RFP 98 OSGi Platform and Java EE integration

Final

18 pages

Abstract

The OSGi platform has attracted considerable attention from Enterprise Java EE application server vendors. The OSGi platform provides a solid foundation for building Enterprise class applications which may access services dynamically provisioned by Java EE containers hosted on an OSGi runtime. A pre-requisite for integrating Java EE with OSGi is a common understanding of the dependencies between the various components that together define the Java EE landscape. This RFP describes those dependencies and specifies common requirements that apply to the integration of any Java EE technology with OSGi. This RFP does not state requirements specific to any Java EE technology or group of Java EE technologies, but refers to other RFPs that do. The requirements stated in this RFP should be satisfied by each RFC that describes any Java EE technology or group of Java EE technologies listed in this RFP.

0 Document Information

0.1 Table of Contents

All Page Within This Box

0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].

```
Source code is shown in this typeface.
```

0.3 Revision History

Revision	Date	Comments
Initial	June 25th 2007	Initial draft created. The focus of this draft is mostly around the second major use case where Java EE is built on top of the OSGi platform. We will add in the other angle in the next revision (or make a separate RFP, to decide in the team meeting). Subbarao Meduri mkv@us.ibm.com Jim Colson jccolson@us.ibm.com
0.2	14 August 2007	Minor update. Improved diagrams in section 2. Remove material now in RFP 100. Subbarao Meduri mkv@us.ibm.com Jim Colson jccolson@us.ibm.com
0.3	6 Aug 2008	Updated Java EE dependency graphs and added additional ones. Removed J2EE 1.3 material to target Java EE 5. Specify correct component technology versions. Added numerous text sections to help describe the dependencies and highlight possible profile options. Jim Colson jccolson@us.ibm.com Michael Keith michael.keith@oracle.com Subbarao Meduri mkv@us.ibm.com Ian Robinson irobins@uk.ibm.com
0.4	1 Oct 2008	Minor changes as a result of EEG review: Added reference to ORB dependencies Footnotes that diagrams are illustrative, not exhaustive Michael Keith michael.keith@oracle.com Subbarao Meduri mkv@us.ibm.com

All Page Within This Box

1 Introduction

Commercial and open source Enterprise Java EE application server vendors have shown considerable interest in building a modular server based on OSGi framework. By decomposing Java EE components and packaging them as OSGi services, these server vendors can build modular dynamic systems rapidly and add incremental support for various constituent (Java EE and non-Java EE) specifications as they evolve independently. This RFP attempts to analyze and understand dependencies of various Java EE specifications in order to better understand and record the decomposition configuration possibilities of EE components.

In addition to the Java EE component technologies there are a number of Java SE technologies that are core to the execution environment of a Java EE application server. The integration of these technologies into OSGi is described in part in other RFPs (e.g RFP 84 and RFP 105) but is also a key part of the experience of any Java EE application. This RFP will outline the expectations on these technologies and touch on the applicable integration points of Java EE applications and their use of these technologies over the underlying OSGi platform.

There are a number of configuration standards and practices, such as the use of annotations and XML descriptors, that affect the deployment of Java EE applications. Although there is currently no broad notion of application deployment in OSGi, there is plainly a level of configuration that is assumed to exist at the bundle level. While the listing of explicit dependencies and enumeration of exported packages, etc, are required parts of living in an OSGi world, Java EE applications developers may or may not be familiar or comfortable with these kinds of practices.

2 Description

A description of the issues is divided into three sections relating to the component technology breakdown, the infrastructure and core services and the configuration of OSGi.

2.1 Java EE 5 Component Taxonomies

To better understand how one can utilize Java EE services in conjunction with an OSGi container, it is useful to look at the constituent elements of Java EE itself.

The Java EE 5.0 specification describes logical relationships between various constituent elements. Such dependency relationships provide insights into how the constituent technologies can be used together to address enterprise use cases.

This section enumerates the entire Java EE 5 landscape and illustrates the dependencies between the Java technologies in Java EE 5 as well as the Java SE technologies these use. In the figures below, “Direct

All Page Within This Box

dependencies" indicate where there is a Java package dependency between the specifications. "Optional dependencies" indicate where there may be a Java package dependency between the specifications if certain optional parts of the specification are used. For example, a JDBC implementation has an optional dependency on JTA if and only if it supports the optional XAResource interface. "Indirect dependencies" indicate a relationship defined in the specification but not manifest in the Java interface, for example where the relationship is through a dependency injection that a Java EE container resolves.

- **Figure** 1 shows the dependencies of Java technologies in a Java EE 5 Web Container. In Java EE 5, Servlet 2.5 and JSP 1.2 MUST be included in a Web Container, while JSP has a mandatory dependency on Servlet APIs. Similarly JSF and JSTL APIs depend on JSP interfaces. These dependencies are depicted as direct dependencies.

- Servlet, JSP, JSF, JAX-WS etc. support common life cycle annotations (JSR 250), and allow for injection of JavaMail, JPA, JTA, JMS, EJB resources. Such dependencies are marked as indirect (soft) dependencies below.

- Additionally, Java EE platform also requires a number of packages (classes and APIs) that must be provided by Java EE containers but are not required to be used. Such dependencies are described as Optional dependencies in the diagrams.

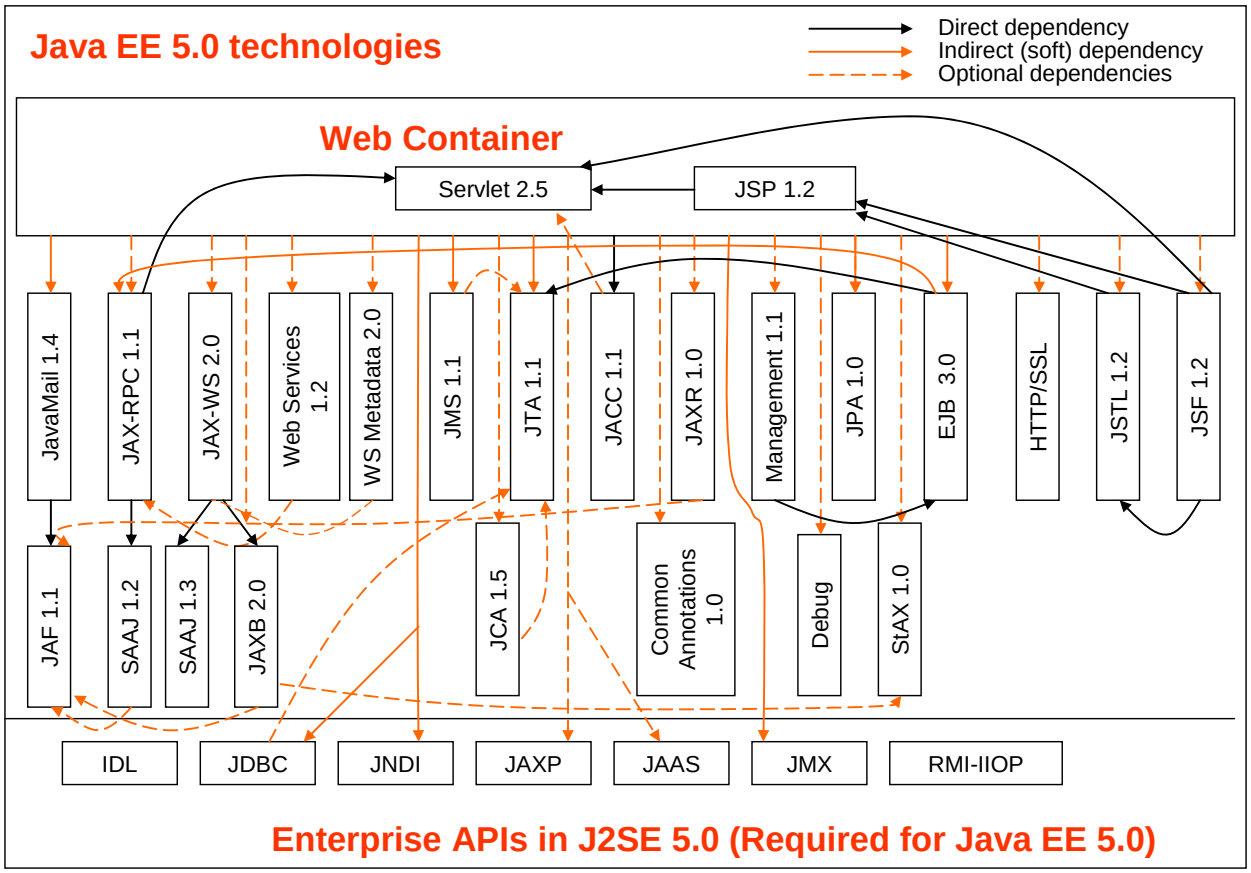

Figure **1** Java EE Web Container[1]

[1] Figure is illustrative in nature and not necessarily an exhaustive depiction

All Page Within This Box

Similarly, Figure 2 shows the relationship between EJB container and other Java EE services:

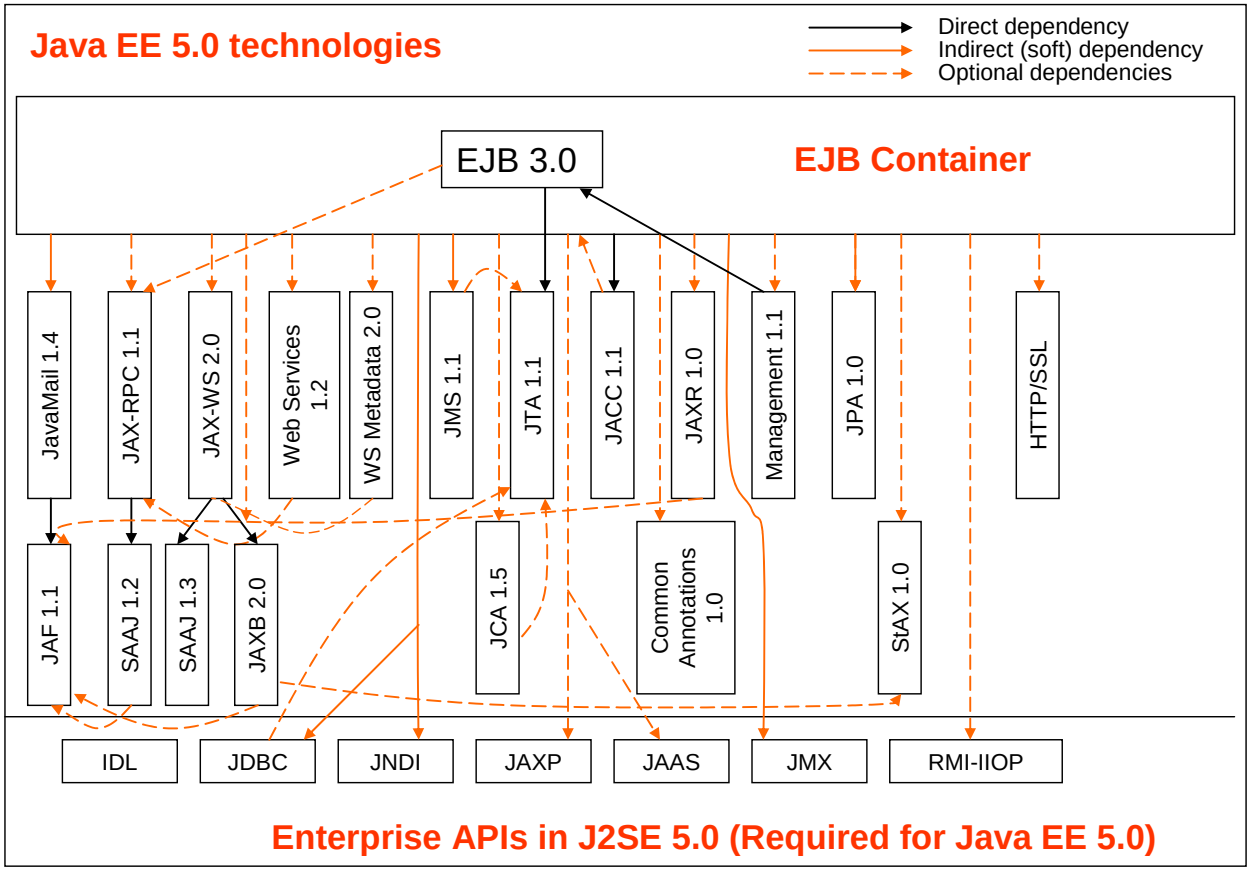

Figure 2 Java EE EJB Container[2]

[2] Figure is illustrative in nature and not necessarily an exhaustive depiction

All Page Within This Box

Figure 3 shows the relationship between Application Client container and Java EE services. Java EE 5 does not require a Java EE client container to provide objects that implement interfaces intended for use in a server but it does require the definitions of such interfaces to be included.

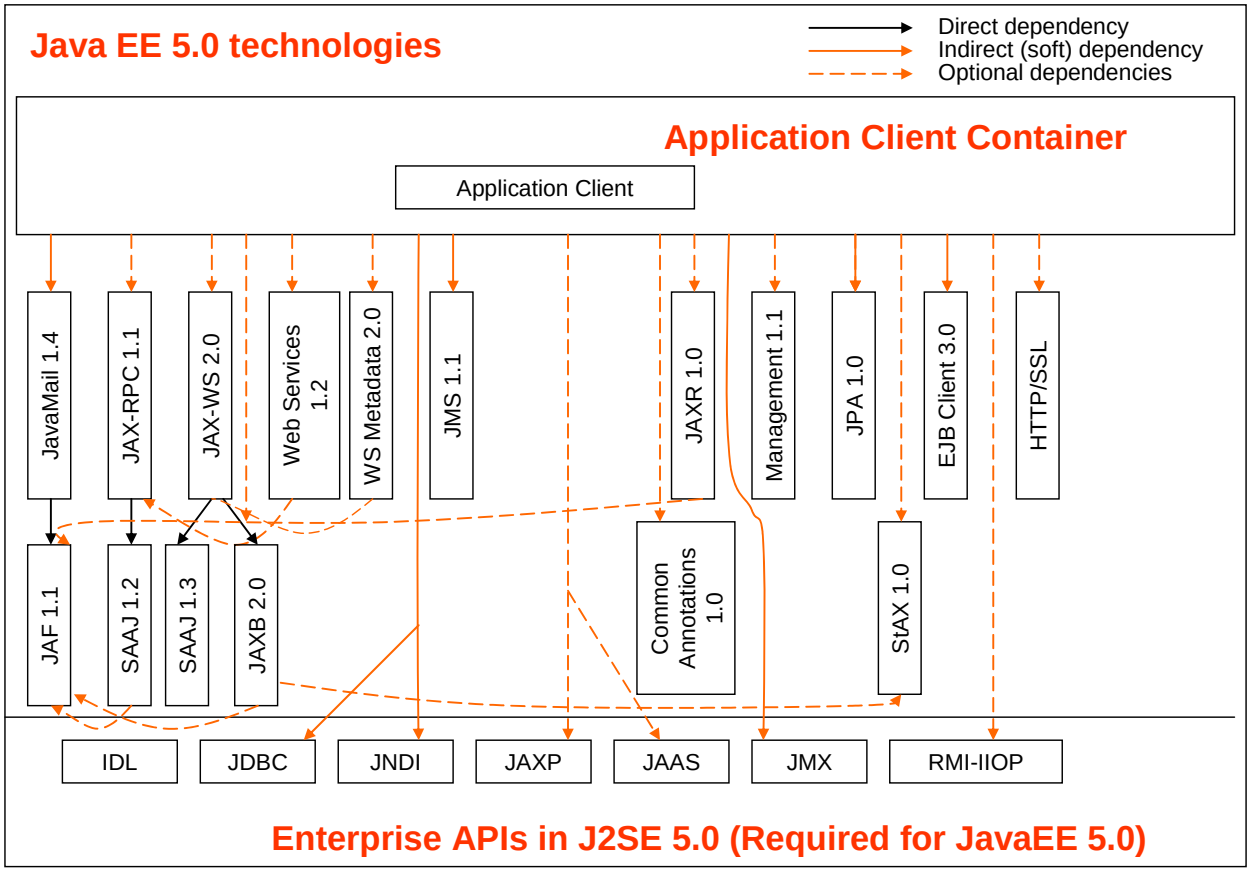

Figure 3 Java EE Application Client Container[3]

[3] Figure is illustrative in nature and not necessarily an exhaustive depiction

Figure 4 illustrates dependencies between the key Java EE and SE technologies across the Java EE landscape without regard to the use by any specific JEE container.

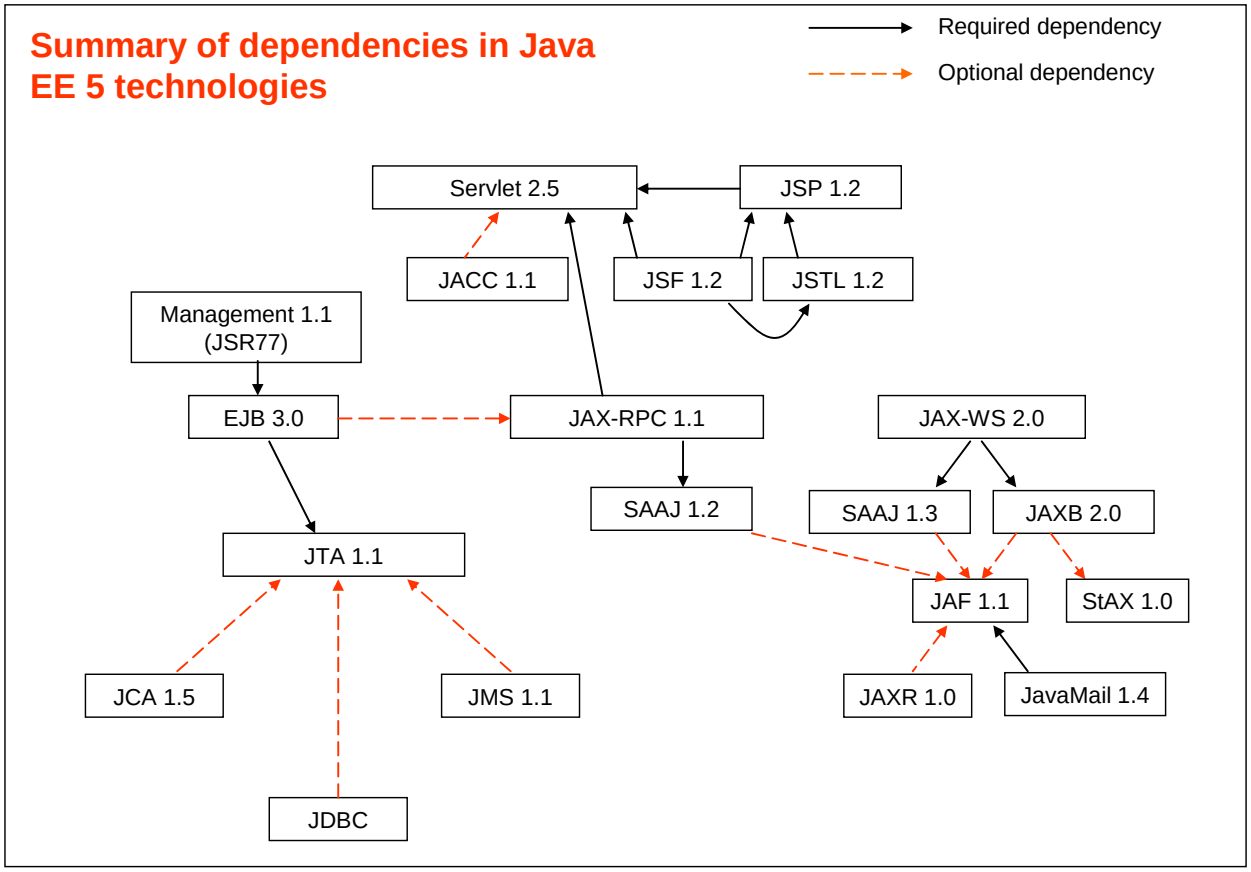

Figure 4 Summary of interface dependencies

All Page Within This Box

The following table lists the technologies that make up Java EE 5 and the JSRs that specify them. Core technologies that are part of Java SE 5 are not included in the list, although JSRs that describe how some of the core technologies function in Java EE are listed.

UMBRELLA TECHNOLOGIES	
Java EE 5	Platform specification - JSR 244
Common Annotations 1.0	Used in both Java EE and Java SE - JSR 250
WEB TECHNOLOGIES	
JSP 2.1	JavaServer Pages, EL – JSR 245
JSTL 1.2	JavaServer Pages Standard Tag Library - JSR 52
JSF 1.2	JavaServer Faces - JSR 252
Servlet 2.5	Java Servlet - JSR 154
APPLICATION TECHNOLOGIES	
EJB 3.0	Enterprise JavaBeans - JSR 220
JCA 1.5	J2EE Connector Architecture - JSR 112
JAF 1.1	JavaBeans Activation Framework - JSR 925
JavaMail 1.4	JavaMail - JSR 919
JPA 1.0	Java Persistence API – JSR 220
JMS 1.1	Java Message Service – JSR 914 + Maintenance Release
JTA 1.1	Java Transaction API – JSR 907 + Maintenance Release
WEB SERVICES TECHNOLOGIES	
Java EE WS 1.2	Web Services for Java EE – JSR 109
WS Metadata 2.0	Web Services Annotations - JSR 181
JAX-WS 2.0	Java API for XML-Based Web Services - JSR 224
JAX-RPC 1.1	Java API for XML-Based RPC - JSR 101
JAXB 2.0	Java Architecture for XML Binding - JSR 222
JAXR 1.0	Java API for XML Registries – JSR 93

All Page Within This Box

SAAJ 1.3	SOAP with Attachments API for Java – JSR 67 + Maintenance Release
StAX 1.0	Streaming API for XML – JSR 173
MGMT/SECURITY TECHNOLOGIES	
MGMT 1.1	J2EE Management – JSR 77
DEPLOY 1.2	J2EE Application Deployment – JSR 88
JACC 1.1	Java Authorization Contract for Containers – JSR 115

Following is a list of currently proposed RFPs/RFCs that cover some aspects of these various Java EE 5 technologies

- RFC 66: OSGi Web Container
- RFP 85: Web Application support
- RFC 98 – Transactions (JTA)
- RFP 84 – JNDI Integration
- RFP 105 – JMX management of OSGi
- RFC 122 – Database access
- RFP 103 – Classloading improvements
- RFC 133 – Context classloader
- New RFP needed for JPA integration

2.1.1 Technologies in Java SE

This group consists of programming artifacts or deployment/runtime requirements, or usage descriptions, of existing Java SE technologies, but applied to Java EE. This category overlaps with the section on infrastructure technologies and will be discussed there.

- MGMT
- DEPLOY
- JACC
- Common Annotations

All Page Within This Box

2.1.2 Prospective SE Components

Some technologies are not only standalone but have in fact already been added to Java SE 6.

This category is worth noting because these technologies will be part of the standard runtime library and therefore "universally available" on the system classpath in Java SE 6. These technologies should be relatively straightforward to integrate by simply placing them on the shared classpath in OSGi. It should be possible to include each of these components individually and independently, taking the noted exceptions into account.

- JAX-WS 2.0 – requires JAXB
- JAXB 2.0
- SAAJ 1.3
- WS Metadata 1.0
- StAX 1.0
- JAF 1.1

2.1.3 Standalone Components

A number of component technologies were/are intended to work in a standalone Java SE environment, or have no required dependencies on other EE components. Although defined as part of the Java EE platform, these technologies have either standalone TCKs, or can be used in Java SE without an EE container. They may (and usually do), have integration points with other Java EE components when they are used in the EE container. It should be possible, when being used in standalone mode, to use these components individually and independently, except where noted.

- JPA
- JTA
- JavaMail (optional Java SE 5 technology; needs JAF)

2.1.4 Web Components

Although web technologies can be used with additional EE components they do not require them and are often used without them. They are listed in a separate category because of their prevalence. The common use case is the web application developed using only the web tier and SE technologies (optionally with other add-on framework technologies). It should be possible to group together these components as a web application bundle since they have inter-dependencies but no hard dependencies on other components. It may also be possible to use only specific components, for example Servlets, JSP and JSPTL, without JSF.

- Servlet 2.5
- JSF 1.2
- JSP 2.1
- JSTL 1.2

All Page Within This Box

2.1.5 Platform Components

The component technologies and descriptions that are intended to run in the context of the (entire) platform are typically are the value-add Java EE technologies, or those that contribute core functionality or portability to the platform. The first two are just prescriptive descriptions of some of the other technologies, while the last three have hard API dependencies on JTA transactions.

- Java EE 5 (no code)
- Java EE WS 1.2 (no code)
- EJB 3.0
- JCA 1.5
- JMS 1.1

2.1.6 Miscellaneous Components

Missing from the above categories is JAX-RPC, which was really in Java EE 5 primarily for the purposes of backwards compatibility. This component has no hard dependencies.

JAXR is only used by web services applications and has no hard dependencies on other components.

The subset of EJB persistence entity bean technology (CMP/BMP) is deprecated in practice (though not strictly until a subsequent EE version), and support for it may arguably be omitted going forward.

2.1.7 Schema Dependencies

Another dependency set worth noting is that of XSD schemas. Most of the schemas associated with the Java EE components rely on the base Java EE platform schema **javaee_5.xsd**, and sometimes others. Given that the suite of schemas is often grouped together and shipped by vendors as resources separate from the actual component APIs, we will not consider these as strict dependencies in the conventional sense. However, if a specific component schema is included by another component schema then there will likely be at least a soft dependency from one to the other, and it can be assumed that this will be captured in the document specific to the respective component.

2.1.8 Soft Dependencies

While the APIs of one component technology may have compile-time dependencies on a subset of other technologies, it may also require other technologies because of implementation requirements that are dictated by the specification. For example, an EJB does not in and of itself need to have a dependency on JAX-WS, but if it calls out to a web service, or is specified as a web service endpoint, then JAX-WS (or JAX-RPC) becomes a required bundle of the implementation. Another example is the ORB that is not part of any component API, but will be required to pass some of the TCK tests associated with certain components, The fan-out of additional component technologies that will be required through the normal course of execution is actually, therefore, much broader and more encompassing than the simple list of hard dependencies.

2.1.8.1 Resource Injection

The Java EE spec allows injection of resources into a predetermined set of component classes. These include:

Servlet, JSP, JSF, JAX-WS, EJB

All Page Within This Box

This implies that any one of these components has to look for and process annotations and then inject the given resource object depending upon the type of the injection target. Thus, each of the components in the above list may have dependencies on the annotations, plus the injectable component technologies in the list of:

Common Annotations, EJB, JPA, JTA, JMS, JavaMail, JAX-WS, JAX-RPC

2.1.8.2 Web Services

Java EE Web Services specifies that a web service may be implemented by an EJB (stateless session bean) or a JAX-RPC/JAX-WS servlet-based class. When using the Web services for Java EE there is a dependency on WS-Metadata, Servlet and EJB during discovery, processing and generating WSDL etc. in the case of a web service implementation class.

Note that in theory an application can invoke almost any API from a given component, thus a matrix showing the dependencies a given component has on others might not be realistic.

2.2 Infrastructure

2.2.1 Security

The minimal permission set must be permitted as per the Java EE 5 specification. Other standard security integrations should also be possible, for example, it must be possible to use JAAS and JACC as required.

2.2.2 ClassLoading

The classloading requirements of Java EE are not particularly strict or well-defined, but there do exist some expectations and guarantees that applications have come to rely on.

Context Class Loader – Thread.getContextClassLoader() method must return a class loader that can be accessed by an application to load any application-specific classes on the application-specific classpath.

Manifest classpath entries in EJB JARs, WARs or RARs must be honoured unless the referencing artifact is being deployed outside of an application EAR.

The JAR files in library directories of application EARs must be placed on the application classpath and be accessible to the application classes.

Classes accessed from a specific component technology used in an application must also be able to access other application-level classes that may not be part of the same component technology.

Given the Java EE model for class loader visibility that is used in conventional EAR and WAR file deployment, there needs to be a common, agreed to, mapping to the standard OSGI class loader framework architecture. This will allow existing Java EE applications to run on any Java EE/OSGi based system unchanged; this use case is considered by RFC 133.

2.2.3 Interoperability

Where necessary, it should be possible for Java EE components to use Java IDL to interoperate with external CORBA facilities or RMI-IIOP to interoperate with external Java EE components from a different vendor. Java EE vendors are further required to provide an ORB implementation. Any implementation of an RFC for a Java EE technology will need to address OSGi enabling such technologies. This effort should not be underestimated and

will need to consider compliance requirements for those technologies, such as CSIv2 for inter-ORB communication.

2.2.4 Common Annotations

Annotations are optional in Java EE, thus the common annotations code used for annotating EE components should also be an optional package. When used, it should be available to all other components.

2.2.5 JNDI

The EE 5 environment naming context (ENC) is defined to be component-based. To adhere to the component isolation of EE the local naming contexts should be different within each component execution context.

Global JNDI is often supported by Java EE containers. Support for global JNDI will not be required in an OSGi Java EE runtime, but wiill not be disallowed in the event that containers wish to provide it.

Whether JNDI lookups return a singleton, or make use of a factory to cause new instantiations of objects, the result of a JNDI lookup should be the same within an OSGi environment as in any other Java EE environment.

2.2.6 Management

The EE management extensions (JSR 77) do not have any hard dependencies on other components, but are prescribed in terms of JMX implementations. Because of the broad scope of managing the server it is not uncommon for the server-side management code to be woven throughout the various EE components, creating a management aspect that crosscuts the server. It may not be practical at this stage to require that the management aspect be unloadable and isolated from the components that are being accessed or operated on, however it should be permitted.

2.3 Configuration and Deployment

Existing Java EE applications are configured through a combination of XML deployment descriptors and annotations. Standard Java EE configurations will continue to be valid in a Java EE/OSGi runtime, but additional artifacts or manifest entries may also be required by vendors in order to complete the OSGi-specific configuration.

Packaging and deploying EAR files, or standalone component jar files, such as a JAR or WAR file, will continue to be a valid deployment configuration, with the same additional artifact proviso as previously mentioned. Existing Java EE artifacts should not be modified or extended for the purposes of deployment into OSGi.

OSGi/Java EE containers may support isolating and unloading the deployment API (JSR 88) from the rest of the server, but this specification does not mandate that it be supported.

2.4 OSGi Services

It should be possible for an Java EE application, written with the assumption of being run within an OSGi runtime, to access and invoke OSGi services. In other words, all of the normal OSGi functionality could be available to a Java EE application. Container runtimes may or may not permit access to services that bypass the typical access restrictions that apply to applications running in Java EE containers. For example, it might not be valid to access a File Reading Service from within an EJB.

3 Use Cases

The following use cases should be considered when specifying how to create a hybrid OSGi/Java EE based system.

3.1 “Fit for purpose” runtime

In this use case, we want to enable the construction of a “fit for purpose” runtime that enables Java EE programming model artifacts without requiring a complete Java EE runtime to be present. This could be used in resource constrained systems such as mobile devices or network intermediaries, systems that require fast boot times and cannot afford superfluous services, or systems that need to be composed with other programming model elements. For example, the most common types of Java EE application are web applications containing no EJB components. RFC 66 is specifically targeted at the web application use case.

3.2 Continuous Availability

In a system requiring continuous availability, the state of the art is to allow for complete redundancy of hardware systems and exploit recycling of the hardware, with subsequent software updates being applied between cycles to repair when a system goes awry. It is useful to consider the recycling of individual Java EE services in order to better utilize hardware resources. This has implications not only on the Java EE components, but on the applications that are written to exploit them if the Java EE services themselves now become dynamic. Consider, for instance, a set of long running web applications that utilize a web container and the desire to update the web container to patch a security hole. It is important to exploit this dynamicity while constraining the impact of the underlying complexity to the administrator and explicitly not impacting the application programmer.

3.3 Incremental Application Update

Applications in Java EE are deployed as components in an EAR file. Incremental and dynamic updates to components inside the application are supported by many Java EE vendors but not a requirement of the Java EE specification. In a Java EE/OSGi based system, an application developer will typically be building applications from fine-grained components and incremental update of those components needs to be considered without requiring the redeployment of the whole Java EE application.

4 Requirements

The following requirements should be considered by any RFC that references Java EE technologies:

All Page Within This Box

1. A Java EE/OSGi system SHOULD enable the standard Java EE application artifacts (e.g. web application) to remain installed when a supporting Java EE runtime element (e.g. web container) is dynamically replaced.

2. RFCs that refer to one or more Java EE technologies MUST NOT impede the ability of an OSGI-compliant implementation to also be compliant with the Java EE specification.

3. RFCs that refer to one or more Java EE technologies MAY define the additional aspects of the technology that are required for the technology to be properly integrated in an OSGi framework but MUST NOT make any syntactic changes to the Java interfaces defined by those Java EE specifications.

4. RFCs whose primary purpose is integration with Java EE technologies MUST NOT require an OSGi Execution Environment greater than that which satisfies only the signatures of those Java EE technologies.

5 Document Support

5.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. Java Platform Enterprise Edition 5 specification http://java.sun.com

5.2 Author's address

Name	Subbarao Meduri
Company	IBM
Address	4205 South Miami Blvd, Durham. NC 27703 USA
Voice	(919) 254-5210
e-mail	mkv@us.ibm.com

All Page Within This Box

Name	Michael Keith
Company	Oracle Corporation
Address	
Voice	(613) 288-4625
e-mail	michael.keith@oracle.com

5.3 End of Document

All Page Within This Box