RFC-131 Multiple Service Provider Separation

Final

24 Pages

Abstract

This document formalizes the requirements and technical solution needed to realize the multiple service provider separation in OSGi world. The “multiple service providers” model is one of the business models known for the OSGi service platform. In the model, multiple service providers offer their services on a single OSGi service platform in the end user’s premises. This document discusses the concept of “separation”, which simplifies the management of “multiple service providers” model and introduces ServicePermission extension and PackagePermission extension.

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

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

Revision	Date	Comments
Initial	*Apr 7, 2008*	Initial draft created. Koya Mori NTT mori.kouya@lab.ntt.co.jp Hiroyuki Maeomichi NTT maeomichi.hiroyuki@lab.ntt.co.jp Ikuo Yamasaki NTT yamasaki.ikuo@lab.ntt.co.jp
2nd draft	*Jun 4, 2008*	Second draft based on REG meeting. InstancePermission is changed to ServicePermission extension. BundlePackagePermission is changed to PackagePermission extension. JavaDocs of the extended ServicePermission and PackagePermission are added to the technical solution. Koya Mori NTT mori.kouya@lab.ntt.co.jp

All Page Within This Box

Revision	Date	Comments
3rd draft	*Sep2, 2008*	Third draft based on REG F2F meeting and conference call. The part of SeparationAdmin is removed from technical solutions. Explanation and Java Docs about ServicePermission extension and PackagePermission extension are modified. In addition, permission check for modification of service properties is added. Ikuo Yamasaki NTT, yamasaki.ikuo@lab.ntt.co.jp
4rd draft	*Nov 7, 2008*	Updated draft based on CPEG F2F meeting in Sep 2008. Sec. 5.4.2, 5.4.3, are 5.5.2 are updated. Sec. 5.4.4 is newly added. TBD1 is kept as it was. Current limitation is described in the Sec. 7. Ikuo Yamasaki NTT, yamasaki.ikuo@lab.ntt.co.jp
5th draft	*Dec 5*	Updated draft based on CPEG & REG F2F meeting in Nov 2008. When modifying properties, checking ServicePermission of not the caller but the registering bundle MUST be required and SecurityException MULT be thrown if it does not have. It keeps backward compatible. Ikuo Yamasaki NTT, yamasaki.ikuo@lab.ntt.co.jp
6th draft	*Jan 9*	Updated draft Javadoc are totally renewed just for better description. It is copied and pasted from reference implementation source code provided by NTT. No change for technical point. Ikuo Yamasaki NTT, yamasaki.ikuo@lab.ntt.co.jp
7th draft	*Feb 12*	Updated draft • PackagePermission supports package attributes filtering. • “osgi.bundle.version” is added for ServicePermission. • Considered Alternative regarding symmetric design was added. • Javadoc and some explanation in the doc are updated. Ikuo Yamasaki NTT, yamasaki.ikuo@lab.ntt.co.jp
8th draft	*Feb 13*	Updated draft • Clarify that current PackagePermission constructor cannot support arbitrary attributes in Export-Package header. Ikuo Yamasaki NTT, yamasaki.ikuo@lab.ntt.co.jp

All Page Within This Box

Revision	Date	Comments
9th draft	*Mar 9*	Updated draft based on long discussion on many conference calls and F2F in March 2009. • Extended ServicePermission supports NO filter String as its name for REGISTER action. • Extended ServicePermission supports filter String on bundle identifiers and service properties as its name for GET action. • Extended PackagePermission supports NO filter String as its name for EXPORT action. • Extended PackagePermission supports filter String on bundle identifiers as its name for IMPORT action. • In extended PackagePermission, EXPORT action has been deprecated and EXPORTONLY action has been newly defined. Ikuo Yamasaki NTT, yamasaki.ikuo@lab.ntt.co.jp

1 Introduction

This RFC defines a separation mechanism for multiple service providers. The requirements are described in RFP-0095.

The “multiple service providers” model is one of the business models known for the OSGi service platform. In the model, multiple service providers offer their services on a single OSGi service platform in the end user’s premises. This document discusses the concept of “separation”, which simplifies the management of “multiple service providers” model. After that, the document introduces ServicePermission extension and PackagePermission extension to realize the separation in OSGi world.

All Page Within This Box

2 Application Domain

In the “Multiple Service Providers” model, three entities play key roles: (1) service provider, (2) end user, and (3) service aggregator. A service provider provides one or more service applications to the end user by delivering them as a bundle on the OSGi service platform in the end user’s premises (Fig. 1). The end user can subscribe to many service applications provided simultaneously by multiple service providers. The service aggregator has the most important role since it offers the following three functions.

(1) Remote management of equipment that realizes OSGi service platform.

(2) Delivery of bundles provided by service providers.

(3) Resolution of conflicts between service applications. This is to prevent improper interaction between service applications, and to keep service applications running correctly and safely.

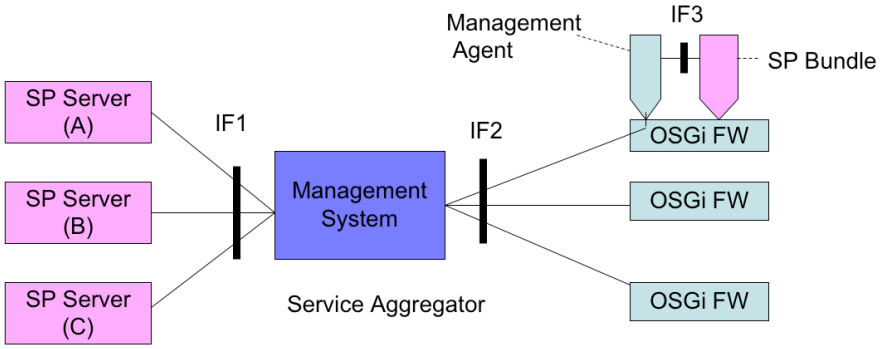

Figure 1: Remote Management Reference Architecture

Several kinds of activities are needed to implement conflict management. Some of them can be greatly simplified by introducing the concept of “separation”. In the concept, there are several areas, each of which is dedicated to one unit, mainly a service provider. Interactions between different areas are restricted, while interactions within the same area are freely accessible.

For example, multiple Service Providers provide Service Applications that consist of bundles and resources through the Service Aggregator. These Service Applications are concurrently deployed on a single OSGi Framework. The Service Aggregator should separate the accesses of the Service Applications to ensure the service operation desired by the Service Providers. Basically, the Service Aggregator should avoid giving permissions across different Service Applications.

All Page Within This Box

3 Problem Description

Multiple Service Applications must be operated on a single OSGi framework to aggregate different Service Providers in remote management architecture. These Service Applications' bundles are able to utilize not only packages and services provided by the same Service Application, but also other resources, as shown below.

1. Packages and services provided by other Service Applications' bundles.

2. Resources such as files, sockets, and system properties, which can be shared between Service Applications.

3. The system bundles and framework, which are defined in OSGi specifications.

Access to the resources can be controlled by setting permissions for each Service Application's bundle. However, if Multiple Service Providers are present, the management of permissions faces the risks below.

- The risk associated with the exporting packages to other Service Providers unexpectedly
- The risk associated with the giving services to other Service Providers unexpectedly
- The risk of information leak and loss from file accesses from another Service Provider's bundle.
- The risk of conflict in the operation of a single device, which is controlled via networks by Multiple Service Providers
- The risk of conflict in the utilization of system bundles that are called by Multiple Service Providers

Introducing "Separation" efficiently reduces these risks. Separation prevents improper interactions between Service Applications provided by different Service Providers, and ensures that each Service Application operates properly and safely by isolating the resources, such as bundles, files, sockets, and system property, that are utilized by the Service Applications. Separation should be implemented by Service Aggregator.

Separation has advantages for the three roles of remote management architecture: Service Provider, End User, and Service Aggregator.

➢ Service Provider

The Service Provider is assured that each Service Application will run properly on any OSGi equipment. Moreover, Service Application's bundles can be developed in a safe manner because the accessible resources have been clarified.

➢ End User

The End User does not need to worry about the Conflict Management for Multiple Service Applications to which he or she has subscribed, because they can delegates it to the Service Aggregator.

➢ Service Aggregator

The Service Aggregator simplifies the permission management to ensure the proper operation of the Service Applications because it does not need to check all permissions to avoid improper interactions between Service

Applications; only permissions that cross Service Applications need be checked. The permission management is a tangled operation, so simplification of management is important for Service Aggregator.

As one approach to implementing Separation, Permission Admin or Conditional Permission Admin can be utilized. The Service Aggregator can ensure Separation by configuring the permissions via Permission Admin or Conditional Permission Admin. Specifically, by using Conditional Permission Admin, the Service Aggregator can group the bundles when assigning permission, so that permission management becomes simpler.

However, problems remain only if Permission Admin or Conditional Permission Admin is used for permission management. The most important function in Separation is detecting permissions that induce improper interactions and determining the appropriate permissions for each Service Application; reducing the number of permissions managed is not the only key goal.

- Permission Admin and Conditional Permission Admin can't support the determination of permission by continuously ensuring the separation of resources among Service Applications. Therefore, to avoid setting permissions across other Service Applications, or to share resources, such as files, sockets, and system bundles with Service Aggregator's policy, Service Aggregator should check the tangled permissions for each Service Application.

- Permission Admin and Conditional Permission Admin have no interface for acquiring the current interaction states among Service Applications

- Permission Admin and Conditional Permission Admin can't detect the permissions that allow improper interactions among Service Applications

4 Requirements

4.1 Requirements for separation

- The solution MUST separate the resources for each Service Application to control the accesses to bundles provided by independent Service Providers.
- The solution MUST control the separation conditions by the Service Aggregator alone.
- The solution MUST control the separation conditions remotely.
- The solution MUST store the separation conditions persistently.
- The solution MUST provide a function to confirm the separation conditions on each OSGi Framework.
- The solution MUST provide a function to detect the violation of separation conditions.
- The solution MUST provide a function to resolve the conflict of common resources.

4.2 Requirements for the permission management

- The solution SHOULD utilize the (Conditional) Permission Admin to configure separation conditions.
- The solution SHOULD cover the existing permissions in the OSGi R4 and Java2 security model.
- The solution SHOULD prepare the mechanisms to accommodate the new types of permissions.

All Page Within This Box

4.3 Limitation for the solutions

- The system resources such as CPU load, memory use and file system use, are beyond the scope of the requirements.
- The security issues such as denial of service attacks, authentication between applications, and prevention of runaway programs, are beyond the scope of the requirements.

5 Technical Solution

5.1 Separation Management Model

The purpose of “Separation” is to protect Service Applications from other Service Applications on the same OSGi framework by restricting interactions between the Service Applications. This is intended to keep each Service Application running properly and safely.

These interactions between Service Applications occur when resources belonging to another Service Application are accessed. Such accesses can be restricted by setting the “permission”, which is the access control mechanism based on the Java2 security model. Therefore, Separation is achieved by managing the permissions that identify the controllable resources.

To realize the Separation, “segment” is logically created for each managed unit for Separation, such as Service Providers or Service Applications. Each segment contains bundles and resources given to the managed unit by Service Aggregator. The resources controlled by permissions are services, packages, sockets, files, and system properties (Fig. 2).

The bundles of a segment can be given permission to all resources in the same segment (solid line), and are not able to get permission for the resources belonging to other segments without authorization by the Service Aggregator (dashed line).

This means that the Service Applications on the same OSGi framework are assured of proper operation due to restriction of interactions between Service Applications by Service Aggregator and the assignment of resources included in the segment that are freely accessible from bundles contained in the same segment. Therefore, Service Providers can simplify the development of Service Applications.

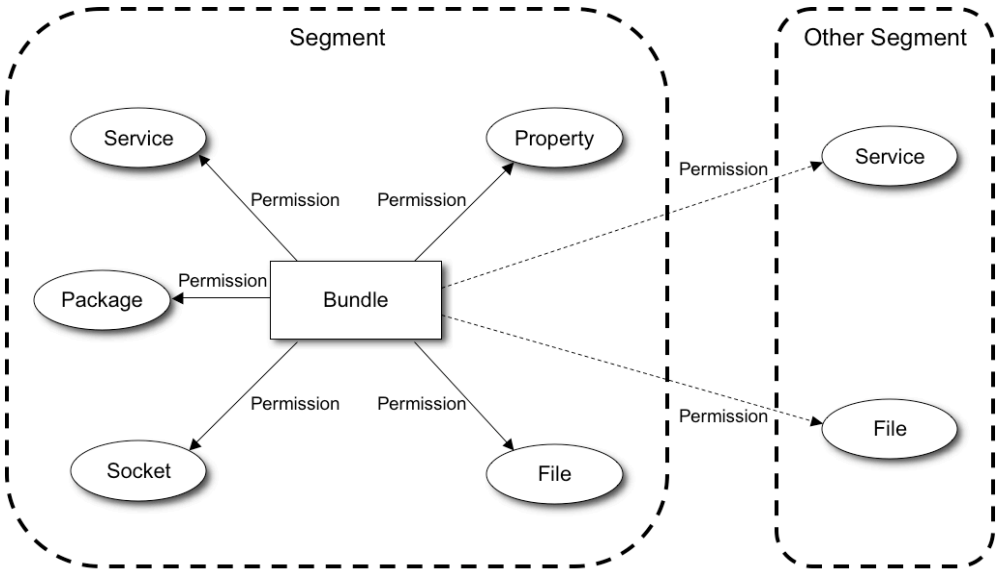

Figure 2: Segment

5.2 Resource Categories

The permission management for the separation is divided into two categories reflecting the types of resources: OSGi resources and Java resources. These resources have different features in term of permission management.

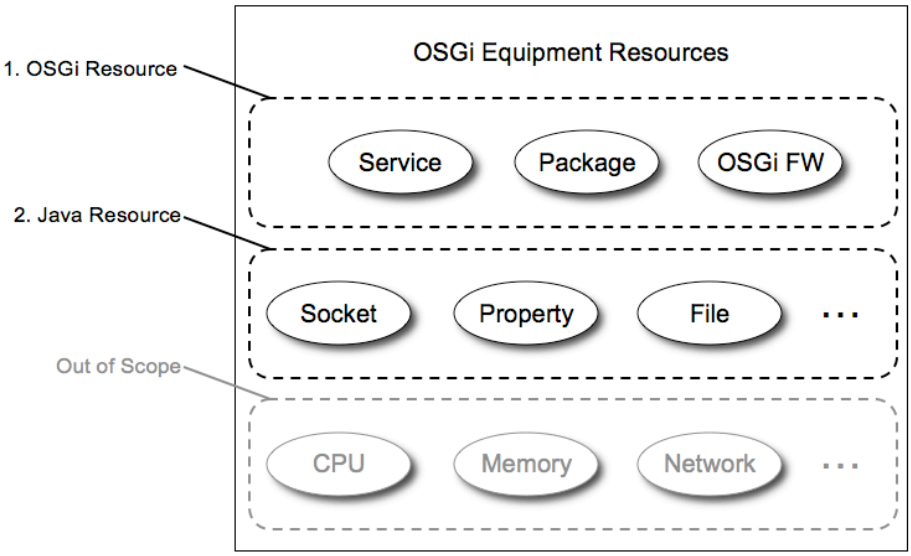

Figure 3: Category for OSGi equipment resources

All Page Within This Box

5.2.1 OSGi Resources

OSGi resources consist of services, packages and OSGi framework API, which are dealt with on an OSGi service platform. These resources can be controlled by OSGi defined permissions such as ServicePermission, PackagePermission and AdminPermission. Therefore, permission management for the resources is formalized in the same manner as for OSGi Equipment.

5.2.2 Java Resources

Java resources consist of such as sockets, files and properties, which are provided by operation system via Java virtual machine. These resources are not controlled by only permissions such as SocketPermission, FilePermission and PropertyPermission because there are many other processes running on the same equipment. For example, the bundle that gets SocketPermissions from OSGi framework cannot open the indicated server socket if another process is listening to the same port. Therefore, the permission management for Java resources should also provide functions to detect conflicts with other processes and to propose appropriate resources to bundles.

5.3 Permission Operation

In this specification, we focus on the OSGi resources described above. The permission management for OSGi Resources is conducted using the following rules.

5.3.1 Service

The goal for the separation of service objects is for an administrator of OSGi framework to be able to restrict or permit access to services from bundles belonging to other segments. Moreover, the administrator should be able to control access to individual service instances that are registered under the same name in order to share services between segments.

OSGi service is separated from other bundles by using ServicePermission to control access to it. ServicePermission has two action attributes to share service objects: “register” and “get”. Therefore, an administrator of OSGi equipment gives ServicePermission with the register attribute to a bundle that is allowed to register a certain service on the OSGi framework. Furthermore, the administrator gives ServicePermission with the get attribute to a bundle if the bundle is allowed to get the service object from the OSGi framework.

The above administrator actions represent the basic operations for service object sharing, but operations are more complicated on the multiple service providers model due to following problems.

5.3.1.1 Problem 1: restricting accesses to services registered under the same name

First, we should consider the relationship between segments and service names under which the service objects are registered. If an administrator allows registering services under the same name for two bundles belonging to different segments, it cannot separate accesses to the service when it gives ServicePermission with get attributes to bundles. Because the two segments have services registered under the same name, this operation causes a problem that the bundle given the ServicePermission is able to access both segments’ services (Fig. 4). In the Fig. 4, even if the Service Aggregator doesn’t want to allow the right arrow, both service instances can be accessed.

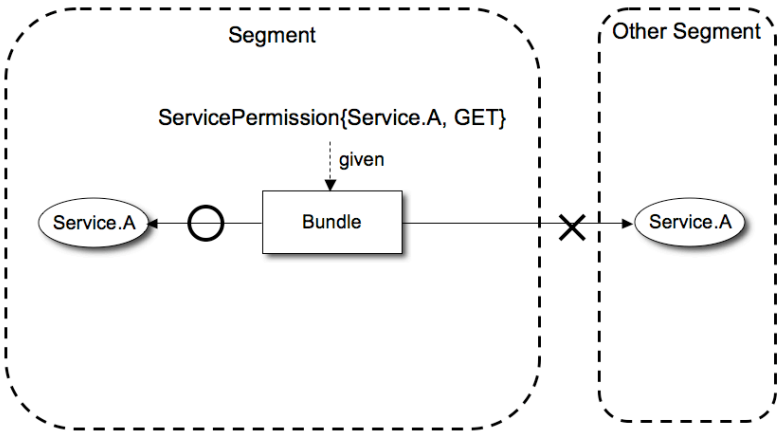

Figure 4: Problem 1 of service separation

5.3.1.2 Problem 2: distinguishing services as instance

Second, we should consider the limitation of ServicePermission. ServicePermission identifies service objects as a class name, not instance. Therefore, if an administrator wants to distinguish the accesses to each service registered under the same class name, it is impossible with only existing OSGi permission mechanisms. Consider, for example, that UPnP Device Service registers several services under the same class name, “org.osgi.service.upnp”, and an administrator wants to allow a bundle to get one of these services. If the administrator gives ServicePermission to the bundle, it can access all services registered under the name of org.osgi.service.upnp (Fig. 5). The administrator cannot control this situation.

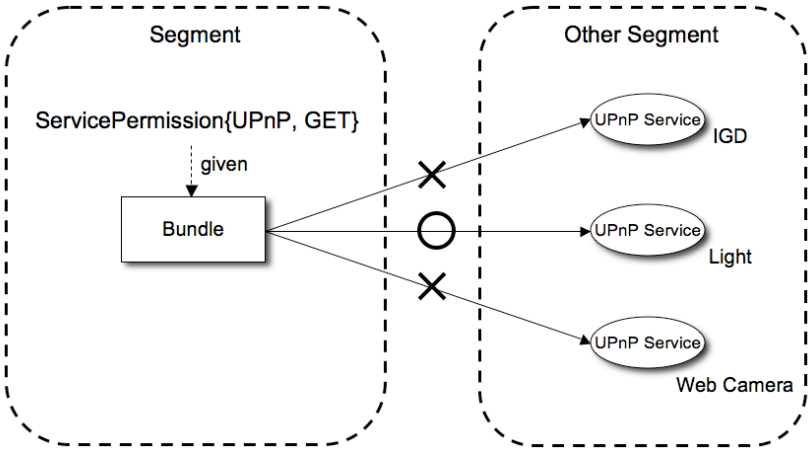

Figure 5: Problem 2 of service separation

5.3.1.3 Possible solutions

To avoid these situations, we have 3 alternatives: (a) registering services under different class names, (b) authenticating caller bundle in service implementation, and (c) extending ServicePermission for distinguishing service instances.

Approach (a) is a way to achieve the goal that can avoid any changes to the current OSGi specification. However, we have to define many classes for the same implementation to solve the problems listed above, and it is difficult

All Page Within This Box

for Service Providers to implement services. Therefore, the approach (a) is impossible to use in the multiple service provider model.

Approach (b) is an effective way to distinguish segments if each caller bundle can be authenticated by digital signatures etc. at the beginning of called method to solve problem 1. However, it is difficult to provide a solution for problem 2 because each service object has to know which bundles can be authenticated as of implementing bundles. Therefore, it is difficult for the administrator of the OSGi framework to control access to service objects because this mechanism depends on the implementation of the service objects.

Approach (c) provides a sophisticated solution to both of the above problems, while it needs extending ServicePermission of the OSGi specification. If an administrator is able to distinguish each service object to set permission, it is able to avoid accesses over other segments by giving the permission to just those bundles in the same segment to solve problem 1 even if the service is registered under the same class name. Moreover, the administrator is able to give bundles the permission for particular service instance to solve problem 2.

According to the above analysis, the approach (c) is the only way to achieve the goal that realizes the separation and solves the problems of the current SerivcePermission. Therefore, we should adopt approach (c) and extend current “ServicePermission”. See *ServicePermission* extension specification on section 5.4.

5.3.1.4 *Separation using ServicePermission* extension

By using ServicePermission extension to identify each service instance, an administrator follows the rules below to realize the separation.

- The administrator gives Extended ServicePermission to bundles to access services in the same segment. Therefore, bundles are freely accessible to the services in the same segment, and the services in the segment are protected from bundles in other segments even if registered under the same class name.

- If the administrator allows a bundle to use a service belonging to other segment, it gives Extended ServicePermission for getting the target service instance to the bundle. Therefore, other services registered under the same class name are protected from improper access by the bundle.

5.3.2 Package

The goal for the separation of packages is enabling an administrator of OSGi framework to control package sharing of packages between bundles in other segments.

Package sharing between bundles is realized by Export-Package Header, Import-Package Header, and Require-Bundle Header. Basically, using Require-Bundle Header between segments is not suitable in multiple service provider model because it enforces strong dependency between segments at development time. Therefore, the goal is assumed as controlling package sharing between bundles using Export-Package Header and Import-Package Header during runtime.

5.3.2.1 Problem 1: restricting sharing of packages with the same qualified name

First, we should consider the relationship between segments and packages. If an administrator allows bundles belonging in different segments to export packages with the same qualified name, it cannot separate package imports when it gives PackagePermission with IMPORT action to bundles. Because two segments have packages exported with the same qualified name, this operation causes a problem that the bundle given the PackagePermission is able to access both segments’ packages (Fig. 6). In the Fig. 6, even if the Service Aggregator doesn’t want to allow the right arrow, both packages can be accessed.

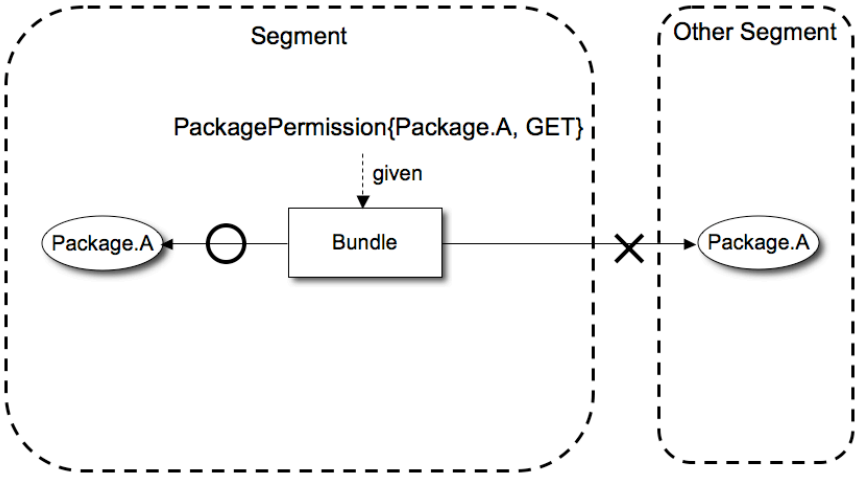

Figure 6: Problem 1 of package separation

5.3.2.2 Possible solutions

There are 3 approaches to separating packages; (a) exporting packages with different qualified names, (b) setting bundle-symbolic-name attribute in its Import-Package Header and controlling package dependencies by PackagePermission and, (c) extending PackagePermission for distinguishing packages based on not only qualified package name but also bundle identifier of exporter bundle..

Approach (a) requires no changes to the current OSGi specification. However, many qualified package names for the same implementation will be needed, and it is difficult to change package names in many cases due to some reasons. The reasons include: the package is standardized one such as org.osgi.util.tracker and XML parser, and the license doesn't allow change of the package name.

Regarding approach (b), it has as similar demerit as using Require-Bundle Header between segments. It makes bundles between segments tightly coupled in development time. The bundles between segments should be loosely coupled.

Approach (c) provides a sophisticated solution, while it needs to extend the OSGi specification with the new function of PackagePermission. While permission check for exporting bundle is as same as the current specification, permission check for import based on not only package name but also the identification of the bundle exporting the package can be realized. This enables an administrator to control sharing over other segments by giving the permission during runtime; the administrator can give the permission to bundles for importing a particular package exported from a particular bundle precisely.

According to the above analysis, the approach (c) is the only way to achieve the goal that realizes the separation regarding Package. Therefore, we should choose approach (c) and extend "PackagePermission". See *PackagePermission* specification on section 5.5.

5.3.2.3 Separation using PackagePermission extension

By giving proper extended PackagePermission regarding package's name and bundle identifier of exporter bundle, an administrator can realize the separation as follows:.

- The administrator gives extended PackagePermission to bundles that import packaged exported by bundles in the same segment. Therefore bundles can freely import the packages from the same segment and the

All Page Within This Box

packages exported are protected from bundles in other segments even if exported packages have the same qualified package name.

- In order to import packages exported by bundles in other segments, administrator should give extended PackagePermission with a proper filter regarding qualified package name and exporter bundle identifier to importer bundles..

5.3.3 OSGi Framework

The goal for the separation of OSGi framework's operation is that an administrator of OSGi framework is able to restrict operations for bundles through OSGi framework from the other segments.

Some operations by OSGi framework API are restricted to check AdminPermission. AdminPermission contains several ways of identifying a bundle and many actions to restrict a specified operation. Bundles in a segment should be able to control bundles and the resources included in the same segment.

5.3.3.1 Separation using AdminPermission

To freely use resources from other bundles belonging to the same segment, an administrator follows the rules below in setting AdminPermissions.

- The administrator gives AdminPermissions to the bundles with bundle identifiers indicating the bundles in the same segment and actions except "extensionLifecycle" for using resources, obtaining information and controlling bundle lifecycle.

- If the administrator allows an operation to a bundle for using resources, obtaining information and controlling bundle lifecycle over a segment, it gives AdminPermissions with the particular operation and bundle identifier.

5.4 Service Permission extension

In OSGi, ServicePermission can restrict an access by only class name, not instance so far. Considering residential use case, it is common that many devices are represented as OSGi services registered into OSGi framework under the same class name, for example UPnPDevice service. To ensure the separation, we need a function that can restrict access based on not only service names but also instances on service properties and which bundle provides.. We introduce ServicePermission extension to realize the function.

ServicePermission has two actions, register and get. Regarding register action, ServicePermission will not be extended. On the other hand, ServicePermission with get action will be extended as followings.

5.4.1 Name for "get" action

The name should be either a LDAP search filter (RFC 1960. See Filter for a description of the filter string syntax.) or a name of objectClass.

A filter string consists of service properties contained in ServiceReference object and/or bundle identifiers including bundle ID, bundle location, signer and bundle symbolic name. When a bundle is going to get a ServiceReference or a Service, and a ServiceEvent is going to be dispatched to a bundle, an OSGi framework checks the name parameter against the service properties of the ServiceReference and bundle identifiers of the bundle registering the service. The properties in the filter string are information as key/value pair. The bundle identifiers are the same ones of Admin Permission. The following bundle identifiers can be used to specify bundles.

All Page Within This Box

- id - The bundle ID of a registering bundle.
- location - The location of a registering bundle.
- signer - A Distinguished Name Chain of a registering bundle.
- name - The symbolic name of a registering bundle.

In order to filter service properties keyed by “id”, ”location”, ”signer”, “name”, the filter syntax can use “@” character as prefix. If a key in the filter String has “@” prefix, the key must be considered as a key for service properties keyed by the key without “@”. For instance, “@name” means neither “name” as bundle identifiers nor a service property keyed by “@name” . It means a service property keyed by “name”. Similarly, “@@name” means a service property keyed by “@name”.

Any key on bundle identifiers and service properties in the filter string is optional.If a name of ServicePermission with get action is not in a format of LDAP filter, it it is dealt as service name as current ServicePermission does. It keeps backward compatibility with the previous ServicePermission specification.

5.4.2 Usage

Filter Examples;

Example s1; A bundle granted the ServicePermission(“(&(objectClass=org.osgi.service.*)(service.pid=sample-service))”, “GET”) can get a service starting with “org.osgi.service.” with a service property “(key, value) = (service.pid, sample-service)”. The bundle can get neither a service starting with “org.osgi.service.” with a service property “(key, value) = (service.pid, other-sample-service)” nor one without service property keyed by “service.pid”.

Example s2; A bundle granted the ServicePermission(“(&(objectClass=org.osgi.service.*)(!(service.pid=sample-service)))”, “GET”) cannot get service starting with “org.osgi.service.” with a service property “(key, value) = (service.pid, sample-service)”. The bundle can get both a service starting with “org.osgi.service.” with a service property “(key, value) = (service.pid, other-sample-service)” and one without service property keyed by “service.pid”.

Example s3; A bundle granted the ServicePermission(“(objectClass=org.osgi.service.*)”, GET”) can get service starting with “org.osgi.service.” with any service properties. The permission is equivalent to ServicePermisson(“org.osgi.service.*”,”GET”).Example s4; A bundle granded the ServicePermission(“(&(objectClass=org.sample.*)(name=com.amce.*)”, “GET”) can get services starting with “org.sample.” registered by bundles whose symbolic name start with “com.amce.” but can neither get services starting with “org.sample.” registered by bundles whose symbolic name starts with “com.other.” nor the ones registered by bundles who have no symbolic name.

Example s5; A bundle granded the ServicePermission(“(&(objectClass=org.osgi.service.upnp.UpnpDeviceService)(signer=*, o=ACME, c=US) (room=kitchen))”, “GET”) can get UpnpDeviceService with a service property “(key, value) = (room, kitchen)” a registered by bundles signed by the specified signers. The bundle can get neither UpnpDeviceServices without the service property registered by bundles signed by the signer nor the UpnpDeviceServices with the service property registered by bundles not signed by the signer. .

Example s6; Even if an administrator grant ServicePermission (“(objectClass=service.A)”, “REGISTER”) to a bundle by (Conditional)PermissionAdmin, the bundle cannot register the service. The reason is, the ServicePermission is ignored by framework because it cannot be constructed. ServicePermission with “register” action must have not filter String as its name.

All Page Within This Box

5.5 Package Permission extension

In OSGi, PackagePermission can restrict the sharing of a package only by qualified package name so far. In other words, an administrator cannot control the wiring between exporters and importers among the same package name. To ensure the separation, we need a function that can restrict access based on not only package names but also bundle identifiers of exporter bundle. We introduce PackagePermission extension to realize the function.

PackagePermission before R4.1 has had two actions, ”export and “import,” where “export” implies “import” . In the new spec, it will have new action “exportonly”, which doesn’t implies “import”. In addition, “export” action will be deprecated.

Regarding export only (and export) action, PackagePermission will not be able to have filter String as its name. On the other hand, PackagePermission with import action will be extended as followings.

5.5.1 Name for “import” action

The name should be either a LDAP search filter (RFC 1960. See Filter for a description of the filter string syntax.) or a package name. A filter string consists of package name and/or bundle identifier including bundle ID, bundle location, signer and bundle symbolic name. When a bundle is going to be resolved, an OSGi framework checks the name parameter against package name to be imported and bundle identifiers of the bundle exporting the package.

The properties in the filter string are information as key/value pair. The following package attributes can be used in the filter:

- package.name – The qualified name of the package to be imported.

The bundle identifiers are the same ones of Admin Permission. The following bundle identifiers can be used to specify bundles.

- id - The bundle ID of a exporting bundle.
- location - The location of a exporting bundle.
- signer - A Distinguished Name Chain of a exporting bundle.
- name - The symbolic name of a exporting bundle.

If a name of PackagePermission with import action is not in a format of LDAP filter, it it is dealt as package name as current PackagePermission does. It keeps backward compatibility with the previous PackagePermission specification.

5.5.2 Usage

Filter Examples;

Example p1; A bundle granted the PackagePermission(“(&(package.name=org.osgi.service.*)(location=http://com.amce.*))”, “IMPORT”) can import package starting with “org.osgi.service.” whose exporter bundle has bundle location starting with “http://com.amce.”.

All Page Within This Box

Example p2; A bundle granted the PackagePermission("(package.name=org.osgi.service.http) ", "IMPORT") can import package "org.osgi.service.http" , no matter what kinds of bundle exported the package. It is equivalent to PackagePermission("org.osgi.service.http","IMPORT").

Example p3; A bundle granted PackagePermission("(&(package.name=org.sample.*)(name=com.amce.*)", "IMPORT") can import package starting with "org.sample." exported by bundles whose symbolic name starts with "com.amce.". But it can import neither a package starting with "org.sample." but exported by bundles whose symbolic name starts with "com.other." nor a package starting with "org.sample." but exported by bundles who have no symbolic name .

Example p4; A bundle granted PackagePermission("(signer=*,o=ACME,c=US)", "IMPORT") can import any packages exported by bundles signed by the specified signer.

Example p5; A bundle granted PackagePermission("org.osgi.service.http", EXPORT") can export the package and import any "org.osgi.service.http" packages exported by any bundles.

Example p6; A bundle granted PackagePermission("org.osgi.service.http", EXPORTONLY") can only export the package but not import the package.Example s7; Even if an administrator grant PackagePermission ("(package.name=package.a)", "EXPORTONLY") to a bundle by (Conditional)PermissionAdmin, the bundle cannot export the package. The reason is, the PackagePermission is ignored by framework because it cannot be constructed. PackagePermission with "exportonly" or "export" action must have not filter String as its name.

5.6 Javadoc

ServicePermission class and PackagePermission class need to be modified to support the separation on OSGi framework. The javadoc should be considered a binding part of this specification. Note that only API that has been modified or added is included in this document.

5.7 org.osgi.framework

Class ServicePermission

java.lang.Object
  └ java.security.Permission
      └ java.security.BasicPermission
          └ **org.osgi.framework.ServicePermission**

All Implemented Interfaces:

java.io.Serializable, java.security.Guard

---

public final class **ServicePermission**
extends java.security.BasicPermission

A bundle's authority to register or get a service.

- The register action allows a bundle to register a service on the specified names.
- The get action allows a bundle to detect a service and get it.

All Page Within This Box

Permission to get a service is required in order to detect events regarding the service. Untrusted bundles should not be able to detect the presence of certain services unless they have the appropriate ServicePermission to get the specific service.

See Also:

Serialized Form

Constructor Summary

ServicePermission(ServiceReference reference, java.lang.String actions) Creates a new requested ServicePermission object to be used by code that must perform checkPermission for the get action.
ServicePermission(java.lang.String filter, java.lang.String actions) Create a new ServicePermission.

Constructor Detail

5.7.1 ServicePermission

public **ServicePermission**(java.lang.String name,
java.lang.String actions)

Create a new ServicePermission.

The name of the service is specified as a fully qualified class name. Wildcards may be used.

```
name ::= <class name> | <class name ending in ".*"> | *
```

Examples:

```
org.osgi.service.http.HttpService
org.osgi.service.http.*
*
```

For the get action, the name can also be a filter expression. The filter gives access to the service properties as well as the following attributes:

- signer - A Distinguished Name chain used to sign the bundle publishing the service. Wildcards in a DN are not matched according to the filter string rules, but according to the rules defined for a DN chain.
- location - The location of the bundle publishing the service.
- id - The bundle ID of the bundle publishing the service.
- name - The symbolic name of the bundle publishing the service.

Since the above attribute names may conflict with service property names used by a service, you can prefix an attribute name with '@' in the filter expression to match against the service property and not one

All Page Within This Box

of the above attributes. Filter attribute names are processed in a case sensitive manner unless the attribute references a service property. Service properties names are case insensitive.

There are two possible actions: `get` and `register`. The `get` permission allows the owner of this permission to obtain a service with this name. The `register` permission allows the bundle to register a service under that name.

Parameters:
`name` - The service class name
`actions` - `get,register` (canonical order)

5.7.2 ServicePermission

`public ServicePermission(ServiceReference reference, java.lang.String actions)`

Create a new `ServicePermission` object to be used by the code that must check a `Permission` object when it obtains `ServiceReference`, gets service object from service registry or dispatches service events.

Parameters:
`reference` - `ServiceReference` object of the service
`actions` - `get,register` (canonical order)

Since:
1.5

5.8 org.osgi.framework

Class PackagePermission

`java.lang.Object`
  `└ java.security.Permission`
    `└ java.security.BasicPermission`
      `└ org.osgi.framework.PackagePermission`

All Implemented Interfaces:
java.io.Serializable, java.security.Guard

`public final class PackagePermission extends java.security.BasicPermission`

A bundle's authority to import or export a package.

A package is a dot-separated string that defines a fully qualified Java package.

For example:

`org.osgi.service.http`

All Page Within This Box

PackagePermission has three actions: exportonly, import and export. The export action, which is deprecated, implies the import action.

See Also:

Serialized Form

Constructor Summary

PackagePermission(java.lang.String name, java.lang.String actions)	Creates a new PackagePermission object.
PackagePermission(java.lang.String name, Bundle exportingBundle, java.lang.String actions)	Creates a new requested PackagePermission object to be used by code that must perform checkPermission for the import action.

Constructor Detail

5.8.1 PackagePermission

public **PackagePermission**(java.lang.String name, java.lang.String actions)

Creates a new PackagePermission object.

The name is specified as a normal Java package name: a dot-separated string. Wildcards may be used.

```
name ::= <package name> | <package name ending in ".*"> | *
```

Examples:

```
org.osgi.service.http
javax.servlet.*
*
```

For the import action, the name can also be a filter expression. The filter gives access to the following attributes:

- signer - A Distinguished Name chain used to sign the exporting bundle. Wildcards in a DN are not matched according to the filter string rules, but according to the rules defined for a DN chain.
- location - The location of the exporting bundle.
- id - The bundle ID of the exporting bundle.
- name - The symbolic name of the exporting bundle.
- package.name - The name of the requested package.

All Page Within This Box

Filter attribute names are processed in a case sensitive manner.

Package Permissions are granted over all possible versions of a package. A bundle that needs to export a package must have the appropriate PackagePermission for that package; similarly, a bundle that needs to import a package must have the appropriate PackagePermssion for that package.

Permission is granted for both classes and resources.

Parameters:
name - Package name or filter expression. A filter expression can only be specified if the specified action is import.
actions - exportonly,import (canonical order).

5.8.2 PackagePermission

public **PackagePermission**(java.lang.String name,
Bundle exportingBundle,
java.lang.String actions)

Creates a new requested PackagePermission object to be used by code that must perform checkPermission for the import action. PackagePermission objects created with this constructor cannot be added to a PackagePermission permission collection.

Parameters:
name - The name of the requested package to import.
exportingBundle - The bundle exporting the requested package.
actions - The action import.

Since:
1.5

6 Considered Alternatives

6.1 "RFC138 Multiple Frameworks in One JVM" and "RFC136 Scope and OSGi"

Although technical solutions on both RFC138[4] and RFC136[5] are not clear at the writing of this RFC, for our use cases, it has a drawback and it cannot realize our requirements.

- Multiple frameworks approach has similar objective in terms of "grouping bundles". It might be able to solve the Problem 1 described in 5.3.1.2 in some use cases. However, it's hard to adopt it in embedded world because the approach needs more memory than what this RFC proposes where memory is limited in embedded world.

All Page Within This Box

- Neither approach solves the Problem 2 described in 5.3.1.2.

6.2 “RFP102 Modifiable Metadata”

Although technical solutions on RFP102[6] is not clear at the writing of this RFC, rewriting bundle-symbolic-name attribute of Import-Package Header would be able to realize what this RFC regarding PackagePermission extension realize: control of package wiring between bundles during runtime by the management agent decision.

In Multiple Service Provider model, there is a management agent who manages OSGi resources on an OSGi framework. If package wiring is realized by the solution only based on RFP102, the management agent needs not only to manage package wiring by the solution but also to manage permissions except regarding package wiring. In case of the proposal in this RFC, all OSGi resources are managed through Permissions in unified way. Therefore, the proposal of this RFC is useful even if solutions of RFP102 become a part of new spec.

7 Security Considerations

After getting a service, no security check for the caller bundle will be done as core spec 1.4.

When a bundle is going to get a service reference or a service, the extended ServicePermission will be checked for the getter bundle as described above. Then, even if the service properties of the service is modified after succeeding in getting a service and the modified service properties doesn’t match the extended ServicePermission granted to the getter bundle, the getter bundle CAN continue to use the service. This is not the drawback newly introduced by the proposed solution but the limitation of the service model of OSGi.

8 Document Support

8.1 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]. Koya, M. et al, RFP 95 Multiple Service Provider Separation.

[4]. Erin, S. et al, RFC 138 Multiple Frameworks In One JVM.

[5]. Subbarao, M. et al, RFC 136 Scopes and OSGi.

[6]. Peter, K. et al, RFP 102 Modifiable Bundle Metadata.

8.2 Author's Address

Name	Koya Mori
Company	NTT Corporation
Address	Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan
Voice	+81-46-859-3446
e-mail	mori.kouya@lab.ntt.co.jp

Name	Hiroyuki Maeomichi
Company	NTT Corporation
Address	Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan
Voice	+81-46-859-3013
e-mail	maeomichi.hiroyuki@lab.ntt.co.jp

Name	Ikuo Yamasaki
Company	NTT Corporation
Address	Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan
Voice	+81-46-859-8537
e-mail	yamasaki.ikuo@lab.ntt.co.jp

8.3 Acronyms and Abbreviations

8.4 End of Document

All Page Within This Box