Virtual OSGi Framework and Telecommunications

Sam Supakkul
Digital Pockets, LLC
Dallas, Texas
ssupakkul@computer.org

Lawrence Chung
Dept. of Computer Science
University of Texas, Dallas
chung@utdallas.edu

ABSTRACT

While embedded and telecommunication equipment is getting smaller, it is required to provide more functionality and flexibility. The success criteria for these equipments are often conflicting: lower cost but higher functionality and flexibility. A new emerging platform called OSGi Framework may be used to meet such high demands. Typical telecommunication or embedded devices for the OSGi Framework are most likely to have small form factor and design to be inexpensive for consumer market. Therefore, large or upgradeable memory may not be common. As the result, number of concurrent services is limited by the physical memory size. This paper proposes a solution to create a virtual memory like effect on the OSGi Framework to allow more concurrent services to run on the OSGi Framework platform. A hypothetical everyday life scenario is used to demonstrate how the Virtual OSGi Framework can be used to enhance the value of the standard OSGi Framework.

1 INTRODUCTION

While embedded and telecommunication equipment is getting smaller,, it is required to provide more functionality and flexibility. This is especially true for consumer communication-based devices such as web phones, navigation aid systems in automobiles, entertainment systems, and networked home services gateways. The success criteria for these equipments are often conflicting: lower cost but higher functionality and flexibility. The equipment manufacturers are experiencing several problems to meet such demands:

- Embedded software is harder to design: increasing networked devices introduce significant complications such as downloadable modules that dynamically reconfigure the system [4].
- Limited development resources: skilled developers and better development tools. The industry is not able to leverage more abundant resources from the traditional IT industry due to the use of different programming languages and specialized real-time operating systems.
- Inflexible: embedded software is typically pre-built and installed on the device. It is difficult to upgrade or add new functionality or applications to the device deployed in the field or purchased by consumers.
- Inefficient use of system storage: all programs must be available in a local persistent storage such as FLASH memory or ROM. Therefore; the number of applications is limited by this storage.
- Poor system serviceability: it is often difficult if not possible to remotely diagnose and fix problems occur in the device.

Recent developments in software and telecommunication industries indicate that these problems are being addressed:

- An industrial consortium called Open Service Gateway Initiative (OSGi) has defined a dynamic embedded application run-time environment based on Java language [12] called OSGi Service Gateway specification [16]. The run-time environment of this specification is called OSGi Framework. OSGi Framework allows applications, in the form of Java JAR files, to be downloaded from the network on-demand without pre-installation. This allows equipment to be highly flexible, makes efficient use of persistent memory storage where it is only used to store system software not the applications, and provides a highly serviceable system where dynamic software upgrades and downloads are naturally supported. Additionally, the use of Java programming language offers greater cost saving and the ability to leverage a larger pool of developers and tools from the traditional IT industry [21].
- A new Java extension called the Real-Time Specification for Java has been released to public through Sun's Java Community Process [7]. It will be possible to develop embedded software using a common language and platform that is independent of underlying real-time

operating system. This will further provide cost saving and the leverage of resource pool from the IT industry.

OSGi Service Gateway specification has gained tremendous support from the telecommunication industry as indicated by the large number of OSGi members from the telecommunication industry and many product initiatives [1], [2], [5], [6], [15], [17]. Also, as high-speed broadband access (such as DSL, cable-modem, and 3G wireless phone) is becoming more affordable and available to consumers, this will be a catalyst of a new type of market for telecommunication services using the OSGi Services Gateway for dynamic on-demand services.

An example of an OSGi Services Gateway application can be depicted in Figure 1.

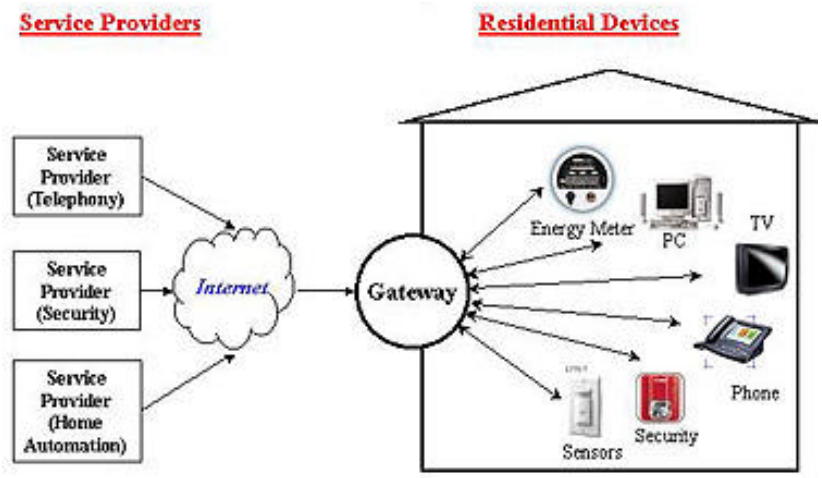

Figure 1. Example of OSGi Services Gateway Application [18]

However, there is a potential problem that could happen to equipment running an OSGi Framework. Because these consumer devices (such as wireless phone and Service Gateway) are most likely equipped with limited system resources (for example CPU and memory storage) for cost purposes but they are still required to provide more functionality and applications to consumer. As the OSGi services gateway and applications proliferate, the equipment may not have sufficient memory to run all concurrent applications desired by the consumer.

This paper proposes Virtual OSGi Framework concept to address this limited memory problem. Similar to virtual memory concept, Virtual OSGi Framework creates a perception of larger memory space for running more concurrent applications on the device. In traditional virtual memory mechanism, a local secondary storage (typically a hard-disk) is used to store inactive memory pages. The inactive memory pages are brought back into main memory when accessed by the application [14]. Unfortunately, an embedded device is typically not equipped with a hard disk that can be used for virtual memory mechanism.

Virtual OSGi Framework concept uses a remote server to store inactive memory. However, due to network latency, swapping applications at memory page level would not be practical as the device may suffer unacceptable response time while swapping in and out memory pages. Therefore, this paper proposes to swap memory at the application level, where the entire application is swapped in from the remote or out when not needed. This is to ensure that the application can provide good response time without the penalty of memory swapping from a remote server. However, swapping memory at the application level requires a different scheduling scheme to ensure that the intended functions and services of the applications are not affected. For example, interactive applications such as an Internet phone application, if being dormant in the remote server, must be swapped into main memory and run to answer a call within a few seconds; otherwise, the caller may perceive that the callee's phone is busy or unoperational. A new scheduling scheme based on the expected Quality of Service (QoS) of the services is also proposed in this paper to facilitate the application swapping.

The telecommunication industry can benefit from the OSGi Service Gateway specification by developing telecommunication equipment (such as web phone, Service Gateway, or telecom switch) using OSGi Framework, and optionally with the real-time for Java extension where needed. The validity of Virtual OSGi Framework proposed in this paper is demonstrated using a walk-through scenario approach based on network

appliance related applications to show that by using the proposed solution, the service gateway can provide more functionality to the consumer without additional cost to upgrade the hardware.

2 OSGi FRAMEWORK

This section describes the goals and motivations of OSGi Framework, its architecture features, and limitation of a typical OSGi Framework platform environment.

2.1 Goals and Motivations

Three key aspects of the OSGi mission are [10]:

- Multiple services,
- Wide-area networks, and
- Local networks and devices

The central component of the OSGi specification effort is the services gateway that acts as the platform for many communications-based services. The services gateway can enable, consolidate and mange voice, data, Internet and multimedia communications to and from the home, office and other locations.

In addition, the services gateway can also function as an application server for a range of high-value services such as energy measurement and control, safety and security services, health-care monitoring services, device control and maintenance, electronic commerce services and more. The gateway provides a focal point for service providers to deliver services to client devices on the local network(s). An implementation of the OSGi gateway may link client devices on the local network(s), such as energy meters, smart appliances or information appliances, to external service providers.

Using an OSGi gateway, a device can dynamically download a Java application, install it, and execute it on demand. When the application is complete, the gateway purges the application to make room for the next request.

2.2 Architecture

The OSGi gateway consists of two primary components: the OSGi Framework and the Services. The OSGi Framework provides a runtime framework that manages the loading, installation, activation, execution, and removal of applications, called services. The services are a set of useful pre-built and customizable applications. The deployment unit of the services is Java JAR file, which contains a deployment descriptor file called Bundle Manifest, and Java classes and resources to implement the services [3], [16], [20].

Figure 2 shows the architecture of the environment. The OSGi Framework is installed on a device running an embedded operating system. The OSGi Framework requires a Java runtime platform, such as Java 2 Micro Edition (J2ME) or Personal Java, or the standard J2SE JVM. The services can be stored anywhere on the network, and they are downloaded and executed as required.

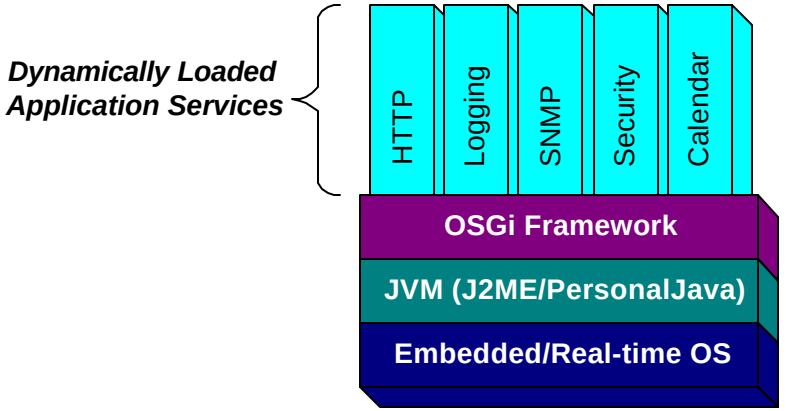

Figure 2. OSGi Framework Architecture [19]

2.3 Limitation of OSGi Framework

Although memory cost is getting lower, but because of small form factor and cost constraints, small telecommunications and embedded devices seldom are equipped with large memory size, and most likely without a

hard disk. For those devices that are equipped with larger memory size, it is likely that it will be outpaced by a growing number of dynamic loadable applications as we have observed that PC large memory size have also been outpaced by the memory requirements of applications that user wishes to run.

3 VIRTUAL OSGi FRAMEWORK

This section describes the goals and motivations of the Virtual OSGi Framework concept and the requirements (functional and non-functional). The non-functional requirements are used as the criteria for deriving architectural solution using NFR Framework methodology. The selected architectural solution is then described in detail using Rational Unified Process (RUP) [9] and Unified Modeling Language (UML) [8]. The UML notations used include use case diagram for requirements, class diagram for structural model, and state chart and collaboration diagram for behavioral model.

3.1 Goals and Motivations

Virtual OSGi Framework is aimed to address the problem that physical memory size limits the number of concurrent applications that may run on the OSGi Framework. The goal is to enable more concurrent applications to run on the services gateway.

3.2 Requirements

This section describes the requirements for the system that implements the Virtual OSGi Framework concept.

3.2.1 Functional Requirements (FR)

The main functional requirements are:

- System deploys services to services gateway as requested by Service Provider
- When target OSGi Framework runs out of memory, system automatically performs memory swapping using the procedure defined by the Virtual OSGi Framework architecture.

3.2.2 Non-Functional Requirements (NFR)

The system that implements Virtual OSGi Framework concept must meet the following non-functional requirements:

- Be compatible and maintain existing features and benefits of standard OSGi Framework
- Must provide acceptable responsiveness
- Must be practical and cost effective

3.2.3 Selection Among Architectural Choices

To determine the architectural solution that meets the non-functional requirements, a technique called NFR Framework is used to compare various architectural alternatives. The main concepts of the NFR Frameworks used in this paper are [11], [13]:

- *Ontology*: Softgoals and interdependencies (hence the name "softgoal interdependency graph" (SIG). A softgoal can be categorized as NFR Softgoal, Operationalizing Softgoal, or Claim Softgoal. NFR Softgoal (depicted as light cloud in SIG) represents a non-functional requirement. Operationalizing Softgoal (depicted as dark cloud in SIG) represents architectural alternative or component. Claim Softgoal (depicted as dotted cloud in SIG) represents the rationale used to justify the decision. To reduce clutter-ness on the diagram, this paper embeds Claim Softgoals in the rationale table. A softgoal may have interdependencies with other Softgoals. An independency represents a type and degree of contribution (-- for BREAK contribution, - for HURT contribution, + for HELP contribution, and ++ for MAKE contribution, etc.). When applied with NFR Softgoals, the Softgoal Decomposition shows lower level or different aspects of non-functional requirements that are concerned by the decision process. Multiple decomposed Softgoals may be considered using either logical AND (depicted as single line arc) or logical OR (depicted as double line arc). When applied with Operationizing Softgoals, the decomposition describes sub-components or different aspects or different aspects or factors of the Operationizing Softgoal. Softgoal Satisficing shows the degree of how an Operationizing Softgoal

satisficing an NFR Softgoal, optionally justified by the rationale described by a Claim Softgoal.

- *Epistemology*: Softgoal Decomposition. A decomposition can be applied to a type and/or a topic, in an object-oriented style, namely, classification/instantiation, aggregation/decomposition, generalization/specialization, views, etc. The decomposition can be observed along the interdependency directed lines and the evolution of the labels of the Softgoals, which are expressed in the following convention:

  Type [Topic1, Topic2, …]

  Where Type is a NFR and Topic is the system or another NFR to which the Type applies.

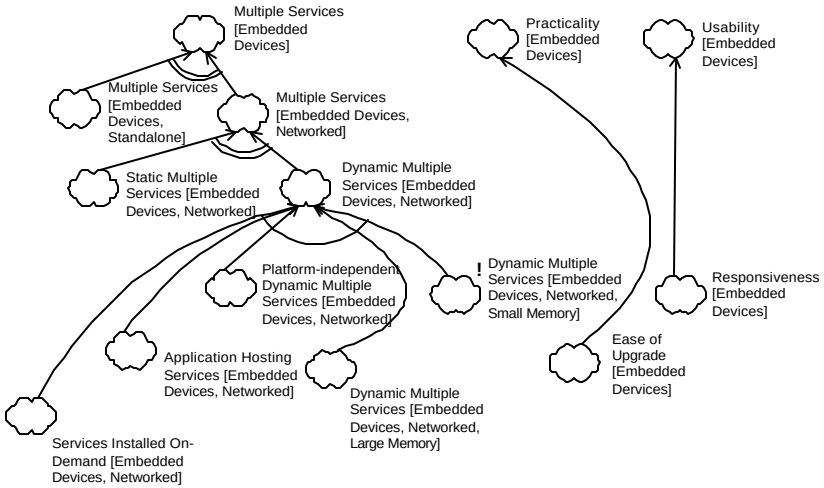

Figure 3. Softgoal Interdependency Graph for NFRs on Virtual OSGi Framework

Figure 3 shows the Softgoal Interdependency Graph for NFRs to be used to determine the architectural solution for the Virtual OSGi Framework concept.

To determine the architectural solution for Virtual OSGi Framework, the softgoals are further decomposed in Figure 4 with architectural alternatives or design elements depicted as dark cloud. The satisficing relationships are shown in diagrams with the degree of satisficing.

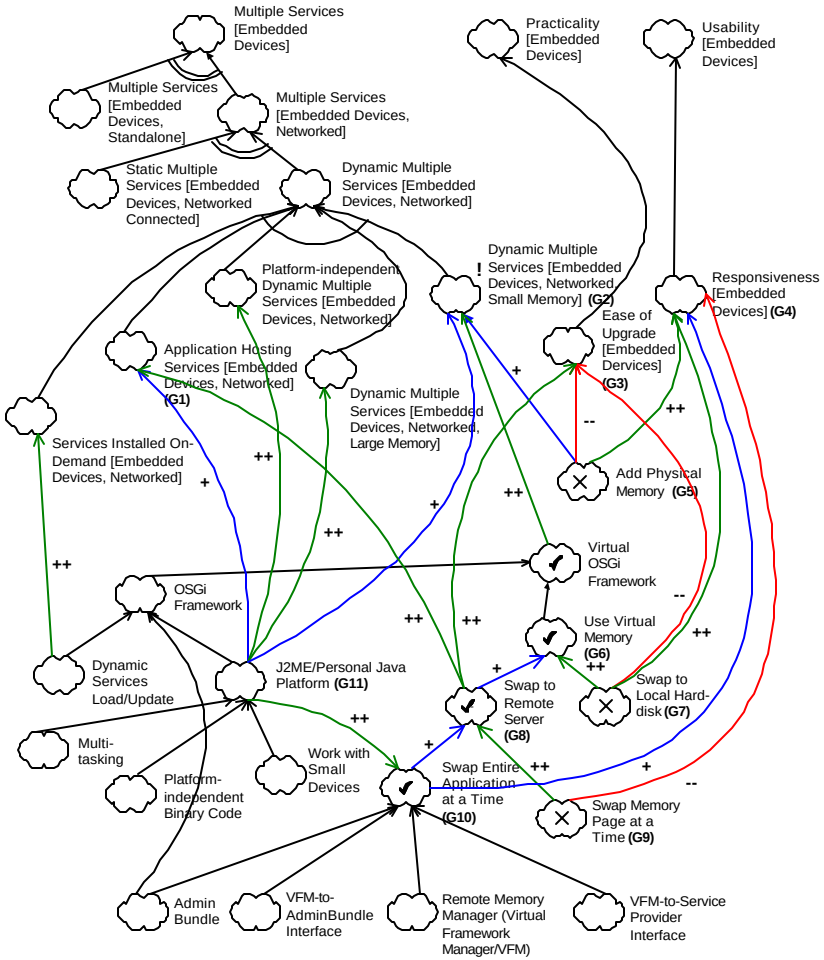

Figure 4. Softgoal Interdependency Graph for Virtual OSGi Framework

The selected architectural solution is to use virtual memory concept. However, instead of swapping at memory page level, the entire application is swapped all together. The rationale of selecting this solution is described in Table 1 below.

Softgoal	Satisficing	Rationale
G11	(+) → G2	OSGi Framework is likely to be deployed on Java JVMs, such as Java 2 Micro Edition (J2ME) or Personal Java, that are suitable for small devices. It is uncommon that these devices operate without any kind of virtual memory. Therefore, number of services run on OSGi Framework would be limited by physical memory. With a device with small memory, such as Web Phone or PDA, this limitation is more severe than larger devices that are equipped with more memory. This is the limitation being addressed by this paper.
G5	(+) → G2	Adding more physical memory to the device would provide the ideal result, that is, more fast memory to host more services. However, larger services and more services would eventually run out of the new larger memory.
	(++) → G4	Adding more physical memory provides the best responsive system.
	(--) → G3	However, adding physical memory is not a practical option as most small telecommunication devices are designed to be small and inexpensive. They often do not offer a memory upgrade option or difficult to do so. Therefore, this design option is denied.
G7	(++) → G6	To support the use of virtual memory, the ideal solution is to use a local mass storage such as hard disk for storing swapped out memory because local I/O is well acceptable for virtual memory technique.
	(++) → G4	Because the data transfer latency is low for hard disk. Using hard disk for virtual memory would yield a very responsive system.

	(--) → G3	However, using hard disk for memory swapping is not a practical solution, as most small embedded devices do not have hard disk. It is also often difficult to add additional hardware such as hard disk to these devices.
G8	(+) → G6	It is possible to swap inactive memory to a remote server, although not ideal due to higher network transmission latency.
	(++) → G3	Using a remote server to facilitate the virtual memory mechanism is the most practical option, as it does not require any hardware upgrade to the embedded device.
	(++) → G1	Because a remote server (Virtual Framework Manager) is used to manage the deployment of services, it can also eliminate the need for Service Provider to perform deployment management function such as periodic deployment of some services that need to run regularly based on schedule.
G9	(++) → G8	The typical virtual memory mechanism would swap a small chunk memory (memory page) at a time based on the virtual memory address space. This is the desirable size for memory swapping.
	(--) → G4	However, because the secondary storage is on a remote server, which data transfer would suffer tremendous delay due to network transmission latency that is much greater than that of a local hard disk. As the result, the active application may yield unresponsive when its memory pages are being swapped in and out between memory on the OSGi Framework and the remote server. *Claim*: Network latency causes page-swapping applications to be unresponsive for interactive or time-critical applications.
G10	(+) → G8	In order to achieve acceptable responsiveness, the entire application image should reside in the main memory of the OSGi Framework. When the application can be swapped out to secondary storage on the server, the service is terminated (similar to memory swapped out effect). Transferring the entire application would take longer than swapping only a memory page; however, it is necessary to make a virtual memory concept with embedded environment.
	(+) → G4	Having the entire application image in the local memory would yield good responsiveness; however it suffers a slight network latency delay during the initial loading of the application.
G11	(++) → G10	One main reason that swapping the entire application is a feasible solution because 1) Programs in Java byte-code are much more compact than other languages, 2) the size of services for embedded device tend to be small as they tend to be communication-based and control-based applications. If they have user interface, the user interface tends to be simpler and much lighter weight than desktop counter part applications. For example, many useful practical services developed at Digital Pockets LLC were found to be in the range of 200KB or smaller. For 1 Mbps connections as found in DSL or 3G wireless technologies, these services would take less than 2 seconds to swap in the entire application, which is reasonable for even interactive services. *Claim*: Good design and Java byte code compactness enable OSGi services to be small enough for acceptable loading delay.

Table 1. Virtual OSGi Framework Architectural Solution Rationale

The Virtual OSGi Framework concept is depicted in Figure 5. The Virtual Framework Manager provides a perception of a larger deployment OSGi Framework platform on the target services by maintaining the current applications on a remote server and swap them to run on the gateway as needed.

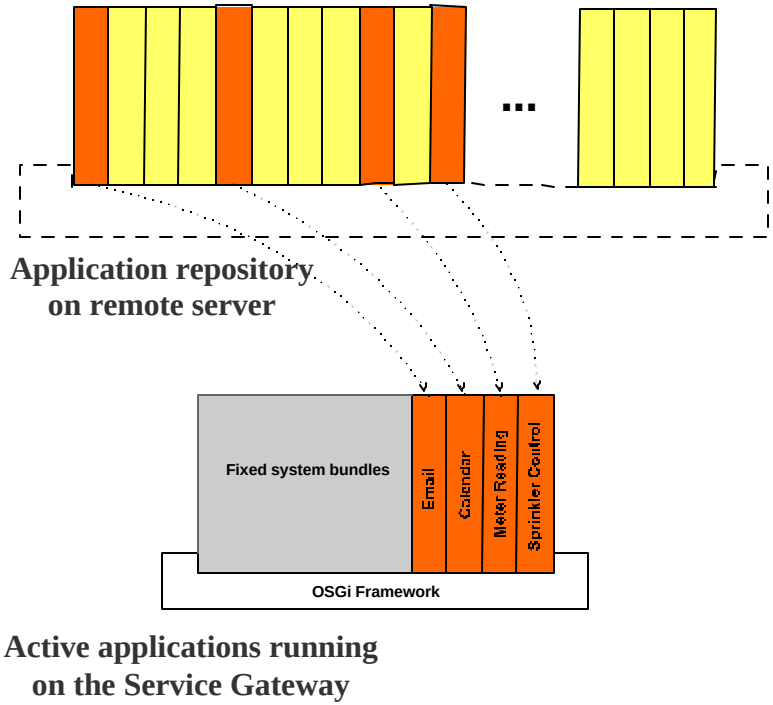

Figure 5. Application-based Swapping Employed by Virtual OSGi Framework Architecture

3.2.4 Revised Functional Requirements

This section presents the revised functional requirements based on the selected architectural solution that is using a remote server to provide virtual memory effect on the OSGi Framework. The requirements are presented using UML Use Case Model as depicted in Figure 6 and described in Table 2.

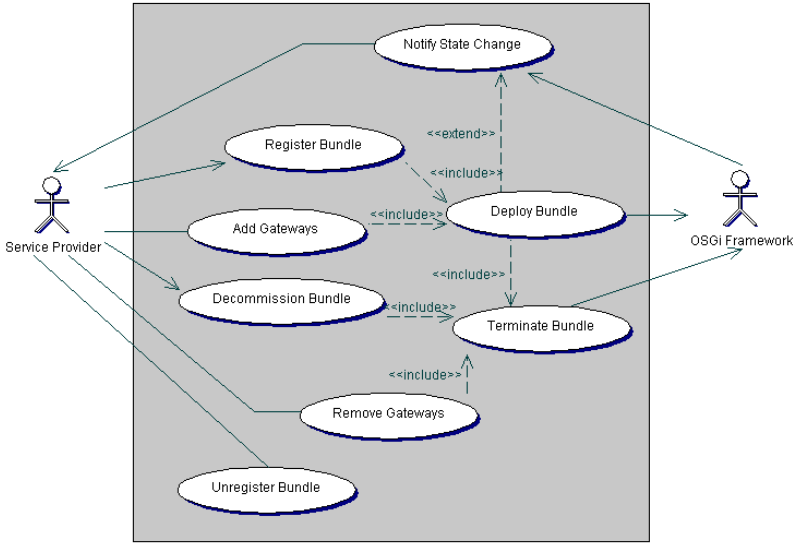

Figure 6. Virtual OSGi Framework System Use Case Diagram

Use Case	Description
Register Bundle	Service Provider registers a new service bundle to be deployed on one or more services gateway(s). The registered bundle is also given a deployment policy to specify how the bundle should be managed. This use case invokes *Deploy Bundle* use case to deploy the new service to the target services gateway(s).
Add Gateways	Service Provider specifies one or more services gateway(s) that should receive an existing registered bundle.
Remove Gateways	Service Provider specifies one or more services gateway(s) that should no longer run an existing registered bundle.

Deploy Bundle	This is an abstract use case. It is invoked by other use cases to deploy a new or sleeping bundle. This use case checks if there is any other Service(s) that this bundle depends. If so, VFM deploys bundles that provided dependent services before deploying this Bundle so OSGi may resolve the Bundle dependency successfully.
Terminate Bundle	This is an abstract use case. It is invoked by other use cases to terminate a registered bundle, which may be running on one or more services gateway(s).
Notify State Change	OSGi Framework notifies the local AdminBundle, which turns notifies the VFM that a bundle residing on the gateway has changed its life-cycle state. If all instances of the bundle running on all target gateways report state changes that causes a change to the master bundle on VFM, VFM would send a notification to the Service Provider using the interface URL provided by the original request.
Unregister Bundle	Service Provider requests VFM to unregister a bundle it has previously registered.

Table 2. Virtual OSGi Framework System Use Cases Description

3.3 Virtual OSGi Framework Structural Model

To implement the Virtual OSGi Framework concept, a new software system is needed to manage the virtual space on the remote server as well as interfacing with the OSGi Framework on the services gateway. The system consists of the following components as depicted in shaded color in Figure 7 and described in Table 3:

- Virtual Framework Manager (VFM)
- Administration Bundle (AdminBundle)
- Service Provider Interface
- Gateway Interface

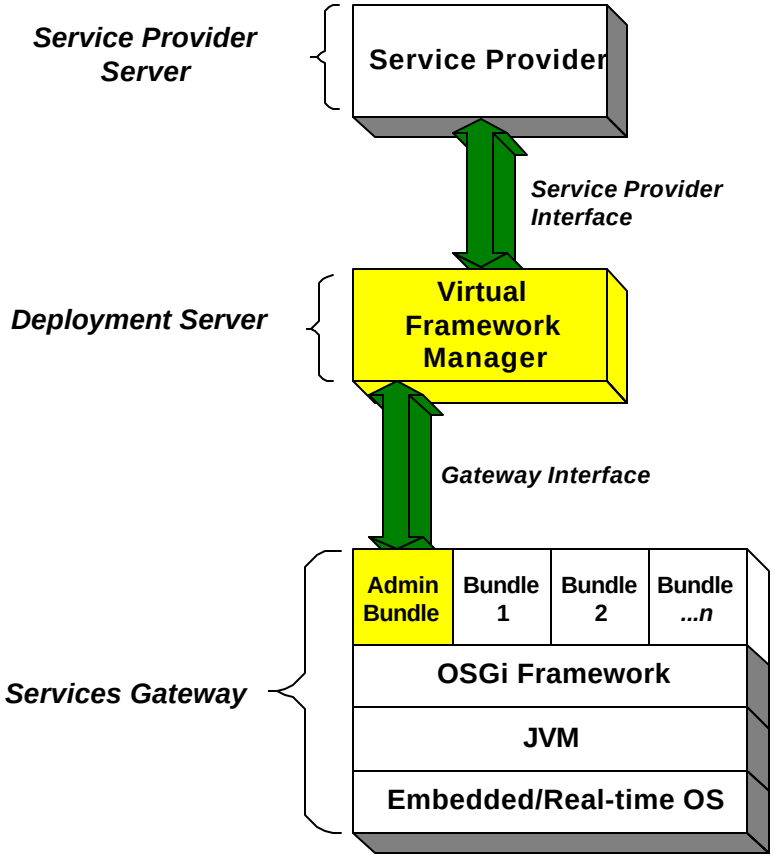

Figure 7. Virtual OSGi Framework Components

Component	Description
Virtual Framework Manager (VFM)	VFM is a remote server that deploys services as requested by Service Provider. The OSGi services are packaged in the deployment unit called bundle, which contains a JAR comprising Java classes that make up the services and a text file that describes properties of the bundle called Manifest. VFM uses the deployment

	policies provided by the Service Provider to determine when swap applications in and out.
Administration Bundle (AdminBundle)	AdminBundle is an OSGi service running on the target OSGi Framework. It acts as a remote management agent to facilitate the deployment and management of OSGi Bundles. It can also be developed to monitor requests or messages sent by consumer devices connected to the services gateway through the local area network. For example, AdminBundle may detect Voice over IP (VoIP) messages from connected IP Phone; it then notifies the VFM that VoIP service should be swapped in to service the request.
Service Provider Interface	This interface supports data exchange and requests between Service Provider and VFM.
Gateway Interface	This interface supports data exchange and requests between VFM and AdminBundle.

Table 3. Virtual OSGi Framework Components Description

3.3.1 Virtual Framework Manager (VFM)

This section presents the structural model of the VFM, the main component of Virtual OSGi Framework architecture. The server is used as the remote memory manager as well as an OSGi service deployment manager.

3.3.1.1 VFM Architecture

The VFM can be further decomposed to sub-components is depicted in Figure 8 and described in Table 4.

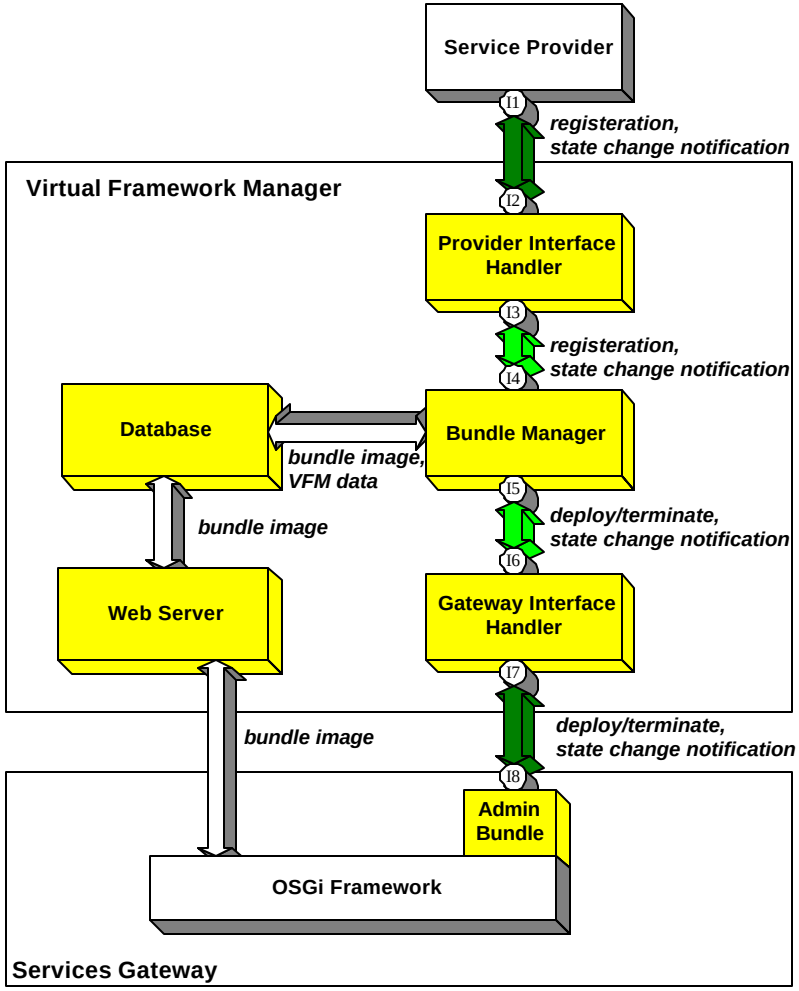

Figure 8. Virtual Framework Manager Architecture

Module	Description
Provider Interface	This module interacts with Service Provider systems for bundle registration/de-registration and bundle life-cycle state

Handler	change notification. It provides standard interface using common protocol and mechanism; for example, the information between Service Provider and VFM may be exchanged using XML over HTTP protocol.
Bundle Manager	This is the main module that manages the bundle deployment and keeps track of bundles running on the target Services Gateways, as well as bundles that are swapped out and waiting to be executed. It uses Database to persistently store bundle information and the executable image to be retrieved by Services Gateway through the Web Server.
Gateway Interface Handler	This module encapsulates the knowledge of OSGi Framework API for VFM. It deploys bundles requested by the Bundle Manager using OSGi API. It monitors the notification that be reported by the Service Gateway. It receives and forwards any notifications to the Bundle Manager.
Database	The Database is used as persistent storage for bundle information and executable image.
Web Server	When VFM installs a bundle on a Services Gateway, it specifies the source of the bundle executable image in URL The OSGi Framework on the Services Gateway would in turn use HTTP protocol to download the bundle image from a remote Web Server using the specified URL. The Web Server through a database connection retrieves the bundle image in the form of Java JAR and forward to the target Service Gateway.

Table 4. VFM Components Description

3.3.1.2 VFM Information Model

This section presents the information to be maintained by the VFM as shown in Figure 9. Detailed description of each class is presented in Table 5.

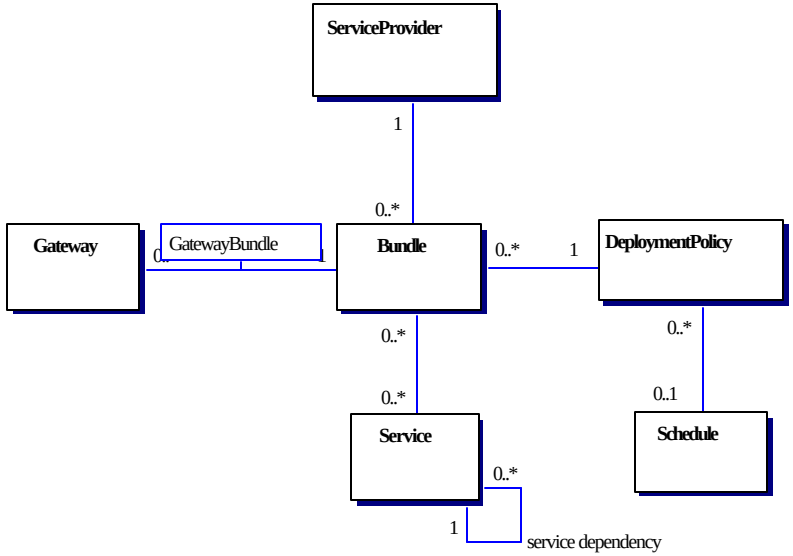

Figure 9. Virtual Framework Manager Information Model

Class	Description
Service Provider	Service Provider represents the organization that provides OSGi bundles and services. **Attribute:** *id* is the unique identifier of Service Provider; *certificate* is the secure digital certificate used to authenticate the Service Provider; *name* is the textual name of this Service Provider. **Relationship:** With *Bundle*: Service Provider may provide one or more *Bundle*(s) to be deployed to services gateways.
Bundle	Bundle is the deployment unit for OSGi applications and services. Each Bundle may perform its function upon the deployment or it may be developed to be component where it provides Services to other

	Bundles. **Attribute:** *vfmAssignedBundleID* is the ID generated by VFM to uniquely identify this Bundle; *name* is the textual name of this bundle as described in the bundle manifest (see Figure 10); *description* is the textual description as described in the bundle manifest (see Figure 10); *bundleJAR* is the bundle deployment image or location inside the internal VFM database where the image is stored; *state* is the current life-cycle state of this bundle (see Figure 11); *notificationURL* is the Service Provider HTTP URL for receiving bundle state change notification sent by VFM. **Relationship:** With *ServiceProvider*: Bundle must be provided by a ServiceProvider. With *DeploymentPolicy*: Bundle must be given a *DeploymentPolicy*. With *Service*: Bundle may contain one or more *Service*(s). With *Gateway*: Bundle may be deployed to one or more *Gateway*(s).
DeploymentPolicy	DeploymentPolicy defines the quality of service and how the bundle should be managed from the deployment perspective. **Attribute:** *QoS* is the expected quality of service for this bundle (see Table 6); *interruptible* indicates whether this Bundle can be preempted and swapped out before it is complete. **Relationship:** With *Bundle*: DeploymentPolicy may be assigned to one or more Bundle. With *Schedule*: DeploymentPolicy may be associated with a repeatable *Schedule*.
Schedule	Schedule defines how Bundle should be periodically re-deployed (see Table 8). The schedule specification is based on Unix cron-job definition. **Attribute:** *minute* specifies at what minutes of the hours the Bundle should be re-deployed; *hour* specifies what hours of the day for re-deployment; *dayOfMonth* specifies what days of the month for re-deployment; *month* specifies what months of the year for re-deployment; *dayOfWeek* specifies what days of the week for re-deployment. **Relationship:** With *DeploymentPolicy*: Schedule may be assigned to one or more DeploymentPolicy.
Service	Service is a functionality that is provided by a Bundle to other Bundle(s). The Service can be registered with the OSGi Framework so it can be invoked by other Bundle(s). The interface to the Service is through Java interface. **Attribute:** *name* is the textual name described in the Bundle manifest; *version* is the version number of this implementation of the Service; *javaInterface* is the exportable Java interface [12] (see *Export-Service* statement in Figure 10). **Relationship:** With *Bundle*: Service must be provided by a Bundle. With *Service*: A Service may depend on one or more other Service(s).
Gateway	Gateway represents the OSGi services gateway. **Attribute:** *ipAddress* is the unique IP address of this Gateway; *adminBundleURL* is the HTTP URL of the AdminBundle that VFM uses as a remote agent to

	facilitate the Bundle deployment. **Relationship:** With *Bundle*: Gateway may host and run one or more *Bundle*.
GatewayBundle	This is a UML association that represents the Bundle instance being deployed on a Gateway. **Attribute:** *state* is the current life-cycle state of the Bundle instance as defined by GatewayBundleState depicted in Figure 12; *priorityOffset* is the offset value may be used by VFM to increase the effective priority when the Bundle has been sleeping longer so that it would be given more consideration to be swapped in to run on the Gateway; *gwAssignedBundleID* is the unique identifier assigned by the Gateway upon deployment. **Relationship:** None (it is already an association class between Gateway and Bundle).

Table 5. VFM Class Description

```
Bundle-Name: AdminBundle
Bundle-Description: Remote Administration Bundle
Bundle-Vendor: Digital Pockets LLC
Bundle-Version: 1.0
Bundle-DocURL: http://www.digitalpockets.com
Bundle-Activator: com.digitalpockets.osgi.adminbundle
Import-Package: org.osgi.service.http,
                javax.servlet; specification-version=2.1.1,
                javax.servlet.http; specification-version=2.1.1
Import-Service: org.osgi.service.http.HttpService
Export-Package: com.digitalpockets.osgi
Export-Service: com.digitalpockets.osgi.admin
```

Figure 10. An Example of OSGi Bundle Manifest

Policy	Value	Description
Quality of Service (QoS)	Critical	For critical services that need to be started as soon as possible. These may be health or security related application that should be started within a second or two.
	Interactive	For services that have user interaction element, they should be responsive to user request. Examples of these applications include device console, IP Phone or home entertainment. These applications should be started within a few seconds.
	High Priority Background	For services that are not critical and do not have user interaction, they may be able to tolerate some delay up to several minutes, such as furnace control or email delivery system.
	Low Priority Background	Some low priority applications may be able tolerate delay up to hours; for example, sprinkler control or once-a-month billing application.

Table 6. Quality of Service Deployment Policy

Policy	Value	Description
Scheduling	Once	For services that do not need repeated deployment, such as control related applications like door or electrical control applications. Once the services are complete and self-terminated, they will not be automatically re-deployed by VFM.
	Repeating	For services that need periodic re-deployment. After the termination, they will be re-activated by VFM based on the schedule defined by this policy.

Table 7. Schedule Deployment Policy

Policy	Value	Description
Preemption	Yes	Bundle can be interrupted and swapped out at any time. Interruptible applications are written to handle any abrupt interruption. Once swapped out, Virtual Framework Manager uses Launching characteristic to determine when the bundle should be swapped in for execution again.
	No	Bundle cannot be interrupted before completion.

Table 8. Preemption Deployment Policy

3.3.2 Admin Bundle

As shown early in Figure 7, Admin Bundle is an OSGi Service running on the target OSGi Framework. It acts as a remote agent to facilitate VFM with the service/bundle management.

3.3.2.1 Information Model

Admin Bundle is a stateless service. It has no need to maintain any information or state about the deployed services. All operations are atomic and have no inter-dependency among them. Therefore, information or state maintenance is not required.

3.4 Virtual OSGi Framework Behavioral Model

This section presents the dynamic behavioral model of the Virtual OSGi Framework system using UML state transition diagrams and collaboration diagrams. The state transition diagrams describe the life cycle of two stateful classes: Bundle and GatewayBundle classes. The collaboration diagrams describes system-wide behaviors based on system level use cases depicted in Figure 6. The names of the messages sent from one object to another on the collaboration diagrams are based on interface operations defined in Section 3.5 and the OSGi Service Gateway specification [16].

3.4.1 Bundle Object Life-Cycle

Each registered Bundle is tracked when it goes through its life-cycle state transitions as depicted in Figure 11 and described in 10. This current state represents the aggregate state of all instances of the bundle being deployed on the target services gateways.

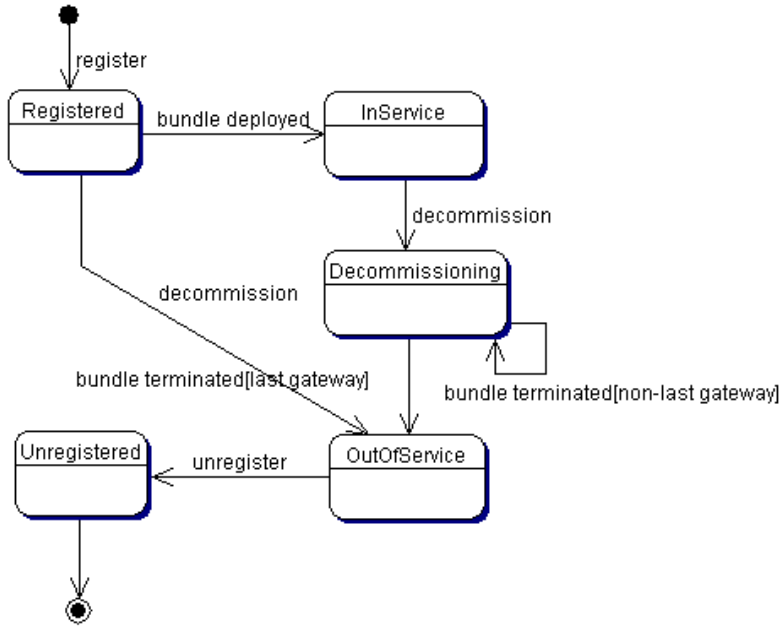

Figure 11. Bundle Object Life Cycle (BundleState)

State	Description
Registered	The bundle has been registered and scheduled for deployment.
InService	The bundle has been deployed to at least one target services gateway.
Decommissioning	Service Provider has requested to stop the bundle. The bundle is being terminated from the target services gateway by the VFM.
OutOfService	The bundle has been terminated from all services gateway.
Unregistered	Service Provider requests VFM to remove the bundle. This is the terminal state of BundleState.

Table 9. Bundle States Description

3.4.2 GatewayBundle Object Life-Cycle

Each instance of bundle deployment is tracked through its state transition as depicted in Figure 12 and described in Table 10.

State	Description
Scheduled	The bundle has been registered and scheduled for deployment to the target services gateway.
Deployed	The bundle is now deployed to the target services gateway. OSGi Bundle must be explicitly installed (Installed state) and started to be active and running (Started state). To remove the bundle from the gateway, the bundle must be stopped (Stopped state), then uninstalled (Uninstalled state).
Sleeping	The bundle is preempted and swapped out (removed) from the services gateway.
Terminated	The bundle is no longer needed for deployment.

Table 10. GatewayBundle States Description

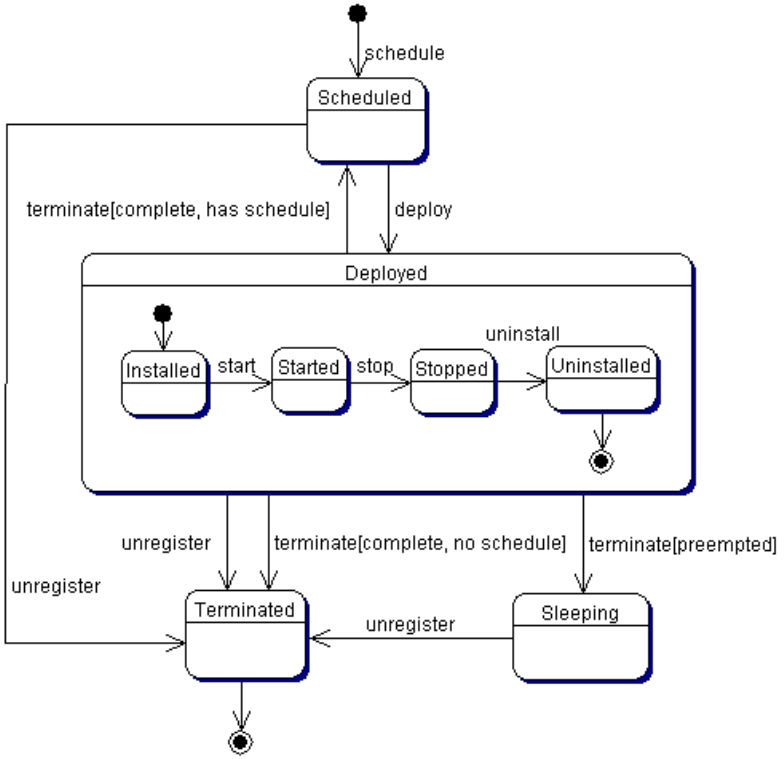

Figure 12. GatewayBundle Life Cycle

3.4.3 Register Bundle Use Case

This use case starts when a Service Provider sends a request to VFM to register a new Bundle for deployment to one or more services gateways. VFM verifies the provided information. If valid, VFM stores the information and schedules the Bundle for deployment. If the target services gateways do not have enough available memory for the Bundle, VFM initiates the swap out process (*Terminate Bundle* abstract use case) to terminate one or more lower priority Bundles to free up the memory for the new Bundle. When the Bundle is installed (by *Deploy Bundle* abstract use

case) to the gateway, the underlying OSGi Framework would automatically retrieves the Bundle JAR file from the Web Server specified in the URL provided by the VFM. Detailed behavior is depicted in Figure 13.

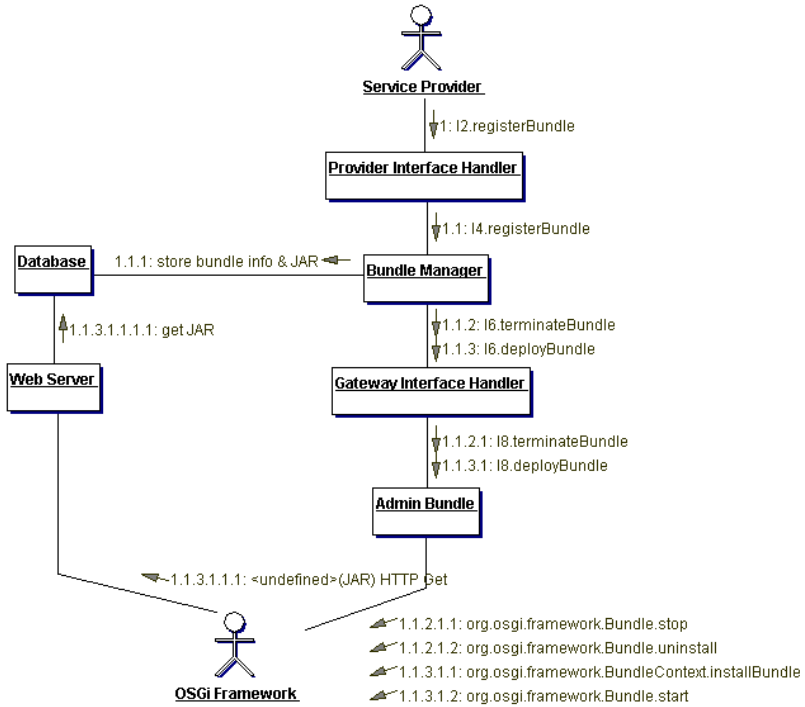

Figure 13. Register Bundle Use Case Collaboration Diagram

3.4.4 Add Gateways Use Case

This use case starts when a Service Provider sends a request to VFM to request to add one or more services gateways to the deployment of a previously registered Bundle. VFM verifies the information provided by the provider. If valid, it schedules the bundle for deployment to the new services gateways, which will be carried out by *Deploy Bundle* use case. The detailed behavior is depicted in Figure 14.

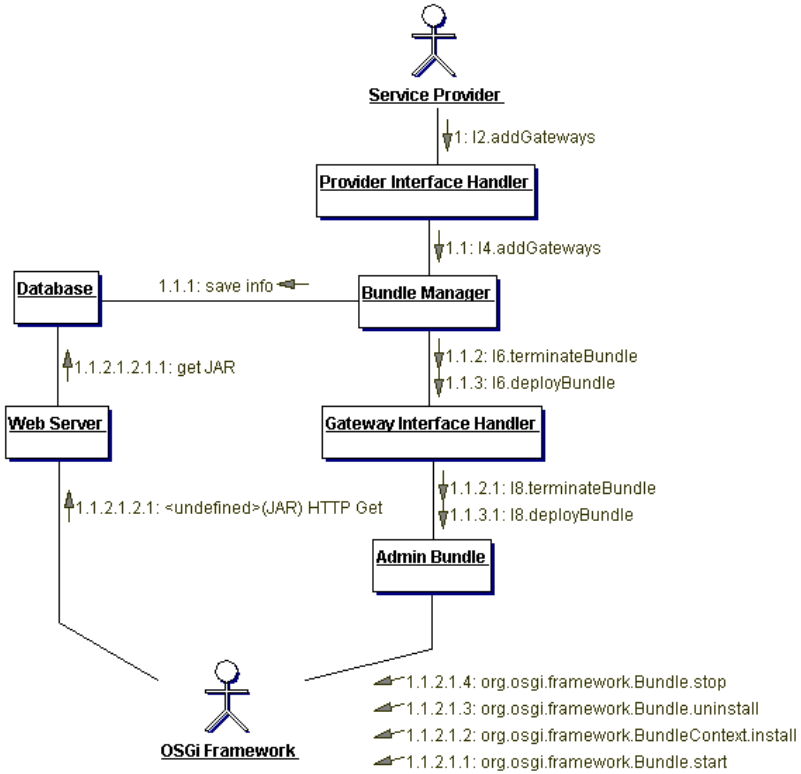

Figure 14 Add Gateways Use Case Collaboration Diagram

3.4.5 Remove Gateways Use Case

This use case starts when a Service Provider sends a request to VFM to remove one or more services gateways from the deployment list of a previously registered Bundle. VFM verifies the information provided. If valid, it initiates the process to terminate the bundle from the specified services gateways (*Terminate Bundle* use case). Detailed behavior is depicted in Figure 15.

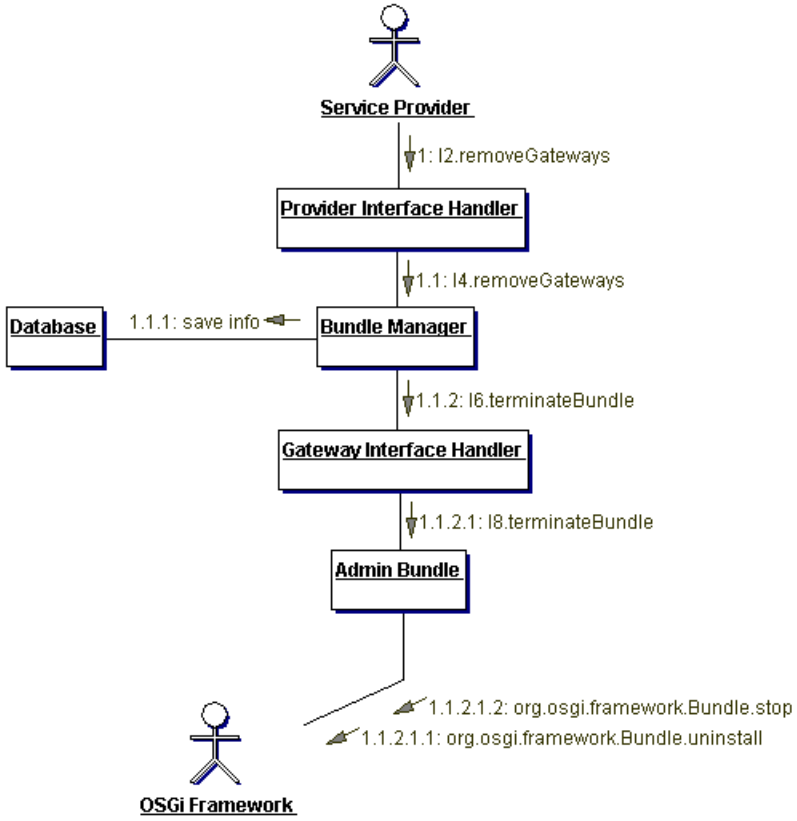

Figure 15 Remove Gateways Use Case Collaboration Diagram

3.4.6 Decommission Bundle Use Case

This use case starts when a Service Provider sends a request to decommission a previously registered Bundle. VFM verifies the provided information. If valid, VFM starts the process to terminate the bundle from all target services gateways. Detailed behavior is depicted in Figure 16.

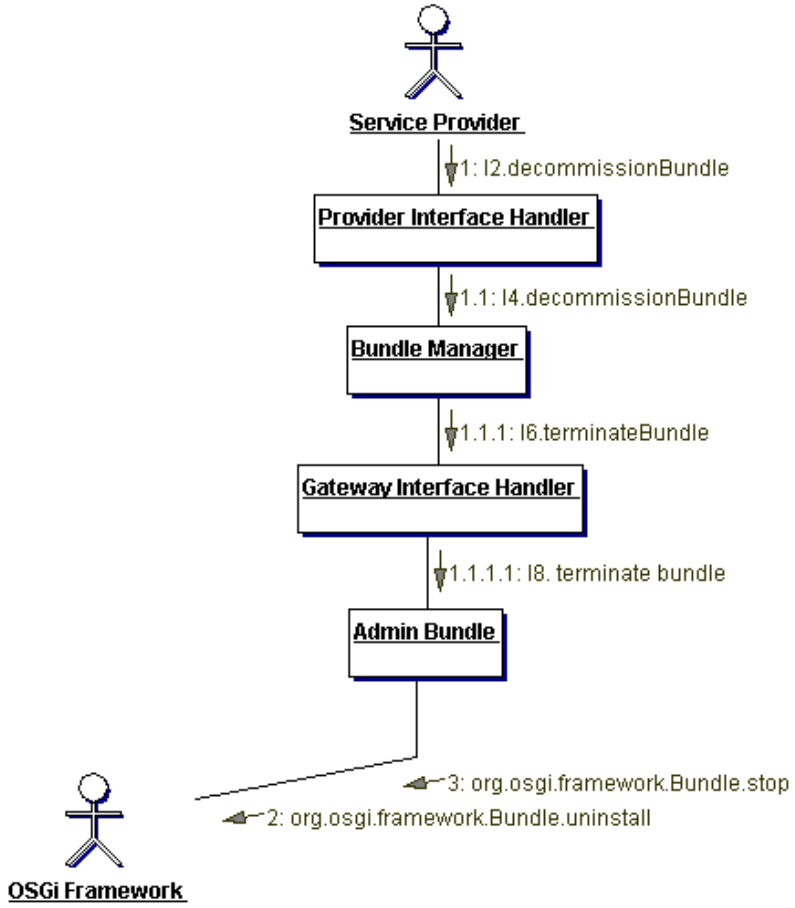

Figure 16 Decommission Bundle Use Case Collaboration Diagram

3.4.7 Unregister Bundle Use Case

This use case starts when a Service Provider sends a request to unregister a previously decommissioned Bundle. VFM verifies the provided

information, including whether the Bundle is in Decommissioned state. If so, VFM removes the Bundle from managed Bundle repository. Detailed behavior is depicted in Figure 17.

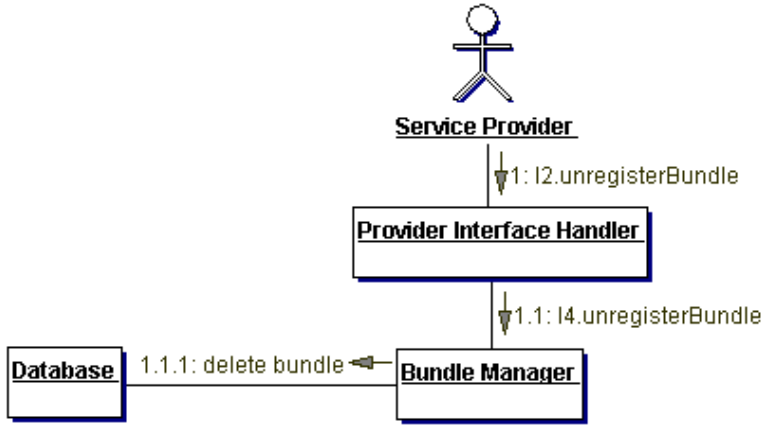

Figure 17 Unregister Bundle Use Case Collaboration Diagram

3.4.8 Notify State Change Use Case

This use case starts when OSGi Framework sends a notification to the local AdminBundle to report that a bundle has changed state. AdminBundle forwards the notification to the VFM, which in turn updates the state of the associated GatewayBundle to reflect the new state of this Bundle instance. The master Bundle object is also checked if it needs to change state as defined by BundleState definition in Table 9. If Bundle state is changed, a notification is sent to the Service Provider who deployed this Bundle. Detailed behavior is depicted in Figure 18.

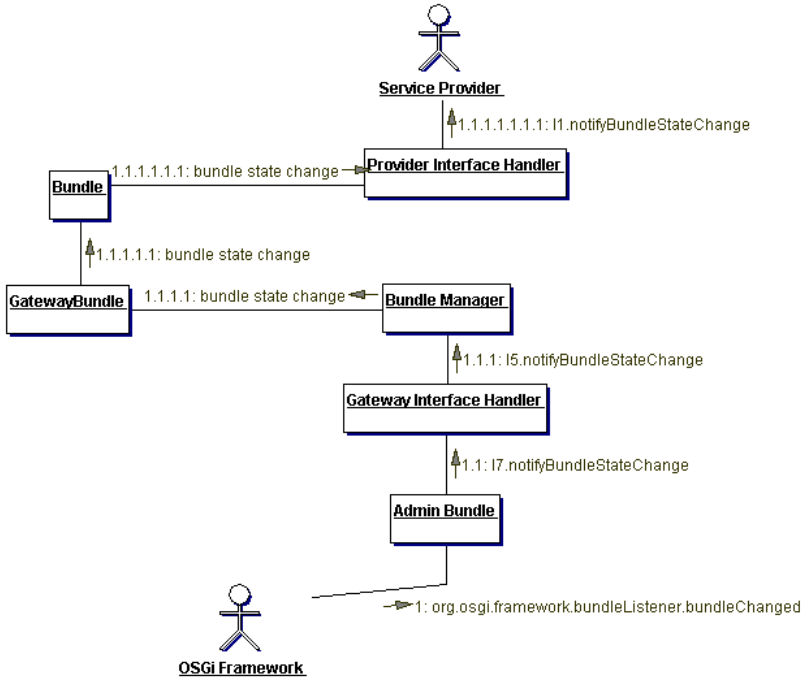

Figure 18. Notify State Change Use Case Collaboration Diagram

3.5 Virtual OSGi Framework Interface Specification

This section presents the details of interfaces between components. The interfaces are labeled *Interface 1* to *Interface 8* as depicted in Figure 8. The interfaces are presented from semantic perspective, without the specific details of the protocol. Each operation of the interfaces is presented with pre-conditions and post-conditions, and input, output, and exception specifications. Pre-conditions describe the states of the system before the operation. Post-conditions describe the states of the system after the operation if successful. Input specifies the operation parameters. Output specifies the return parameters of the operations. Exception specifies the error conditions reported if the post-conditions cannot be met.

3.5.1 Interface I1

This interface is protocol dependent based on the protocols to be supported by the system.

3.5.1.1 *notifyBundleStateChange Operation*

Pre-conditions: None
Post-conditions: None

Input:
Integer *bundleID*,
String *oldState*,
String *newState*
Output:
None
Exception: None

VFM sends a notification to Service Provider to report that a bundle has changed its life-cycle state.

Input: *bundleID* is the unique ID of the bundle whose state has changed; *oldState* is the previous state before the transition; *newState* is the new state after the transition.

Output: None.

Exception: None

3.5.2 Interface I2

This interface is protocol dependent based on the protocols to be supported by the system. It allows Service Provider to make requests using different protocol.

3.5.2.1 registerBundle Operation

Pre-conditions: None.
Post-conditions: Bundle related information and executable image (JAR) are created, stored, and scheduled to be deployed as specified by the deployment policy.

Input:
Integer *providerID*,
Certificate *providerCertificate*,
String *bundleURL*,
DeploymentPolicy *bundlePolicy*,
StringList *gatewayIPs*,
String *notificationListenerURL*
Output:
Integer *bundleID*
Exception:
InvalidProvider,
InvalidBundleLoc,
InvalidBundlePolicy,
InvalidGateways(StringList *gatewayIPs*),
InvalidEventListener,
BundleNotDeployable

Service Provider registers a new bundle for deployment.

Input: *providerID* uniquely identifies the Service Provider making the request; *providerCertificate* securely authenticates the provider using digital certificate; *bundleURL* is the URL where the bundle JAR file is located; *bundlePolicy* specifies policy for managing the bundle; *gatewayIPs* contains IP addresses of one or more target services gateways for the deployment of this bundle; *notificationListenerURL* is the URL of HTTP web page to post bundle state change notification unless *notificationListenerURL* is undefined.

Output: *bundleID* is assigned identifier of the bundle once it has been successfully registered.

Exception: *InvalidProvider* indicates that the provided service provider certificate is invalid; *InvalidBundleURL* indicates that the provided URL does not exist or not reachable; *InvalidBundlePolicy* indicates that the provided deployment policy is missing or invalid; *InvalidGateways* indicates that the IP addresses of some target gateways are invalid or not reachable; *InvalidEventListener* indicates the event listener web page is invalid or not reachable; *BundleNotDeployable* indicates that VFM has

verifies and determined that the bundle is not deployable, for instance the service with the specified service signature already registered and deployed by other Service Provider.

3.5.2.2 addGateways Operation

Pre-conditions: The bundle exists and was registered by this Service Provider.
Post-conditions: Bundle is scheduled to be deployed to the new services gateway.

Input:
Certificate *providerCertificate*,
Integer *bundleID*,
StringList *gatewayIPs*
Output: None
Exception:
InvalidProvider,
InvalidBundle,
InvalidGateways(StringList *gatewayIPs*)

Service Provider requests to deploy a previously registered bundle to one or more services gateways.

Input: *providerCertificate* authenticates that the Service Provider is indeed the same provider that previously registered the Bundle; *bundleID* is identifier of the bundle for this request; *gatewayIPs* contains IP addresses of one or more services gateways to deploy the bundle.

Output: No output is required.

Exception: *InvalidProvider* indicates the provided certificate is invalid or the provider is not the provider that has originally registered the bundle; *InvalidBundle* indicates the bundle ID is unknown or not currently active; *InvalidGateways* indicates that the gateways with IP addresses defined in *gatewayIPs* are invalid or unreachable.

3.5.2.3 removeGateways Operation

Pre-conditions: The bundle exists and was registered by this Service Provider.
Post-conditions: Bundle is scheduled to be removed from the specified services gateways.

Input:
Certificate *providerCertificate*,
Integer *bundleID*,
StringList *gatewayIPs*
Output: None
Exception:
InvalidProvider,
InvalidBundle,
InvalidGateways(StringList *gatewayIPs*)

Service Provider requests to remove one or more services gateways from bundle deployment of a previously registered bundle. The gateways would no longer host the specified bundle.

Input: *providerCertificate* authenticates the Service Provider making sure the request is indeed the original provider that registered; *bundleID* is identifier of the desirable bundle for this request; *gatewayIPs* contains IP address of one or more services gateways to deploy the bundle.

Output: No output is required.

Exception: *InvalidProvider* indicates the provided certificate is invalid or the provider is not the provider that has originally registered the bundle; *InvalidBundle* indicates the bundle ID is unknown or not currently active; *InvalidGateways* indicates that the gateways with IP addresses defined in *gatewayIPs* are invalid or unreachable.

3.5.2.4 decommissionBundle Operation

Pre-conditions: The bundle exists and was registered by this Service Provider.
Post-conditions:
The bundle is in *Decommissioning* state.

Input:
Certificate: *providerCertificate*,
Integer *bundleID*,
Output: None
Exception:
InvalidProvider,
InvalidBundle

Service Provider requests to stop the service of a particular Bundle. After the control is returned to the request, VFM starts to terminate the bundle from all target services gateways of this Bundle.

Input: *providerCertificate* authenticates that the provider making the request is indeed the original provider that registered the Bundle; *bundleID* identifies the bundle to be decommissioned.

Output: no output is required.

Exception: *InvalidProvider* indicates the provided certificate is invalid or the provider is not the provider that has originally registered the bundle; *InvalidBundle* indicates the bundle ID is unknown or not currently active.

3.5.2.5 unregisterBundle Operation

Pre-conditions:
The bundle is in *Decommissioned* state.
Post-conditions:
The bundle is removed and no longer managed by VFM.

Input:
Certificate: *providerProvider*,
Integer *bundleID*

Output: None
Exception:
InvalidProvider,
InvalidBundle
InvalidBundleState

Service Provider requests to remove a particular bundle from the system.

Input: *providerCertificate* authenticates that Service Provider is the correct owner of the Bundle; *bundleID* identifies the bundle the provider wishes to unregister.

Output: no output is required.

Exception: *InvalidProvider* indicates the provided certificate is invalid or the provider is not the provider that has originally registered the bundle; *InvalidBundle* indicates the bundle ID is unknown or not currently active; *InvalidBundleState* indicates the bundle is not in the correct state for this operation.

3.5.3 Interface I3

3.5.3.1 notifyBundleStateChange Operation

Pre-conditions: None
Post-conditions: None

Input:
String *providerListenerURL*,
Integer *bundleID*,
String *oldState*,
String *newState*
Output: None
Exception: None

Bundle Manager module sends a notification to a provider's listener located at *providerListenerURL* to indicate that the bundle identified by *bundleID* has performed a state transition from *oldState* to *newState*.

3.5.4 Interface I4

Semantically, I4 is the same as Service Provider Interface I2. The difference is that I4 is an internal interface, where I2 is protocol specific interface based on the protocols to be supported by the system. For instance, HTTP, IIOP, or RMI.

3.5.5 Interface I5

3.5.5.1 notifyBundleStateChange Operation

Pre-conditions: None
Post-conditions: None

Input:
String *gatewayIP*,,
Integer *bundleID*,
String *newState*
Output: None
Exception: None

Bundle Manager is notified that a bundle (identified by *bundleID)* on a gateway (identified by *gatewayIP*) has changed bundle state to *newState.*

3.5.5.2 requestService Operation

Pre-conditions: None
Post-conditions: None

Input:
String *serviceName*
Output:
String *bundleURL*
Exception:
UnknownService

Bundle Manager is requested for a bundle (would be available at location *bundleURL*) that supports OSGi service identified by *serviceName*.

3.5.6 Interface I6

3.5.6.1 deployBundle Operation

Pre-conditions: None.
Post-conditions: The bundle is successfully installed and started on the target services gateways.

Input:
String *bundleURL*,
StringList *gatewayIPs*
Output:
List of {String *successfulGatewayIPs*, Integer *bundleID*},
List of {String *failedGatewayIPs*, String *description*}
Exception:
InvalidGateways(StringList *gatewayIPs*)

Bundle Manager requests to deploy a bundle to one or more target services gateways.

Input: *bundleURL* specifies the location on VFM where the bundle JAR is located; *gatewayIPs* contains IP addresses of one or more target services gateways to run the bundle.

Output: A list of data structure, each contains *successfulGatewayIPs* to indicate the gateways that have successfully deployed the bundle, and *bundleID* to report the bundle ID assigned by the gateway, and a list of *failedGatewayIPs* to indicate the gateways that failed to start the bundle with the reason described in *description*.

Exception: *InvalidGateways* indicates that some of the services gateways identified by the provided *gatewayIPs* are invalid or unreachable.

3.5.6.2 terminateBundle Operation

Pre-conditions: Bundle is deployed to the target services gateways.
Post-conditions: The bundle is not running on some of the specified services gateways.

Input:
List of {String *gatewayIPs*, Integer *bundleID*}

Output:
StringList *successfulGatewayIPs*,
List of {String *failedGatewayIP*, String *description*}
Exception:
InvalidGateways(StringList *gatewayIPs*)

Bundle Manager requests to terminate a bundle on target services gateways.

Input: A list of *gatewayIPs* that specifies the target services gateways to terminate the bundle identified by bundleID on that gateway.

Output: A list of data structure, each contains *successfulGatewayIPs* to indicate the gateways that have successfully deployed the bundle, and *bundleID* to report the bundle ID assigned by the gateways and a list of *failedGatewayIPs* of the gateways that failed to start the bundle and *description* of the failure.

Exception: *InvalidGateways* indicates that some of the services gateways identified by the provided *gatewayIPs* are invalid or unreachable.

3.5.7 Interface I7

Semantically, I7 is the same as Bundle Manager Interface I5. The difference is that I5 is an internal interface, where I7 is protocol specific interface based on the protocols to be supported by the target services gateways, for instance, HTTP, Jini, or RMI.

3.5.8 Interface I8

Semantically, I8 is the same as Bundle Manager Interface I6. The difference is that I6 is an internal interface, where I8 is protocol specific interface based on the protocols to be supported by the AdminBundle and the target services gateway, for instance, HTTP, Jini, or RMI.

4 APPLICATION OF VIRTUAL OSGi FRAMEWORK

This section shows an example how Virtual OSGi Framework architecture can be used to provide a perceived larger memory space for OSGi Framework on a hypothetical services gateway using a real-life connected home scenario.

4.1 Scenario

One evening not so far in the future in a networked home connected to the Internet through a device called Residential Services Gateway, Mary has finished her dinner and loaded the dishes into the dishwasher. She instructs the washer to start in economy mode, meaning it would let the utility company start the machine automatically when the electricity rate is at the lowest of the day. The built-in application in the dishwasher sends the request to the "Appliance Manager" application running on the Gateway to handle the request. The "Appliance Manager" contacts the server at TXU, the utility company, and registers an agent program to be activated when the electricity rate is dropped to discounted rate.

Mary is now relaxing with a glass of her favorite wine. She feels like talking to her mother to check to see if her mother has received the birthday present Mary sent last week, so she picks up the home remote control, which is equipped with a voice recognition feature, and says, "call mom". The remote control recognizes the command and forwards the translated English text to the "Internet Phone" application on the Gateway to handle the command. The application looks up the Internet phone book from the directory then makes a call to Mary's mother. Unfortunately, there is nobody to answer the phone, so Mary places a request for her mother to call back when she is available.

Sipping on her wine and winding down from her busy day, maybe a romantic movie would be nice Mary thinks. So she picks up the home remote control and pushes "Movie" button. The Java TV applet running on the digital TV contacts the "TV manager" application on the Gateway for her movie profile, which includes the watch-list items, to display on the screen. Mary browses through the watch-list, and clicks "Play" after highlighting "Sleepless in Seattle" her favorite movie she wanted to watch one more time. The TV applet requests the "TV manager" for an MPEG stream of the selected movie title. The "Movie Manager" downloads a "Movie Finder" application, then requests it to find "Sleepless in Seattle" from the Internet. The "Movie Finder" application issues a request for lowest price bidding on a designated movie auction site. Using automated agents deployed at the site, the participating movie suppliers take less a minute to conclude the bidding with "Movie.com" as the winner, which won

the bidding by just 5 cents. The auction web site then exchanges payment information with the “Movie Finder”, which in turn retrieves the digital wallet and digital certificate from the smart card inserted to the smart card reader on the digital TV. Once the transaction has been confirmed, “Movie.com” starts streaming the movie to “Movie Finder” application, which in turn forwards the stream to the digital TV to be played on the screen. The “Movie Manager” then notifies the “Smart Home Control” that a movie has started. The “Smart Home Control” looks up the profile database and finds that the “Movie Watching” home control setting is enabled. The “Smart Home Control” then sends a series of commands to “Basic Home Control” application to dim the light in the living room, close the blind, postpone the dish washer, set the phone to ringer-off and block all calls except for the ones from her family. Mary is now enjoying her favorite movie on the comfy couch.

Half way through the movie, a call is made from Mary’s mother. The “IP Phone” application notices from the caller ID that it is from her mother, so it sends a request to the Java TV applet to indicate on the screen that a phone call from her mother is ringing and prompts whether she wants to answer. Since Mary has watched this movie many times and has been waiting for the call, so she would not mind the interruption. Mary pushes the “Pause” button on the remote and clicks the “Yes” button on the screen to answer the phone. The “IP Phone” then streams her mother’s voice and video to the digital TV, which in turn streams Mary’s voice and video captured from the microphone and camera equipped on the TV back to her mother’s IP phone. In the meantime, the “Movie Finder” application requests the “Movie.com” server to pause the movie stream while Mary is talking to her mother.

Having talked for a long time, her mother excuses herself to prepare dinner. Mary says good-bye and clicks the “Hang-up” button on the screen and pushes “Play” button on the remote. The movie stream is then resumed from the “Movie.com” server and then played on the screen. Mary continues watching and enjoys her favorite happy ending of the movie.

4.2 Walk-through of OSGi Framework Memory Snap-shots

For illustration purposes, the memory usage is simplified as follows and depicted in Figure 19.

- Target OSGi Framework has 1000 KB available memory for all bundles
- Each bundle consumes approximately100 KB of Framework memory
- There are 6 non-interruptible bundles that are resident in the memory to perform system level or core functions of the Gateway device, including HTTP services, Servlet services, Logging services, Remote Administration services (AdminBundle), System Monitoring services, Device Monitoring services. These services take up 600 KB combined.
- The remaining 400 KB is available for 4 concurrent bundles to support Mary’s activities mentioned in the scenario.

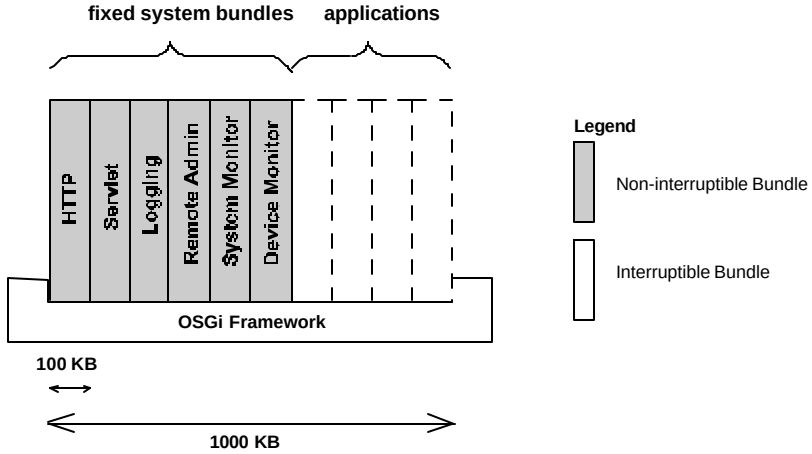

Figure 19 Example of an OSGi Framework Memory Layout

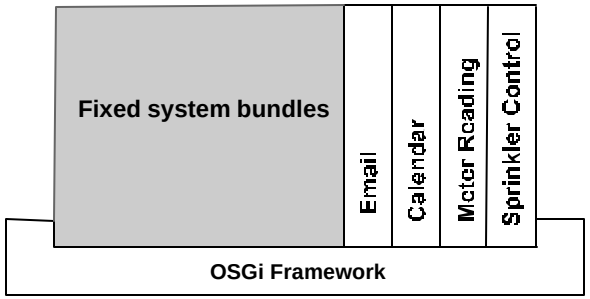

Figure 20 Before Mary Turns on the Dishwasher Memory Layout.

Figure 20 shows the active bundles before the scenario starts, that is, before Mary has turned on the dishwasher.

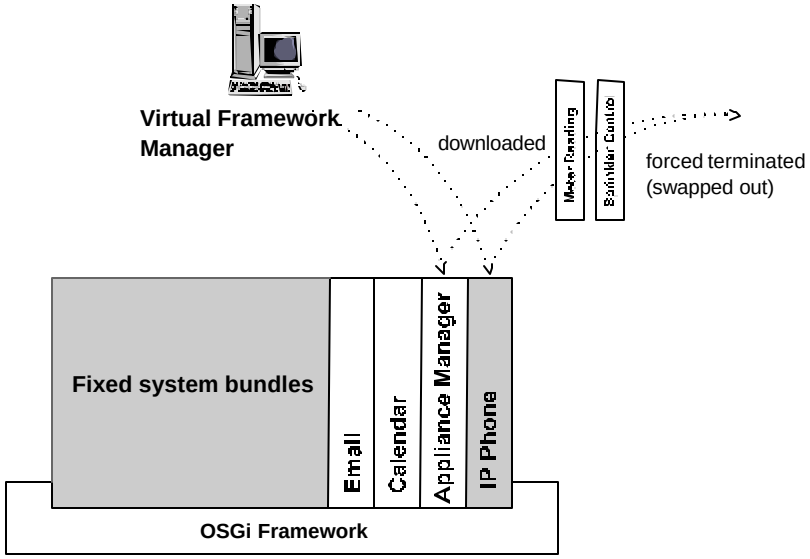

Figure 21 After Mary Activates Dishwasher

Figure 21 depicts the active bundles after Mary has activated the dishwasher and is placing an Internet phone call to her mother. Appliance Manager is downloaded from the bundle repository on the Virtual Framework Manager. To free up memory for the two new bundles, the lowest priority bundles (Meter Reading and Sprinkler Control bundles) are temporarily terminated (swapped out).

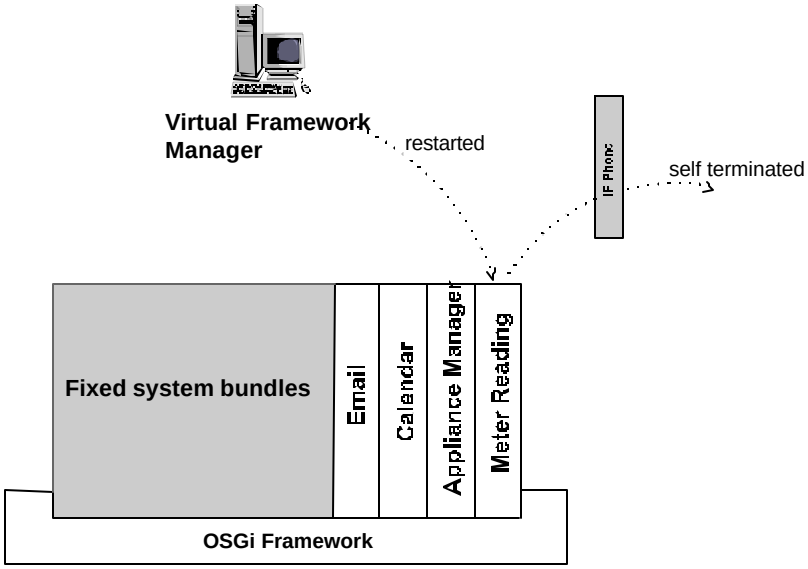

Figure 22 After Mary hung up the Telephone Memory Layout

Figure 22 depicts the active bundles after Mary has hung up the phone after placing a request for her mother to call back. The IP Phone bundle terminates and removes itself from the Framework. The bundle termination event prompts VFM to restart Meter Reading bundle that it has temporarily terminated (swapped out).

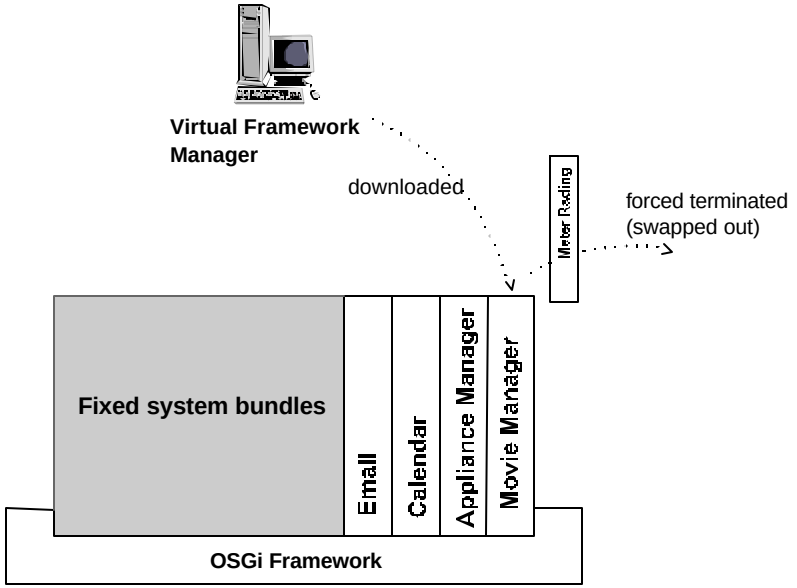

Figure 23 Memory Layout After Mary has Requested to See Her Movie

Figure 23 depicts the active bundles after Mary has requested to see her movie profile. Movie Manager bundle is downloaded from VFM's bundle repository. Being a low priority bundle, The Meter Reading bundle is again swapped out to free memory for Movie Manager bundle.

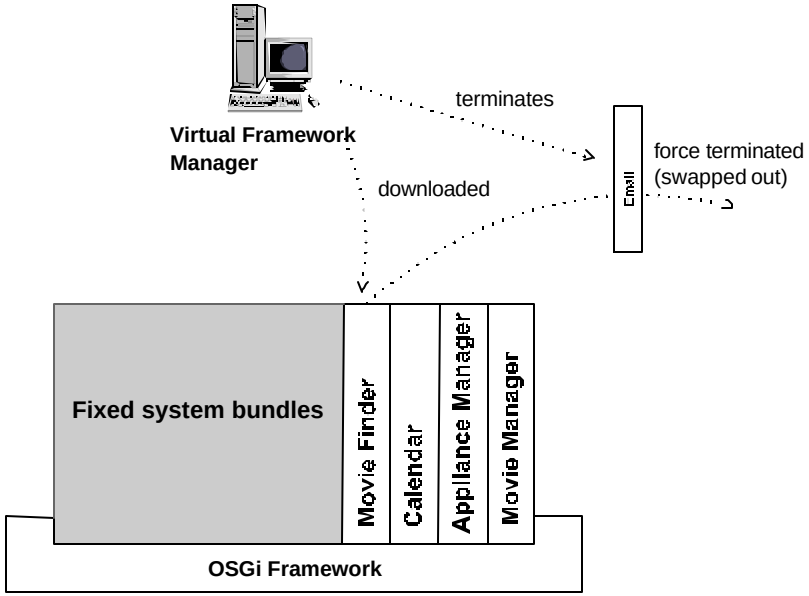

Figure 24 Memory Layout after Mary Requested to See "Sleepless in Seattle"

Figure 24 depicts the active bundles after Mary has requested to see "Sleepless in Seattle" the movie. Email bundle is temporarily terminated (swapped out) to free up memory for Movie Finder bundle.

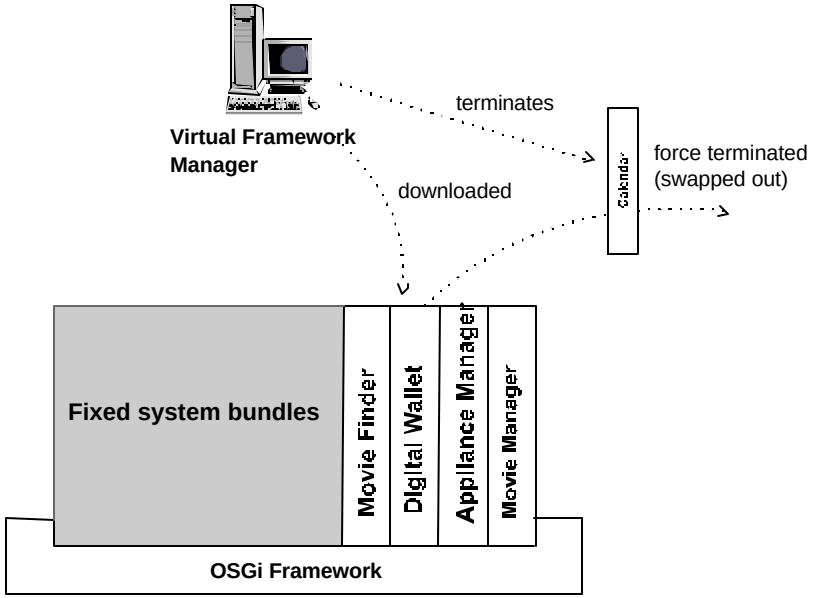

Figure 25 Memory Layout after the Movie is Found

Figure 25 depicts the active bundles after Movie Finder has found a movie supplier from the auction site. Being a lower priority, Calendar bundle is temporarily terminated. The Digital Wallet bundle is then downloaded to exchange electronic payment information with "Movie.com" site that has won the bidding.

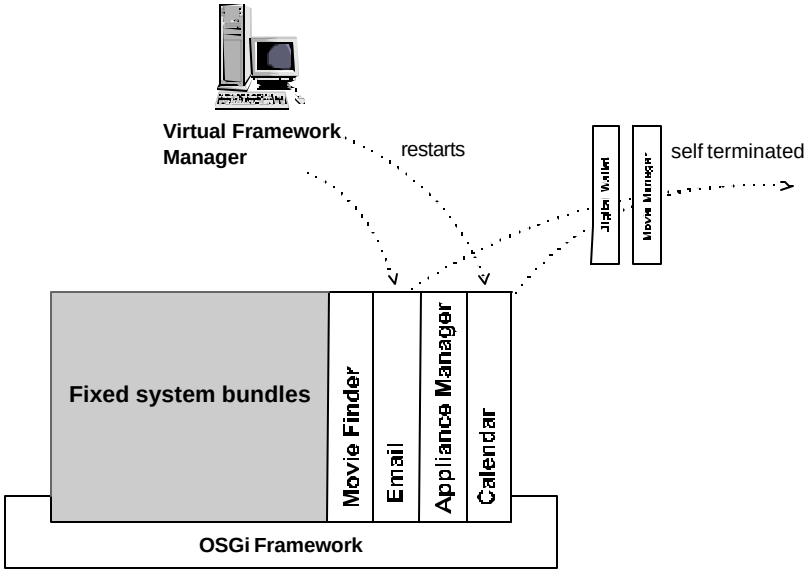

Figure 26 Memory Map after the Movie has Started

Figure 26 depicts active bundles after the movie has started. Having completed their intended tasks, Movie Manager and Digital Wallet bundles automatically terminate and remove themselves from the Framework. The Email and Calendar bundles are then restarted.

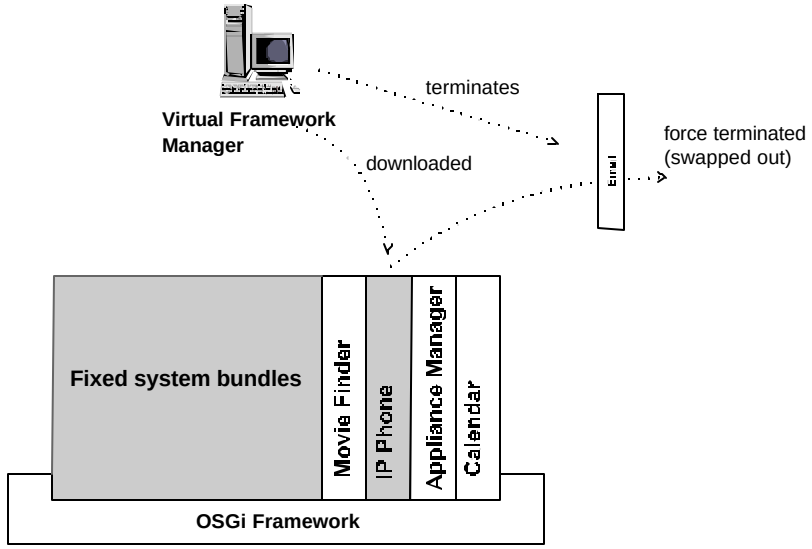

Figure 27 Memory Map after Mary has Accepted a Call from Mom

Figure 27 depicts active bundles after Mary has accepted the call from her mother. Email bundle is swapped to free up memory for non-interruptible IP Phone bundle.

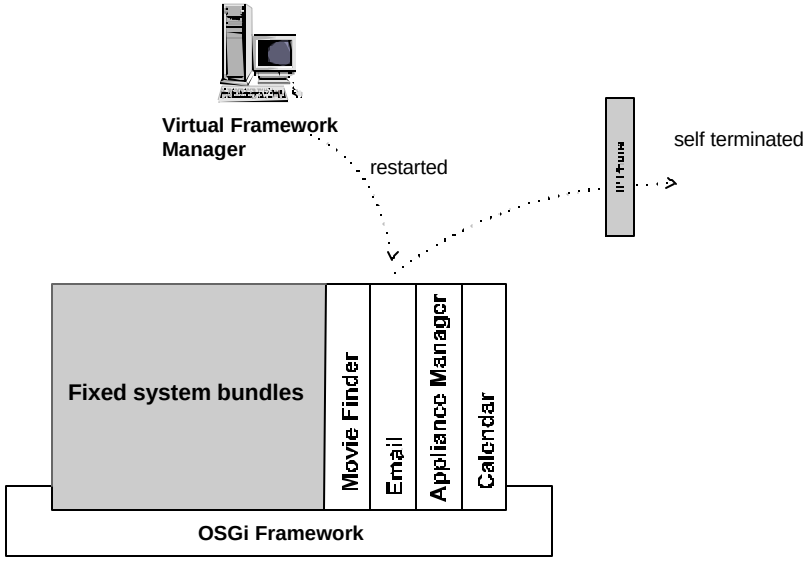

Figure 28 Memory Map after Mary has Hung up the Phone

Figure 28 depicts active bundles after Mary has hung up phone with her mother. IP Phone automatically terminates and removes itself from the Framework. Email bundle is then restarted to resume its function.

5 CONCLUSION

This paper makes the following technical contributions:

- A solution to address fixed limited memory in small telecommunication and embedded devices for dynamic loadable application services environment using a remotely swappable application concept,
- A demonstration on 2 software methodological frameworks: NFR Framework to derive architectural decisions, and RUP/UML to develop software architectural designs. It has been the first time that the NFR Framework has been used together with RUP/UML in a software requirement engineering and software architectural design process.

Virtual OSGi Framework architecture is proposed as an example how the technique may solve the problem in the emerging Services Gateway market. The Virtual OSGi Framework uses a remote server to manage the deployment of OSGi Bundles and ensures the right services are brought into the main memory of OSGi Framework for execution at the right moment. The concept of *Deployment Policy* is introduced to facilitate the services scheduling so that services with different Quality of Service (QoS) requirements can be scheduled accordingly so that the intended functions and acceptable responsiveness are maintained.

This paper uses the NFR Framework to systematically evaluate various architectural options in order to meet the established goals. The solution is selected based on NFR satisficing with extensive rationale and claims given to support the decision. With the selected architectural solution, a detailed architectural design is modeled using RUP/UML to demonstrate how the Virtual OSGi Framework architecture may be constructed. It is the first time that the two software development frameworks have been used together to demonstrate how a system may be developed from requirement engineering phase to architectural and software design phase. Finally, a hypothetical everyday life scenario is used to demonstrate how Virtual OSGi Framework may be used in a connected home using a Services Gateway connected to the Internet.

There is room to improve the proposed architecture. For example, this paper has defined an attribute for GatewayBundle class called *priorityOffset* so that services that are sleeping for a long period of time may be given higher effective priority the longer it waits. However, this paper does not define the algorithm for such dynamic scheduling. Also, for simplicity, this paper has shown only the design for the core concept of Virtual OSGi Framework. Some more detailed design may be needed for complex commercial environment. For example, Bundle life cycle requires that a Bundle may be unregistered only from *Decommissioned* state. However, in some situations, a Service Provider or Services Gateway Operator may wish to unregister the Bundle even though some services gateways may be considered having the bundle running. A more elaborate architectural solution (such as leasing concept [22], [23]) may be used to solve dangling services on the services gateways that have no managed Bundle counterpart on the remote server (VFM).

REFERENCES

[1] allNetDevices.com, *Coactive Unveils New Series Of Residential Gateways*, http://www.allnetdevices.com/news/0005/000503coactive.htm

[2] allNetDevices.com, *IBM Ships OSGi Toolkit*, http://www.allnetdevices.com/news/0005/000505ibm.htm

[3] D. Jordan, "Java in the Home: OSGi Residential Gateways", *Java Report*, September, 2000, pp 38-42, 104.

[4] E. A. Lee, "What's Ahead for Embedded Software?", *Computer*, September 2000, pp. 18-26.

[5] ElectronicNewsOnline, *Novell, TI demonstrate broadband gateway technology*, http://www.electronicnews.com/news/3377-129NewsDetail.asp

[6] Ericsson, *Ericsson and Skandia Set Up New Company*, http://www.ericsson.com/press/20010115-0027.html

[7] G. Bollella, *The Real-Time Specification for Java*, Addison-Wesley, 2000.

[8] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[9] I. Jacobson, G. Booch, J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[10] J. Barr and R. Mata, "OSGi: spec basics, interface issues", *EE Times*, December 11, 2000, Issue 1144, http://www.eetimes.com

[11] J. Mylopoulos, L. Chung, S. S. Y. Liao, H. Wang and E. Yu, "Extending Object-Oriented Analysis to Explore Alternatives", *IEEE Software*, Jan./Feb., 2001. pp. 2-6.

[12] K. Arnold and J. Gosling, *The Java Programming Language Second Edition*, Addison-Wesley, 1998.

[13] L. Chung, B. A. Nixon, E. Yu and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, Boston, 2000.

[14] M. Bach, *The Design of The UNIX Operating System*, Prentice-Hall Software Series, 1986.

[15] OSGi, *Home Gateway Platform Unveiled*, http://www.osgi.org/news/news271100.html

[16] OSGi, *OSGi Service Gateway Specification*, release 1.0, May, 200.

[17] OSGi, *The Internet-Enabled Car*, http://www.osgi.org/news/news020101.html

[18] Sun Microsystems, *Java Embedded Server Software Overview*, http://www.sun.com/software/embeddedserver/overview/index.html

[19] Sun Microsystems, *Java Embedded Server Software White Papers #1*, http://www.sun.com/software/embeddedserver/whitepaper1.html

[20] Sun Microsystems, *Java Embedded Server Software White Papers #2*, http://www.sun.com/software/embeddedserver/whitepaper2.html

[21] Sun Microsystems, *SOFTSWITCH: ON. Nortel Wins with Java™ Technology*, http://www.java.sun.com/features/2001/02/nortel.html

[22] The Internet Engineering Task Force, *RFC 2131 Dynamic Host Configuration Protocol*, http://www.dhcp.org/rfc2131.html

[23] W. K. Edwards, *Core JINI*, Prentice-Hall PTR, 1999