RFC-140 Residential Management Tree

Draft

48 Pages

Abstract

Different industries are interested in the application of OSGi for their business, in which remote management is a key issue. Telecom operators (both fixed and mobile ones), server managers, and automotive manufactures, etc. need solutions to remotely manage their instances of OSGi frameworks. The main problem with that need is that is very difficult to solve with a single solution, taking into account that the management protocols are many and different for the different industries. To achieve this goal, a management object model should be defined to expose OSGi framework manageable information through a management agent.

1 Document Information

1.1 Table of Contents

1 Document Information....2
1.1 Table of Contents....3
1.2 Terminology and Document Conventions....3
1.3 Revision History....3

2 Introduction....7

3 Application Domain....7

4 Problem Description....8
4.1 Management agent making use of OSGi standardized Java interfaces....8
4.2 Mobile specification approach....9
4.2.1 Support for TR-069 notifications....9
4.2.2 Limitations in the number of services available in the DMT....10
4.2.3 Mapping TR-069 to the OMA-DM inspired DMT model....11
4.2.4 Conclusion....11

5 Requirements....11

6 Technical Solution....12
6.1 Legend....15
6.2 Framework Object....16
6.3 BundleState Object....23
6.4 PackageState Object....28
6.5 ServiceState Object....30
6.6 Filters Object....33
6.7 BundleResources Object....41
6.8 Limitations....43
6.8 Maximum and Minimum values of numeric parameters....43
6.8 Future Work....43

7 Considered Alternatives....44

8 Security Considerations....45

9 Document Support....45
9.1 References....45
9.2 Author's Address....46
9.3 Acronyms and Abbreviations....46
9.4 End of Document....46

1.2 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 9.

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

1.3 Revision History

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

Revision	Date	Comments
Initial	*Sep. 12 2008*	Initial Draft Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp
0.2	*Oct. 9 2008*	2nd Draft Overall architecture is changed based on REG meeting and RFC-139 definition. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
0.3	*Nov. 10 2008*	3rd Draft Definition tables for all nodes in Management Objects are added. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
0.4	*Jan. 19 2009*	4th Draft Revising tree architecture based on the discussion at the REG meeting. Filter format description is added. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
0.5	*Feb. 12 2009*	5th Draft Revising Bundles Object based on discussion at the REG meeting. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp

Revision	Date	Comments
0.6	*Mar. 26 2009*	6th Draft Revising Filters Object specification based on discussion and analysis. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
0.7	*May. 14 2009*	7th Draft Revising Resources sub-tree and Filters Object specification based on discussion at the REG meeting. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp
0.8	*Jun. 04 2009*	8th Draft Revising wording and ServiceState Object specification based on discussion at the REG meeting. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp
1.0	*Sep. 18 2009*	9th Draft Bundle StartLevel configuration is moved to the Framework Object. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp
1.1	*Oct. 16 2009*	10th Draft Lifecycle sub-tree of Bundle is moved ti the Framework Object. Revising BundleState Object specification and BundleResources Object specification based on discussion at the REG meeting. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp

Revision	Date	Comments
1.2	*Jan. 20 2010*	11th Draft Revising example of the Filter and TargetSubtree (Case1 and Case4). Declaring Residential management tree Data plug-in root path. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp
1.3	*Feb. 19 2010*	12th Draft Revising BundleState Object and PackageState Object, ServiceState Object. Adding NumberOfEntries nodes below the $/OSGi/<instance-id>/. Moving Filters Object and Local node to below the $/OSGi/<instance-id>/. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp
1.4	*Apl. 1 2010*	13th Draft Revising Certificate information format section and Service Property Dictionary nodes section based on discussion at the REG meeting. Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp
1.5	*May. 12 2010*	14th Draft Removing Policy Object and Configuration Object based on discussion at the Conference Call at May 12. Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.6	*Aug. 8 2010*	15th Draft Revising all section based on discussion at the REG F2F meeting in Gerona. Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp

Revision	Date	Comments
1.7	*Sep. 6 2010*	16th Draft Revising Framework Object, BundleState Object, Filter Object and BundleResource Object based on comments for Ver. 1.6. Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.8	*Oct. 12 2010*	17th Draft Revising Framework Object, BundleState Object based on comments for Ver. 1.7. Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.9	*Nov. 24 2010*	18th Draft Revising Framework Object and Filters Object based on comments for Ver. 1.8 and agreements in F2F meeting in Heidelberg. Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Koya Mori, NTT Corporation, mori.kouya@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.10	*Dec. 1 2010*	19th Draft Revising Framework Object and adding the Ext node to each Management Object based on comments for Ver. 1.9. • Bug 1797/1807/1808/1806 Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.11	*Dec. 20 2010*	20th Draft Revising Framework Object and each table of Management Object based on comments for Ver. 1.10. • Bug 1814/1820/1827/1829/1830/1835/1837/1847 Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp

Revision	Date	Comments
1.12	*Feb. 1 2011*	21th Draft Revising Filters Object and each table of Management Object based on comments for Ver. 1.11. • Bug 1858/1860/1862 Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp
1.13	*Feb. 15 2011*	22th Draft Revising minor issues based on comments in Bugzilla for Ver. 1.12. • Bug 1871/1872/1873/1874/1875/1876/1877/1878/1879/1881 Shigekuni Kondo, NTT Corporation, kondo.shigekuni@lab.ntt.co.jp Ikuo Yamasaki, NTT Corporation, yamasaki.ikuo@lab.ntt.co.jp

2 Introduction

Traditionally, fixed telecommunication operators don't have knowledge about what runs in the customer's local area network (LAN). They provide connectivity and manage the wide area network (WAN) that provides this connectivity, but they do not know anything about the devices and networks behind the gateway (xDSL mainly) that interconnect WAN and LAN. Recently the need for management of customer networks and devices is increasing in order to make the deployment of new complex services at home (home automation, tele-health, VoIP, IPTV, surveillance, etc) feasible with reasonable costs. For example to avoid sending technicians to the customer premises for solving problems.

There are two main kinds of devices that need to be managed remotely in operator's business: those which come from the fixed line business managed via the TR-069 protocol, that is standardized by the Broadband FORUM [6], and those which come from the mobile business managed by OMA DM, which is standardized by the Open Mobile Alliance. Up to now, the OSGi specifications cover the OMA DM ones, but lacks of specification about how an OSGi framework on a device could be managed by TR-069.

One key question is: why do we need two technologies to manage devices in a converged scenario in which the two kinds of devices are going to interact between them? The question has a better answer from the business point of view than from a technical point of view. Fixed and mobile businesses have evolved independently during the last years. Both technologies have acquired enough critical mass not only for having success in product implementation and roll-out, but also to be a de facto standard in their respective applications domains.

Despite the fact that the best solution would be a single protocol to manage all kind of devices, we are aware that at least for the short-medium term both technologies must coexist. For the future, the model shouldn't be closed to add new solutions, for example a mixed model that unifies both worlds.

3 Application Domain

Driven by triple play service delivery in the home network, fixed line access service providers have the need to configure home devices to ensure the proper service delivery. Broadband Forum's CPE WAN Management Protocol (CWMP, alias TR-069) enables them to do this. By using a remote management server (Auto Configuration Server, ACS), they are able to manage TR-069 enabled devices. TR-069 provides them with possibilities to configure parameters, be informed of parameter changes (notification), start diagnostic tools, update firmware, etc.

Similarly, for the mobile world, the OMA defined the OMA-Device Management specification for remote management of mobile devices. OMA-DM offers similar tools to the mobile service providers as TR-069 to fixed line service provider, but OMA-DM is of course tailored to the specifics of the mobile environment.

As OSGi technology offers a flexible software environment for all these different devices, the remote management of the platform is of interest for both fixed and mobile service providers. As such, it should be possible to integrate the remote management of the OSGi platform, and the applications running on top of it, in the existing management infrastructure.

The DMT Admin service with its mobile management tree in the Mobile specification for OSGi R4 standardizes the remote management of an OSGi platform. As it is largely inspired by OMA-DM, it needs to be evaluated for multi protocol support.

4 Problem Description

In a scenario in which service providers offer a growing number of services, to use specific solutions for the management of those services is not the most suitable option. To speed up the deployment of these services, such as triple play, home automation or tele-health, it is essential to offer general management solutions that allow for the management of a large number of services and the flexible life-cycle management of applications.

These devices usually are already managed by a standard protocol, so it makes sense that an OSGi framework, which hosts the services, running on a device could be managed in the same way as the other resources of the device. Of course, the remote management should be fully integrated in the existing remote management

solutions of the service provider to avoid duplicating management infrastructure and to increase performance on the devices.

Currently, there are two options in OSGi for remote management:

- create a management agent bundle making use of the Java object interfaces,
- create a protocol adapter bundle that interacts with the DMT Admin service, as defined in the OSGi Mobile specification.

4.1 Management agent making use of OSGi standardized Java interfaces

Currently, for the management of a bundle, the OSGi specifications define different Java objects with which a management application can interact. Using this approach, a management agent can implement extensive management of the OSGi framework, as well as any service standardized. Mapping the Java interfaces to the specific remote management protocol and data model tree is up to the management agent.

For runtime interaction with a bundle, a bundle can register a service in the service registry. However, this service interface is not standardized. Also, mapping the service interface to a general management model is not standardized. A current approach is to implement a proprietary service interface on all bundles to be managed. By tailoring this interface so that it easily maps to the management protocol primitives, it is simple for the management agent to map remote management commands to the bundle's service interface. The disadvantage is the proprietary service interface, so that 3rd party bundles might not be compliant.

As a conclusion we can say that this current approach allows for extensive remote management of any aspect of the OSGi platform, but lacks a standardized service interface definition for bundles to implement.

4.2 Mobile specification approach

The Mobile Expert Group has provided its own solution based on the OMA [3] Device Management [4] specification to provide a remote management solution. The OSGi Mobile specification contains two chapters related to remote management:

- chapter 3: detailing the mobile management tree
- chapter 117: detailing: the DMT Admin service, bundle plugin interface specification, the notification service

The Device Management Tree model of OMA-DM was chosen as meta-data model and operational model. However, it was intended to be mappable to other protocols.

An analysis of mapping the Mobile specification DMT model to TR-069, however, shows that the current DMT model approach (as defined in the OSGi R4 Mobile Specification) introduces some issues. For example:

- Limitations for active or passive notifications on any parameter in the object tree
- A limited number of services have been mapped to the DMT model
- The complexity of mapping a new protocol to the OMA-inspired DMT model, which could imply performance issues on limited devices.

4.2.1 Support for TR-069 notifications

TR-069 offers the feature of active and passive notifications. By setting a parameter's notification attribute, a remote manager requests to be notified with the parameter's new value at the time the value changes (active notification) or at the next periodic inform (passive notification). Notification can be configured on any parameter of the TR-069 object tree. This approach enables the remote manager to be informed not only of changes in status variables of the platform, but also of configuration changes performed by a local manager, e.g. through a local Web interface.

The Mobile specification offers a few features that could help to implement TR-069 notification support:

- The DMT Admin service sends events using the Event Admin service when operations have been performed on nodes (nodes added, removed or copied; node values changed etc.)

- The OSGi Notification service defines a way to alert a remote management server. Protocol adapters on their turn have to implement a RemoteAlertSender interface (and register it) for use by the notification service. Notifications are sent by calling sendNotification on the notification service:

- The Monitor Admin service: A bundle can register a Monitorable service, to be used by the Monitor Admin service. By registering a Monitorable service, the bundle exposes access to a number of status variables. Notification can be implemented by the StatusVariable provider. If it does, it will call the update method on the Monitor Listener. The Monitor Admin service then generates an event on the Event Admin service. The Monitor service is currently also represented in the DMT tree.

Two problems arise when trying to map the current approach to TR-069:

- TR-069 defines that notification is applicable to any parameter in the object tree.

  Currently, the DMT Admin service only send events for operations on DMT nodes that were performed using the DMT Admin API. For example: if configuration changes are performed by using the Configuration Admin service API, no events will be sent. Most of the current implementations do not perform all changes via the DMT Admin service. Therefore, the events sent by the DMT Admin service are an only subset and thus not very reliable as single source of events (and thus as single source of TR-069 notifications).

  The OSGi Monitor service only supports notification of changes on Status Variables, exposed through a Monitorable service, and enabled by the bundle to support on-change notification (i.e. dynamic Status Variables).

- Requesting notification is not fully under the control of the remote manager. In the case of a bundle using the notification service, there is no standardized way to configure the bundle to send alerts when the value of one of the implemented DMT nodes changes. In the case of the monitor service, the sending of events can be controlled, but is limited to dynamic Status Variables.

  The current DMT Admin service has no attributes properties on its nodes to be used to configure notification behavior, such as active notification and passive notification defined in TR-069. Therefore, a remote manager cannot control the notification behavior of DMT nodes in a standardized way.

To conclude, the current options, as provided in the Mobile specification, limit notification of parameter changes to StatusVariables, explicitly enabled for monitoring. There is no standardized approach available to monitor changes on any node in the DMT.

4.2.2 Limitations in the number of services available in the DMT

The OSGi R4 Mobile specification mapped a number of services to the DMT tree. However, these services are limited to the services listed in the Mobile Specification. Other interesting services, as listed in the OSGi R4 Service Compendium are not yet mapped (standardized) in the DMT tree:

The DMT tree defined in the Mobile Specification contains objects for the following services:

- Configuration Admin service
- Log service
- Monitor Admin service
- Application Admin service
- Conditional Permission Admin service
- Deployment service

A number of areas that could be of interest to a remote manager are currently missing in the DMT tree:

- Startlevel management
- Bundle management: managing individual bundles as opposed to deployment packages (inventory, life-cycle management, exported services, …)
- Service management: getting a remote view on services registered in the service registry
- Permission Admin management
- Home Gateway Core Function management: handling Home Gateway core functions, such as firewall configuration and port forwarding control, from bundles running on an OSGi framework

4.2.3 Mapping TR-069 to the OMA-DM inspired DMT model

Within the OSGi Mobile specification, the choice has been made to model the DMT after OMA-DM.

As a result, creating an OMA-DM protocol adapter is quite straightforward. Although no major hurdles have been identified in creating a TR-069 protocol adapter, it is less straightforward:

- The TR-069 RPC primitives have to be translated to the DMT Admin service interface methods (which are OMA-DM RPC inspired).
- The TR-069 tree has to be mapped to the DMT tree. Translating object model specific features like DMT meta nodes, or TR-069 attributes is not straightforward. It might require specific extensions to the DMT, e.g. to support TR-069 attributes, or a single node in the DMT might result in multiple objects in a TR-069 data model, etc.
- The TR-069 data types have to be mapped to the DMT Admin data types. However, TR-069 data types, such as “unsignedint” and “dateTime” (ISO 8601), cannot be translated appropriately into DMT Admin data types defined in the current specification. Translating these data types might result a limitation of the available value range and a complex object that consists of multiple nodes, respectively.

4.3 Conclusion

The OSGi Mobile specification delivers a standardized data model (the DMT), and standardized interface (on the DMT Admin service) to enable remote management through a protocol adapter. However, the current specification lacks management objects for a number of interesting areas. Also, there is some support lacking for TR-069 notifications. Furthermore, since the DMT model is OMA-DM inspired, implementing a TR-069 protocol adapter is not straightforward, although not impossible.

5 Requirements

REQUIREMENT[1]: A management tree, which is mappable to multiple remote management protocols, MUST be standardized. The solution MUST be mappable at least to OMA DM and TR-069 protocols. The model MUST be open to add new protocols in the future, like a possible common solution to substitute OMA-DM and TR-069.

REQUIREMENT[2]: A bundle MAY implement a non-standard sub-tree of the management tree. The solution MUST support the management of this type of sub-trees. As such, it MUST define the interface to be implemented by the bundle to support this management.

REQUIREMENT[3]: The management tree SHOULD cover bundle life cycle management and service monitoring.

REQUIREMENT[4]: The management tree SHOULD cover all services defined in the release 4 of the OSGi specifications. If prioritization is needed, at least the following services MUST be covered: Start Level , Package Admin and Log Service.
[REMARK]: Though the services which are Configuration Admin and Permission Admin, Conditional Permission Admin are required as the following services in RFP87, they are removed from this requirement because they are not supported in this RFC.

REQUIREMENT[5]: The solution MUST have a good performance in order to run on devices with limited resources.

REQUIREMENT[6]: The solution SHOULD enable bundles to hook into Internet Gateway functions such as DHCP.

6 Technical Solution

6.1 Legend

All nodes of the Residential Management Tree are described in a table format. This table format defines the following meta information:

- Add – An x indicates that the implementation must support the creation of the given node by the management system.

- Get – An x indicates that the implementation must support retrieval of the properties of the given node (including the value).

- Replace - An x indicates that the implementation must support setting the value of the given node. Support for changing other properties is optional. Note, that this column does not correspond to the node attribute changing, which can be provided by an implementation even if the node value cannot be changed, for example in case it supports setting the Title property.

- Delete - An x indicates that the implementation must support deletion of the given node by the management system.

- Type - The node type for an interior node, or the data type for a leaf node. The following node and data types are defined.

    - Node type: node, list_node

    - Data type: string, int, float, date, bin, bool, b64, long, unsigned_int, node_uri, date_time

The “list_node” is the interior node must have node type of "org.osgi/1.0/ListSubtree", which is defined in RFC141 as URI identifying an OMA DM Device Description Framework (DDF) file. The node type indicates that the sub-tree under this interior node contains “list”. The name of child nodes must be a continuous numerical string from 1 and the values of the child nodes must be the element of the list. In addition, the Meta Node of child nodes must have the same format. See RFC141 for the details.

- Cardinality - The range of occurrences of the given node. * means infinite.

- Scope - The scope indicates the creation strategy. It can have the following values:

    - P - Permanent. A permanent node cannot be changed by the management system. It can, however, appear due to an internal device event, for example, the addition of a network interface.

    - D - Dynamic. A node that must be created by the management system. Such a creation can then automatically create other nodes.

    - A - Automatic. A node that is created automatically by a managed object if its parent node is created.

6.2 Overall Residential Management Tree

This RFC defines the Residential Management Tree which is handled via DMT Admin service as it is available on an OSGi Residential Platform. The protocol used between the remote server and the device is not specified, but it is expected that the TR-069 protocol will be the management protocol used to manipulate this tree.

Although the top level nodes of this tree depend on the user policy, this RFC supposes that TR-106 structural requirements defined by the Broadband Forum [6] will be adopted by the Residential Management Tree architecture. The top level nodes of a tree adhering to TR-106 are depicted in Figure 1 (*See TR-106 Amendment1 [7]*). The partial tree, enclosed by the dashed line, is specified by TR-106, which is expected to be used with the Residential Management Tree in many circumstances in residential service domain. This RFC does not define any restrictions on the architecture enclosed in the dashed line, and the ancestor nodes of *OSGi* node can be arbitrarily defined by users. Therefore, the parent node of the *OSGi* node is referred to as "$" in the following sections. This RFC defines that the "$" is relative path from root node. (i.e. the absolute path of the parent node of the *OSGi* node is "./$")

The OSGi Residential Management Tree is a relative tree. Devices can place the root of this tree anywhere in the Device Management Tree. In this RFC, this relative location in the Device Management Tree is indicated with the "./$". The root of the OSGi tree is set in the System property that must not change during runtime:

residentialmanagement.osgi.root

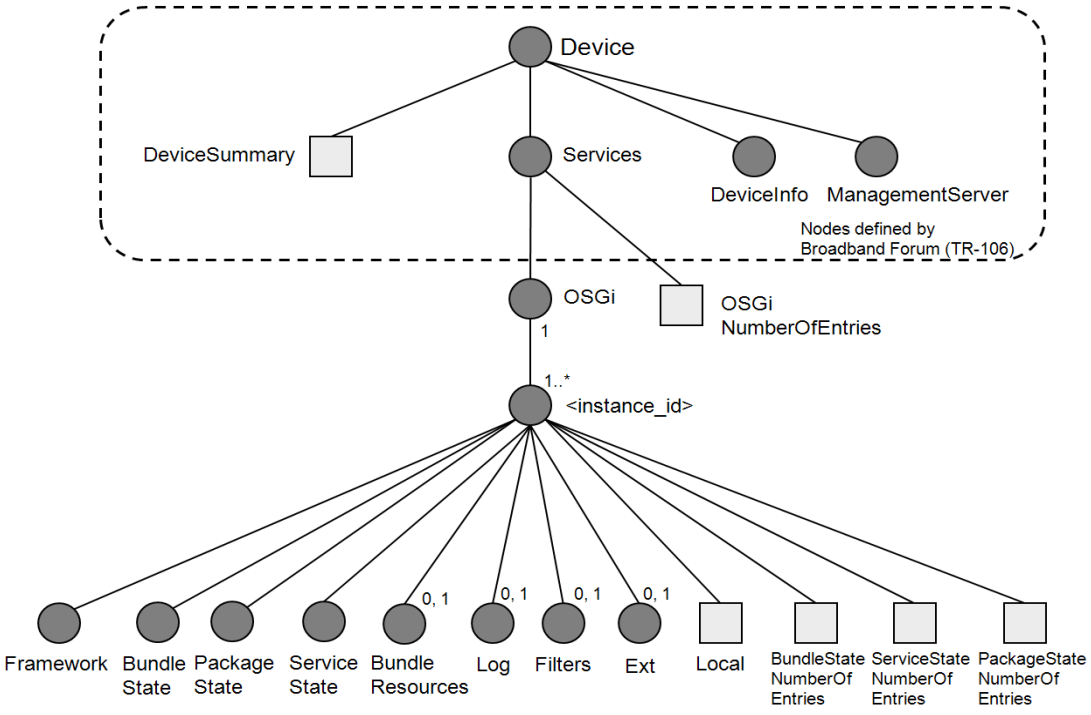


Fig.1 Overall Management Tree

When “residentialmanagement.osgi.root” placed in “./Device/Services/CustomNode/CustomNode2”, The TR-069 path "Device.Services.OSGi.1.Framework" is mapped to "./Device/Services/CustomNode/CustomNode2/OSGi/1/Framework".

The OSGi Residential Management Tree consists of a number of distinct parts as shown in Figure 1. Each of the sub-trees in the figure is explained in the following sections. Users of this tree may add a user-defined sub-tree under the *./$/OSGi/<instance_id>/Ext* node. Moreover, there is no restriction on the use of sub-trees defined in Mobile Management Tree [8] as a user-defined sub-tree in the Residential Management Tree.

./$/OSGi/<instance_id> node is used to represent an instance of the *OSGi* framework on a device. In most cases there is only one instance of *OSGi,* so only *./$/OSGi/1* is available. However, in some cases such as multiple OSGi Frameworks run on the residential home gateway, there exist several *OSGi* instances in the Residential Management Tree. In this case, the local OSGi framework on which the bundle registering Data Plugin of the Residential Management Tree is running should be identified, so that the DMT Admin implementation can access the indicated node path via appropriate way. Therefore, The leaf node, *./$/OSGi/<instance_id>/Local* node , represents whether the OSGi Framework object is local or not as boolean value. How the Data Plugin of the Residential Management Tree on the local OSGi Framework finds and communicates with the other OSGi Frameworks is out of scope of this document.

The pairs between each OSGi Framework and the corresponding <instance_id> must be kept persistently beyond restart of the local OSGi Framework and reconnection of other OSGi Frameworks. OSGiNumberOfEntries and BundleStateNumberOfEntries, ServiceStateNumberOfEntries, PackageStateNumberOfEntries are represent the number of those instances existing at the moment. When the instances are added or deleted, the value of the nodes must be incremented or decremented, respectively.

The Residential Management Tree adopts the complemental data model with the Mobile Management Tree defined in the OSGi Mobile Specification. On the one hand, *Log* which is one of top level node has definitions similar to the Mobile Specification. On the other, the Residential Management Tree has some original sub-trees: *Framework, BundleState, PackageState, ServiceState* ,*Filters, and BundleResouces*. Especially, all of *Framework, BundleState, PackageState, ServiceState* Subtrees are designed to be compliant to TR-106 defined by Broadband Forum[7]..

The Framework sub-tree is to manipulate the OSGi framework on which this management tree is implemented and to control the life cycle of installed bundles instead of the *Deployment* sub-tree defined in the Mobile Specification. The BundleState sub-tree is used to derive information of individual bundles. The PackageState and ServiceState sub-trees provide information of available Packages and Services on the OSGi framework, respectively. The Filters sub-tree is used to filter information contained in a tree. The BundleResouces sub-tree is used to derive resources in a bundle jar file.

Some sub-trees have the extension node “Ext” for vendor extension. The vendor extension must not be defined outside of Ext nodes.

Basically, the structure of the sub-trees in the Residential Management Tree corresponds to the one defined in “RFC-139 JMX Control of OSGi”, which defines four interfaces for OSGi framework core APIs; FrameworkMBean, BundleStateMBean, PackageStateMBean and ServiceStateMBean. Although the fundamental architecture adheres to the API of RFC-139, some features such as command execution and structural data exchange are adjusted to realize functionality in a hierarchical object tree, because the Java interfaces are difficult to map completely to the Residential Management Tree which adheres to the DMT Admin model. One of the biggest differences is that returned values of OSGi Core API's methods cannot be returned in DMT Admin interfaces.

The nodes for Overall Management Tree are explained in Table 6.1.

Table 6.1 Nodes for Overall Management Tree

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
OSGi		X			node	1	P	This node is root node of the Residential Management Tree.
OSGiNumberOfEntries		X			unsigned_int	1	P	A leaf node that represents the number of ./$/OSGi/<instance_id> nodes.
OSGi/**<instance_id>**		X			node	0..*	P	This node is used to represent an instance of the OSGi framework on a device. The name must be a numerical string from 1. The range of the node name number must be from 1 to Long.MAX_VALUE.
OSGi/<instance_id>/ **Local**		X			bool	1	P	A leaf node that represents whether the OSGi Framework object is local or not as boolean value.
OSGi/<instance_id>/ **Ext**		X			node	0,1	P	Interior node that can contain values for vendor extensions.
OSGi/<instance_id>/ **BundleStateNumberOfEntries**		X			unsigned_int	1	P	A leaf node that represents the number of ./$/OSGi/<instance_id>/BundleState/<id> nodes.
OSGi/<instance_id>/ **ServiceStateNumberOfEntries**		X			unsigned_int	1	P	A leaf node that represents the number of ./$/OSGi/<instance_id>/ServiceState/<id> nodes.
OSGi/<instance_id>/ **PackageStateNumberOfEntries**		X			unsigned_int	1	P	A leaf node that represents the number of ./$/OSGi/<instance_id>/PackageState/<id> nodes.

6.3 Framework Object

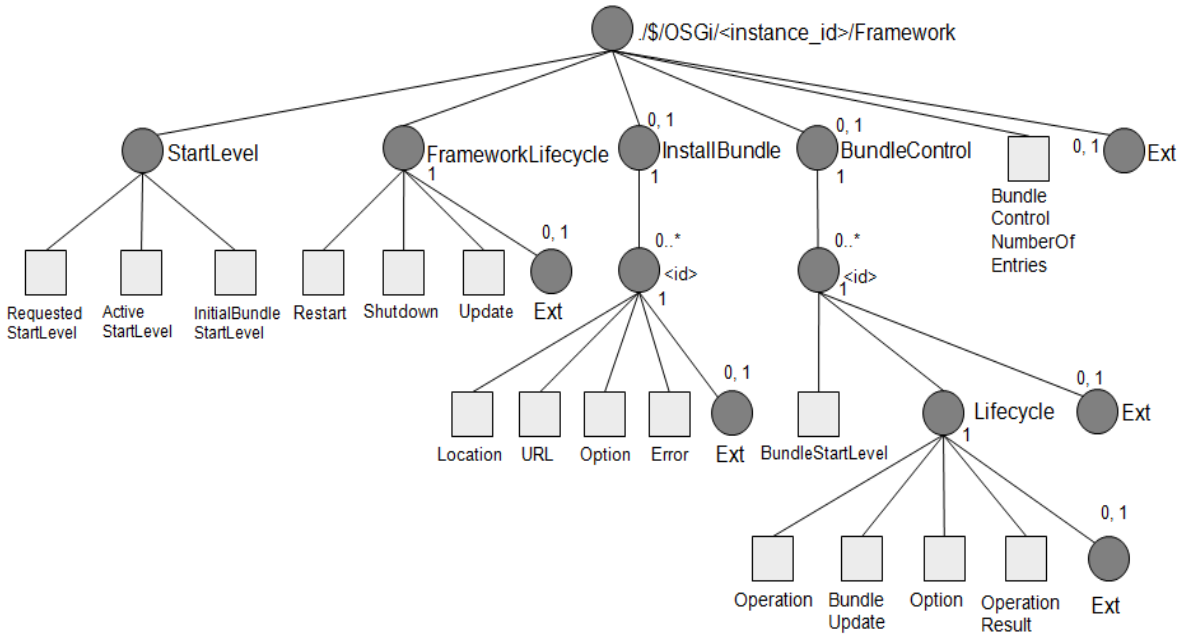


Fig.2 Framework Object

The *Framework* Object is a managed object that allows manipulation of the OSGi framework, StartLevel configuration, Framework Lifecycle control and Bundle lifecycle control. The *Framework* Object is an mandatory, therefore the manipulation of the OSGi residential platform is always available.

The tree structure of *Framework* Object can be accessed from the *./$/OSGi/<instance_id>/Framework* sub-tree. Figure 2 shows the structure of the *Framework* Object sub-tree.

The *Framework* Object consists of 5 parts; *StartLevel, InstallBundle, FrameworkLifecycle, BundleControl* and *Ext*.

The *Framework* Object must support transactions because all changes to the *./$/OSGi/<instance_id>/Framework* tree must be done in an atomic session to keep the OSGi Framework consistent. Only atomic sessions can perform the required changes to the node values.

The following introduces the steps taken to install a bundle. At first, ./$/OSGi/<instance_id>/Framework/InstallBundle/<id> node must be created by ReadWriteDataSession#createInteriorNode() method, its children will be created automatically by the Framework Object. Next, the value of Location must be set by ReadWriteDataSession#setNodeValue() method. The URL is optional. After that, when TransactionalDataSession#commit() method is called, InstallBundle sub-tree should call BundleContext#installBundle(String) method where the first argument is the Location or BundleContext#installBundle(String, InputStream) method where the first argument is the Location and the second argument is the InputStream object retrieved from the specified URL.

[REMARK] If commit() method is called BEFORE any value of Location node is set by setNodeValue() method, InstallBundle sub-tree must NOT call installBundle() method. If commit() method is called AFTER either of them is set, installBundle() method must be called. Multiple bundles can be installed by just one commit() if multiple ./$/OSGi/<instance_id>/Framework/InstallBundle/<id> are created before one commit() is called.

The *BundleControl* sub-tree provides bundle start level and life cycle operations of an individual bundle after installation. Regarding bundle life cycle control, the operation is basically conducted by indicating the desired

state of the bundle; “Active”, “Resolved” or “Uninstalled”. The appropriate state can be reached with the help of the operation specified in the ./$/OSGi/<instance_id>/Framework/BundleControl/<id>/Lifecycle/Operation node. This sub-tree supports lazy activation and transient start (or stop). When commit() is called, Bundle#start(int option) or Bundle#stop(int option) is called where the argument is the value set in the ./$/OSGi/<instance_id>/Framework/BundleControl/<id>/Lifecycle/Option node. The value of *Option* node must be retained beyond the data plug-in bundle reboot. Details of the values are shown in Table 6.2.

[REMARK] A Protocol Adapter (or local manager) should set not only the Operation in which “Resolved” or “Active” is set but also *Option* node in same transactional session. The reason is that if a protocol adapter (or a local manager) sets only the *Operation* node, not the *Option* node, a value that the remote manager does not expect might be kept in the *Option* node. In that case, the operation resulting from this value is likely to differ from what the remote manager expects.

Note that *Bundle*Update is indicated directly as a command because this kind of operation does not represent the specific desired state of a bundle. The value set to *Budnle*Update node is the string of the new bundle's URL. The *Error* node represents the result of the latest operation on this bundle. The error message will be set to the *Error* node when Operation or BundleUpdate operation fails. On the other hand, the empty String will be put to the *Error* node when the operation succeeds.

To use the TR-069 protocol to control the Residential Management Tree, the node name of *./$/OSGi/<instance_id>/Framework/InstallBundle/<id>* should be represented by a numeric character string, not a literal node name such as Location or URL. The reason is that TR-069 has only the RPC called “AddObject” to create a new node in a tree, and the RPC can take as an argument the path name of the collection of objects for which new nodes are to be created.

[REMARK] According to TR-069, a dynamic node path in the Residential Management Tree should be defined as numeric character. Therefore, the path name of the new node should be assigned as an instance number by incrementing the number. The management system cannot specify the identifier of the new node.

All nodes for the *Framework* Object sub-tree are explained in Table 6.2.

Table 6.2 Framework sub-tree Nodes

URI	**Add**	**Get**	**Replace**	**Delete**	**Type**	**Cardinality**	**Scope**	**Description**
Framework		X			node	1	P	Framework Root node. This node must have the node type of "org.osgi/1.0/FrameworkManagementObject", including the Management Object version.
Framework/**StartLevel**		X			node	1	P	Interior node that contains the values for StartLevel configuration. StartLevel service must be available on OSGi framework where Residential Management Tree is working.
Framework/StartLevel/ **RequestedStartLevel**		X	X		int	1	P	A leaf node used to configure the Framework's StartLevel. When this node value is replaced, StartLevel#setStartLevel with the specified value must be called. This value must be kept persistently.

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
Framework/StartLevel/ **ActiveStartLevel**		X			int	1	P	A leaf node that contains the Framework's current StartLevel. This node is read-only to get the Framework's StartLevel.
Framework/StartLevel/ **InitialBundleStartLevel**		X	X		int	1	P	A leaf node used to configure the initial bundle StartLevel. When this node value is replaced, StartLevel#setInitialStartLevel with the specified value must be called.
Framework/**FrameworkLifecycle**		X			node	1	P	Interior node that contains the values for Lifecycle control of the OSGi Framework.
Framework/FrameworkLifecycle/**Restart**		X	X		bool	1	P	A leaf node used to restart the OSGi Framework. This node is writable to set the restart command. If this node value is replaced with 'TRUE', the Framework sub-tree must restart the OSGi Framework. When commit() is called, Framework Object MUST fire the DmtEvent. The event SHOULD be fired by MountPoint#sendEvent(). The destructive operation and the event notification should be fired in a new thread. After the method returns, Framework Object restarts the framework. The topic of the event MUST be set to “info/dmtree/DmtEvent/DESTRUCTIVE_OPERATION”.
Framework/FrameworkLifecycle/**Shutdown**		X	X		bool	1	P	A leaf node used to shutdown the OSGi Framework. This node is writable to set the shutdown command. If this node value is replaced with 'TRUE', the Framework sub-tree must shutdown the OSGi Framework. When commit() is called, Framework Object MUST fire the DmtEvent before the framework shutdown operation. The event SHOULD be fired by MountPoint#sendEvent(). Framework Object shutdown the framework after the method returns. The destructive operation and the event notification should be fired in a new thread. The topic of the event MUST be set to “info/dmtree/DmtEvent/DESTRUCTIVE_OPERATION”.
Framework/FrameworkLifecycle/**Update**		X	X		bool	1	P	A leaf node used to update the OSGi Framework. This node is writable to set the update command. If this node value is replaced with 'TRUE', the Framework sub-tree must update the OSGi Framework. Framework Object MUST fire the DmtEvent

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
								before the framework update operation. The event SHOULD be fired by MountPoint#sendEvent(). The topic of the event MUST be set "info/dmtree/DmtEvent/DESTRUCTIVE_OPERATION". The destructive operation and the event notification should be fired in a new thread.
Framework/FrameworkLifecycle/**Ext**		X			node	0,1	P	Interior node that can contain values for user extensions.
Framework/**InstallBundle**		X			node	0,1	P	Interior node that contains the values for bundle installation.
Framework/InstallBundle/**<id>**	X	X		X	node	0..*	D	Interior node that represents a bundle which will be installed to the OSGi Framework. This node is created for each bundle to be installed. When a transaction is committed, the Framework Object attempts to install the bundle specified under this node. See details of the installation operation described in the ***Location*** definition. This node must be dynamically deleted if the installation succeeds and must be stored persistently unless it is deleted after installation succeeds. When multiple bundles are installed by just one commit(), the nodes with lower <id> will be installed before the nodes with higher <id>. The name must be a numerical string from 1. The range of the node name number must be from 1 to Long.MAX_VALUE.
Framework/InstallBundle/<id>/**Location**		X	X		string	1	A	A leaf node used to set the bundle location under which the specified bundle is installed. This node is writable because the node is used to install a bundle. If the ***Location*** is replaced by setNodeValue() method and the ***URL*** is empty String when commit() method is called, BundleContext#installBundle(*location*) must be called where *location* is the specified value in this node. When the transaction is committed where both this node ***Location*** and ***URL*** are replaced and commit() method is called, BundleContext#installBundle(*location*, *in*) must be called where *location* is the specified value of this node and *in* is an InputStream retrieved from

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
								the specified ***URL***. InstallBundle sub-tree must call installBundle() method only if the value of Location node is set by setNodeValue() method and called commit() method. Optionally, the value of URL node can be set too. Even if commit() is called, installBundle() method must not be called if Location is not set. If the installation of a bundle succeeds, the Framework Object must delete the ***Framework/InstallBundle/<id>*** node corresponding to the bundle and its descendants. If the installation of a bundle fails, the Framework Object will NOT retry the operation and must set the reason for operation failure to the ***Error*** node. Once the bundle install process fails, the <id> object is not checked when commit() method is called next time. If the ***Option*** is set to the value except default one, bundle start will be attempted after the installation. The default value of this node is an empty string.
Framework/InstallBundle/<id>/**URL**		X	X		string	1	A	A leaf node used to be set the bundle's jar file URL from which the specified bundle is installed. This node is writable because the node is used to install a bundle. The default value of this node is an empty string. See details of the installation operation described in the ***Location*** definition.
Framework/InstallBundle/<id>/**Option**		X	X		int	1	A	A leaf node that can be used to start the bundle immediately after its installation. When the node value is set to the value except default one, the bundle will be started with the option specified in the node. The possible node values are: -1 : default value -- only install without start 0 : no start options are specified -- the bundle will be started without options. 1 : start transient -- the bundle will be started with START_TRANSIENT option. 2 : start with activation policy -- the bundle will be started with START_ACTIVATION_POLICY option. Both START_ACTIVATION_POLICY and START_TRANSIENT can be used simultaneously to start the bundle with activation policy and transiently (The node value is 3).
Framework/InstallBundle/<id>/		X			string	1	A	A leaf node used to represent the reason for an

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
Error								installation failure. This node will be created at the same time as the Framework/InstallBundle/<id> node creation. The Framework Object must set a value specifying the error reason to this node when bundle installation fails. The default value is the empty string.
Framework/InstallBundle/<id>/**Ext**		X			node	0,1	A	Interior node that can contain values for user extensions.
Framework/**BundleControl**		X			node	0,1	P	Interior node that contains the control functions for each bundle.
Framework/BundleControl/**<id>**		X			node	0..*	A	A node that represents a Bundle instance. This number must equal the bundle id plus 1. The bundle id is returned by Bundle#getBundleId(). For example, the name of <id> node which represents the system bundle instance is "1" because the number of system bundle id is "0". In case that TR-069 is used, the number "0" which is the name of <ID> node cannot be handled by ACS. That's why this RFC adopts this format. The name must be a numerical string. The range of the node name number must be from 1 to Long.MAX_VALUE.
Framework/BundleControl/<id>/**BundleStartLevel**		X	X		int	1	A	The configuration node for the StartLevel of the bundle. When the node value is gotten, this object must return the current StartLevel of the bundle. If this value is changed, StartLevel#setBundleStartLevel with the specified value must be called.
Framework/BundleControl/<id>/**Lifecycle**		X			node	1	A	This is a parent node for commands related to the bundle's life-cycle control. This sub-tree is described in detail later.
Framework/BundleControl/<id>/Lifecycle/**Operation**		X	X		string	1	A	A leaf node used to control the bundle's life-cycle. When this node value is replaced with the state described below, the BundleControl sub-tree must control the bundle life-cycle to the specified life-cycle operation. The BundleControl sub-tree must return an error if the specified life-cycle operation cannot be understood. This life-cycle operation must be one of the following: - START

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
								- STOP - UNINSTALL - RESOLVE The Framework Object does not have to retry the operation if life-cycle control fails, but the error status should be written in the ***Error***. The default value of this node is an empty string. When this node value is an empty string which means no operation.
Framework/BundleControl/ <id>/Lifecycle/**BundleUpdate**		X	X		string	1	A	A leaf node used to update the bundle. This node is writable to set the update command. When this node value is replaced with an URL string, Bundle#update(InputStream) must be called where InputStream is the specified URL. If the specified URL is an empty string, Bundle#update() must be called so that the specified bundle is updated with the jar-file indicated by the BundleLocation. If the update fails, the Framework object does not have to retry the operation. If Operation and BundleUpdate node values are replaced during same transaction, BundleControl sub-tree must put update operation ahead of life cycle operation attributed by changing Operation Node value. The default value of this node is an empty string.
Framework/BundleControl/ <id>/Lifecycle/ **Option**		X	X		int	1	A	A leaf node used to set start or stop options. A Protocol Adapter (PA) (or local manager) should set not only Operation but also this node in the same transactional session. After called commit() method, Lifecycle sub-tree must call Bundle#start(int option) or Bundle#stop(int option). The value of this node must be kept beyond the data plug-in bundle reboot. Option node contains integer format data as below. 0:default 1:START_TRANSIENT or STOP_TRANSIENT 2:START_ACTIVATION_POLICY 3:START_ACTIVATION_POLICY | START_TRANSIENT (Example 1) If a Protocol Adapter wants to start a Bundle transiently, the Protocol Adapter should set “START” to Operation node and “1” to Option node. (Example 2)If a Protocol Adapter wants to start a

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
								bundle not transiently, the Protocol Adapter should set Operation node to “START” and Option node to “0”. (Example 3)If a Protocol adapter wants to stop a bundle transiently, it should set Operation node to “STOP” and this node to “1”.
Framework/BundleControl/<id>/Lifecycle/**Error**		X			string	1	A	This node holds the latest operation's result which is conducted through the Framework/BundleControl/<id>/Lifecycle sub-tree, so that the Management System can derive the result of the bundle life-cycle operation. If the operation succeeds, the empty String will be set to this node. On the other hand, the error message will be set to this node when the operation fails. The default value is the empty string.
Framework/BundleControl/<id>/Lifecycle/**Ext**		X			node	0,1	A	Interior node that can contain values for vendor extensions.
Framework/BundleControl/<id>/**Ext**		X			node	0,1	A	Interior node that can contain values for vendor extensions.
Framework/**BundleControlNumberOfEntries**		X			unsigned_int	1	A	A leaf node that represents the number of ./$/OSGi/<instance_id>/Framework/BundleControl/<id> nodes.
Framework/**Ext**		X			node	0,1	P	Interior node that can contain values for vendor extensions.

6.4 BundleState Object

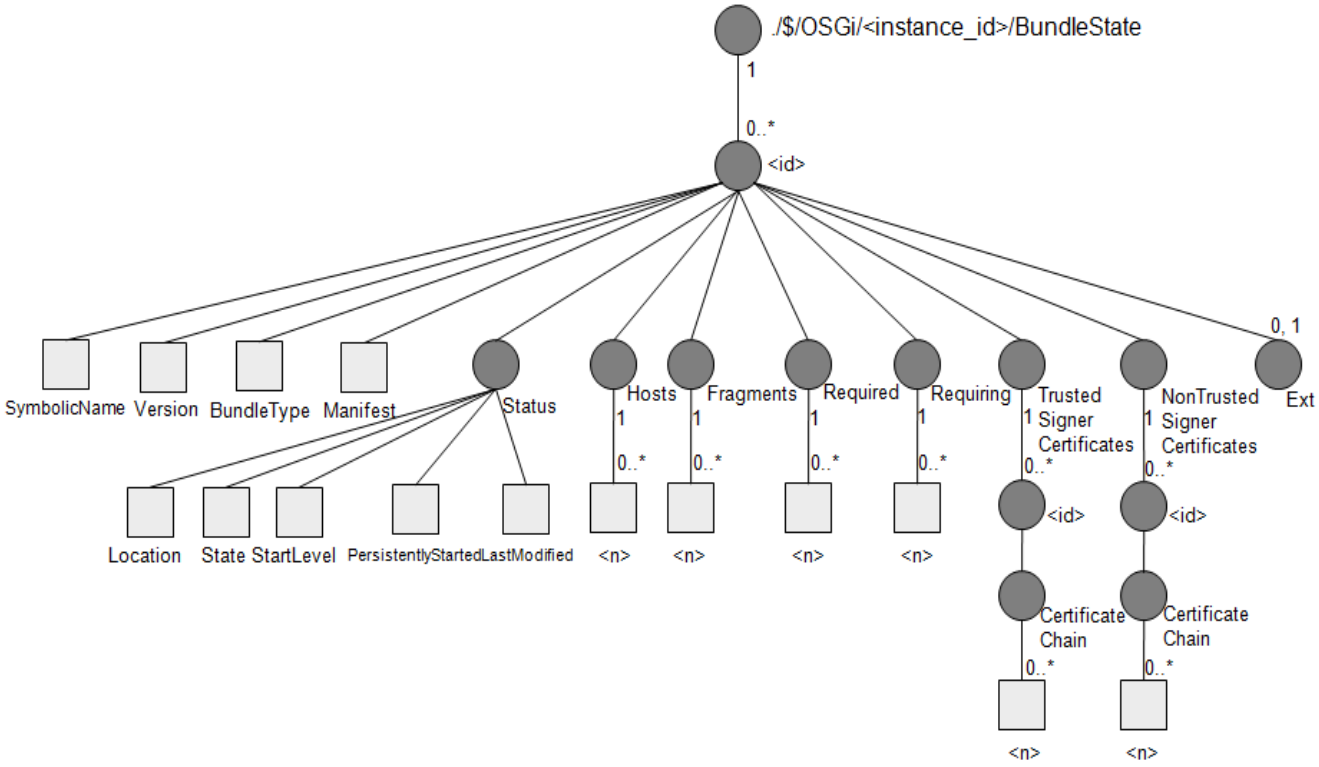


Fig.3 BundleState sub-tree

Figure 3 shows the *BundleState* sub-tree architecture. This sub-tree is used to obtain the information of a bundle from the OSGi framework. When a bundle is installed on the OSGi framework, a new node of *./$/OSGi/<instance_id>/BundleState/<id>* is automatically added and *<id>* is incremented. In other words, *<id>* is equivalent to *Bundle#getBundleId() plus 1*. The identifiers including SymbolicName and Version are used to identify a bundle on the OSGi framework. *BundleType*, an integer value, represents the type of the bundle. *Manifest* represents the Manifest header by String value. *StartLevel* shows the bundle's start level on this OSGi framework. *State* shows the bundle state; Installed, Resolved, Starting, Active, Stopping and Uninstalled. *Location,* a String value, represents the BundleLocation of the bundle. *Fragments, Hosts, Required, Requiring sub-trees* contain the corresponding bundle identifiers. *TrustedSignerCertificates* and *NonTrustedSignerCertificates sub-trees* contain the Certificate chains. The one is only trusted certificates, the other is only NOT trusted certificates by the framework.

All nodes for the BundleState Object sub-tree are explained in Table 6.3.

Table 6.3 BundleState sub-tree Nodes

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
BundleState		X			node	1	P	This is a parent node for status information of the bundle. This sub-tree is detailed later. This node must have the node type of "org.osgi/1.0/BundleStateManagementObject" including the Management Object version.
BundleState/**<id>**		X			node	0..*	A	A node that represents a Bundle instance. This number must equal the bundle id plus 1. The bundle id is returned by Bundle#getBundleId(). For example, the name of <id> node which represents the system bundle instance is "1" because the number of system bundle id is "0". In case that TR-069 is used, the number "0" which is the name of <ID> node cannot be handled by ACS. That's why this RFC adopts this format. The name must be a numerical string. The range of the node name number must be from 1 to Long.MAX_VALUE.
BundleState/<id>/ **SymbolicName**		X			string	1	A	The Bundle-SymbolicName of the bundle which is specified in Bundle-SymbolicName manifest header. If Bundle#getSymbolicName() returns "null", an empty string is set as the value.
BundleState/<id>/ **Version**		X			string	1	A	The version of the bundle.
BundleState/<id>/ **BundleType**		X			int	1	A	A node indicating the type of the bundle. The node value must be equivalent to the value of PackageAdmin#getBundleType().
BundleState/<id>/ **Manifest**		X			string	1	A	This node is a leaf node that contains Manifest headers as String. The node implementation must get Dictionary object by Bundle#getHeaders(). For converting the Dictionary object to String, this node must have the value adhering to the jar file manifest specification[10]. The order of appearance depends on its implementation. Example:"<key>:<value><newline character><key>:<value><newline character>..."
BundleState/<id>/**Location**		X			string	1	A	The *BundleLocation* of the bundle.
BundleState/<id>/ **Status**		X			node	1	A	This node is the parent of the nodes that represent the status of the bundle.

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
BundleState/<id>/ Status/**State**		X			int	1	A	The state of the bundle as returned by Bundle#getState(). This state is one of the following: 2 – Installed 4 – Resolved 8 – Starting 16 – Stopping 32 – Active
BundleState/<id>/ Status/**StartLevel**		X			int	1	A	The StartLevel of the bundle. Because this value is a read-only node in this sub-tree, user must use the Framework Object to configure the StartLevel.
BundleState/<id>/ Status/**PersistentlyStarted**		X			bool	1	A	The status of the bundle at the last shutdown of the OSGi Framework. This node returns whether the specified bundle's autostart setting indicates the bundle must be started. In case the bundle must be started, the value must be TRUE. Otherwise, the value must be FALSE.
BundleState/<id>/ Status/**LastModified**		X			date_time	1	A	The latest time at which the bundle has been modified.
BundleState/<id>/**Hosts**		X			list_node	1	A	The sub-tree under this interior node contains the list of the bundles hosting this bundle. When this node has the child nodes, each of which represents the bundle hosting this bundle. Even if there is no host bundle, this node must exist.
BundleState/<id>/Hosts/**<n>**		X			long	0..*	A	A leaf node that represents the bundle ID of the host bundle. If there is no host bundle, this node must not exist. The name must be a continuous numerical string from 1. The value is corresponding bundle ID.
BundleState/<id>/ **Fragments**		X			list_node	1	A	The sub-tree under this interior node contains the list of fragment bundles. When this node has the child nodes, each of which represents the fragment bundles. Even if there is no fragment bundle, this node must exist.
BundleState/<id>/Fragments /**<n>**		X			long	0..*	A	A leaf node that represents the bundle ID of the fragment bundle. If there is no fragment bundle, this node must not exist. The name must be a continuous numerical string from 1. The value is corresponding bundle ID.
BundleState/<id>/		X			list_no	1	A	The sub-tree under this interior node contains the

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
Required					de			list of the bundles requiring this bundle (i.e. the bundle is required by those bundle). When this node has the child nodes, each of which represents the bundles requiring this bundle. Even if there is no required bundle, this node must exist.
BundleState/<id>/Required/**<n>**		X			long	0..*	A	A leaf node that represents the bundle ID of the required bundle. If there is no required bundle, this node must not exist. The name must be a continuous numerical string from 1. The value is corresponding bundle ID.
BundleState/<id>/ **Requiring**		X			list_node	1	A	The sub-tree under this interior node contains the list of the bundles required by this bundle (i.e. the bundle requiring those bundles). When this node has the child nodes, each of which represents the bundles required by this bundle. Even if there is no requiring bundle, this node must exist.
BundleState/<id>/Requiring/ **<n>**		X			long	0..*	A	A leaf node that represents the bundle ID of the requiring bundle. If there is no requiring bundle, this node must not exist. The name must be a continuous numerical string from 1. The value is corresponding bundle ID.
BundleState/<id>/ **TrustedSignerCertificates**		X			node	1	A	Interior node that contains the trusted signer certificates of the bundle. Even if there is no signature, this node must exist.
BundleState/<id>/ TrustedSignerCertificates/**<id>**		X			node	0..*	A	A interior node that represents the X.509 Certificate chain. This node name must be a continuous numerical string from 1.
BundleState/<id>/ TrustedSignerCertificates/<id>/**CertificateChain**		X			list_node	1	A	The sub-tree under this interior node contains the list of the Distinguished Names in trusted signers of the bundle. When this node has the child nodes, each of which represents the Distinguished Names obtained from X509Certificate.
BundleState/<id>/ TrustedSignerCertificates/<id>/CertificateChain/**<n>**		X			string	0..*	A	A leaf node that represents the entries of the list of the Distinguished Names obtained from X.509Certificate, only trusted by the framework. Those can be gotten by Bundle#getSignerCertificates(SIGNERS_TRUSTED). The order of X509Certificate is the same order as the List contained in the Map which returned by Bundle#getSignerCertificates(). The name must be a continuous numerical string

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
								from 1.
BundleState/<id>/ **NonTrustedSignerCertificates**		X			node	1	A	Interior node that contains the NOT trusted signer certificates of the bundle. Even if there is no signature, this node must exist.
BundleState/<id>/ NonTrustedSignerCertificates/**<id>**		X			node	0..*	A	A interior node that represents the X.509 Certificate chain. This node name must be a continuous numerical string from 1.
BundleState/<id>/ NonTrustedSignerCertificates/<id>/**CertificateChain**		X			list_node	1	A	The sub-tree under this interior node contains the list of the Distinguished Names in NOT trusted signers of the bundle. When this node has the child nodes, each of which represents the Distinguished Names obtained from X509Certificate.
BundleState/<id>/ NonTrustedSignerCertificates/<id>/CertificateChain/**<n>**		X			string	0..*	A	A leaf node that represents the entries of the list of the Distinguished Names obtained from X.509Certificate, only NOT trusted by the framework. Those can be gotten by Bundle#getSignerCertificates(SIGNERS_ALL) except ones in TrustedSignerCertificates node. The order of X509Certificate is the same order as the List contained in the Map which returned by Bundle#getSignerCertificates(). The name must be a continuous numerical string from 1.
BundleState/<id>/ **Ext**		X			node	0,1	A	Interior node that can contain values for vendor extension.

6.5 PackageState Object

The *PackageState* Object is a managed object that allows the package information to be derived. This object can be used to retrieve package dependencies between bundles.

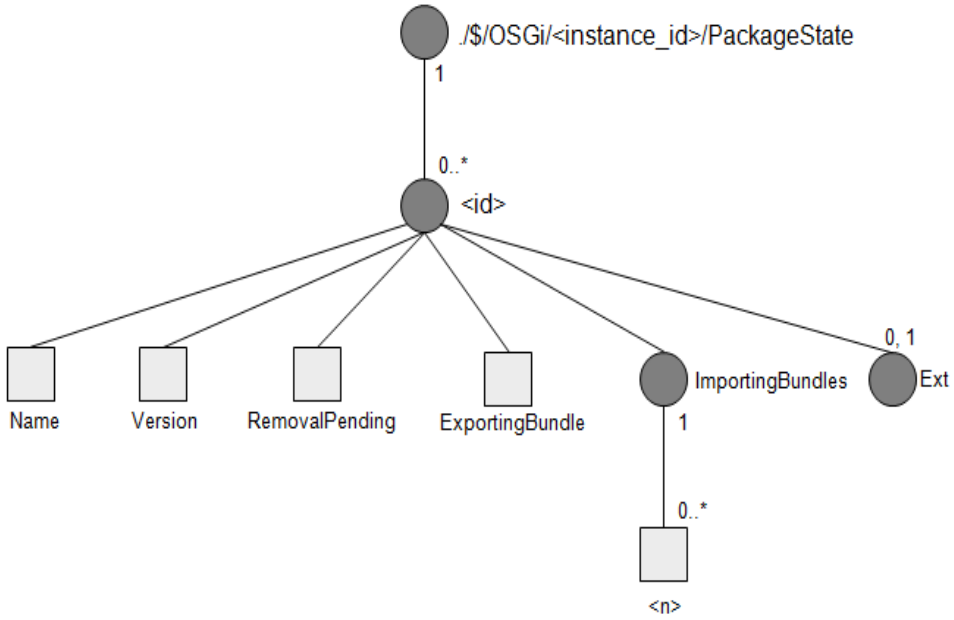


Fig.4 PackageState Object

Figure 4 shows the over all architecture of the *PackageState* object. The *./$/OSGi/<instance_id>/PackageState/<id>* node is created for an individual package existing on the OSGi framework. This node represents a package's information including bundle dependencies. The pairs between each package and the corresponding <id> must be kept persistently as long as the package is exported from the same bundle, which means the same bundle ID, beyond restart of the OSGi Framework.

The Name node contains a qualified package name, and the Version node contains the version number of the package. The ExportingBundle node shows a bundle identifier exporting the package. On the other hand, the ImportingBundles sub-tree shows the bundles importing the package. These nodes can be used to get information on packages shared between bundles.

The *PackageState* Object may support only readable session because it does not contain any writable node.

All nodes for the *PackageState* Object sub-tree are explained in Table 6.4.

Table 6.4 Pack*ageState sub*-tree Nodes

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
PackageState		X			node	1	P	PackageState Root node containing package information existing on the OSGi Framework. The children of this node must represent the actual package status when this sub-tree is accessed. This node must have the node type of "org.osgi/1.0/PackageStateManagementObject " including the Management Object version.
PackageState/**<id>**		X			node	0..*	A	A node that represents the Package instance. This is equivalent to org.osgi.service.packageadmin.ExportedPackage object. The name must be a numerical string from 1. The range of the node name number must be from 1 to Long.MAX_VALUE.
PackageState/<id>/**Name**		X			string	1	A	The qualified name of the package.
PackageState/<id>/**Version**		X			string	1	A	The version of the package.
PackageState/<id>/ **RemovalPending**		X			bool	1	A	A leaf node that represents the removal status of the package. If a bundle exporting the package has already been uninstalled or updated but the package is still used, this node must be TRUE.
PackageState/<id>/ **ExportingBundle**		X			long	1	A	A leaf node that represents the bundle ID of the exporting bundle. The value is a bundle ID.
PackageState/<id>/ **ImportingBundles**		X			list_node	1	A	The sub-tree under this interior node contains the list of the ids of bundles importing the package. When this node has the child nodes, each of which represents the ids of bundles importing the package. Even if there is no importing package, this node must exist.
PackageState/<id>/ ImportingBundles/**<n>**		X			long	0..*	A	A leaf node that represents the bundle ID of the importing bundle. If there is no importing bundle, this node must not exist. The name must be a continuous numerical string from 1. The value is corresponding bundle ID.
PackageState/<id>/**Ext**		X			node	0,1	A	Interior node that can contain values for vendor extension.

6.6 ServiceState Object

The Service*State* Object is a managed object that allows the service information to be derived. This object can be used to retrieve service dependencies between bundles.

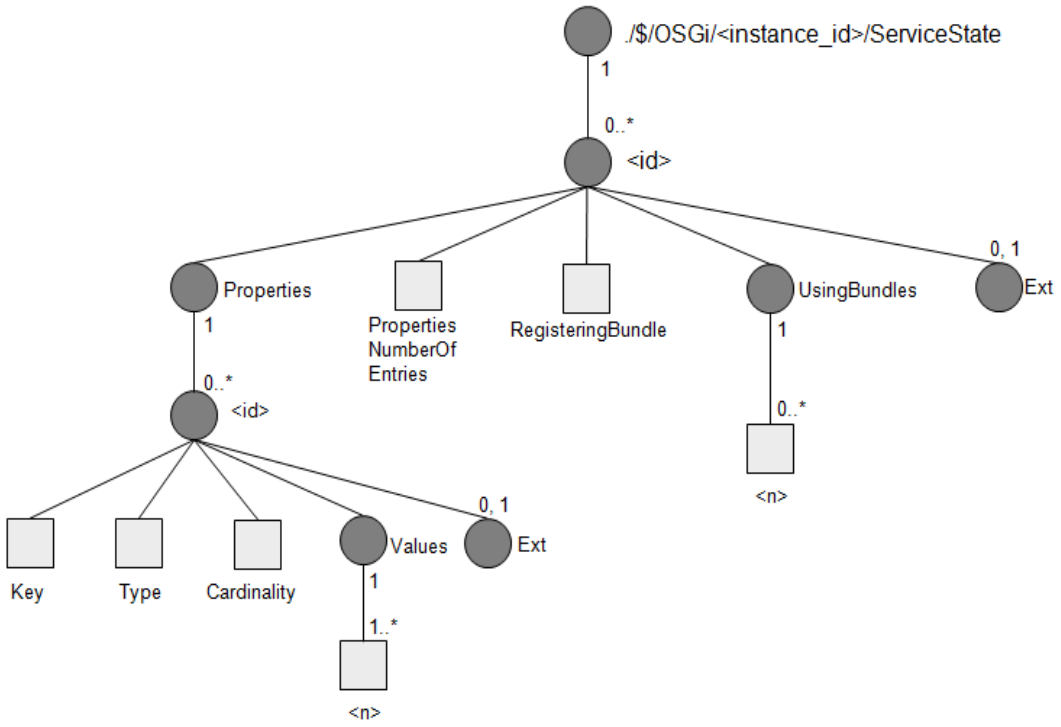


Fig.5 ServiceState Object

Figure 5 shows the over all architecture of the *ServiceState* object. The *./$/OSGi/<instance_id>/ServiceState/<id>* node is created for an individual service instance existing on the OSGi framework. This node represents a service's information including registering bundle and using bundles. Registering a service to the OSGi framework automatically adds a new *Service* instance under the *./$/OSGi/<instance_id>/ServiceState* node with incrementation of *<id>*. The name must be a continuous numerical string from 1. The name of *<id>* doesn't correspond to the *service.id* of the service instance in the OSGi framework.

The *Properties* node contains service properties of the service, which include the interface names implemented by the service. The *Properties* node must contain all service properties which consist of string, boolean or numeric data types including single-dimension arrays or vectors. However non-serializable data types can be discarded from the Properties sub-tree, since these types of properties are difficult to be represented in object trees. See 6.5.1 for the detail of it. The *RegisteringBundle* node shows the id of the registering bundle. On the other hand, the *UsingBundles* sub-tree shows bundles using the service. These nodes can be used to get information on the relationships between registering bundle and using bundles.

In order to get service interface names of a service, you should first find the “<id>” to obtain the node *./$/OSGi/<instance_id>/ServiceState/<id>/Properties/<id>/Key* whose value is “objectClass”. After that, the service interface name is obtained from *./$/OSGi/<instance_id>/ServiceState/<id>/Properties/<id>/Values/<n>.*

The *ServiceState* Object may support only readable session because it does not contain any writable node.

All nodes for the Service*State* Object sub-tree are explained in Table 6.5.

Table 6.5 ServiceState sub-tree Nodes

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
ServiceState		X			node	1	P	ServiceState Root node containing service information existing on the OSGi Framework. The children of this node must represent the actual service status when this sub-tree is accessed. This node must have the node type of "org.osgi/1.0/ServiceStateManagementObject" including the Management Object version.
ServiceState/**<id>**		X			node	0..*	A	A node that represents the Service instance. The name must be a numerical string from 1. The range of the node name number must be from 1 to Long.MAX_VALUE.
ServiceState/<id>/ **Properties**		X			node	1	A	Holds the <id> nodes that contain the key and values for the service properties.
ServiceState/<id>/ Properties/**<id>**		X			node	0..*	A	A interior node that represents the service property instance of this sub-tree. This node name must be numerical string from 1. The range of the node name number must be from 1 to Long.MAX_VALUE. The node holds the key and value of an entry in the service property Dictionary. The value is defined by its type, cardinality and value node.
ServiceState/<id>/ Properties/<id>/**Key**		X			string	1	A	A leaf node that represents the key of the service property.
ServiceState/<id>/ Properties/<id>/**Type**		X			string	1	A	List of types of the properties' values. See *Service Property Dictionary nodes* section.
ServiceState/<id>/ Properties/<id>/**Cardinality**		X			string	1	A	Cardinality of the property. See *Service Property Dictionary nodes* section. The value is either: scalar, collection or array.
ServiceState/<id>/ Properties/<id>/**Values**		X			list_node	1	A	The sub-tree under this interior node contains the list of the actual values; this node has the child nodes, each of which represents the actual values.
ServiceState/<id>/ Properties/<id>/Values/**<n>**		X			string bin int long bool	0..*	A	A leaf node that contains a value which is defined by the ***Type*** node in this sub-tree. When the value is empty array or collection, this node is not cleated. The name must be a continuous numerical string from 1.

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
					float			See *Service Property Dictionary nodes* section.
ServiceState/<id>/Properties/<id>/**Ext**		X			node	0,1	A	Interior node that can contain values for vendor extension.
ServiceState/<id>/ **PropertiesNumberOfEntries**		X			unsigned_int	1	A	A leaf node that represents the number of <id> nodes.
ServiceState/<id>/ **RegisteringBundle**		X			long	1	A	A leaf node that represents bundle ID of the registering bundle. The value is a bundle ID.
ServiceState/<id>/ **UsingBundles**		X			list_node	1	A	The sub-tree under this interior node contains the list of the ids of bundles using the service. When this node has the child nodes, each of which represents the ids of bundles using the service. Even if there is no using bundle, this node must exist.
ServiceState/<id>/ UsingBundles/**<n>**		X			long	0..*	A	A leaf node that represents the bundle ID of the using bundle. If there is no using bundle, this node must not exist. The name must be a continuous numerical string from 1. The value is corresponding bundle ID.
ServiceState/<id>/**Ext**		X			node	0,1	A	Interior node that can contain values for vendor extension.

6.6.1 Service Property Dictionary nodes

The service property Dictionary consists of key-value pairs. The service property Dictionary is mapped to a sub-tree. The URI for a service property item is the following:

./$/OSGi/<instance_id>/ServiceState/<id>/Properties/<id>

These sub-nodes are:

- Key – Represents the key of the service property.
- Type – Type contains only one Java type name of Values such as *java.lang.Float, float, java.lang.String*, char, etc.
- Cardinality – Defines if the value is a scalar, an array, or a collection. It can take the following values:
  - scalar – For simple, unstructured values, like a string or a byte[].
  - array – When the value is a Java array (but not byte[]).

- collection – When the value must be a Java Collection object.

- Values – If Cardinality is array, Values contains the list of values of the service property. Otherwise if Cardinality is collection, Values contains the list of values of the service property, where all values have the same Java type. That is the limitation because it would be too complex to handle different Java types in Collection's values. Otherwise, Values contains only one value.

The actual value (Values child nodes) is mapped to a DmtData type if possible. If this mapping is not possible, the node must be a string node and the Java class of the given type must be able to parse the value in a constructor.

[Remarks] The mapping of data types between DMT Admin and TR-069 are described in RFC149.

6.7 Filters Object

The Filters Object is a managed object that searches the nodes in a tree that correspond to the filter expression. This Filter Object is a generic mechanism for the whole management tree below the *./$/OSGi* node although Filters sub tree is located under the *./$/OSGi/<instance_id>* node. In other words, the Filter Object plugin must be able to filter all nodes in the tree except Filters Object itself.

The Filters Object can be used to group bundles, packages, services and other information in the tree. Since the filter string set in the *Filter* node has no restriction in terms of its usage policy, users can use this function in accordance with their needs.

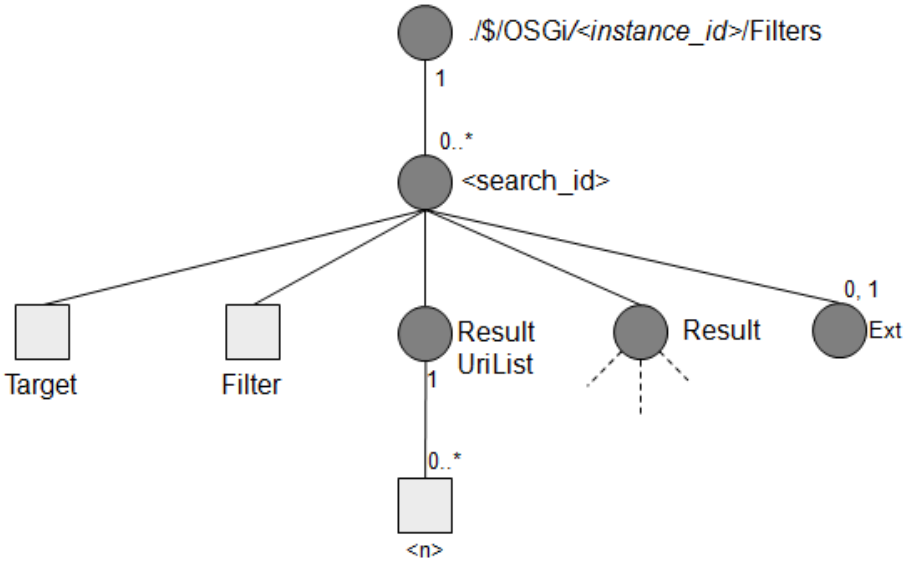


Fig.6 Filters Object

The *./$/OSGi/<instance_id>/Filters* object is used to search nodes by filtering values or names of nodes located under the sub-tree specified by *./$/OSGi/<instance_id>/Filters/<search_id>/Target*. At first, the user needs to create the *./$/OSGi/<instance_id>/Filters/<search_id>* node by incrementing the *<search_id>* number, which should be a numeric character string as demanded by the TR-069 protocol. Then the user sets the desired partial path in the *./$/OSGi/<instance_id>/Filters/<search_id>/Target* node, that specifies the sub-tree required to provide information as the result, and sets an appropriate filter string in *./$/OSGi/<instance_id>/Filters/<search_id>/Filter.* When *./$/OSGi/<instance_id>/Filters/<search_id>/ResultUriList* node is accessed, the Filter Object plugin must search all sub-trees that match the filter specified in *./$/OSGi/<instance_id>/Filters/<search_id>/Filter*. The scope of this search is not any nodes in the sub-tree under *Target* but only any child interior nodes of the *Target* , which has the properties that matches the filter. Here, the child leaf nodes and the child interior nodes which represents list are considered as the properties of the interior node, while the child interior nodes which does not represent list are not. Then, *./$/OSGi/<instance_id>/Filters/<search_id>/ResultUriList* node contains the list of the absolute paths of the matched child interior nodes from the $ node.

When either of *./$/OSGi/<instance_id>/Filters/<search_id>/Result* node is accessed, the search must be done and *./$/OSGi/<instance_id>/Filters/<search_id>/ResultUriList* node must be set as described above and then the copied subtree under the matched child interior nodes must be aligned under the *./$/OSGi/<instance_id>/Filters/<search_id>/Result* node with the absolute path from the *$* node. Remark that the copied Subtree must be the one not when the *./$/OSGi/<instance_id>/Filters/<search_id>/Target* node or the *./$/OSGi/<instance_id>/Filters/<search_id>/Filter* node is set, but when the *./$/OSGi/<instance_id>/Filters/<search_id>/Result* node is accessed.

The <*search_id*> node is a dynamic node which means that the path name of the new node should be assigned as an instance number by incrementing the largest existing number as demanded by TR-069. Therefore, the node name is defined as numeric string in the Residential Management Tree.

All nodes for the Filters Object sub-tree are explained in Table 6.5.

Table 6.5 Filters Object sub-tree Nodes

URI	**Add**	**Get**	**Replace**	**Delete**	**Type**	**Cardinality**	**Scope**	**Description**
Filters		X			node	0,1	P	The root node of Filters Object. This node must have the node type of "org.osgi/1.0/FiltersManagementObject" including the Management Object version.
Filters/**<search_id>**	X	X		X	node	0..*	D	Represents a filter search request. This node is created to set the filter expression and to get the sub-trees, which match with the filter expression. The ***<search_id>*** is assigned in ascending order to ***Filters*** instances when they are set. This is a unique ID of ***Filters*** instance and must be kept persistently. The name must be a numerical string from 1. The range of the node name number must be from 1 to Long.MAX_VALUE.
Filters/<search_id>/**Target**		X	X		string	1	A	Partial path to the sub-tree under which ***Filter*** is going to be matched. This must be the absolute path of the top node name of the Device Management Tree.. The default value of this node is an empty string, which means no filtering must be done. This value must be kept persistently.
Filters/<search_id>/**Filter**		X	X		string	1	A	Contains the filtering expression. The filter must be given in the OSGi Filter format. See *Filter and target Expression* section. An empty string indicates that no filtering must be done. An empty string is the default value for this node.
Filters/<search_id>/**Result**		X			node	1	A	Matched sub-trees are stored under the ***Result*** node. Any children of this node must be contained as as a read-only snapshot. It will be created only when Result sub-tree or ResultUriList is accessed at the first time and never changed. The result must be only those sub-trees that satisfy the specified ***Filter*** node. And the filtered sub-trees must include all nodes located under

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
								the sub-tree specified by the ***Target*** node. See *Filter and Target Expression* section. The nodes under Result node must be read only.
Filters/<search_id>/**ResultUriList**		X			list_node	1	A	The sub-tree under this interior node contains the list of the matched node URIs. This node has the child nodes, each of which represents the matched node URIs. Even if there is no matched URI, this node must exist.
Filters/<search_id>/ResultUriList/**<n>**		X			node_uri	1	A	A leaf node that represents the matched interior node path which is contained as a snapshot. The snapshot will be created only when Result sub-tree or ResultUriList is accessed at the first time. If there is no matched path, this node must not exist. The name must be a continuous numerical string from 1. The value is corresponding node path. See *Filter and Target Expression* section.
Filters/<search_id>/**Ext**		X			node	0,1	A	Interior node that can contain values for vendor extension.

6.7.1 Filter and Target Expression

The expression of *./$/OSGi/<instance_id>/Filters/<search_id>/Target* must be a partial path from the "./$" node to the top node of the target sub-tree to be filtered and must end with "/", which implies that a partial path must indicate an interior node. The characters "*" and "-" must be allowed to be used in the path as a wild-card; "*" indicates a wild-carded node for single level, while "-" indicates multiple levels in a node path respectively. Multiple wild-card can be used in a node path. There are some rules using "*" and "-" in this RFC:

1. When "-" is included in the Target, all possible paths of the Target must be matched.
   This is an example subtree:
   ./Example/B/A/A/A/b
   ./Example/B/A/A/A/A/c
   When the following Target is given,
   ./Example/-/A/
   the matched path of the Target is as below.
   ./Example/B/A/
   ./Example/B/A/A/
   ./Example/B/A/A/A/
   ./Example/B/A/A/A/A/

2. The "-" must not be use at end of the path.

3. The Target path must start with ./$/OSGi/<instance_id>/<child_node_name> where <child_node_name> must not be Filters.

<Examples>
The valid Target path:
./$/OSGi/1/BundleState/
./$/OSGi/1/ServiceState/*/Properties/

The invalid Target path:
./$/OSGi/1/Filters/
./$/OSGi/1/
./$/OSGi/*/BundleState/
./$/-/PackageState/

If an inappropriate string, such as a string including an incorrect format path, is specified in *./$/OSGi/<instance_id>/Filters/<search_id>/Target*, the Filters Object should throw a DmtException with an error code COMMAND_FAILED to the caller and should keep the old Target node value. In case that the path which does not exist in the current subtrees is specified in *./$/OSGi/<instance_id>/Filters/<search_id>/Target*, the Result subtree must have no children.

The filter expression of *./$/OSGi/<instance_id>/Filters/<search_id>/Filter* should be a LDAP search filter (RFC 1960. See Filter for a description of the filter string syntax.). This expression is equivalent to the OSGi Filter format.

The key string in a filter must be only name of leaf nodes or list sub-tree nodes, in other words, the nodes are property of interior node. The node names which are not valid keys in the LDAP filter cannot be matched. The Filters Object must search properties of the interior nodes which are children of ./$/OSGi/<instance_id>/Filters/<search_id>/Target. Therefore it must not contain the node path separator "/" which is not escaped by backslash. If the indicated node does not exist, the Filters Object must not create sub-trees under the *Result* node and *ResultUriList* node. A wild-card must not be allowed to be used in a key string. Both interior node which has leaf child nodes as list and leaf node can be specified as the key string of a filter. The “*” can be set to the Filter node. When “*” is set to the Filter node, all of nodes in the scope much the filter.

If an interior node which has the node type of "org.osgi/1.0/ListSubtree" is specified as the key string, node values of child nodes of the specified interior node must be recognized as the target values to be matched. Other interior node is out of scope of searches. On the other hand, if a leaf node is specified, the value of the specified leaf node must be recognized as the target value.

In a filter expression, types of value should be ignored because the Filter Object must recognize the filter strings as String. Therefore, the filter matching operation will translate each leaf node value with the help of DmtData#toString() method.

Filters object must contain the result of search as a snapshot in *./$/OSGi/<instance_id>/Filters/<search_id>/Result* sub-tree and *./$/OSGi/<instance_id>/Filters/<search_id>/ResultUriList*. It means that Result sub-tree and ResultUriList node do not always have actual status of the sub-tree. The snapshot will be created when *./$/OSGi/<instance_id>/Filters/<search_id>/Result* sub-tree or *./$/OSGi/<instance_id>/Filters/<search_id>/ResultUriList* is accessed at the first time. The snapshot has the time out which is set in system property.

residentialmanagement.osgi.filters.result.timeout

The unit of it is second. The default value depends on the implementation. The Filters object must remove the <serch_id> object when the snapshot times out. The snapshot will be removed when deleteNode() method is called for".../Filters/<search_id>". In addition, all of <search_id> sub-tree should not be persistent beyond the restrart of OSGi Framerok or Filters Management Object.

The Filters Object searches nodes that satisfy the specified keys and values in *./$/OSGi/<instance_id>/Filters/<search_id>/Filter* among the child nodes of the node specified by *./$/OSGi/<instance_id>/Filters/<search_id>/Target*, and must create the *./$/OSGi/<instance_id>/Filters/<search_id>/ResultUriList* and the *./$/OSGi/<instance_id>/Filters/<search_id>/Result* sub-tree. The absolute node path from the $ node must be created with the actual node name of the wild-card appearing in the node path specified by *./$/OSGi/<instance_id>/Filters/<search_id>/Target*.

The sub-trees under the *./$/OSGi/<instance_id>/Filters/<search_id>/Result* and the *./$/OSGi/<instance_id>/Filters/<search_id>/ResultUriList* must be read-only in order to keep consistency among data-plugins related to the filter search. The Filter Object must prevent attempts to access the sub-tree to change node value or properties, and must throw DmtException DmtException with an error code COMMAND_NOT_ALLOWED to the caller.

After once the snapshot is created, it must be removed in case that the value of *./$/OSGi/<instance_id>/Filters/<search_id>/Target* or *./$/OSGi/<instance_id>/Filters/<search_id>/Filter* is replaced. After that, the new snapshot must be created when *./$/OSGi/<instance_id>/Filters/<search_id>/Result* sub-tree or *./$/OSGi/<instance_id>/Filters/<search_id>/ResultUriList* is accessed at the first time.

The node of ./$/OSGi/<instance_id> and its ancestors must represent only node names and parent-child relation. On the other hand, the node of ./$/OSGi/<instance_id>/<child_node_name> and its descendants which are matched result nodes must represent full meta info, for example, node type, meta data, value, and so on.
See the following example.

<Example>
We assume that following node exists in the BundleState Object.
./$/OSGi/1/BundleState/1/Status/StartLevel=2
The given Target:
./$/OSGi/1/BundleState/
The given Filter:
(StartLevel=2)

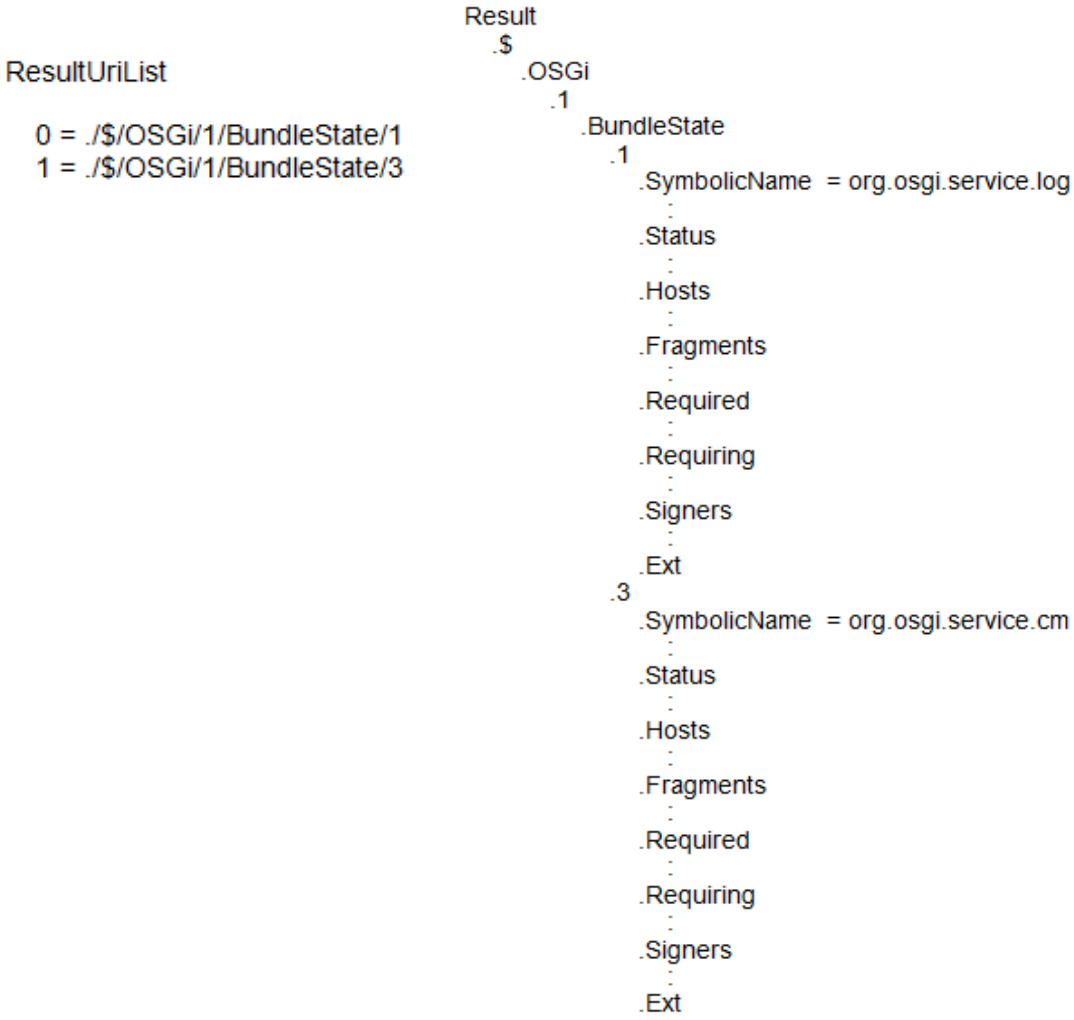
In this case, the result read-only sub-tree can be as below:
The nodes represent only node name and parent-child relation:
.../Filters/Result
.../Filters/Result/$
.../Filters/Result/$/OSGi
.../Filters/Result/$/OSGi/1
The nodes represent full meta info:
.../Filters/Result/$/OSGi/1/BundleState
.../Filters/Result/$/OSGi/1/BundleState/1
.../Filters/Result/$/OSGi/1/BundleState/1/SymbolicName
.../Filters/Result/$/OSGi/1/BundleState/1/Fragments
.../Filters/Result/$/OSGi/1/BundleState/1/Host
.../Filters/Result/$/OSGi/1/BundleState/1/Status
and the rest...

[REMARKS] The matched result nodes in snapshot must have meta node because the value included meta node can be changed (See RFC141).

The following expressions are examples of the Filter and Target:

[Case1] Simple Target and Filter usage

- Target: ./$/OSGi/1/BundleState/
- Filter: (SymbolicName=org.osgi.*)
    - Bundles whose Bundle-SymbolicName correspond to “org.osgi.*” are matched. The Result node contains corresponding sub-trees that have absolute node paths descending from $ node. A possible sub-tree under the Result node is described below ($ node should be changed to the actual node path depending on each execution environment):



[Case2] Filter for service properties filtering

- Target: ./$/OSGi/1/ServiceState/*/Properties/
- Filter: (&(Key=application)(Values=automation))
    - Services that have properties including “application” as a key and “automation” as a value are matched. The Result node contains corresponding sub-trees that have relative node paths descending from the .../Properties/<id> node. A possible sub-tree under the Result node is described below ($ node should be changed to the actual node path depending on each execution environment):

```
ResultUriList
    0 = ./$/OSGi/1/PackageState/27
    1 = ./$/OSGi/3/PackageState/11

Result
  .$
    .OSGi
      .1
        .PackageState
          .27
            .Name
            .Version
            .RemovalPending
            .ExportingBundle = 5
            .ImportingBundles
              .0 = 10
              .1 = 13
      .3
        .PackageState
          .11
            .Name
            .Version
            .RemovalPending
            .ExportingBundle = 5
            .ImportingBundles
              .0 = 10
              .1 = 22
```

6.8 BundleResources Object

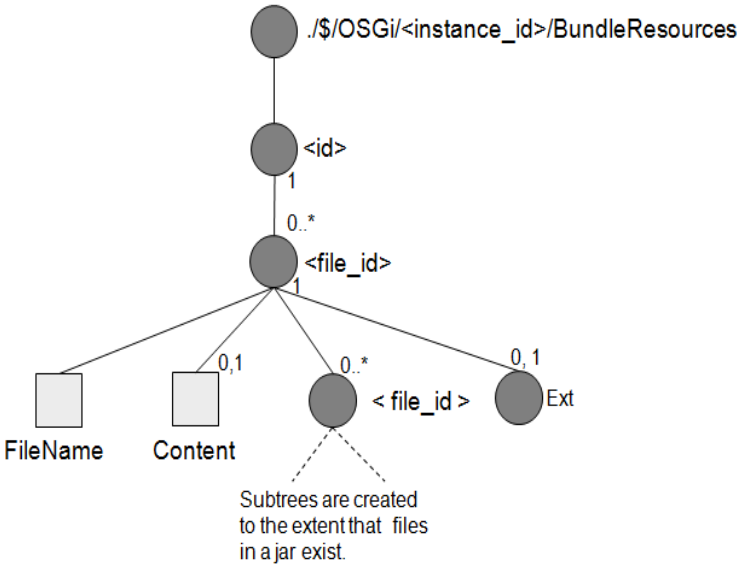


Fig.7 BundleResources sub-tree

The BundleResources sub-tree is optional and it is used to derive the bundle entries. The BundleResources sub-tree consists of interior and leaf nodes that correspond to actual directory and file entries, so that a remote manager can derive a bundle's resources by a simple operation. For example, suppose there is /META-INF/ MANIFEST.MF file in a jar file, The Bundle Resources sub-tree will specify the file with:

./$/OSGi/<instance_id>/BundleResources/<id>/<dir_id>/FileName = META-INF

./$/OSGi/<instance_id>/BundleResources/<id>/<dir_id>/<file_id>/FileName = MANIFEST.MF

./$/OSGi/<instance_id>/BundleResources/<id>/<dir_id>/<file_id>/Content = the content of the MANIFEST.MF file

The <file_id> node is automatically created and represents a bundle entry. The Content leaf node exists only for files and is missing for the directories. The <*id*> node must indicate the root directory of the jar file. The name of <file_id> should be put a mangled file name which is obtained by Uri#mangle(String). The reason why the file or directory name is not used as the node name is that the node name length might have its max size in Dmt Admin specification. See ,getMaxSegmentNameLength() method of info.dmtree.Uri. Then The file or directory name is placed in the leaf node *FileName*. On the other hand, *Content* node is also created automatically as a leaf node and represents the content of the file in a jar file. The content is encoded in base64 format. If the size of the content exceeds a limit which depends on the BundleResources Object implementation, the BundleResources Object should abort the reading of file content and should throw a DmtException with an error code COMMAND_FAILED.

Because the *BundleResources* sub-tree is created automatically based on the actual file architecture of the indicated bundle, this sub-tree must be read-only. A remote manager can not add, replace, delete and exec nodes in this sub-tree.

Note: Assume that adopted remote management protocol implements a method that retrieves data of all nodes under the specified node recursively (such as “GetParameterValues” RPC of TR-069 protocol). When a remote manager indicates *./$/OSGi/<instance_id>/BundleResources/<id>* node or ancestor nodes of it, heavy data transaction between the remote manager and client would occur because the BundleResources sub-tree contains all entries of the bundle. Therefore, when using such kind of method in an operation, a remote manager should carefully specify the node in terms of performance.

[REMARK] A considered alternative of the BundleResources sub-tree architecture is shown in section 7.

All nodes for the BundleResources sub-tree are explained in Table 6.6.

Table 6.6 Resources sub-tree Nodes

URI	**Add**	**Get**	**Replace**	**Delete**	**Type**	**Cardinality**	**Scope**	**Description**
BundleResources		X			node	0,1	P	This is a parent node for resources included in the bundle. This sub-tree is detailed later. This node must have the node type of "org.osgi/1.0/BundleResourcesManagementObject" including the Management Object version.
BundleResources/**<id>**		X			node	0..*	A	A node that represents a Bundle instance. This number must equal the bundle id plus 1. The bundle id is returned by Bundle#getBundleId(). For example, the name of <id> node which represents the system bundle instance is "1" because the number of system bundle id is "0". In case that TR-069 is used, the number "0" which is the name of <id> node cannot be handled by ACS. That's why this RFC adopts this format. The name must be a numerical string. The range of the node name number must be from 1 to Long.MAX_VALUE.
BundleResources/<id>/**<file_id>**		X			node	0..*	A	Represents the entry identifier of a file or a directory. This node is automatically created to include children with details about file or directory. The name of <file_id> should be put a mangled file name which is obtained by Uri#mangle(String).
BundleResources/<id>/<file_id>**FileName**		X			string	1	A	Represents the file or directory name.
BundleResources/<id>/<file_id>/**Content**		X			b64	0,1	A	The content of the file specified in the FileName node. This leaf node is automatically created for

URI	Add	Get	Replace	Delete	Type	Cardinality	Scope	Description
								an individual file in a bundle. The node value is encoded in base 64 format. The node exists only for files and is missing for directories.
BundleResources/<id>/**Ext**		X			node	0,1	A	Interior node that can contain values for vendor extension.

6.9 Limitations

The Residential Management Tree has several limitations due to the specification of a remote management protocol (TR-069) utilized with the DMT Admin service. The following section describes these limitations.

The Residential Management Tree events are not specified in this RFC. This task will be investigated as a future work.

6.9.1 Life-cycle control of the Framework

The life-cycle control of the OSGi Framework depends on the implementation of the framework on which the Residential Management Tree can run. This RFC doesn't specify the details about the implementation of the operations placed under *./$/OSGi/<instance_id>/Framework/FrameworkLifecycle/*.*

6.9.2 Service Properties Expression

The following list summarizes the restrictions and rules that should be obeyed in order to maintain a valid ServiceState Object.

- Complex data types, multiple-dimension arrays or vectors might be discarded from the Properties sub-tree, since these types of properties are difficult to be represented in the object tree if the data isn't serializable.

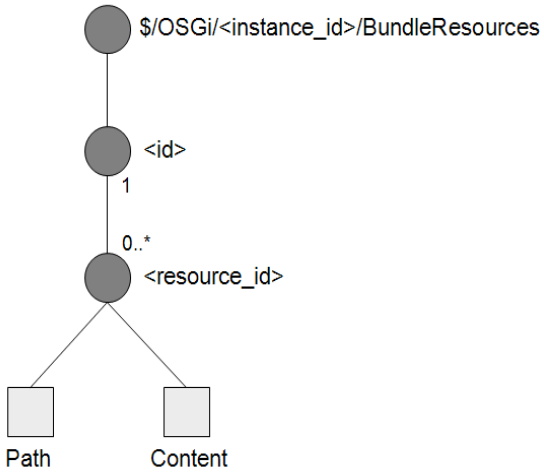
6.10 Future Work

In TR-069 protocol way, Management Server (ACS) cannot provide enough information to the Configuration Object and Policy Object which are defined in the Mobile Management tree. Therefore we will investigate this task as a future work.

7 Considered Alternatives

7.1 BundleResources sub-tree architecture

There is considerable alternative for designing the BundleResources sub-tree to retrieve resources inside the bundle jar file. The following architecture is another design of the BundleResources sub-tree, which was eventually discarded as the result of discussion.



The *<resource_id>* node is a dynamic node, which means the path name of the new node is automatically assigned as an instance number by the Bundles Object. In advance, the management system needs to create the *<resource_id>* node. Then it has to indicate the file path by setting the *Path* node parameters and can retrieve the contents of the specified file as the value of the *Content* node.

This architecture has pros and cons compared to the proposed architecture in Section 6.8.

First, this architecture enables a scalable implementation because the remote manager is able to decide which resources should be retrieved from bundles arbitrarily. The remote manager, therefore, can avoid heavy data traffic between remote manager and client, when an ancestor node of the Resources sub-tee is specified by the GetParameterValues RPC defined in the TR-069.

Secondly, the typical usage of the Resources sub-tree would be for diagnostics senarios; when some problems occur, a remote operator who needs to find the cause of the problem retrieves the content of the bundle JAR file by using this sub-tree.

But this architecture limits the diagnostics ability of a remote manager. The remote manager has to check many files by repetitively creating <resources_id> nodes until the cause of problem is detected. This situation prevents an effective diagnostics through the Resources sub-tree.

On the other hand, the proposed architecture in Section 6.8 provides a better diagnostics ability than this but the performance decreases in terms of data transactions when retrieving data recursively.

Consequently, the architecture proposed in Section 6.8 is chosen for the Resources sub-tree due to the effectiveness of the diagnostics ability, even though there is a risk of a performance drawback when a remote manager indicates *./$/OSGi/<instance_id>/BundleResources/<id>* node or ancestor nodes.

8 Security Considerations

All security requirements follow the DMT Admin specification.

9 Document Support

9.1 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]. OMA, Open Mobile Alliance. The mission of the Open Mobile Alliance is to facilitate global user adoption of mobile data services by specifying market driven mobile service enablers that ensure service interoperability across devices, geographies, service providers, operators, and networks, while allowing businesses to compete through innovation and differentiation. http://www.openmobilealliance.org/

[4]. OMA Device Management specification v1.2. The goal of the Device Management Working Group is to specify protocols and mechanisms that achieve management of mobile devices including the necessary configuration to access services and management of the software on mobile devices. http://www.openmobilealliance.org/release_program/dm_v1_2C.html

[5]. CPE WAN Management Protocol, TR-069 amendment 2, December 2007. http://www.broadband-forum.org/technical/download/TR-069Amendment2.pdf

[6]. The Broadband Forum is a global consortium of nearly 200 leading industry players covering telecommunications, equipment, computing, networking and service provider companies. Established in 1994, originally as the ADSL Forum and later the DSL Forum, the Broadband Forum continues its drive for a global mass market for broadband, to deliver the benefits of this technology to end users around the world over existing copper telephone wire infrastructures. http://www.broadband-forum.org/about/forumhistory.php

[7]. Data model template for TR-069 enabled devices, TR-106 amendment 1, November 2006

[8]. OSGi Service Platform Mobile Specification, Release 4 Version 4.0, July 2006

[9]. RFC1779 http://www.ietf.org/rfc/rfc1779.txt

[10]. http://download.oracle.com/javase/1.5.0/docs/guide/jar/jar.html#JAR%20Manifest

9.2 Author's Address

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

Name	Shigekuni Kondo
Company	NTT Corporation
Address	Y320C, 1-1 Hikari-no-oka, Yokosuka, Kanagawa, Japan
Voice	+81-46-859-3444
e-mail	kondo.shigekuni@lab.ntt.co.jp

9.3 Acronyms and Abbreviations

9.4 End of Document