RFC 94 MEG Deployment Configuration

Confidential, Draft

21 Pages

Abstract

This document describes how initial configuration can be done when a deployment package is deployed on a MEG-compliant platform. It does NOT discuss post-deployment configuration via a “Remote Manager” – that is covered in the MEG Device Management work stream.

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	August 3, 2004	Initial version by Joe Rusnak, IBM (jgrusnak@us.ibm.com)
0.1	August 9, 2004	Revised the proposed XML schema definition to make it less verbose. Other minor editorial changes. Joe Rusnak, IBM
0.2	August 16, 2004	Accepted previous changes. Fixed indentation for the XML schema definition and the example AUTOCONF.XML files. Rewrote Section 1 (Introduction) to be more readable (hopefully!). Other minor editorial changes. Joe Rusnak, IBM
0.3	September 7, 2004	Added a proposal for how deployment-time configuration security should be handled (based on suggestion from Gabor Paller). Added a discussion of backing out deployment-time configuration changes when a deployment package is undeployed. Added an ”optional” element to the <TS> tag in AUTOCONF.XML to specify whether or not the inability to configure the specified <TS> should cause deployment to fail or not. Joe Rusnak, IBM
0.4	November 19, 2004	Major revision of the document. Changed the XML schema so that a single common XML schema definition can be shared between RFC 62 and RFC 94. Also made changes to make RFC 94 consistent with the new RFC 88… Joe Rusnak, IBM

All Page Within This Box

Revision	Date	Comments
0.5	December 17, 2004	Minor editorial changes. Removed unused definitions and acronyms. Updated the "Security" section based on discussions at the Boca meeting, and consistent with RFC 88.
0.6	January 10, 2005	Minor editorial changes (e.g., changed "resource package" to "deployment package"). Also changed the example to make it consistent (MS-A is now a ManagedServivceFactory).
0.7	February 3, 2005	Changed XML schema definition to be much simpler, and consistent with RFC 62. Added an informative section describing the relationship with RFC 62. The basic principles are unchanged.
0.8	February 14, 2005	Updated the XML schema definition based on error detected in RFC 62. Per Nokia's request, added a "merge" attribute to <Designate>
0.9	March 23, 2005	Minor updates due to feedback from the RFC 62 and RFC 94 RIs. Added a "Value" tag to help define arrays and vectors, and clarified how to specify a zero-length array or an empty vector. Changed the type of "cardinality" and "size" to "integer" (from "int").

1 Introduction

When a deployment package is deployed in the MEG environment [4], there is a requirement to provide configuration parameters to bundles automatically, as part of the deployment operation. The bundles to receive configuration parameters may be bundles in the deployment package being deployed, or they may be other bundles (outside the deployment package) on which the deployment package depends.

This document describes a technique that allows the MEG Deployment Administrator (MDA) to perform this deployment-time configuration based on an XML configuration document that is contained in the deployment package being deployed. This declarative approach to configuration provides a simpler alternative to programmatic configuration via the Configuration Admin service. This is conceptually similar to what is being done in Declarative Services [3][1].

[1] RFC 80 Declarative Services simplifies the programming model for using services in OSGi so that new programmers can declare their service relationships and then have them managed by the runtime rather than having to write code to call OSGi APIs.

All Page Within This Box

2 Application Domain

2.1 Description

Mobile devices are rapidly becoming more complex as their capabilities increase. One such capability is the ability to install new software components after the time of manufacture. For example, device users already are, or soon will be, accustomed to installing applications on their devices from remote servers wirelessly or via local connectivity. Operators and other providers also wish to perform software repair and upgrades remotely to subscriber devices. To enable these deployment capabilities, a powerful management framework is required. OSGi provides such a framework, and this document will address the use-cases and requirements for extending OSGi for software component deployment for mobile devices

A software component lifecycle can be divided into several phases. First, a Developer must create and package a software component. The Developer then delivers the packaged component to a means of distribution, called a component Store. On the Store, the packaged component may be modified further. For example, the Store Owner may put multiple components into a single package and tailor them for a particular configuration. Once the packaged component is available on the Store, an Administrator or User initiates a request to a management server to deliver the component to the device. The management server has access to the Store and distributes a packaged component to the mobile device, which subsequently installs it. **During installation, the component package provides all the necessary environmental setup required, such as initial configuration and data store setup.** The device may also need to resolve issues such as dependency of component on available services and libraries. Once successfully installed, the component's runtime is managed by a management framework. Eventually, the component reaches end-of-life, perhaps due to obsolescence or undesirability by the user, and is uninstalled from the device by the action of an Administrator or User. **When uninstalled, the component is completely and automatically removed from the device.**

This document focuses on the initial configuration aspect of deployment. Other RFCs (e.g., RFC 88[4]) address the other aspects of deployment.

2.2 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].

Configuration parameters – Parameters specific to software components that allow configuration of certain software component functionalities.

Data store – A collective term for all data storage areas that may be used by a software component, such as configuration data, user preferences, and other component-specific data files.

Deployment package – A named and versioned entity that groups resources into one management unit. Deployment packages are the unit of deployment and uninstallation defined in RFC 88 [4]. Deployment packages can contain bundles and associated deployment-time resources (for example, RFC 94 configuration files).

Resource Processor -- An entity defined in RFC 88 [4] that allows deployment-time customization or configuration. A resource processor is a service that implements the `ResourceProcessor` interface and that registers its interest in some resource associated with an RP with the MEG Deployment Admin. A configuration

All Page Within This Box

resource processor would register its interest in RFC 94 AUTOCONF.XML files, and would provide the processing described in this document. See RFC 88 for more information about resource processors.

Script – Program code written in a scripting language that satisfies the install, uninstall and upgrade script requirements on functionality and security. The script is normally part of the application package.

Service – A software component that can be accessed by services and other units of execution and has its own execution state.

Software component – A unit of software functionality (including code and static data resources). Services, libraries, and units of execution are collectively called software components.

Store – A database that stores packaged components, ready for delivery to the device.

URL – Uniform Resource Locator

User – The end user of the device. Depending on the device policy, an end user may also have access to management functionality over a local API.

2.3 Acronyms and Abbreviations

API – Application Programming Interface

DMT – Device Management Tree (see OSGi RFC 87 [8])

MEG – Mobile Expert Group

MDA – MEG Deployment Administrator (see RFC 88 [4])

OSGi – Open Services Gateway Interface

3 Problem Description

Deployment covers a wide range of topics, including:

- Packaging of software components in a format suitable for deployment to a device.
- Initiating deployment-related management operations from the server or from the device
- Interaction between the management server and the management system on the device within the framework of existing standards

All Page Within This Box

- Interaction between the local management entities and the management system on the device
- Operation of the management system during the deployment-related management operations.

The relevant actions are installation, updating, uninstallation, and reinstallation. Interaction with the policy system is also an important concern.

Deployment scenarios can be triggered from many sources such as a User or Administrator and can happen over many types of transports such as a radio network or short-range connection.

This document focuses on the configuration of bundles when a deployment package is deployed (initial installation or refresh). When deployment packages are deployed in the MEG environment, there is a requirement to provide configuration parameters to bundles in the deployment package (and perhaps other bundles installed on the framework) automatically as part of the deployment operation. This document describes a technique that allows the MEG Deployment Administrator (MDA) to perform this deployment-time configuration based on an XML configuration document that is part of the deployment package being deployed. This provides a declarative approach to configuring bundles that is simpler that the programmatic approach of dealing directly with Configuration Admin [1].

4 Requirements

These requirements are a subset of the deployment requirements in RFP 53 [9].

REQ-DEP-04-01. It MUST be possible to customize the installation process by scripts, add-ons, etc

REQ-DEP-04-05. It SHOULD be possible to associate component-specific data to installed components. This data is loaded into the appropriate data store during installation process. *For example, it is possible to declare that 3 database tables need to be created during installation and they are initialized from initial data that the package contains.*

REQ-DEP-05-01. It MUST be possible to package all data (code, settings, resources, and other component-specific data) into one file that can be expanded into individual files on the device.

Additionally, there is a requirement to share the XML schema definition with RFC 62. The relationship between RFC 62 and RFC 94 is described in Section 5.3.

All Page Within This Box

5 Technical Solution

Suppose bundles B and C are members of a deployment package. Suppose A is another bundle (not a member of the deployment package!) upon which C depends. Further suppose that for C to operate properly, C requires A and B to have certain configuration properties set correctly when the deployment package is deployed (so that everything is set up before bundle C is started).

In order for bundles A and B to be "pre-configured" on behalf of bundle C by the MDA, they must use the OSGi Configuration Admin Service [5] In other words, A and B must register a `ManagedService` or a `ManagedServiceFactory` with the OSGi Service Registry. As described in RFC 88, there are some rules in this area that must be followed to ensure proper cleanup when a deployment packaged in uninstalled (or when installation fails):

- Since B is in the deployment package, B can register a `ManagedService` or a `ManagedServiceFactory` (or both).
- Since A is NOT in the deployment package, A can only be configured via a `ManagedServiceFactory`.

It is not necessary for A or B to be started before they are configured.

As described in RFC 88 [4], when the MDA is deploying the bundle suite the MDA looks for an AUTOCONF.XML file in the deployment package. This AUTOCONF.XML file contains configuration information that allows a Configuration Resource Processor registered with the MDA to "pre-configure" A and B via the OSGi Configuration Admin service.

The format of the AUTOCONF.XML file is based on an XML schema definition that is shared with OSGi RFC 62, "Meta Type Repository" [6].

5.1 AUTOCONF.XML

Here is the XML schema definition for AUTOCONF.XML configuration documents. RFC 62 and RFC 94 share a common schema definition. Some of the elements and attributes are used only for RFC 62. The parts of the schema definition used by RFC 94 are shown in bold text. Other elements and attributes in the schema should be ignored by an RFC 94 implementation.

```
<?xml version="1.0" encoding="UTF-8" ?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.osgi.org/xmlns/metatype/v1.0.0"
    xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0">

    <complexType name="MetaData">
        <sequence>
            <element name="OCD" type="metatype:OCD" minOccurs="0" maxOccurs="unbounded" />
            <element name="Designate" type="metatype:Designate" minOccurs="1" maxOccurs="unbounded" />
        </sequence>
        <attribute name="localization" type="string" use="optional" />
```

All Page Within This Box

```
</complexType>

<complexType name="OCD">
    <sequence>
        <element name="AD" type="metatype:AD" minOccurs="1" maxOccurs="unbounded" />
        <element name="Icon" type="metatype:Icon" minOccurs="0" maxOccurs="1" />
    </sequence>
    <attribute name="name" type="string" use="required" />
    <attribute name="description" type="string" use="optional" />
    <attribute name="id" type="string" use="required" />
</complexType>

<complexType name="AD">
    <sequence>
        <element name="Option" type="metatype:Option" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
    <attribute name="name" type="string" use="optional" />
    <attribute name="description" type="string" use="optional" />
    <attribute name="id" type="string" use="required" />
    <attribute name="type" type="metatype:Scalar" use="required" />
    <attribute name="cardinality" type="integer" use="optional" default="0" />
    <attribute name="min" type="string" use="optional" />
    <attribute name="max" type="string" use="optional" />
    <attribute name="default" type="string" use="optional" />
    <attribute name="required" type="boolean" use="optional" default="true" />
</complexType>

<complexType name="Object">
    <sequence>
        <element name="Attribute" type="metatype:Attribute" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
    <attribute name="ocdref" type="string" use="required" />
</complexType>

<complexType name="Attribute">
    <sequence>
        <element name="Value" type="string" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="adref" type="string" use="required" />
    <attribute name="content" type="string" use="optional" />
</complexType>

<complexType name="Designate">
    <sequence>
        <element name="Object" type="metatype:Object" minOccurs="1" maxOccurs="1" />
    </sequence>
    <attribute name="pid" type="string" use="required" />
    <attribute name="factory" type="boolean" use="optional" default="false" />
    <attribute name="bundle" type="string" use="optional" />
    <attribute name="optional" type="boolean" default="false" use="optional" />
    <attribute name="merge" type="boolean" default="false" use="optional" />
</complexType>

<simpleType name="Scalar">
    <restriction base="string">
        <enumeration value="String" />
        <enumeration value="Long" />
        <enumeration value="Double" />
        <enumeration value="Float" />
        <enumeration value="Integer" />
        <enumeration value="Byte" />
        <enumeration value="Char" />
        <enumeration value="Boolean" />
        <enumeration value="Short" />
    </restriction>
</simpleType>

<complexType name="Option">
    <attribute name="label" type="string" use="required" />
```

```
        <attribute name="value" type="string" use="required" />
    </complexType>

    <complexType name="Icon">
        <attribute name="resource" type="string" use="required" />
        <attribute name="size" type="integer" use="required" />
    </complexType>

    <element name="MetaData" type="metatype:MetaData" />

</schema>
```

Suppose A and B provide managed services MS-A (a `ManagedServiceFactory`) and MS-B (a `ManagedService`) respectively, where MS-A and MS-B expose the following configuration properties:

MS-A	
Property	**Type**
gear	Int
ratio	Vector of Float

MS-B	
Property	**Type**
foo	String
bar	Short [ ]

Further suppose that for proper operation C needs:

- “gear” set to 3
- “ratio” set to {3.14159, 1.41421, 6.032E23}
- “foo” set to “Zaphod Beeblebrox”
- “bar” set to {1,2,3,4,5}

The corresponding AUTOCONF.XML document associated with C would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<metatype:MetaData xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.osgi.org/xmlns/metatype/v1.0.0 metatype.xsd ">

  <OCD id="ocd1" name="Managed Service A">
    <AD id="gear" name="%gear" type="Integer" cardinality="0" />
    <AD id="ratio" name="%ratio" type="Float" cardinality="-3" />
  </OCD>

  <OCD id="ocd2" name="Managed Service B">
    <AD id="foo" name="%foo" type="String" cardinality="0"/>
    <AD id="bar" name="%bar" type="Short" cardinality="5"/>
  </OCD>
```

```
  <Designate pid="org.osgi.example.bundleA.MS-A" factory="true"
             bundle="BundleASymbolicName-BundleAVersion">
    <Object ocdref="ocd1">
        <Attribute adref="gear" content="3" />
        <Attribute adref="ratio">
            <Value>3.14159</Value>
            <Value>1.41459"</Value>
            <Value>6.023E23"</Value>
        </Attribute>
    </Object>
  </Designate>

  <Designate pid="org.osgi.example.bundleB.MS-B" factory="false"
             bundle="BundleBSymbolicName-BundleBVersion">
    <Object ocdref="ocd2">
        <Attribute adref="foo" content="Zaphod Beeblebrox"/>
        <Attribute adref="bar">
            <Value>1</Value>
            <Value>2</Value>
            <Value>3</Value>
            <Value>4</Value>
            <Value>5</Value>
        </Attribute>
    </Object>
  </Designate>

</metatype:MetaData>
```

5.1.1 <OCD> (Object Class Definition)

This element, and the <AD> element discussed in the next section, provide type information about a managed service to be configured by a Configuration Resource Processor. Some of the elements and attributes in the XML schema definition are for RFC 62 use, and are not used in RFC 94 AUTOCONF.XML documents.

`name`: A human-friendly name for the OCD.

`id`: a globally unique id that identifies this OCD (this object structure).

5.1.2 <AD> (Attribute Definition)

This element defines an attribute (name-value pair) in an OCD. In RFC 94 terms, it defines a configuration parameter for a managed service.

`name`: A user-friendly name for the attribute definition.

`id`: The name of the configuration parameter.

`type`: The Java type of the parameter. Valid values are String, Long, Double, Float, Integer, Byte, Char, Boolean, and Short.

`cardinality`: an integer “n” describing the “shape” of the parameter:

- n = Integer.MIN_VALUE: the parameter is a java.util.Vector of the specified `type`, no length limit.
- n < 0: the parameter is a java.util.Vector of the specified `type`, maximum length is –n.

All Page Within This Box

- n = 0: the parameter is a scalar (single object of the designated `type`).
- n > 0: the parameter is an array of the specified `type`, maximum size = n.
- n = Integer.MAX_VALUE: the parameter is an array of the specified `type`, no maximum size.

5.1.3 <Object>

<OCD> and <AD> provide type information about configuration parameters. The <Object> and <Attribute>elements provides the value(s) to "fill in" the parameter(s). An <Object> can be considered an instance of an <OCD> and an <Attribute> can be considered an instance of an <AD>.

`ocdref`: the name of the corresponding <OCD> (the <OCD> of which this <Object> is an instance).

5.1.4 <Attribute>

The value(s) for an <AD> in an <OCD>, i.e., the value(s) of a configuration parameter.

`adref`: the name of the <AD> for which this is a value.

`content`: The value (or one of the values) of the attribute. This is a "convenience attribute" when defining scalar attributes (`cardinality` == 0) or single-element arrays or vectors (`cardinality` == 1 or -1). It should not be used when defining larger vectors and arrays (that is, it should not be used in conjunction with the <Value> element.

5.1.5 <Value>

This element is used to specify elements of an array or a vector. For a single-element array or vector, the following XML snippets are equivalent:

- `<Attribute name="foo" content="some_content"/>`
- ```
  <Attribute name="foo">
       <Value>some_content</Value>
  </Attribute>
  ```

The following construct denotes a zero-length array or an empty vector:

- `<Attribute name="foo"/>`

(Whether it is an array or vector depends on the value of the "`cardinality`" attribute on the corresponding <AD> element.)

5.1.6 <Designate>

The <Designate> element encloses one or more <Object> elements and associate them with a pid and a bundle location.

`pid` : the pid (or factory pid) of the managed service to be configured.

All Page Within This Box
  ```

`factory`: a `boolean` describing the type of managed service. If `factory` is `false`, the managed service is a `ManagedService`. If it is `true`, the managed service is a `ManagedServiceFactory`.

`bundle`: the bundle symbolic name and version of the bundle providing the managed service to be configured. This information will be used to generate a bundle location to which to bind the corresponding `Configuration` object (see Section 7 for more information).

`optional`: this boolean attribute allows a section of an AUTOCONF.XML file to be designated optional. If it is true, then if the Configuration Resource Processor encounters a problem processing this section of AUTOCONF.XML, it simply skips it and continues with the rest of the document. If it is false, processing stops and an exception is thrown.

`merge`: this boolean attribute defines what to do when the `pid` attribute refers to a `Configuration` object that already contains a `Dictionary` of configuration parameters. If `merge` is false, the information in AUTOCONF.XML over-writes the existing configuration information – a new `Dictionary` object is created from the contents of AUTOCONF.XML, and it replaces the previous `Dictionary` in the `Configuration` object. If `merge` is true, then:

- an attribute (key) already in the `Dictionary` that is also mentioned in AUTOCONF.XML has its value(s) overwritten with the new value(s) in AUTOCONF.XML.
- an attribute (key) already in the `Dictionary`, but not mentioned in AUTOCONF.XML, is left unchanged.
- an attribute (key) not present in the `Dictionary`, but mentioned in AUTOCONF.XML, is added to the `Dictionary`.

5.2 AUTOCONF.XML Processing

The MDA processes sections of AUTOCONF.XML differently, depending on whether the target managed service is a `ManagedService` or a `ManagedServiceFactory`.

5.2.1 ManagedServiceFactories

`ManagedServiceFactories` have certain advantages for deployment-time configuration:

1. The configuration changes that are made when a deployment package is deployed are easier to “clean up” when a deployment package is undeployed.

2. It is easier to avoid conflicts between multiple deployment packages that all need their own version of a service.

Suppose we had a bundle that provided a translation service – it translates from English to a variety of other languages (French, German, Chinese, etc.). Obviously the service needs to be configured with the appropriate target language. Consider the following scenario if the translation service is configured via a `ManagedService`:

1. A deployment package X is deployed. It requires a translation service that converts English to French. Consequently its AUTOCONF.XML file contains a section that configures the translation service with French as its target language.

2. Later, deployment package Y is deployed. It requires a translation service that converts English to German. Consequently its AUTOCONF.XML file contains a section that configures the translation service with German as its target language.

All Page Within This Box

3. When MDA processes Y's AUTOCONF.XML file, X is "broken".

To avoid this problem RFC 88 specifies that an AUTOCONF.XML document associated with a deployment package can only configure `ManagedServices` provided by bundles in the deployment package[9]. If it wants to configure a managed service of a bundle NOT in the Deployment package, the managed service MUST be a `ManagedServiceFactory`.

Implementing the translation service as a `ManagedServiceFactory` avoids the problem:

1. When deployment package X is deployed, its AUTOCONF.XML file causes a "copy" of the translation service to be created for X to use (see, for example, Section 10.6.4 of the OSGi R3 Specification). This copy translates English to French.

2. When deployment package Y is deployed, its AUTOCONF.XML file causes another "copy" of the translation service to be created for Y to use. This copy translates English to German.

3. Both X and Y are happy.

When processing an AUTOCONF.XML file that refers to a `ManagedServiceFactory`, the Configuration Resource Processor does the following:

- Calls `ConfigurationAdmin.createFactoryConfiguration(factory pid)`. This returns a `Configuration` object that contains a new PID assigned by the Configuration Admin Service. The Configuration object is bound to the location of the bundle providing the service[11], as declared by the "bundle" attribute on the <Designate> element of AUTOCONF.XML.

- Makes an entry in a persistent store that associates the PID obtained above with the ID of the deployment package being deployed. This is to facilitate cleanup when the deployment package is uninstalled, as described below.

- Creates a new `Dictionary` and populates it with the configuration information specified via the <Object> and <Attribute> elements

- Calls `Configuration.update(Dictionary)` to pass the new values to the Configuration Admin service.

If at any point an exception (e.g., `SecurityException`) is thrown, then if the `optional` attribute of <Designate> is `true`, MDA should stop processing this section of the AUTOCONF.XML document and should proceed to the next segment. If, on the other hand, the `optional` attribute is `false` (the default), the Configuration Resource Processor should abort deployment of the deployment package and back out any configuration changes made to this point.

`Configuration.update(Dictionary)` triggers normal Configuration Admin processing: The Configuration Admin service will call the `updated()` method of the `ManagedServiceFactory` once for every Configuration

[9] Since a bundle cannot be in more than one deployment package, this restriction prevents the problem.
[11] Location binding for `Configuration` objects is described in Section 10.11.2 of the Configuration Admin Service section of the OSGi R3 spec. The Configuration Resource Processor derives the bundle location from the bundle symbolic name and version.

All Page Within This Box

object registered under the factory PID. In particular, one of the calls to `updated()` will carry the `Dictionary` from the new `Configuration` object created above, along with the new PID assigned by the Configuration Admin service. As described in Section 10.6.4, the `ManagedServiceFactory` can create a new service object (e.g., a translation service object) and register it in the OSGi Service Registry under the new PID. It can also register other configuration information in the Service Registry to allow the deployed deployment package to find the right instance of the service object.

When the deployment package is uninstalled, MDA will invoke the Configuration Resource Processor, which will look in its persistent store and find the PID(s) associated with the deployment package. For each PID associated with the root bundle, the Configuration Resource Processor does the following:

- Calls `ConfigurationAdmin.getConfiguration(`*`pid`*`)`. This returns the `Configuration` object associated with the PID.
- Calls the `Configuration` objects' `delete()` method. This removes the configuration information from the Configuration Admin's persistent store.

From here, it is normal Configuration Admin processing. Configuration Admin calls the `ManagedServiceFactory`'s `deleted()` method, and the `ManagedServiceFactory` can unregister and clean up the copy of the service object corresponding to the PID.

5.2.2 ManagedServices

There are times when a `ManagedServiceFactory` is not appropriate, like when a service is a singleton. An example might be a TCP/IP service that needs one (and only one) IP address, or an HTTP service that needs one (and only one) port to listen on for HTTP requests. In these cases, a `ManagedService` is more appropriate

As described in the previous section, it is very possible for two deployment packages that share a `ManagedService` to conflict with each other when both try to configure the `ManagedService`. This is why RFC 88 does not allow two or more deployment packages to share a ManagedService from the configuration standpoint. RFC 88 restricts the AUTOCONF.XML file associated with a deployment package to configuring `ManagedServices` that are provided by bundles in the deployment package. Since RFC 88 prohibits having a bundle in more than one deployment package, the possibility of conflict is eliminated.

When processing a <Designate> section of AUTOCONF.XML where `factory` equals "`false`", the Configuration Resource Processor should:

- Verify that the bundle exposing the `ManagedService` is part of the deployment package being processed. If it isn't, processing of the AUTOCONF.XML file should be aborted.
- Call `ConfigurationAdmin.getConfiguration(`*`pid`*`)` to get a `Configuration` object for `ManagedService`. If the `Configuration` object is not bound to a bundle location, the Configuration Resource Processor should bind it to the bundle location derived from the "bundle" attribute of <Designate>.
- Create a `Dictionary` with configuration info extracted from the <Object> and <Attribute> elements.
- Next, the Configuration Resource Processor should call `Configuration.update(Dictionary)` to pass the new values to the Configuration Admin service.

All Page Within This Box

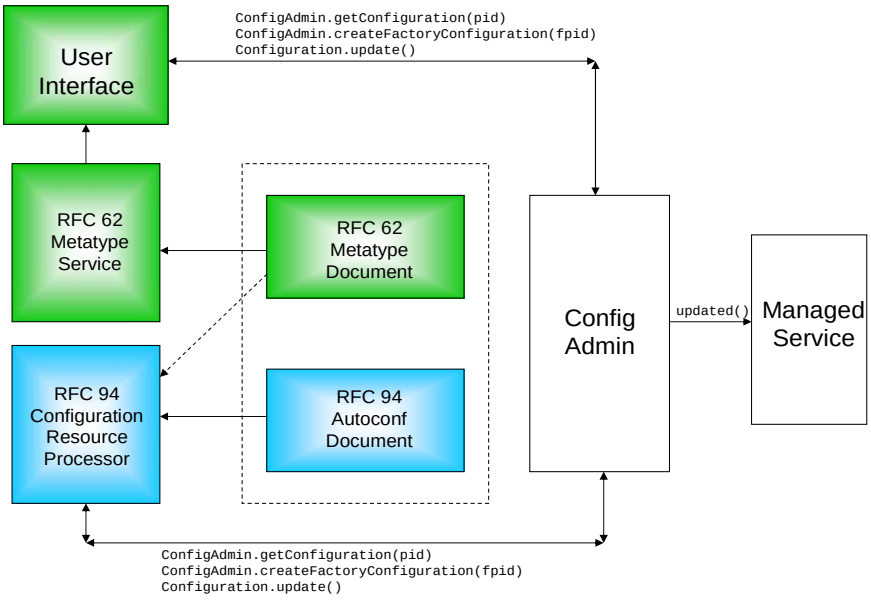
What follows is normal Configuration Admin Service processing. Configuration Admin calls the `updated()` method of the `ManagedService`, which reconfigures itself based on the new configuration parameters in the `Dictionary`.

If at any point in the process the Configuration Resource Processor gets an `Exception` and `optional = false` is specified in the <Designate> element, the Configuration Resource Processor should abort the deployment operation and back out any configuration changes that were made up to this point. Otherwise, the Configuration Resource Processor should ignore this section of AUTOCONF.XML and skip to the next section (if any).

It is strongly advised that the managed service register the properties from the `Dictionary` object in the OSGi Service Registry, as described in Section 10.4.3 of [5].

5.3 Relationship to RFC 62 (Metatype Service)

As shown in the diagram below, the OSGi Metatype Service (RFC 62) and the MEG Configuration Resource Processor (RFC 94) are related. Both use XML documents to provide configuration information to managed services through the Configuration Admin service. Because of their relationship, these two services share a common XML schema definition



The Metatype Service reads an XML document that provides type information about groups of attributes representing configuration parameters for a managed service. The document can also provide information to help a user interface prompt the user for values for the configuration parameters. This includes menu choices and allowed ranges for configuration parameters. Localization information can also be included. This XML document accompanies the bundle providing the managed service.

The Configuration Resource Processor processes an XML document (AUTOCONF.XML) that also contains type information and values for the configuration parameters. This document might be associated with a MEG deployment package (see RFC 88) wants to use a particular managed service. In other words, an

All Page Within This Box

AUTOCONF.XML file is not a resource in the bundle providing the managed service. As shown in the diagram above, an AUTOCONF.XML file might be completely self-contained, incorporating both type definition and values. On the other hand, an AUTOCONF.XML file might simply provide the values for configuration parameters, and use the type definitions provided by a Metatype Service document (this is depicted by the dotted arrow in the diagram).

An example will help clarify these concepts: Suppose a bundle A provided an anagram service, and exposed it as a `ManagedServiceFactory`. The configuration parameters for the anagram service are:

- The language in which to generate anagrams
- The maximum number of anagrams to be returned
- The maximum number of words in each anagram
- Whether or not a word can appear more than once in an anagram
- The minimum and maximum number of letters in each word of an anagram

When the anagram bundle is deployed, it can be accompanied by the following metatype document that:

- Allows a user interface to prompt a user for values for the configuration parameters
- Provided default values for some or all of the configuration parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<metatype:MetaData xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.osgi.org/xmlns/metatype/v1.0.0 metatype.xsd ">

  <OCD id="anagram-ocd" name="Anagram Service Configuration">
    <AD id="language" name="Language" type="String" default="english" >
        <Option label="English" value="english" />
        <Option label="French" value="french"/>
        <Option label="German" value="german"/>
        <Option label="Spanish" value="spanish"/>
    </AD>
    <AD id="max_returns" name="Number of Returned Anagrams"
                         type="Integer" min="1" max="50" default="5" />
    <AD id="max_words" name="Max. Words In Anagram"
                         type="Integer" min="1" max="10" default="3"/>
    <AD id="multiple" name="Multiple Word Occurences Allowed"
                         type="Boolean" default="true"/>
    <AD id="letters" name="Min. and Max. Letters Per Word"
                         type="Integer" cardinality="2"/>
  </OCD>

  <Designate pid="com.acme.anagram_service.pid"/>

</metatype:MetaData>
```

Now suppose that a MEG application is deployed, and it wants to configure an instance of the anagram service for its use. The deployment package containing the MEG application would contain an AUTOCONF.XML file that looks like:

```
<?xml version="1.0" encoding="UTF-8"?>

<metatype:MetaData xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.osgi.org/xmlns/metatype/v1.0.0 metatype.xsd ">

  <Designate pid="com.acme.anagram_service.pid"
                          bundle="AnagramBundleSymbolicName-Version">
    <Object type="anagram-ocd">
        <Attribute name="language" content="French"/>
        <Attribute name="max_returns" content="10"/>
        <Attribute name="max_words" content="3"/>
        <Attribute name="multiple" content="false"/>
```

All Page Within This Box

```
        <Attribute name="letters" content="2"/>
        <Attribute name="letters" content="7"/>
    </Object>
  </Designate>

</metatype:MetaData>
```

In this example, the AUTOCONF.XML file reuses type information in the anagram bundle's metatype file.

Alternatively, the AUTOCONF.XML file might not want to depend on the anagram bundle's metatype file, and might be completely self-contained, in which case it would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<metatype:MetaData xmlns:metatype="http://www.osgi.org/xmlns/metatype/v1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.osgi.org/xmlns/metatype/v1.0.0 metatype.xsd ">

  <OCD id="anagram-ocd" name="Anagram Service Configuration">
    <AD id="language" name="Language" type="String" default="english" >
        <Option label="English" value="english" />
        <Option label="French" value="french"/>
        <Option label="German" value="german"/>
        <Option label="Spanish" value="spanish"/>
    </AD>
    <AD id="max_returns" name="Number of Returned Anagrams"
                         type="Integer" min="1" max="50" default="5" />
    <AD id="max_words" name="Max. Words In Anagram"
                         type="Integer" min="1" max="10" default="3"/>
    <AD id="multiple" name="Multiple Word Occurences Allowed"
                         type="Boolean" default="true"/>
    <AD id="letters" name="Min. and Max. Letters Per Word"
                         type="Integer" cardinality="2"/>
  </OCD>

  <Designate pid="com.acme.anagram_service.pid"
                            bundle="AnagramBundleSymbolicName-Version">
    <Object type="anagram-ocd">
        <Attribute name="language" content="French"/>
        <Attribute name="max_returns" content="10"/>
        <Attribute name="max_words" content="3"/>
        <Attribute name="multiple" content="false"/>
        <Attribute name="letters" content="2"/>
        <Attribute name="letters" content="7"/>
    </Object>
  </Designate>

</metatype:MetaData>
```

All Page Within This Box

6 Alternatives

Previous versions of the schema attempted to detect cases where a configuration parameter was assigned a value of the wrong type. These problems could be detected when the AUTOCONF.XML document was validated against the schema. However, we decided to drop this because (a) it made the schema significantly bigger, more complex, and harder to read, and (b) it provided incomplete protection (it caught some errors, but didn't catch others). Since AUTOCONF.XML documents will probably be generated by tools, these tools can detect the errors that the schema definition attempted to detect.

7 Security Considerations

Allowing a deployment package's AUTOCONF.XML file to (re)configure arbitrary managed services creates some security exposures.

1. As described in [5] it is possible for a malicious bundle to register a `ManagedService` under a PID used by another (legitimate) bundle. This essentially "hijacks" the `ManagedService` PID, and constitutes a denial of service attack on the legitimate bundle (since it never receives the configuration information it needs). Section 10.11.2 of [5] describes a location binding technique that can be used to prevent this. The RFC 94 Configuration Resource Processor will bind `Configuration` objects to locations specified in AUTOCONF.XML[14].

2. As we discussed in an earlier section, suppose we had a bundle that provided a translation service – it translates from English to a variety of other languages (French, German, Chinese, etc.). Obviously the service needs to be configured with the appropriate target language. Consider the following scenario if the translation service is configured via a `ManagedService`:

   - A "good" deployment package X is deployed. It requires a translation service that converts English to French. Consequently its AUTOCONF.XML file contains a section that configures the translation service with French as its target language.

   - A malicious deployment package developer develops a malicious deployment package Y. Y's AUTOCONF.XML file reconfigures the translation service to translate English to Chinese.

[14] The bundle location will be derived from the bundle symbolic name and version declared in the "bundle" attribute of <Designate>.

All Page Within This Box

- When (malicious) deployment package Y is deployed, MDA processes Y's (malicious) AUTOCONF.XML file, and X is "broken".
- For this reason, AUTOCONF.XML is not allowed to configure `ManagedServices` that are not part of the deployment package.

3. The problem is somewhat less severe if a service is configured via a ManagedServiceFactory, since each user of the service can get its own version of the service. But while it is more difficult for a malicious deployment package to break a good deployment package's configuration, it is still possible for a malicious deployment package (with a malicious AUTOCONF.XML file) to mount "denial of service" attacks by creating instances of services with parameters set maliciously high or low. For example, a malicious AUTOCONF.XML file could create several bogus translation services. While these bogus copies wouldn't interfere with translation services configured for good deployment packages, if the translation dictionaries used by the translation service were large, these bogus, unused instances of the translation service would use up memory. Or consider an email checking service that checked every "x" seconds to see if a user had email. A malicious AUTOCONF.XML file could create one or more instances of the email checking service and set "x" to 1, which would consume communications bandwidth and increase communications costs for the user.

   To hinder such attacks, a configuration resource processor's capabilities should be limited by the permissions granted to the signer of a deployment package. The MEG Deployment Admin must enforce this limitation by performing two operations:

   - The MEG Deployment Admin must construct an `AccessControlContext` that represents the permissions associated with the signer of the deployment package. If the MEG implementation on the device includes the Conditional Permission Admin [7] (which is optional in MEG), the MEG Deployment Admin can invoke `getAccessControlContext()` on Conditional Permission Admin to get such an `AccessControlContext`. Otherwise, the MEG Deployment Admin must construct the `AccessControlContext` itself, using permission information in the DMT.
   - Before calling a configuration resource processor, the MEG Deployment Admin must perform a `doPrivileged(AccessControlContext)` operation. This will limit the permissions of the configuration resource processor to the permissions granted to the signer of the deployment package.

   For example, suppose there were two entities that could deploy deployment packages onto a mobile phone. The wireless carrier AcmeWireless.com wants to deploy games, address book/PIM applications, and other "carrier applications". An enterprise SpacelySprockets.com wants to deploy mobile enterprise applications on a phone.

   Among the permissions granted to AcmeWireless.com might be `ConfigurationPermission("com.AcmeWireless.*, SET_ACTION")`. This permission would allow AcmeWireless to configure any managed service it provides on the platform. In particular, AUTOCONF.XML files contained in deployment packages signed by AcmeWireless.com could configure any managed service provided by AcmeWireless. However, deployment packages signed by AcmeWireless could not configure managed services provided by SpacelySprockets.

   SpacelySprockets might be granted `ConfigurationPermission("com.SpacelySprockets.*",SET_ACTION)` and `ConfigurationPermission("com.AcmeWireless.fooService",SET_ACTION)`. These permissions would allow deployment packages signed by SpacelySprockets to configure any managed service from SpacelySprockets and also the "fooService" provided by AcmeWireless. However, configuring other managed services provided by AcmeWireless.com would not be allowed.

All Page Within This Box

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]. OSGi RFC 80, “Declarative Services”.

[4]. OSGi RFC 88, “MEG Deployment”.

[5]. OSGi R3 Specification, Chapter 10, “Configuration Admin Service Specification”.

[6]. OSGi RFC 62, “Meta Type Repository”

[7]. OSGi RFC 95, “Conditional Permissions”

[8]. OSGi RFC 87, “OSGi DMT”

[9]. OSGi RFP 53, “MEG WS Deployment Model”

8.2 Author’s Address

Name	Joe Rusnak
Company	IBM
Address	
Voice	919 543 3317
e-mail	jgrusnak@us.ibm.com

8.3 Acronyms and Abbreviations

See Section 2.3.

8.4 End of Document

All Page Within This Box