RFP 80 OSGi Framework Booting

Open Distribution, Draft

6 Pages

Abstract

OSGi frameworks should have a standard mechanism for booting.

0 Document Information

0.1 Table of Contents

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 6.1.

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

0.3 Revision History

The last named individual in this history is currently responsible for this document.

All Page Within This Box

Revision	Date	Comments
Initial	April 16, 2007	Initial draft created. John Wells, BEA Systems, Inc. jwells@bea.com
0.1	July 11, 2007	Reviews based on review

1 Introduction

Every OSGi framework has a different mechanism to get booted inside a java process. If there were standards around how an OSGi framework may be booted and used within a JVM many vendor lock-in concerns could be addressed.

2 Application Domain

The problem domain is any person or company who wishes to use an OSGi framework for their application but who does not want to have the risks associated with vendor lock-in.

2.1 Terminology + Abbreviations

- Compliant OSGi framework: In this paper, a compliant OSGi framework is one that implements a standard solution that accounts for the use cases defined in this document.

3 Problem Description

Every OSGi framework must invent its own technology for getting started. Therefore, code that may need to boot the framework from various places must write code that is proprietary for the particular OSGi vendors' implementation, if they wish to support more than one framework.

In enterprises, where the issues of vendor lock-in can cause a barrier to adoption of the system, this issue becomes magnified. While it should certainly be the case that each vendor can supply add-ons and extra

features, the standard portions of all OSGi frameworks could be encapsulated in a Framework Booting specification. Having such a specification would increase the consistency and quality of compliant OSGi frameworks, and allay fears about vendor lock-in.

4 Use Cases

This is not an exhaustive list of use cases that should be supported by any RFC in response to this proposal, but should instead be treated as representative of typical scenarios.

4.1 Script tooling

An IT shop would like to write shell scripts that integrated the booting of the OSGi framework with the boot of the hardware platform. They would like to write a Korn shell script for their Solaris boxes, a DOS CMD script for their window boxes and a Perl script for their Linux boxes. However, it should be configurable which OSGi platform should be the one that is booted.

These script writers would like to have a standard java command line (that optionally takes vendor specific arguments) that will be guaranteed to work across all different OSGi frameworks.

4.2 Generic OSGi Launcher Capability

A company wants to write a general purpose GUI launcher of OSGi frameworks. This utility must have the following features:

1. It must be able to run any compliant OSGi framework. This includes, but is not limited to:
    a. Open-source OSGi frameworks such as Equinox, Knopflerfish and Felix
    b. For-pay frameworks
    c. Frameworks that do not yet exist today
2. It must be able to run the OSGi framework inside the same process as itself, or start a new process
3. It is ok to make the user pass special arguments for individual frameworks
4. The user must be able to pass a list of bundles that should be installed and/or started. This list must be able to be chosen programmatically, and not require construction of an internal or external ZIP file.
5. The utility should be able to determine if the framework is restarting, and list the existing bundles to be started without starting the framework. In this way, the user can decide if they would like to change the list of bundles that should be started on this launch of the framework.

4.3 Standard Applications

JEE and various other standards bodies have defined application packaging formats which are deployed onto server containers. Examples of these application packaging formats include but are not limited to:

1. Web Applications (war files - http://java.sun.com/products/servlet/)
2. Enterprise Java Beans (ear files - http://java.sun.com/products/ejb/)

All Page Within This Box

3. JEE Connectors (rar files - http://java.sun.com/j2ee/connector/)

Any instance of any of these applications may have chosen to include an OSGi implementation which they start when the application is first loaded into the container. The application then gets to use the many benefits of OSGi inside their application code including package hiding and use of the service registry. This works very well and there are real examples of this in open source projects (for example, see http://www.eclipse.org/birt/phoenix/).

However, this practice causes a problem when the server container itself is already running OSGi. In particular, the following scenarios are possible:

1. An application of one of the types listed above wishes to start an OSGi runtime and is deployed to a server that is already running inside an OSGi runtime.
2. Multiple applications of the same type (say two "WAR" files) both want to start an OSGi runtime and they are deployed to a single server that is not running OSGi.
3. Applications of different types (say an "EAR", a "WAR" and a "RAR") all wish to start OSGi runtimes and are deployed to a single server that is not running OSGi.
4. Multiple applications of the same and different types all wish to start OSGi runtimes and they are deployed to a single server that is already running inside an OSGi runtime.

There are various combinations of things which may occur in this use case:

1. The OSGi implementation in the various applications may be from different vendors.
2. The OSGi implementation in the application may differ in version or vendor from the OSGi implementation used by the server

There may also be different qualities of service that applications need from the OSGi environment. For example for one application it may be acceptable to simply "join in" to an existing OSGi registry and package name-space if there is already one running in the process. However, another application may wish to have their own "virtual" OSGi runtime which does not share the service registry or package namespace with other OSGi environments within the same JVM.

5 Requirements

1. The solution SHOULD allow for scripting languages to behave similarly for all compliant OSGi frameworks.
2. The solution MUST allow for starting, restarting and stopping compliant OSGi frameworks without prior knowledge of the framework.
3. The solution SHOULD allow for starting compliant frameworks in the same or different processes (though it would be OK to fall back on the Java Process verbs if necessary)
4. The solution SHOULD handle the problems associated with booting multiple OSGi frameworks inside the same JVM process space.
5. The solution MUST be able to handle vendor specific extensions.

6 Document Support

6.1 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

6.2 Author's Address

Name	John Wells
Company	BEA Systems, Inc.
Address	140 Allen Road, Liberty Corner NJ 07938
Voice	+1 908 580 3127
e-mail	jwells@bea.com

6.3 End of Document