also released.


Hudson - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://kohsuke.sfbay/hudson/
Go
Hudson
Hudson
ENABLE AUTO REFRESH
New Job
Configure
Reload Config
All
JAX-WS
JAXB
Tango
java.net
+
Internet

Build Queue	
hudson	🚫
jaxb-ri	🚫

Build Executor Status

No.	Status	
1	Idle	
2	Idle	
3	Building javanet-maven-repository-daemon #826	🚫
4	Building jaxb-ri #3181	🚫
5	Building glassfish #105	🚫
6	Idle	

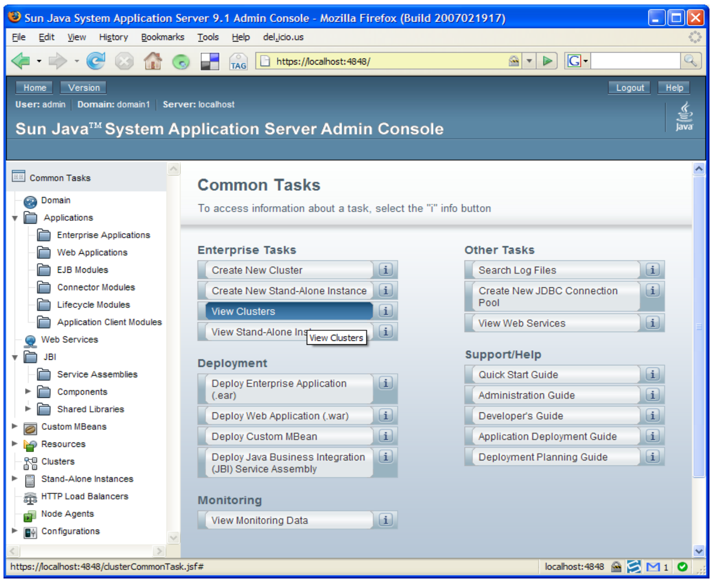
	Job	Last Success	Last Failure	Last Duration
●	Common annotations	4 days (#16)	9 months (#3)	39 seconds
●	bsh	6 months (#11)	10 months (#2)	59 seconds
●	dtd-parser	6 months (#8)	N/A	1 minute
●	fi	28 days (#586)	1 month (#567)	7 minutes
●	fi (weekly)	6 days (#53)	13 days (#52)	5 minutes
○	glassfish	4 hours (#104)	1 day (#88)	1 hour
●	hudson	4 minutes (#201)	N/A	1 minute
●	istack-commons	12 days (#19)	16 days (#5)	14 seconds
🔴	japex	3 days (#55)	9 hours (#64)	1 minute
●	java-ws-xml community discussion updater	4 minutes (#16146)	10 hours (#16125)	1 minute
●	java.net acl processor	18 hours (#162)	N/A	0 seconds

In a different angle, the GlassFish community includes a number of tools and some of them have become quite popular. on their own right.

One tool worth mentioning here is *Hudson*, a *Continuous Integration* tool that is being used to perform builds, execute tests, and more. Hudson is simple to setup, supports multiple slave machines and has a plug-in architecture that can be used to extend its functionality.

Administration

One of the strengths of GlassFish v2 is its administration. It includes a fully featured task-driven GUI Administration, as well as a command-line interface with a complete TCL interpreter and such niceties as a *do what I mean* mechanism for correcting typos. GlassFish also includes JMX-based management, as well as management for Clusters and Load Balancing.



Sun Java System Application Server 9.1 Admin Console - Mozilla Firefox (Build 2007021917)
File Edit View History Bookmarks Tools Help del.icio.us
https://localhost:4848/
Home Version
Logout Help
User: admin Domain: domain1 Server: localhost
Sun Java™ System Application Server Admin Console
Common Tasks
Domain
Applications
Enterprise Applications
Web Applications
EJB Modules
Connector Modules
Lifecycle Modules
Application Client Modules
Web Services
JBI
Service Assemblies
Components
Shared Libraries
Custom MBeans
Resources
Clusters
Stand-Alone Instances
HTTP Load Balancers
Node Agents
Configurations
Common Tasks
To access information about a task, select the "i" info button
Enterprise Tasks
Create New Cluster
Create New Stand-Alone Instance
View Clusters
View Stand-Alone Ins
View Clusters
Deployment
Deploy Enterprise Application (.ear)
Deploy Web Application (.war)
Deploy Custom MBean
Deploy Java Business Integration (JBI) Service Assembly
Monitoring
View Monitoring Data
Other Tasks
Search Log Files
Create New JDBC Connection Pool
View Web Services
Support/Help
Quick Start Guide
Administration Guide
Developer's Guide
Application Deployment Guide
Deployment Planning Guide
https://localhost:4848/clusterCommonTask.jsf#
localhost:4848

Installation Experience

As mentioned earlier, GlassFish v2 has a single Installer that can be installed in any of 3 different user profiles: *developer*, *cluster* or *enterprise*. The install process using the bits from http://glassfish.java.net is an easy two-step command-line process: install, then configure. Note that it is possible to upgrade from a *developer* profile to a *cluster* profile.

The bundle itself is substantially smaller (around 55 MB) than in the past thanks to its use of the *Pack200* compression technology (also used in the Java SE 6 installation bundle).

In its Sun Java System Application Server incarnation, the installer is a little more sophisticated and graphical.

On Ubuntu, the installation is, as expected, available through *apt-get*.

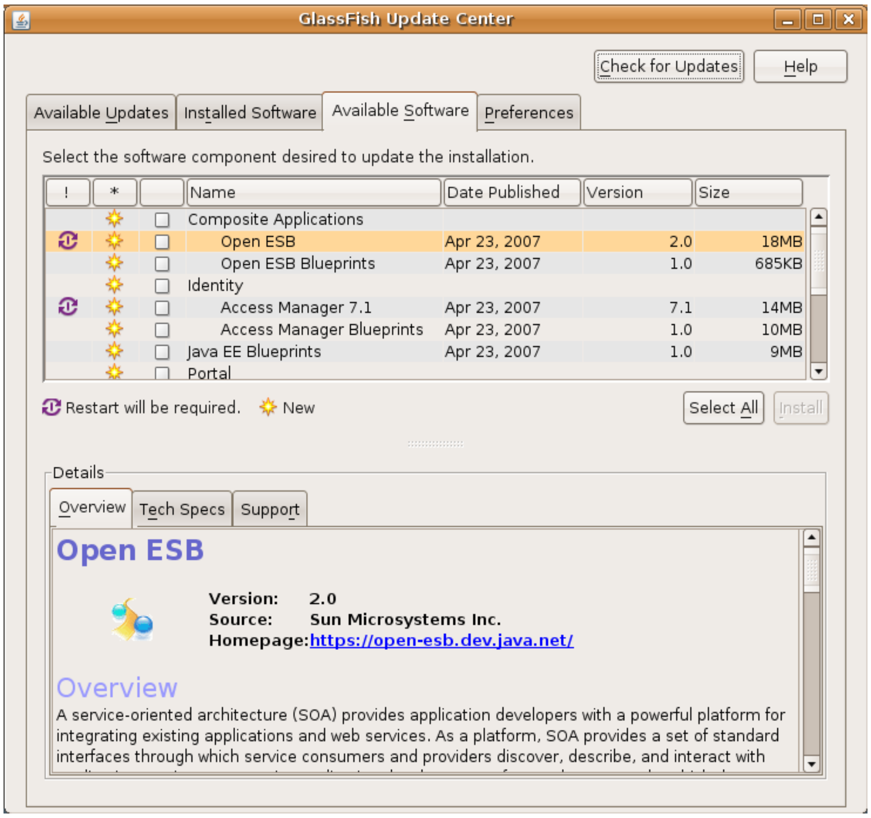
Sun Java System Application Server 9.1 is also available and supported as part of Java Enterprise System 5 Update 1.

Update Center

Update Center to deliver GlassFish add-on components, updates and to provide a mechanism to enable community contributions. Various different add-on components are delivered through Update Center today.

The GlassFish Update Center is an open source project hosted on http://updatecenter.dev.java.net. The initial release with GlassFish V2 is focusing on providing simple download and installation experience of additional applications for GlassFish.

The Update Center functionality in GlassFish is implemented across two tiers. On the server side, we have Update Center modules and corresponding catalog XML file. The client side is part of GlassFish V2 installation. When started, Update Center client (located in the **`/updatecenter`** directory of the GlassFish distribution) connects to a predefined Update Center server URL. It then downloads available catalog file, compares it to local installed component registry and uses catalog information to show components available for installation or update. Afterwards, based on user's selection, components are downloaded and installed.



Select the software component desired to update the installation.

!	*		Name	Date Published	Version	Size
	✸	☐	Composite Applications			
↻	✸	☐	Open ESB	Apr 23, 2007	2.0	18MB
	✸	☐	Open ESB Blueprints	Apr 23, 2007	1.0	685KB
	✸	☐	Identity			
↻	✸	☐	Access Manager 7.1	Apr 23, 2007	7.1	14MB
	✸	☐	Access Manager Blueprints	Apr 23, 2007	1.0	10MB
	✸	☐	Java EE Blueprints	Apr 23, 2007	1.0	9MB
	✸	☐	Portal			

↻ Restart will be required. ✸ New

Open ESB

Version: 2.0
Source: Sun Microsystems Inc.
Homepage: https://open-esb.dev.java.net/

Overview

A service-oriented architecture (SOA) provides application developers with a powerful platform for integrating existing applications and web services. As a platform, SOA provides a set of standard interfaces through which service consumers and providers discover, describe, and interact with

This mechanism can be used for deploying new GlassFish features, framework libraries, and even entire web applications. You can create your own update center add-on module, package it, test it and made it available by simply adding additional Update Center definitions to the client software.

GlassFish in Production

A common property of Open Source projects is an accelerated adoption cycle as users can decide when to use the technology based on their assessment of the readiness of the technology and their own feature/risk status. In the case of GlassFish we are seeing this accelerated adoption on GlassFish v1, and even on GlassFish v2. Deployment stories are just the last stage of feedback loop that is critical to the success of GlassFish, and we are collecting them at *http://blogs.sun.com/stories*.

An adoption example is Peerflix (*peerflix.com*), the media-sharing community, which recently re-architected their infrastructure and switched from a Microsoft infrastructure to GlassFish v1. The newer site uses JSF with Facelets and Apache Shale for the presentation layer and Kodo JDO for persistence on an MySQL database, all running on Solaris 10. Initial experience with the site has been very positive. Another great example is the high-volume (Australia's third largest commercial site in Australia) wotif.com website choosing GlassFish over JBoss.

Developing GlassFish

GlassFish is developed in a *transparent* manner with community *participation* and *cooperating* with other communities. The community and the development process have had a core impact on the result and have created a product that, when compared with one created through our earlier, traditional, development process, is *better suited* to the needs of the customers, has *higher quality* through a faster and more accurate feedback loop, and goes through a *faster* and *more agile* development cycle.

Some of the impact has been unexpected; for instance, the use of external communication vehicles (web sites, blogs, mailing lists, etc) enables the use of tools like external search engines, blog readers and mailing list archives, and leverages on the activities of the users in the Web. The result is that information flows better across the heterogeneous GlassFish community than across a homogeneous group within a single company even when the group is distributed geographically and across diverse time zones company.

Future Directions

GlassFish is a work in progress. We expect the community and the product to continue to evolve. The community is formalizing its governance (the interim governance board should be announced by the time you read this) and will continue to expand. We also expect the *fabric* of the community to continue to strengthen in multiple ways such as through *GlassFish Days* (like the one before JavaOne 2007) and we want to start some *GlassFish Users Groups*. Note that if the Sun TechDays conference (http://developers.sun.com/events/techdays/) is coming to a city near you, chances are there will be a *GlassFish Day* as well.

Now that the community has delivered high-quality version 2 features, the short-term goal is to increase adoption and deployments. GlassFish projects such as Jersey and Sailfin (see paragraphs below) are important incremental additions. We also want to see more applications and frameworks working with GlassFish, and increase the relationships with other groups and communities.

The medium term is v3, a highly modular container code-named HK2 that will start up very quickly and will use few resources, only activating the services needed to execute the application it is managing. Such a container would be suitable in many scenarios, from very light-weight uses to high-availability ones. Early builds of HK2 are available at http://hk2.dev.java.net.

Jersey – RESTful Web Services

JAX-RS is a new technology being developed within the JCP (JSR 311) to define a POJO-based programming model for RESTful Web Services. Resources are simple annotated POJOs and HTTP verbs are mapped to java methods (see "JAX-RS annotations and POJOs" inset). Consumer and producer annotations further simplify the manipulation of MIME types.

The specification work is going on at http://jsr311.dev.java.net/ (early drafts available there). JAX-RS is listed as a potential component of the upcoming Java EE 6

JAX-RS annotations and POJOs

This is a typical JAX-RS resource using a POJO:

```
@UriTemplate("/helloworld")
public class HelloWorldResource {
  @HttpMethod("GET")
  @ProduceMime("text/plain")
  public String getMessage() {
      return "Hello World";
  }
}
```

Once deployed, the resource will be available using the `"/helloworld"` URI path.

specification (JSR 316).

Jersey (http://jersey.dev.java.net) is the reference implementation for JSR 311 and is closely following the evolution of the specification. The Jersey runtime can be deployed in a variety of environments: GlassFish servlet container, Metro JAX-WS endpoint, Grizzly, Java 6 (using its lightweight web server), Tomcat, etc...

Jersey is really a set of libraries and the installation in GlassFish can be trivial using the Update Center. NetBeans 6.0 is set to support the creation of resources from POJOs as well as the creation of test AJAX-based clients for JAX-RS resources..

Sailfin – SIP Servlets technology for GlassFish

Project SailFin (http://sailfin.dev.java.net) is based on robust and scalable SIP Servlets Technology contributed by Ericsson and the GlassFish v2 application server.

SIP (Session Initiation Protocol) and SIP Servlets are behind many popular services such as Voice-over-IP (VoIP) phone services, instant messaging, presence and buddy list management, and web conferencing. SIP Servlets are expected to play an even bigger part in building the next generation of Telecommunications services.

Combined with Java EE, SIP Servlets can be used to add rich media interactions to Enterprise applications. SIP Servlet 1.1 (JSR 289) updates the SIP Servlets API and defines a standard application programming model to mix SIP Servlets and Java EE components. Leveraging Java EE services like web services, persistence, security and transactions, would enable faster development of smarter communications-enabled applications. Project Sailfin's objective is to implement JSR 289 in its first release.

The daily and milestone builds available from the SailFin website (http://sailfin.dev.java.net) are self-contained and install just like a GlassFish application server. The web site also provides sample documented applications.

The first release of SailFin is scheduled towards the middle of 2008.

Getting Started

A good place where to track the activities of the GlassFish community is *TheAquarium* (*blogs.sun.com/theaquarium*). This group blog provides daily spotlights of the most important activity in the community. A related blog is *Stories* (*blogs.sun.com/stories*) a collection of GlassFish adoption stories related.

The home page for the GlassFish Community is at *java.net* (*glassfish.java.net*); documentation, mailing lists, forums, Wikis are all available there. All GlassFish builds are also available there.

GlassFish builds come in different forms: Nightly and Weekly builds are regular builds from the sources; Milestone builds go through a stabilization cycle and usually include new features – the specific details are described in the associated *Highlight* notes; *Final* builds are the most stable and are the ones we encourage for deployment. Sun provides support for these final builds.

Stay in Touch

We hope this article has succeeded in sketching the benefits of the GlassFish community and Application Server. We hope you will give it a try and find it useful... and we are always very interested in any feedback you may have. See you at: http://glassfish.java.net/ and http://blogs.sun.com/theaquarium.

About the Authors

Eduardo Pelegri-Llopart is a Distinguished Engineer at Sun Microsystems and the overall lead for the GlassFish Community.

Eduardo has worked in a number of Open Source projects and has participated in the Java Community since 1995. He resides in the San Francisco Bay Area.

Yutaka Yoshida is a GlassFish engineer.

Alexis Moussine-Pouchkine is a GlassFish evangelist based in Paris, France.

Table of Contents