Multi-Tenancy

Early Draft

8 Pages

Abstract

An investigation into the possibility for a multi-tenancy setup that can be useful in the context of cloud, grid, and utility computing. The shared platform of cloud computing requires control over isolation as well as mechanisms to avoid interference between unrelated applications of different tenants.

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.

Revision	Date	Comments
Initial	*May 03 2013*	*Initial* *Steffen Kächele, University of Ulm steffen.kaechele@uni-ulm.de*
0.1	*Sept. 06 2013*	*Prepared version for Southampton F2F* *Steffen Kächele, Univerisity of Ulm, steffen.kaechele@uni-ulm.de*

1 Introduction

OSGi was used to modularize single applications. New use-cases introduce scenarios where applications run on shared third-party hardware. One example is a cloud computing scenario with multiple tenants. Umbrella RFP 133 initially discusses the potential of OSGi in cloud computing setups. RFC 183 discusses a technical architecture to integrate OSGi frameworks in a distributed cloud like ecosystem. However, the deployment of unrelated application of different tenants is currently left open.

Yet, when deploying multiple applications on the same host, there is the need for isolation. One option to isolate applications are Subsystems as proposed in RFC 152 and RFC 201. Subsystems are a collection of bundles for a particular feature. They have a declarative model for resource collections and an API to install and manage them. On a shared multi-tenant platform, subsystems only provide minor isolation. They control import and export and access to services. Subsystems of the type “application” are defined to have no exports to the outside and prefer internal bundles when resolving imports. When there is need for stronger isolation, technical solutions such as dedicated hardware, hardware visualization, processes, and separation in the runtime environment can help to separate OSGi applications. Yet, such multi-tenancy aspect are not covered by OSGi. Isolation might seen as an orthogonal aspect, but may highly affect the component model. Subsystems focus on separating a distinct set of bundle in more subsets, but does not consequently isolate them in case of control and resource access. Moreover, they lack a concept for a distributed multi-tenant platform.

For management purposes, RFC182 defines a REST interface to control framework instances remotely. Yet, it does not cover multi-tenant scenarios.

RFP 200 proposes several solutions for resource management that can be valuable when establishing isolation.

This RFP discusses the integration of multi-tenancy platforms with regard to OSGi. Subsystems can see as one solution to isolate applications (alternatives are virtual machines, framework instances and separate JVMs) and discusses with other options are there. It furthermore discusses how we can archive isolation between applications and access to resources. It addresses the management of isolated deployment on multiple hosts.

1.1 Design options

The OSGi Core specification currently does not define how to deploy multiple applications on shared infrastructure. Yet, there are multiple solutions in order to do that.

1.1.1 Subsystems

The first option is to use the Subsystem specification to deploy multiple application within one framework. In this case, Subsystems control access of wiring, but resource isolation is quite low. Systems give a 'keep out of each other's way" isolation model. Thus, it is simple to punch though the isolation to see what is in other subsystems.

1.1.2 OSGi framework instances

Another option is to run multiple (virtualized) framework instances, one for each application. These frameworks can still run in a single JVM. Additional runtime support can help to isolate different applications.

1.1.3 JVMs and bare metal JVMs

A higher isolation can be achieved by isolating framework instances using separate JVM processes. Thus, applications isolated on operating system level.

1.1.4 Hardware or hardware virtualization

The easiest and most isolated way to separate applications is to use dedicated (virtual) machines. Frameworks are well isolated on (virtualized) hardware level. For scalability and in conjunction with RFC 183 multiple (virtualized) nodes can be combined to a distributed ecosystem.

2 Application Domain

2.1 Introduction

IT outsourcing and consolidation of infrastructure is a major IT trend. New cloud computing platforms help to increase server utilization and lower IT costs. In these scenarios, typically a lot of unrelated applications of different tenants are running on a shared third-party infrastructure. The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Yet, running applications on a shared infrastructure requires isolation. Tenants should only see their own environment while being isolated to other tenants. This affects computing, memory, storage and network.

A big advantage of OSGi is fine-grained deployment of applications. Using this fine-grained deployment on application level may improve the resource utilization and save resources. Integrated multi-tenancy support can further improve utilization and ease application deployment.

2.2 Terminology

Provider	The entity that manages the infrastructure to host applications.
Application	An application is a composition of OSGi bundles for a particular purpose.
Tenant	A tenant uses the platform provided by the provider and deploys one or applications. Tenants are unrelated to each other.
Node	A computing node that hosts the framework.

Framework An instance of an OSGi framework implementation.

3 Problem Description

3.1 Isolating applications

Originally, OSGi was not designed as a multi-tenant framework to host different unrelated applications. Subsystems already present an OSGi-like way to aggregate bundles to applications. It furthermore can control import and export dependencies as well as wiring. Depending on the use-case, applications however may require a higher level of isolation. Examples are the restriction to special framework functionality as well as limits for CPU and memory usage.

There are several design options for a multi-tenancy deployment. As demonstrated in Section 1.1, applications can be separated by Subsystems, with dedicated OSGi framework instances, on JVM level or by using (virtualized) hardware. The RFC has to integrate current solutions in a common OSGi appropriate way.

3.2 Resource access

Usage of resource may interfere between users. In conjunction with the OSGi security manager, code loading can be limited to particular resources. Examples are:

3.2.1 Network access and Sockets

When using multiple application on the same host, they can interfere when using the same IP address. For example two applications may want to start a webserver on port 8080 (This problem does not exist when using multiple (virtual) hosts).

3.2.2 Files

Even though OSGi defines bundle own storage areas, it does not prohibit to access the local file system. When a file system is available OSGi should limit access to file system for each application to a dedicated storage area.

3.2.3 Database and other resources

When using database and other resources services, the framework must ensure that only the tenant that is allowed to use the partition of data can read and write it.

3.3 Resource management

Related to resource access is resource management, in case when multiple application access limited resources of a single computer. Examples are memory, CPU resources and storage. Multi-tenancy management may limit it for particular applications.

3.4 Scaling support

The solution should propose a way to attach applications to users even in a complex distributed environment.

3.5 Programming language level interference

There are also some programming language level issues. In particular, when different applications share code, there might be conflict. (“org.osgi.framework.bsnversion” in Core R5 can help here)

3.5.1 Static Variables

One problem are static variables. If multiple bundles of separate applications are internally wired to the same bundle, the may use different instances but share the same static variables.

3.5.2 Singletons

Another problem are singletons.

3.6 Tenant and application management

New tenants appear and disappear, they create applications and remove them. With OSGi as a lightweight container, this can done very rapidly. However, currently there is no multi-tenant API to realize these jobs.

3.7 Access control for bundles and framework

In the OSGi framework functions are accessible independent of the tenant it uses. Bundles can be installed, uninstalled and updated. However, current API does not require authentication. The API should provide a way to authenticate tenants and provide dedicated access to their resources.

4 Use Cases

4.1 Company

A company organizes a big server park. It hosts component-based applications of different departments with changing workloads. For better utilization, the company shares machines between the departments with a lightweight isolation mechanism. Thus, it is able to scale-out and move applications quickly.

4.2 Cloud Provider

A cloud provider hosts several applications. It uses a strong isolation mechanism.

4.3 Development

A company hosts a large component-based application. For development, the use the same infrastructure but a dedicated isolated environment. During software development, each department can work in its own testing environment.

5 Requirements

5.1 Applications

APP-1 The solution MUST restrict access control to bundles of one tenant.

APP-2 Framework instances MUST provide exactly the same interface as specified in Core specification.

APP-3 Each application has to be wired and control as it runs in a dedicated framework.

APP-4 Bundle must be isolated. Wiring between applications have to be prohibited. In particular, it have to be possible to use the same bundle in the same version multiple times, but isolating static variables.

APP-5 The solution MUST define a descriptor that defines the requirements for resource isolation (hardware, process, OSGi, subsystem) and resource access (network, file).

5.2 Management

MAN-1 The solution MUST define how providers can handle multiple applications.

MAN-2 The solution SHOULD propose a way to map applications to tenants even in a complex distributed environment.

MAN-3 Each framework MUST be fully controllable in the way to add/remove and update bundles.

MAN-4 The solution MUST define how to provide isolation when using resources (i.e. network, files, ...).

MAN-5 The solution MUST define an API for providers to create new applications for a dedicated tenant.

MAN-6 The solution MUST define interfaces to generate new framework instances.

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	Steffen Kächele
Company	University of Ulm
Address	Albert-Einstein-Allee 11 89081 Ulm
Voice	
e-mail	steffen.kaechele@uni-ulm.de

6.3 End of Document