RFC 0078 MEG High-level architecture

Draft

25 Pages

Abstract

This document provides an overview of how OSGi can be applied in mobile devices. The detailed specification is available in a number of RFCs created by MEG and CPEG. This document serves as a guide for reader of those RFCs by providing a context where the most important relationships and structures are visible. This RFC is INFORMAL.

0 Document Information

0.1 Table of Contents

0.2 Status

This document describes the Mobile Expert Group's high-level architecture for the OSGi Alliance, and requests discussion and suggestions for improvements. Distribution of this document is unlimited within the OSGi Alliance. The document is informal.

All Page Within This Box

0.3 Acknowledgement

This document is the result of the collaborative effort of the following individuals:

- Vadim Draluk, Motorola
- Peter Kriens, aQute
- Gabor Paller, Nokia

0.4 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [1].[1]. `Source code is shown in this typeface`.

0.5 Revision History

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

Revision	Date	Comments
Initial	03/25/2004	Initial version based on the March 23 and 24 phone conferences Gabor Paller, Nokia, gabor.paller@nokia.com
0.1	03/30/2004	Changed according to the March 29 phone conference Gabor Paller, Nokia, gabor.paller@nokia.com
0.2	04/05/2004	Changed according to the April 2 phone conference Gabor Paller, Nokia, gabor.paller@nokia.com
0.3	04/07/2004	Removed template texts, proposed sections for Application Domain, Problem Description, and Security. Wordsmithing and added missing components. Peter Kriens
0.4	04/09/2004	Added sections and drawings explaining Native-MEG. Structured the function block descriptions. Div. changes. PKR
0.5	04/13/2004	Remote management/native relations better explained. Div details. PKR
0.6	04/15/2004	Removed most explanations of the native management. Vadim Draluk
0.7	04/25/2004	Added ADMPRM and update drawing, pkr
0.8	03/01/2005	Major rework. The original goal of the document was to kick off the work in MEG. Now the scope is changed to provide a high-level overview of the work done, and guide the readers of other specifications by providing a context for them. Gábor Pécsy
0.9	03/09/2005	Some rework of Gabor's 3/01/2005 revision. Vadim Draluk, Bill Gengler

All Page Within This Box

1 Introduction

The OSGi Alliance's Mobile Expert Group has been specifying a system based on OSGi framework that is suitable for mobile devices like mobile phones. In the RFP phase, the requirements were collected in 7 documents. These are: RFP 52 (Application Model), RFP 53 (Deployment Model), RFP 54 (Non Functional Requirements), RFP 55 (Policy Framework), RFP 56 (Security), RFP 57 (MIDP), RFP 58 (Device Management).

As the result of the specification efforts made in the different works streams of MEG, a number of RFCs have been created to address the requirements. The architecture described in the document represents only one possible realization of Mobile OSGi; it isn't meant to be complete or normative. The goal of this document is to provide an overview of the solution, which is specified in details in the different RFCs and guide the reader by providing a context where the most important relationships and structures are visible. This RFC is therefore INFORMAL.

2 Application Domain

3 Problem Description

OSGi Service Platform Release 4 and the services defined by the Mobile Expert Group (MEG) have been created through the cooperation of many different players in the computing and mobile telecommunication industries with focus on the special needs of these industries including

- *Mobile terminal manufacturers,* whose devices are to provide an open platform for application developers
- *Mobile network operators* with well-established management systems for mobile devices. This management happens online – over the air –, and usually affects large volumes (up to millions, sometimes tens of millions) of devices.

OSGi Mobile Expert Group (MEG) was responsible for making sure that the next revision of OSGi Service Platform meets the needs of these constituents. Therefore MEG:

All Page Within This Box

- Extended the available set of management services with a number of new ones, such as monitoring and application management
- Defined a new application model for OSGi mobile applications, which simplifies the work of application developers and administrators
- Specified the OMA DM-based device management by specializing the generic remote management architecture of OSGi.

With these new services and specifications, it is possible to create mobile device specific configurations of the OSGi Service Platform. RFC-104 OSGi Mobile Device Profile [1].[18] defines the minimum requirements for such configurations.

In the rest of the document, the term *Mobile OSGi* will be used to denote such a configuration. This document provides an architecture overview of Mobile OSGi. The goal of the document is to serve as a starting point for the interested reader, and put the elements of Mobile OSGi into a context. The reader is assumed to be familiar with the basic concepts of OSGi. While the document focuses on describing only the high-level architecture of MEG, in each case references are provided to full detailed specifications of corresponding areas.

4 Requirements

This document doesn't contain normative material. The requirements that are specified in the MEG RFPs are addressed in other RFCs (see the references).

5 Technical Solution

The architecture of Mobile OSGi is described at two levels. First a high-level ***logical view*** is given, where the platform's functionality is introduced. After that, the ***realization*** of this functionality is discussed at the level of corresponding OSGi services. Wherever needed, pointers are given to the detailed specifications.

5.1 Logical view

OSGi Service Platform is used for two purposes in mobile devices:

- It is an application development platform, which enables the creation of mobile applications and shared services – middleware – that can simplify the development of applications.

All Page Within This Box

- It provides a management framework, which enables the remote and local management of these applications and services. Remote management via OMA DM is of particular interest but OSGi generic remote management architecture enables the use of other management protocols as well.

This first release of Mobile OSGi focuses on two critical functional areas: device management – especially via OMA DM – and the mobile application model:

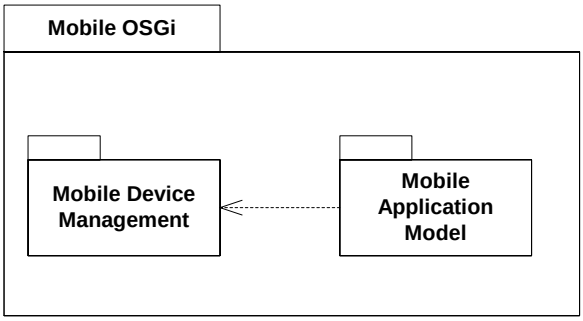


Figure 1: Mobile OSGi functional areas

5.1.1 Mobile Application Model

Applications interacting with human users have different needs than the basic service bundles, which have been the focus of previous OSGi releases. In order to use OSGi as a mobile application development platform, we need an application model, which can both benefit from flexibility/power of OSGi and introduce sufficient rigor and discipline to enable an average developer to create new applications and middleware components. This simplification is achieved by building the application model on top of OSGi declarative services introduced in Release 4. This hides dynamic aspects of the framework from developers. Applications of this new model are called Meglets.

The applications' lifecycle model is different from the service lifecycle model in that it has clearly defined time boundaries and various states of activity, and is often controlled by the user activating and stopping applications. To support this, a new lifecycle model for applications has been defined.

The interaction of applications is also different from the interaction of services. While the latter are subject to only discovery and invocation, meglets can:

- Be content handlers (see Content Handler API [JSR-211])
- Launch other meglets via the application management framework, passing startup arguments if necessary.
- Interact with each other and with the execution environment via events.

Application model is specified in RFC-91 [1].[16], declarative services are defined in RFC-80 [1].[12].

5.1.2 Device Management

Device Management of Mobile OSGi includes management of all of the software components and execution environment under the control of OSGi on the device.

Remote management was always a central concept of OSGi, but both client and server were expected to be owned by the same party, so their interaction has never been standardized. Mobile devices are different because network infrastructure is potentially supplied and owned by parties different from ones writing the device-based software. This necessitates alignment of the management model with the standards prevalent in the industry, and results in broad support OMA DM models receive in MEG.

Figure 2 describes the logical architecture of OMA DM-based remote management.

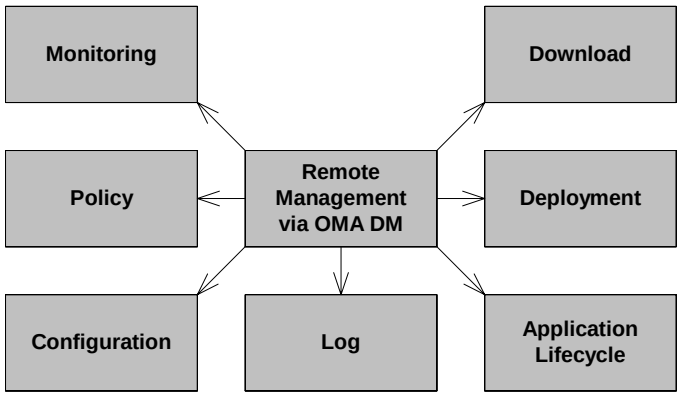


Figure 2: Logical architecture of device management

5.1.2.1 Remote Management via OMA DM

OSGi Release 3 defined a generic – management protocol agnostic – remote management architecture. In Release 4, the application of this generic architecture with OMA DM management protocol is specified, which is widely used in the mobile industry.

The Device Management Tree structures for the different management functions are defined. Device Management Tree (DMT) is the management object model used in OMA DM protocol and in its nodes, all of the operating characteristics of the OSGi environment and installed components are mapped.

An API is defined which provides local programmatic access to the different parts of DMT. This API enables the creation of 3rd-party management agents, which can use a protocol of choice and which can be deployed on mobile devices as an alternative to OMA DM if required.

In addition, a plugin interface has been defined, which enables the extension of the DMT to support expanded management functionality. The plugins are responsible for mapping OMA DM commands to operations on the related management service. Through the use of plug-ins, virtual subtrees are added to the DMT that correspond to actual underlying OSGi entities.

5.1.2.2 Management services

The following management services are included in Mobile OSGi.

Configuration

Configuration Admin service enables the management of configuration objects. These objects can be associated with different entities (service, applications). Configuration Admin dynamically notifies the configured entities – if they are running – about all changes made to the corresponding configuration object.

Logging

Log Service provides facilities for creating log entries. Logs can be queried via remote management to enable the remote administrator to analyze possible problems on the device.

Application Lifecycle Management

This open management framework provides remote access to application lifecycle management operations, such as starting, stopping, and scheduling. This framework, which supports different application models, can be customized to support additional, application-model specific management operations as needed. For example, the framework has been extended to support the suspend and resume operations on Meglets.

This framework enables the creation of application model agnostic application managers, which are capable of performing the basic management tasks on their application types. It also enables the creation of application model aware application managers, which can utilize the specific features of the supported application models.

Policy Management

Policy defines managed access to management operations, applications, services and shared libraries.

Access control for applications is an extension of Java 2 security framework. It utilizes the new conditional permission feature of OSGi. The use of conditions gives a very high level of flexibility to the policy system.

Access control of management objects uses both Java 2 security and Access Control Lists defined in OMA DMT. For remote management, first ACLs applied. ACLs provide a coarse-grained access control mechanism, which is then complemented with the fine-grained access control of Java 2 permissions. When accessing the management services locally, only the Java 2 permissions are used.

Further details of OSGi policy and conditional permissions are available in RFC-92 [1].[17] and RFC-95 [20].

Monitoring

The ability to monitor the behavior of applications and services is provided. In Mobile OSGi, application and services can define Key Performance Indicators (KPI), which provide important statistics about their state. KPIs can be monitored both locally via the new Monitor Admin service and remotely via DMT or a custom agent.

The detailed definition of monitoring is available in RFC-84 [1].[13].

Deployment

In Mobile OSGi, deployment is not handled at the bundle level. Instead, bundles are grouped with other resources (e.g., Configuration Objects) into a larger unit of deployment called a Deployment Package (DP). The contents of these DPs are installed, updated, and uninstalled as a unit atomically. The Deployment Admin Service handles the overall processing and administration of DPs including bundle management and the processing of other resource contained within the DP. Resource Processors (RPs) can be defined to handle processing of other non-bundle resources contained in a DP. MEG Deployment Configuration is an example of a MEG defined RP. RPs can also be written by third parties to handle their own resources.

See RFC-88 [1].[15] for detailed definition of deployment functionality and RFC-94 [21] for details on the MEG Deployment Configuration RP.

Download

All Page Within This Box

Deployment Packages can be downloaded to the device using OMA DL 1.0 protocol. OSGi defines only a simple extension to the OMA DL download descriptor file format, which enables the download subsystem to recognize that the downloaded JAR file contains a DP and pass it to the Deployment Admin of OSGi.

The download model supports the pull-like model, which is fully compatible with the existing MIDP download process. This enables the device user to use the browser of the device to discover the DP to be downloaded. The download model also supports the push model, where some management server initiates the download (and installation) of the DP. This functionality is available via the DMT using OMA DM protocol or via a custom agent.

Further details of download are available in RFC-105 [1].[19].

5.2 External View

The following figure describes the connections of Mobile OSGi to its environment. Some of the listed elements may be implemented on top of OSGi, or separately. This external architecture view is just a reference; concrete implementations may look very different from this. The interfaces identified in Figure 3 are logical names, possible mappings to standard OSGi APIs is given in section 5.2.2.

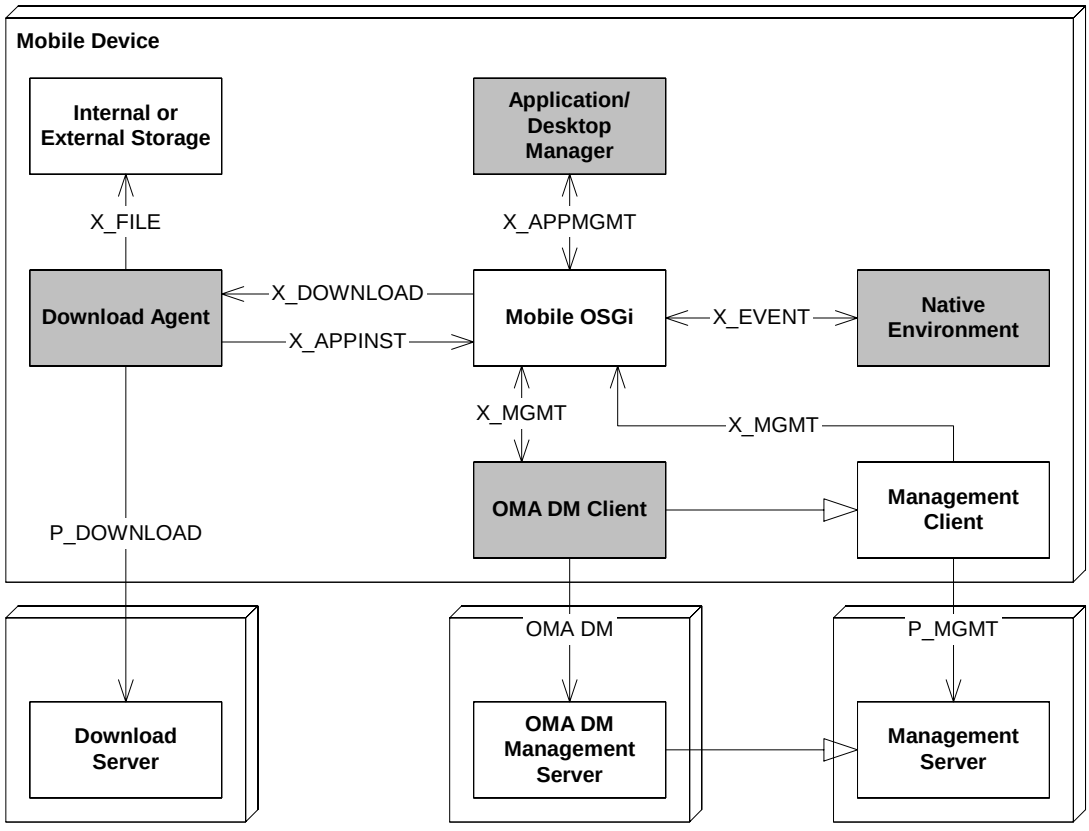


Figure 3: External view of Mobile OSGi

5.2.1.1 Application / Desktop Manager

An Application/Desktop Manager typically interacts with the user via some graphical user interface, e.g. by displaying icons and names of applications. This manager would provide means of starting/stopping (i.e.,

launching/closing) applications. It may also expose more sophisticated management features, like scheduling, suspending/resuming operations, etc. To provide all this functionality, the manager uses the application inventory and lifecycle management services provided Application Management Framework of OSGi *X_APPMGMT* interface.

5.2.1.2 Native Environment

X_EVENT is the interface between the OSGi subsystem and the native environment (if present). This interface is not standardized in OSGi.

This interface provides a means for OSGi services and applications to adapt to changes happening within native parts of the device. Depending on the implementation needs, the notification may be bidirectional, i.e. event of the OSGi subsystem are forwarded to the native environment as well.

5.2.1.3 Management Client

Management Clients are the client-side agents that can receive management operations from a management server via some *P_MGMT* management protocols and perform the corresponding operations on the management objects of the device. In order to enable remote management, the device should have at least one Management Client installed.

In particular, **OMA DM Client** implements the client side of the OMA DM management protocol. It is responsible for the connection to remote management servers. It may be implemented on top of the OSGi framework. Typically, this client will be some external entity because it is the common access point for the whole DMT of the device. This approach enables the non-OSGi parts of the DMT to be accessible independent of the state of the OSGi framework, the DMT Admin and the DMT Plugins.

Management Clients can use *X_MGMT* interface to interact with the management services of OSGi. *X_MGMT* interface is bi-directional.

5.2.1.4 Download Agent

Download Agent is responsible for downloading Deployment Packages or other application packages to the mobile device. OSGi specifies the use of OMA DL 1.0 protocol for downloading DPs from a remote download server. Different download protocols (e.g. MIDP OTA for MIDlets) may also be used.

Download Agent uses *X_APPINST* interface to install the downloaded packages in OSGi. *X_DOWNLOAD* interface is used by OSGi to initiate download of specific packages.

P_DOWNLOAD interface is the download protocol used between the download server and the mobile device. In addition, Download Agent may support other media types, for example “downloading” from local file system or via some local PC connectivity. DA may use *X_FILE* interface for accessing such local media.

See RFC-105 MEG Download [1].[19] specification for further details on Deployment Package download.

5.2.2 Interface Summary

The following table summarizes the interfaces in the figure:

Interface	Description
X_APPMGMT	This interface provides access to application management operations, which are generic, independently from the underlying application model. This enables the creation of

All Page Within This Box

Interface	Description
	application model agnostic application managers. The interfaces defined in the `org.osgi.service.application` and `org.osgi.service.application.meglet` packages can be used for this purpose.
X_EVENT	This optional interface enables the interoperability between the native and OSGi event mechanisms. The functionality of the interface is implementation specific; it may be uni- or bidirectional. The interface is implementation specific.
X_MGMT	Management Clients can interact via this interface with OSGi. The interface is bidirectional; OSGi management services may be able to initiate communication with the management server. Possible realization of this interface is DMT Admin API as defined in `org.osgi.service.dmt`, but the Management Client can use the APIs of the different management services (e.g. Configuration Admin, Permission Admin etc.) directly as well. The OSGi initiated communication takes place over some implementation specific interface.
P_MGMT	P_MGMT is a remote management protocol, which is used for the communication between a Management Server and the corresponding Management Client on the device. OMA DM is such a management protocol. Other possibilities include for example SNMP or HP OVO.
X_DOWNLOAD	OSGi framework can use this interface to initiate the download of some package. This function used for example to implement the server-assisted download use cases, where a management server initiates the download and installation of a package. There is no standard for this interface. Actually, it may be different for the different download agents of the device.
X_APPINST	This interface is used by the Download Agent to pass the downloaded package to the corresponding OSGi service. This interface is differentiated from *X_DOWNLOAD*, because the download process may have been initiated by some external means (e.g. user browsing). If the downloaded package is an OSGi Deployment Package, this interface corresponds to Deployment Admin API as defined in `org.osgi.service.deploymentadmin` package. For other types of packages, implementation specific interfaces can be used.
P_DOWNLOAD	*P_DOWNLOAD* interface is the download protocol used between the download server and the mobile device. *P_DOWNLOAD* is OMA DL 1.0, when downloading DPs from a remote server. Other application models may require or enable the use of other download protocols. In addition, Download Agent may support downloading via some local PC connectivity. DA may use implementation specific protocols for accessing such local media.

All Page Within This Box

Interface	Description
X_FILE	Download Agent may support local media types, for example “downloading” from local file system or from a memory card. *X_FILE* interface is used for accessing such media. One realization of this interface is the API defined in `java.io` package, but other implementations are possible as well.

5.3 Detailed Architecture

This section gives an overview how the logical architecture is realized in mobile OSGi.

5.3.1 OMA DM Device Management

OSGi defines management services that provide a set of APIs. The generic remote management architecture is protocol independent. Management Servers connect to a protocol-specific Management Client via some management protocol. The management client is responsible for mapping the management operations received via the management protocol to the method calls of the different management APIs. The following figure describes how this generic model has been applied to the specifics of OMA DM.

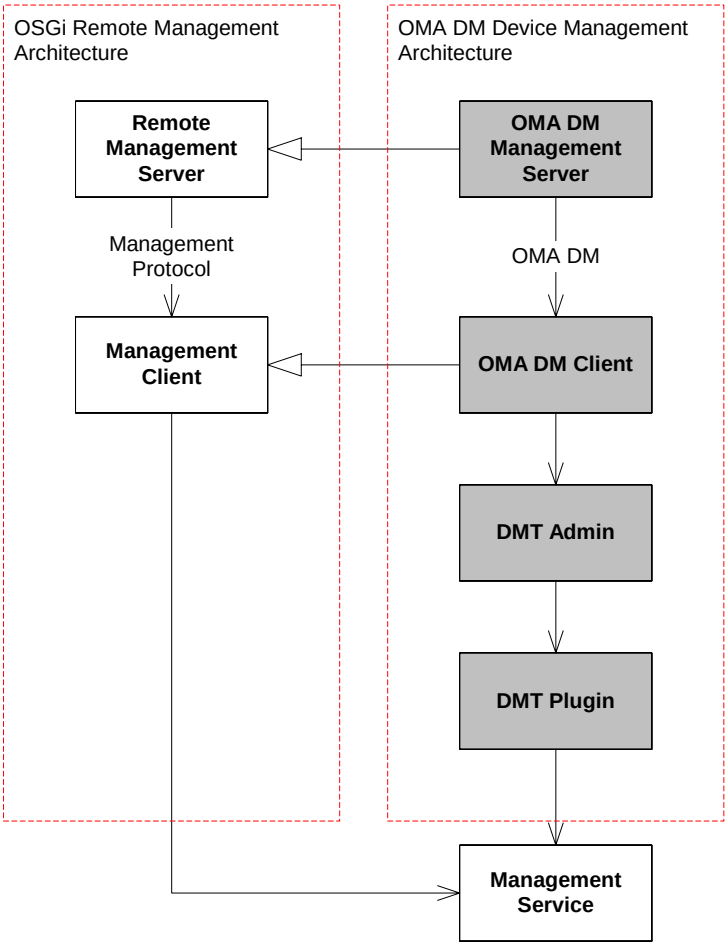


Figure 4: Generic Remote Management Framework Applied for OMA DM

In Figure 4 the role of Management Client is implemented in three entities:

- OMA DM Client is responsible for the client-side protocol implementation. It authenticates the management server, parses the received protocol messages. If the request targets the OSGi subsystem then it delegates the request to DMT Admin. This delegation happens via the DMT Admin API (defined in `org.osgi.service.dmt` package).

- DMT Admin and the DMT Plugins are responsible for implementing the management object model (DMT) and mapping it to the different management services. See the following section for details.

5.3.1.1 Device Management Tree

The management object model of OMA DM management protocol is the OMA Device Management Tree (DMT). In DMT all the management objects are mapped into tree-like structure. The nodes of the tree can be addressed by URIs. The URI is the list of the nodes along the path from the root of the tree to the addressed node. The node names are separated by the '`/`' character, the name of the root node is '`.`' (dot). For example:

```
./OSGi/log
./Device/Battery/Level
```

All Page Within This Box

The nodes can be interior or leaf nodes. The leaf nodes can store typed data; the available set of data types is defined by OMA. Other kinds of metadata can be associated with the nodes as well. One of the most important kinds is Access Control List (ACL).

OMA DM defines five possible operations on the nodes of DMT. These are Add, Get, Replace, Delete and Execute. ACL of a node specified which entity is permitted to perform the different operations on that node.

DMT doesn't necessarily exist on the device as some kind of database. Different nodes or subtrees of DMT correspond to different management objects. The connection between the subtree and the node is implemented in DMT plugins.

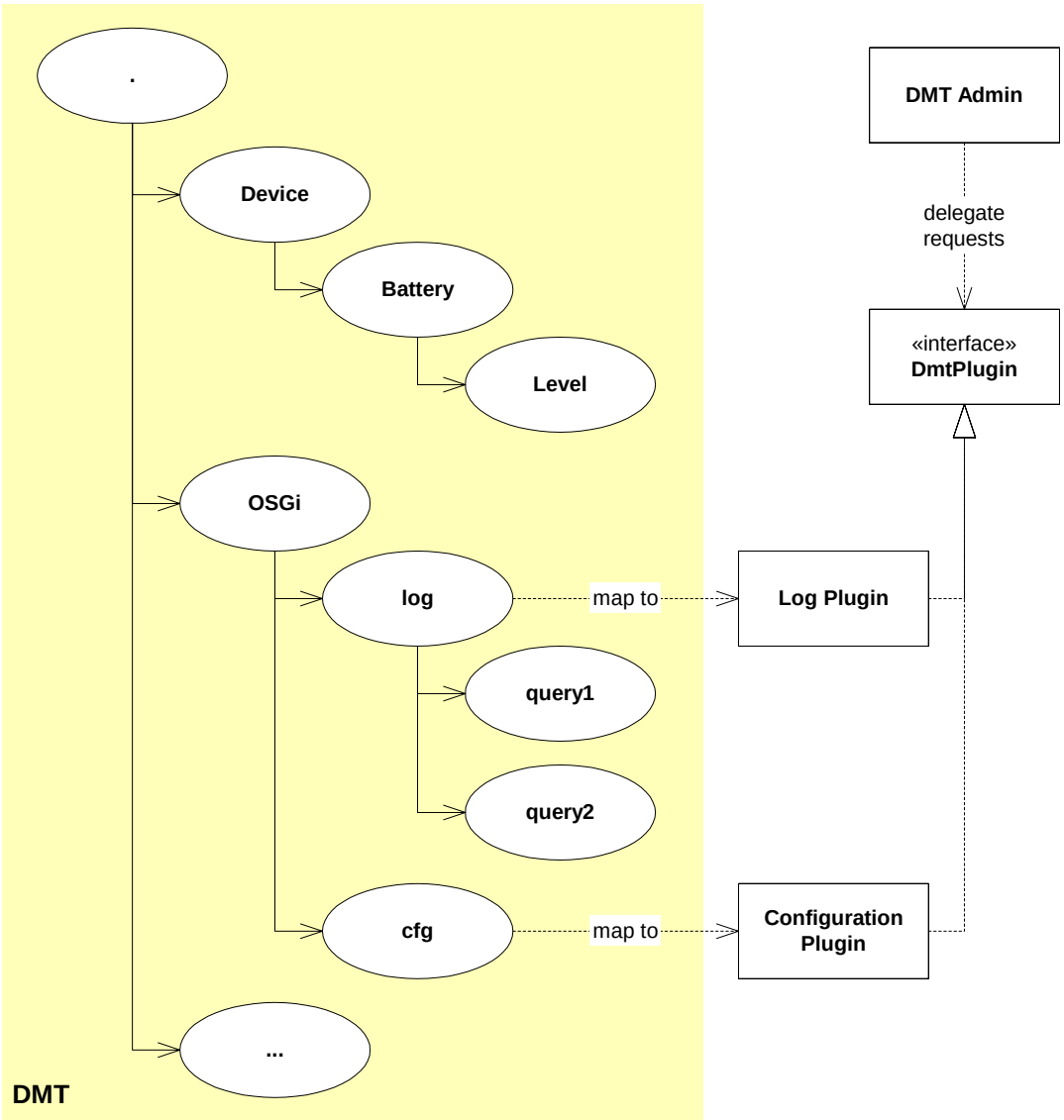


Figure 5: Device Management Tree plugins

DMT Admin is responsible for the management of the DMT plugins. The OMA DM operations received from the OMA DM Client are dispatched by DMT Admin to the corresponding plugin, which then translates the OMA DM

request to actual operations of the corresponding management object. The semantics of the different OMA DM commands depends on the managed object.

The execution of OMA DM operations takes place in a transactional context. DMT Admin is responsible for the implementation of this context, it notifies the plugins about the creation of a new transaction, and the successful or unsuccessful termination of the transaction.

The DMT structure for OSGi management is defined in RFC-87 [1].[14].

5.3.1.2 Device Management Service Architecture

The following figure describes a possible architecture of the Mobile Device Management functionality. The different DMT plugins are managed by DMT Admin (the connecting lines were omitted for clarity). Some management operation relies on low-level services provided by the OSGi core, others are implemented in different management services. The DMT plugins map the different management services to DMT. Notice that the described structure of DMT Plugins is just one possible realization. In other implementations, the number of plugins may differ, just like the distribution of responsibilities. In some implementation, DMT Admin may integrate all the listed functionalities as well.

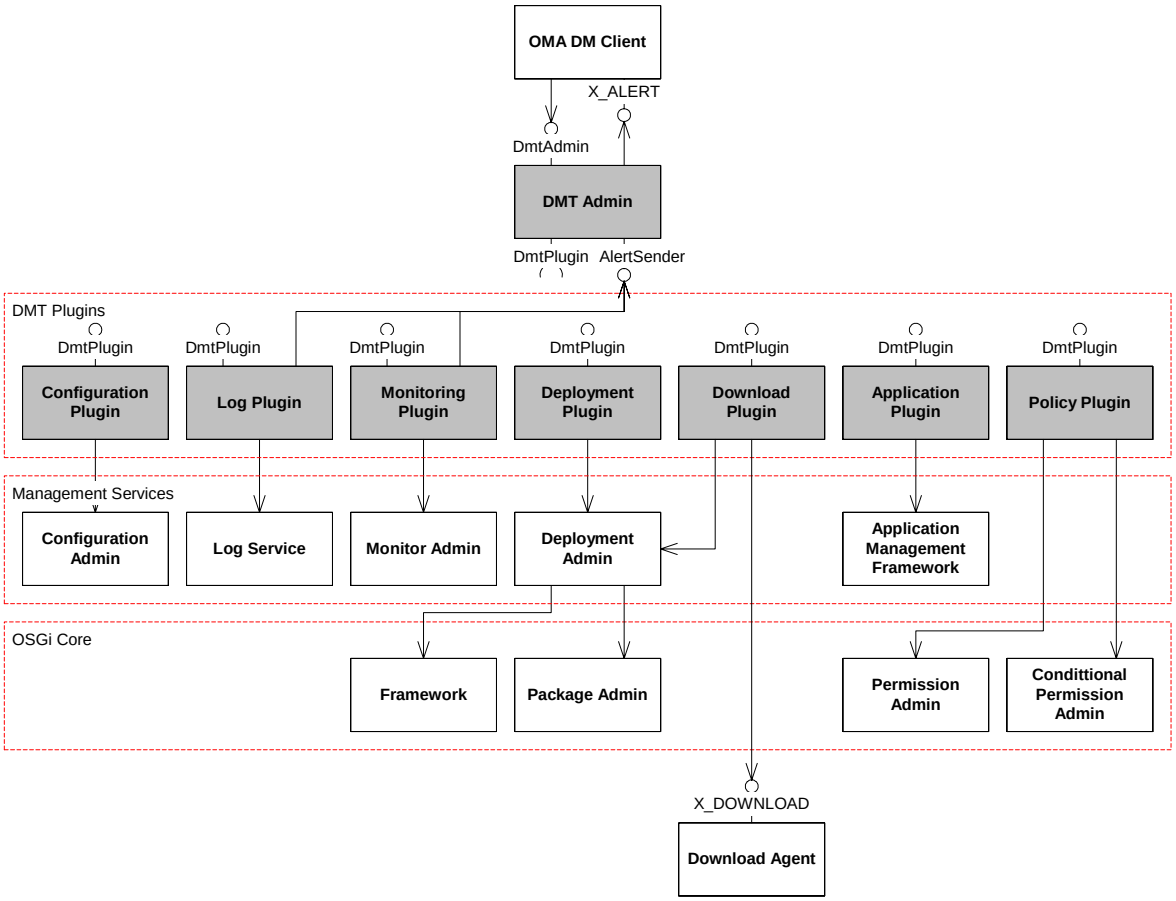


Figure 6: Device Management Service Architecture

Figure 6 describes Mobile Device Management Architecture. The elements of the figure are described in the following sections.

DMT Admin Service

OMA DM Client delegates the OMA DM requests targeting OSGi subsystem to DMT Admin service via the *DmtAdmin* interface.

DMT Admin is responsible for the management of the OSGi-defined subtree of DMT. Some part of this subtree may be maintained directly by DMT Admin, but typically it delegates the management operations to DMT plugins.

DMT Admin uses *DmtPlugin* interface to access the DMT plugins. Each plugin has an associated URI, which determines to root of the subtree managed by the plugin. Plugins cannot be nested.

DMT Admin implements the *DmtAdmin* interface and uses the *X_ALERT* interface to send alerts to the management server. In addition, DMT Admin implements the *AlertSender* interface which enables the different DMT Plugins to send alerts – asynchronous notifications – to the management servers.

DMT Plugins

DMT plugins handle subtrees of DMT identified by the URI of the root node of the subtree. DMT Admin uses the DmtPlugin interface to delegate the different management operations to the corresponding plugin. The actual distribution of responsibilities between the plugins, and the number of plugins is implementation specific, this list is provided as reference only.

The different plugins use the corresponding management service APIs to implement the calls.

Download plugin

Management servers can initiate the download (and installation) of Deployment Packages to the mobile device. The plugins forwards the download requests to a Download Agent, which can then execute the download process. The actual state of the download can be monitored in the DMT during the process.

Configuration plugin

Configuration plugin enables the manipulation of Configuration objects stored in Configuration Admin via DMT. The plugin maps the content of Configuration objects to the DMT, so OMA DM operations can be used to query and modify their content or to create new configurations.

Log plugin

Log plugin enables the setup and execution of log queries in the device. Management server can specify different attributes of the query:

Filtering attributes to select the desired log records

The list of interesting fields of the records

Limitations on the size of query results

Queries can be set up well in advance and executed in a separate OMA DM management command. Old queries can be reused as well. The results of the query are sent back to the server in an asynchronous notification.

Monitoring plugin

Monitoring plugin maps all defined KPIs to the DMT. Management servers can simply query their values or create monitoring jobs to periodically monitor the changes. Monitoring plugin uses asynchronous notification mechanism to deliver the updates to the server.

Deployment plugin

Deployment plugin enables the management of Deployment Packages. It maps all installed DPs to DMT, which enables the management servers to query different attributes of package, including the list of bundle it contains, its dependencies, the signatures that certified the origin of the package etc. DPs can be uninstalled via this plugin.

Application plugin

Application plugin provides access to the services of the Application Management Framework and its Meglet-specific extensions. The plugin makes visible all installed applications in the DMT. OMA DM commands can be used to launch new instances or schedule applications for execution.

All the existing application instances are mapped to the DMT as well. Management servers can stop them and, if they are Meglets, suspend and resume them.

Policy plugin

Access Control Lists are already part of DMT; they can be manipulated via OMA DM. Policy plugin defines the mapping of Java 2 permissions stored in the Permission Admin and Conditional Permission Admin services to DMT. Permissions can be created, removed or modified using OMA DM via this plugin.

5.3.1.3 Interface summary

The following table summarized the interfaces in the figure:

Interface	Description
DmtAdmin	DmtAdmin provides access to the DMT of the device. Management agents – such as OMA DM Client – can use this API to navigate in the tree and perform the management operations defined in OMA DM. This API is defined in `org.osgi.service.dmt` package.
DmtPlugin	DMT plugins need to implement this interface. DMT Admin uses this interface to delegate the OMA DM management commands to the corresponding plugin. The interface is defined in `org.osgi.service.dmt` package.
AlertSender	This interface can be used by the DMT Plugins to send asynchronous alerts to management servers. The interface is defined in `org.osgi.service.dmt` package.
X_ALERT	This interface is used by DMT Admin to forward the requests received via *AlertSender* interface to the OMA DM Client.

All Page Within This Box

Interface	Description
	The interface is left implementation specific.
X_DOWNLOAD	*X_DOWNLOAD* interface is defined in section 5.2.2.

5.3.2 Application Management Framework

OSGi Release 4 specifies an application model (see section 5.3.3) to address the different needs of application developers. To enable starting, stopping, scheduling of applications, application management was defined. Nevertheless, application management is not restricted to the new model only. An Application Management Framework was created instead, which is applicable for the management of many different application types. Figure 7 describes the elements of the framework and how it can be applied to manage an arbitrary application model X.

The application management framework has been based on the OSGi platform service model. Applications and their running instances can be found as services in OSGi service registry.

The application management framework supports the basic lifecycle management – i.e. start and stop – and scheduled execution of application. It also allows extensions with new lifecycle management operations for particular application models. This section provides an overview of this management framework, for further details see RFC-91 [1].[16].

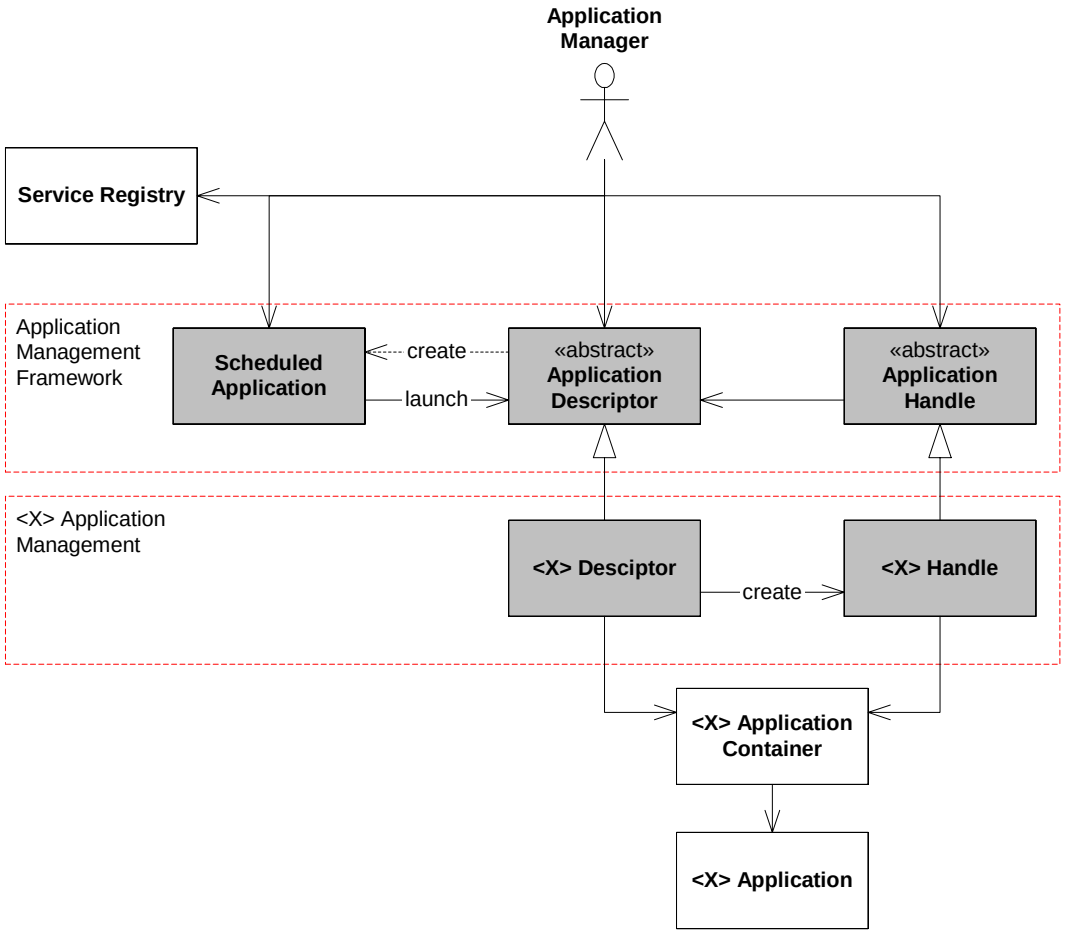


Figure 7: Application Management Framework

Application Manager in the figure denotes any client of the management framework, for example Application/Desktop Manager from section 5.2.1.1, or the Application Plugin from section 5.3.1.2. The management framework consists of three entities.

Application Descriptor

Application Descriptors represent the installed application. Whenever an application is installed, the corresponding descriptor is registered in service registry. It remains registered as long as the application is installed. The descriptor is removed from the registry when the application is uninstalled. The application descriptor can be used for the following purposes:

- Obtaining different attributes of the installed application, e.g. its icon, name, version etc.
- Launching a new instance of the installed application
- Scheduling the application to be launched on some event

Application Descriptor is defined as an abstract class. It implement policy enforcement and scheduling but leaves the lifecycle management (i.e. launching an application) undefined. Different application models need to provide

their concrete realization of those parts by providing their application model specific descriptor class. Therefore, always a subclass of Application Descriptor is registered in the service registry.

Application Handle

Application Handle represents an instance of an application. When a new instance is launched, using the corresponding descriptor, an Application Handle is registered in the service registry to represent that particular instance. It remains in the service registry as long as the corresponding application instance is not destroyed. Application handle can be used for the following:

- Query different characteristics of the running application instance, for example its application descriptor, state, startup arguments etc.
- Destroy the corresponding instance.

Just like Application Descriptor, the handle is given as an abstract class, which implements policy enforcement, but leaves the lifecycle management operations to the concrete realizations. Therefore, always subclass of Application Handle is registered in service registry.

Scheduled Application

Applications can be scheduled for execution on a particular event. Application descriptor provides the means for this. When the corresponding method is called, a Scheduled Application object is registered in the service registry. This object can be used for the following:

- Query different characteristics of the scheduling, for example the triggering event specification, the startup parameters, the descriptor of the application to be launched etc.
- Cancel the scheduling.

5.3.2.1 Applying the Framework

In order to support the management of a new application model (X) in the framework the application model needs to be integrated with the framework.

<*X*> *Application Container* (container) denotes the implementation of the runtime environment specified in the X application model. Notice, that this container may or may not be part of the OSGi subsystem. It is responsible for the execution of <*X*> *Applications*.

The integration of the container – thereby enabling the management of X applications – happens by providing X model specific realizations of Application Descriptor and Application Handle. Let these classes be <*X*> *Descriptor* and <*X*> *Handle* respectively.

These classes are responsible for forwarding the lifecycle management operations to the application container. <X> Handle can extend the basic set of lifecycle operations with new ones, specific to application model X.

This way a generic Application Manager can perform the basic management operations (start and stop) on all applications, independently of their types, whereas an X application model aware Application Manager can utilize the advanced functionalities provided by the X model.

5.3.3 Application Model

In order to become a real application development platform, OSGi defined a new application model. This application model defines:

- A lifecycle model for the applications
- An interaction model for the applications
- A convenience layer, which helps the application developer to deal with the dynamic nature of OSGi

On the other hand, it doesn't define the UI; it can be applied with different UI toolkit (e.g. AWT, SWT, Swing, LCD UI etc).

The applications of this new model are called Meglets. Meglets are included in bundles. A bundle can contain any number of Meglets. The bundles are deployed to the framework in deployment packages. Each Meglet is a Service Component as defined in RFC-80 Declarative Services specification [1].[12]. A bundle must be started before any Meglet it contains can be used. Once the bundle is started, the Meglets have their own lifecycle.

Meglets need to extend the abstract `org.osgi.meglet.Meglet` class. This class:

- is provided as part of the Meglet runtime environment implementation,
- ensures that all Meglets comply with different contracts regarding lifecycle management. This includes the lifecycle management methods required from service components as well.
- provides convenience methods for interacting with the OSGi framework.

Lifecycle Model

The Meglet model fits into the generic lifecycle model defined in the Application Management Framework (see section 5.3.2) but it extends the possible lifecycle operations. Running Meglets can be suspended. The state of a suspended Meglet instance is saved persistently but otherwise it consumes no resources until it is resumed. Suspended instances survive even framework restarts. Figure 8 describes the lifecycle model of Meglets.

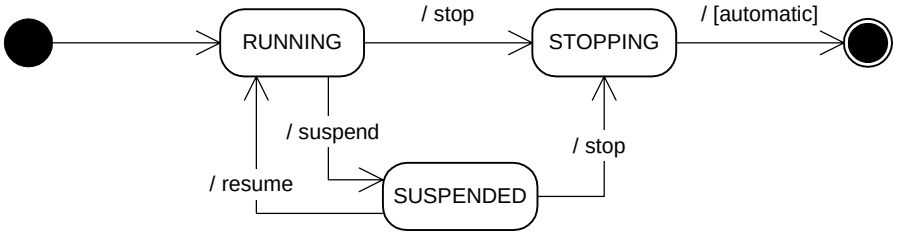


Figure 8: Lifecycle model of Meglets

The Meglet base class defines extension points (methods) for applications, where they can react to the lifecycle state transitions of the application.

Convenience Methods

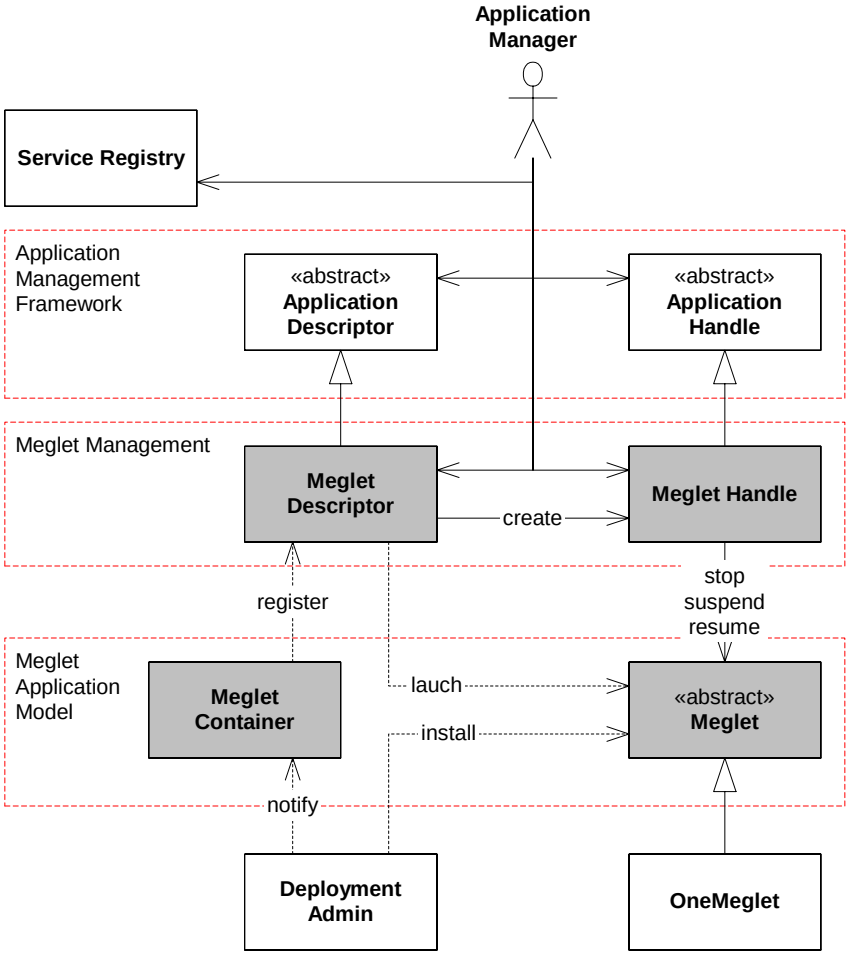
All Page Within This Box

The `Meglet` base class provides a number of convenience methods that help dealing with the different OSGi services. Applications can easily get their configuration, access Log Service, send or receive event. These methods significantly simplify the work of application developers; nevertheless, they can still access the full flexibility of the framework, if needed.

5.3.3.1 Meglet Management

OSGi defines how the generic framework should be applied for Meglets. This process outlined in section 5.3.2.1 is applied. MegletDescriptor and MegletHandle classes are defined in the `org.osgi.service.application.meglet` package. MegletHandle class is extended with `suspend()` and `resume()` methods, so that Meglet-aware Application Managers can use them to suspend and resume Meglet instances. The following figure describes the elements of the mapping.

The figure also describes the installation of Meglets. Meglets are installed as part of bundles by the Deployment Admin. The Meglet container, which listens for bundle events, recognizes when bundles containing Meglets are installed and uninstalled. Based on these events, it register/unregisters the MegletDescriptors for the Meglets in the bundle. As Meglets are native to OSGi, they do not require any special runtime environment – beyond what OSGi provides anyway. Responsibilities of MegletContainer are therefore limited. It is responsible for keeping the service registry up to date with the descriptors of installed Meglets and the handles of the existing Meglet instances.



All Page Within This Box

Figure 9: Meglet Management

6 Considered Alternatives

No alternatives have been considered so far. However, there have been discussions about the level of detail and the semantics of the diagram. It was decided to stay on a high level and fill in the details in the workgroups. These discussions have not been recorded.

7 Security Considerations

The MEG Security architecture is an extension of the OSGi Security Architecture, which, in its turns, is based upon the Java 2 Security model.

Until now, security has been an option in OSGi. MEG, however, makes the use of the OSGi security mechanisms mandatory. A number of RFCs must be developed or adopted to create an end-to-end security chain because some aspects of this chain have been left to the implementers.

Install packs must be signed. RFC 73 currently works on a signing model. It must be decided if this RFC is adopted or another RFC is required. This is an open issue. A related issue is the assignment of the Policies to the Downloaded Bundle. RFC 73 discusses a Permission based model, where the signer is associated with a set of permissions. Unfortunately, this RFC is still in flux due the differences between Java 2 and OSGi application model and the lack of real world usage of fine-grained permissions.

An extension to the OSGi Security is the Policy Admin. The Policy Admin is envisioned as a powerful mechanism to restrict the usage of the Mobile Device depending on the user, the time and date and potentially other conditions. The Policy Admin is based on the Permission Admin. All permission checks are performed using standard Java 2 Permissions. The use of the Policy Admin is optional.

DMT has its own security policy mechanism in form of ACLs. MEG will use this mechanism to define DMT access policies, while extending the set of actors from mere servers under OMA DM to some local subjects such as code signers, subscription IDs etc.

The MEG promotes the use of services for the interfaces. Services can be easily protected with ServicePermission. PackagePermisson will be used to limit the contribution of bundles to the shared code base.

Downloaded Bundles must attempt to minimize the sharing of implementation classes because this increases the risk of opening up access points to other bundles.

8 Document Support

8.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. RFP 0052 MEG WS Application Model

[4]. RFP 0053 MEG Deployment Model

[5]. RFP 0054 MEG Non Functional Requirements

[6]. RFP 0055 MEG Policy Framework

[7]. RFP 0056 MEG Security

[8]. RFP 0058 Device Management

[9]. SyncML Device Management Tree Description

[10]. OMA Device Management (http://www.openmobilealliance.org/tech/wg_committees/dm.html)

[11]. JSR-211 – Content Handler API

[12]. RFC-80 – Declarative Services

[13]. RFC-84 – Monitoring

[14]. RFC-87 – DMT Structure

[15]. RFC-88 – Deployment Manager

[16]. RFC-91 – Application Model

All Page Within This Box

[17]. RFC-92 – Policy Admin

[18]. RFC-104 OSGi Mobile Device Profile

[19]. RFC-105 MEG Download

[20]. RFC-95 Conditional Permissions

[21]. RFC-94 MEG Deployment Configuration

8.2 Author's Address

Name	Gabor Paller
Company	Nokia
e-mail	Gabor.Paller@nokia.com

Name	Vadim Draluk
Company	Motorola
e-mail	vdraluk@motorola.com

Name	Peter Kriens
Company	aQute/OSGi
e-mail	Peter.Kriens@aQute.biz

Name	Gábor Pécsy
Company	Nokia
e-mail	Gabor.Pecsy@nokia.com

Name	Bill Gengler
Company	IBM
e-mail	gengler@us.ibm.com

8.3 End of Document

All Page Within This Box