Multiservice Home Gateways: Business Model, Execution Environment, Management Infrastructure

Yvan Royon, Stéphane Frénot

Multiservice Home Gateways: Business Model, Execution Environment, Management Infrastructure

Yvan Royon and Stéphane Frénot, INRIA Ares/CITI, INSA-Lyon

ABSTRACT

The home gateway market is undergoing deep changes. On one side, home networks are evolving, getting dynamic and federating more and more devices. On the WAN side, new actors appear, such as multimedia content providers. From both sides emerge new features and new management needs. In this article we highlight challenges and benefits of moving more intelligence to the home gateway, making it more than a simple interconnection device. We argue that we need a full-fledged execution environment on gateways to address the evolution of business models. We present this evolution and summarize existing solutions for execution environments and management for home gateways. Then we propose two improvements that reflect the new requirements. The first improvement is a high-level virtualization of service gateways, and the second one recommends an end-to-end dynamic multiprovider management system.

INTRODUCTION

During the last few years, the home gateway market has evolved at a very fast pace. For a time, it only consisted of bringing IP connectivity to the home. Available services were common application-level programs, such as Web or email clients. Today, operators are moving to integrate value-added services. Multicast TV and voice over IP services are widely advertised; their integration with data transport is referred to as *triple play*. These three network-enabled services are provided through either the same connectivity box (the modem) or a dedicated set-top box.

In the near future, operators plan to open the service delivery chain.[1] The roles currently assumed by the Internet access provider will be split between connectivity provisioning and service provisioning. Instead of being tied to a single service provider for television and phone over IP, the end user will have a variety of choices and will gain from competition in both price and diversity. This business model is referred to as *multiplay*.

A model with multiple actors outlines new challenges. The home gateway hosts a set of configurable services, which can be operated, controlled, or monitored by different service providers. However, these important issues are not sufficient to get the full picture. Indeed, voice and video are not the only services that can be of interest for home users: domotics, entertainment, home security, and health care are also potential markets. For a long time these other services have been considered individually, with separate networks, devices, and user control. By integrating them with the home network, new use cases are possible.

As an example, a network-enabled white good (e.g., an oven or a dishwasher) can be monitored by its manufacturer. In case of failure (e.g., if the temperature in a fridge rises above a certain threshold), an alarm would be raised to the home user and the nearest repairman. The contents of a fridge or wine cellar can also be monitored using sensors. The home user would input preferences, or orders would be sent to the nearest retailer. Many current projects focus on providing help for seniors and the disabled at home; these issues are known as *quality of life*. For instance, the home gateway would monitor a pacemaker or similar medical appliances, and alarms would be forwarded to family members and the nearest hospital.

All these examples show that the definition of *service* must be broadened. It should not be limited to network multimedia flows, but also include management facilities for in-home devices (configuration, preferences, monitoring), human-machine interfaces, and any kind of application the home gateway may host. This article refers to this extension as *multiservice*. We keep this definition open so that future ideas and applications can be seamlessly integrated.

The home gateway must undergo changes so that the multiservice model can be implemented. Indeed, home users should be able to use in-home devices and set preferences regardless of the status of the access provider's network. This means that most home-related settings should be hosted inside the home. In the triple play model, the home gateway already hosts or uses information and configuration from both home users (e.g., WiFi access point settings) and the access

1 See the MUSE European Project, IST-FP6-507295, which partially supports this work: http://www.ist-muse.org provider (e.g., last mile settings). Therefore, it seems a natural choice to enhance home gateways to be multiservice ready.

Our goal in this article is to provide mechanisms for multiservice enhancements, which target both the OS layer and the management plane. Figure 1 shows a management perspective of the multiservice home environment. Three *realms* interact around home gateways. The access realm is the usual access provider, who provides layer 1–4 connectivity to the gateway. The user realm contains all interactions the end user and local devices may have with the home gateway. Third, the service provider realm represents the interactions between a service provider and a service hosted on the gateway.

From an execution layer point of view, these multiple realms and entities also coexist on the gateway. This means that some level of software isolation must be defined. The article is structured as follows. Each section focuses on both the management layer and the execution environment. We present existing solutions related to home gateways. We define new requirements that result from the multiservice model and the enhancements that must be made to existing solutions. We describe our testbed implementation of a multiservice gateway; results are detailed later. Finally, we conclude the article.

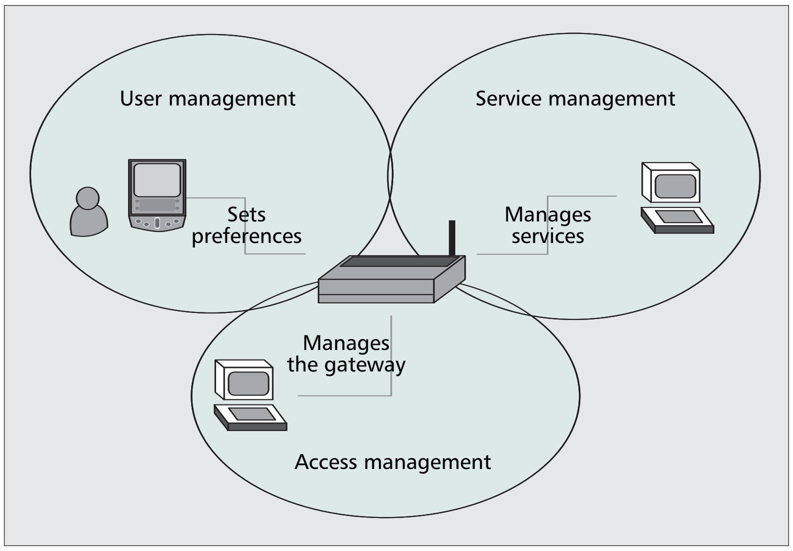


Figure 1. *Management realms.*

OVERVIEW OF EXISTING HOME GATEWAY ENVIRONMENTS

New business models bring new technical requirements. With the connectivity-only model, an auto-configuration sserver (ACS) provides layer 1–4 parameters to the home gateway (e.g., Dynamic Host Configuration Protocol, DHCP). There is only one ACS, hosted by the access provider. With the triple play model, the ACS must store additional intelligence and parameters related to voice and video subscriptions. Moreover, it manages not only the home gateway, but also other customer premises equipment (CPE), such as set-top boxes for TV.

With the multiplay model, there might be more than one ACS. Coherence and aggregation of management activities from different service providers to one particular home gateway become a problem. With the multiservice model described above, it is clear that more intelligence must be put inside the home gateway, making it a pivotal device in the home network. In this section we first summarize existing management solutions that evolve around the home gateway. Then we present a panel of execution environments applicable to this particular device.

MANAGEMENT SOLUTIONS

With the triple play model, home users configure their gateway and their services through the Access Provider's Web site. However, with the multi-service model, there can be more than one service provider; the home user may also interact with in-home devices, whether the Internet access is active or not. As a result, a single management server, located outside of the home, is no longer desirable. The solutions that exist for this problem can be categorized in two groups: wide area network (WAN) management protocols and application/service management protocols.

WAN management protocols are network-centric solutions that follow a three-tier architecture: the manager (e.g., an ACS), the agent (on the managed device), and the connector (for translation between the former two). Existing management technologies answer three questions: how the data is structured, which data is available, and how the agent and connector interact. Common examples are TR-069 from the Digital Subscriber Line (DSL) Forum, CIM/WBEM from the Internet Engineering Task Force (IETF), and Simple Network Management Protocol (SNMP).

Application management protocols focus on home device management, such as service discovery and autonomous service description. Examples are JMX in the Java world and UPnP.

An implementation of the multiservice model can make use of several of these WAN and application management solutions. The other side of the story is that the home gateway must support the management agent, but also other applications, configuration facilities, and so on. Such support is provided by the execution environment.

EXECUTION ENVIRONMENTS FOR THE HOME GATEWAY

With triple play and multiplay business models, home gateways are becoming more than just modems: they are resource-constrained computers. Their task is not only to transport network packets, but also to filter them, support multicast, be upgradeable, host application services, and support user interaction. Two kinds of execution environments are popular in academia and industry: GNU/Linux and OSGi.

Linux — The obvious trend at the moment is that constructors are abandoning proprietary operating systems that used to power their modems in favor of Linux variants. There are several strong advantages:

Integrating applications from different providers, making them work together and ensuring their correct behavior, is time consuming. One possible answer is to use type-safe languages and sandboxing techniques.

- Linux exists in different flavors, such as uCLinux, which makes it adaptable to new architectures [1], including resource-constrained environments.
- Source code is open and mainstream, and so is tested by lots of users.
- The programming model is standard C or C++. Finding able developers is very easy.
- It is then trivial to reuse existing applications for networking (firewalls), multimedia (codecs), or management (agents for various protocols, e.g., SNMP).

Current operating systems still suffer drawbacks in multiservice environments. Integrating applications from different providers, making them work together and ensuring their correct behavior, is time consuming. One possible answer is to use type-safe languages and sandboxing techniques. The mobile phone industry went this way with Java. The home gateway industry is following the same path with OSGi.

OSGi Technology — In its early stages, OSGi was designed specifically for open service gateways.[2] The OSGi service platform is a container built on top of a Java virtual machine. It hosts deployment units called *bundles*, which contain code, other resources (pictures, files), and a specific start method. A descriptor file expresses dependencies on other software pieces, among other meta-data.

The OSGi platform automatically checks and resolves dependencies among bundles. It also dynamically controls bundles' life cycles, and enables hot deployment and update. This simplifies the administrator's work and improves stability.

The programming model is service-oriented programming (SOP) [2], which is derived from object-oriented programming (OOP). The driving motivation is to minimize coupling among pieces of software. A service is an interface, or a contract, that describes what a piece of software does. How it is implemented is hidden internally.

The OSGi specifications keep the management layer open, in the sense that it does not impose a specific management protocol or technology. In real-life situations, several technologies can be combined to provide an end-to-end management solution [3].

OSGi is becoming popular in academic communities, but more so in industry. For instance, some BMW cars use OSGi for noncritical tools. Another example is the Eclipse development kit, which rebuilt its whole plug-in architecture around an OSGi framework, and most J2EE implementations, which are or plan to be redesigned as OSGi bundles. Actors interested in home gateways are also adopting OSGi, through the Home Gateway Initiative (HGI), a telco-driven consortium.

HGI — The home gateway performs various tasks, from protocol translation to quality of service (QoS) enforcement to remote access. Since most of these tasks are dynamic, configurable, manageable, and can change over time, they should be implemented by updateable software. HGI [4] has defined operating system requirements for home gateways, which are largely based on the OSGi specifications. They operate on three levels. The first one is software management. It includes:

- Configuration of software pieces called *modules*
- Management of their life cycles: install, update, uninstall
- Dynamicity and security enforcement: modules registered and verified; their dependencies resolved, then linked
- Resets for default configuration parameters, and for firmware debug in case of low-level failures

The second level is performance management and diagnostics, which include:

- Remote diagnostic tests for hardware and software elements
- Performance monitoring
- Support for events sent by the gateway.

The third and last level gathers definitions of users in presence:

- A super-user (the ACS), in control of everything that is manageable
- A local administrator, in control of local management (e.g., firewall, users)
- End users, with permissions set by the local administrator

Current projects on home gateways are led by the HGI specifications, the OSGi platform, and management technologies. However, they lack the integration of the multiservice business model. They still focus on models with a single access and service provider hosting a single ACS. Our goal is to enable the multiservice model, from both the management layer and execution environment points of view. The next section lists the requirements for these two issues.

Requirements for Multiservice Home Gateways

A multiservice home gateway hosts various services, applications, and so on from several service providers. The fact that these service providers can be separate entities has an impact on both the management layer and the execution environment.

Management Requirements

Figure 1 shows that the actors around the home gateway use the management layer for different purposes. In the multiservice model, management should not be classified according to network span (LAN, WAN) or open systems interconnection (OSI) layer (IP, TCP), but according to the goals and features of each actor.

For example, a home user will set preferences and configure devices using UPnP technologies, flash interfaces, or proprietary solutions. An access provider will configure layer 1–4 parameters and manage QoS using TR-069. Service providers may prefer JMX or SNMP to monitor their services.

With these examples in mind, we can express management requirements as follows:

- Each actor around the gateway (home user, access provider, and each service provider) can use its own dedicated management

[2] *http://www.osgi.org*

agent.
- Technologies used for management agents are heterogeneous.

These two points allow management activities to be isolated from different actors in terms of network flows. We must also address separation local to the gateway in terms of runtime execution.

OS Requirements

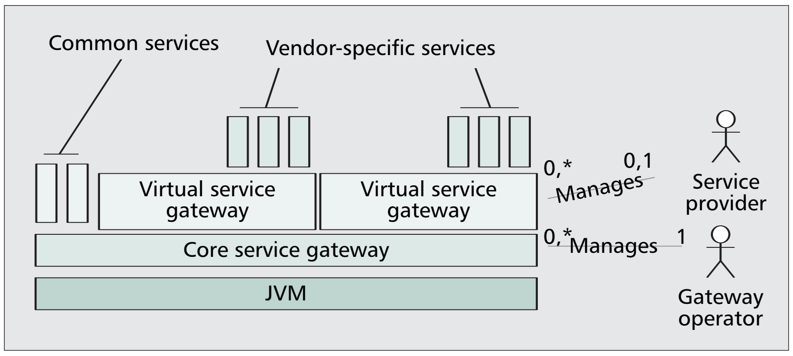
HGI requirements work well in a single-provider scenario with a single ACS. However, they fall short on several points in the case of an open multitier environment. Indeed, different service providers, potentially competitors, may deploy modules on the same home gateway. These modules may contain business-critical code or data, or simply give away information on the client base, configurations, or implementation performance. They may also contain malicious or buggy code, which could endanger all modules present.

Therefore, the need for isolation among actors present at the software level dictates the following additional requirements:
- Definition of users:
  - –Service or software providers can be seen as users on the home gateway, similar to multi-user execution environments. These users need a secure authentication phase and a secure session for all activities on the gateway, such as module deployment.
  - –A root user controls users allowed on the system. S/he also checks overall resource consumption, and may take coercive measures against users that threaten to starve resources.
- Isolation and sharing of modules:
  - –Modules from different providers (users) must be isolated by default; that is, a user can only see and interact with its own modules.
  - –A single instance of some modules may be shared among all users on the home gateway. This is useful for libraries (e.g., codecs) or common modules (e.g., a Web server).
- Remote access for management features.
- Preferences and configuration can be stored locally if they represent private information or are not related to an ACS.

Now that we have described the changes multiservice gateways must undergo, the next section shows how we implemented these changes using OSGi.

A Framework Implementation for Multiservice Home Gateways

OSGi offers important features for home gateways: service-oriented programming, dynamic life cycle management, and dependency checking. Unfortunately, it lacks concepts for the multiservice business model. To enable this model, we propose improvements on two levels. At the runtime level we implement a virtualization of the OSGi framework [5]; at the management level we provide an end-to-end JMX-based solution.



■ **Figure 2.** *Multiservice, multiprovider home gateway.*

OSGi Virtualization

Software elements that belong to different providers must be separated. In other words, we need some level of runtime isolation. Various work already exists on this topic, such as BSD jails, Xen, VServer or VMware. They propose different levels of enforcement in terms of resource isolation and namespace isolation, which are a trade-off with performance.

For instance, Xen [6] runs separate copies of a whole operating system for each isolated environment. VServer [7] does not duplicate the kernel, but still duplicates inodes on the file system for each isolated environment. A BSD jail duplicates only a portion of the file system, but does not provide strong resource isolation.

In our case, following the multiservice model, the chosen level of isolation must still allow to share services on demand among different service providers. This means that isolation should be permissive. Moreover, we focus our implementation on Java/OSGi environments. As a consequence, sharing applies to OSGi services (which are Java interfaces). For scalability reasons, duplicating a whole JVM for each isolated environment seems overkill. Last, a home gateway has limited resources; typically 16 Mbytes of permanent storage and 64 Mbytes of RAM. These reasons led us to opt for a weaker but lighter level of isolation than Xen's or VServer's.

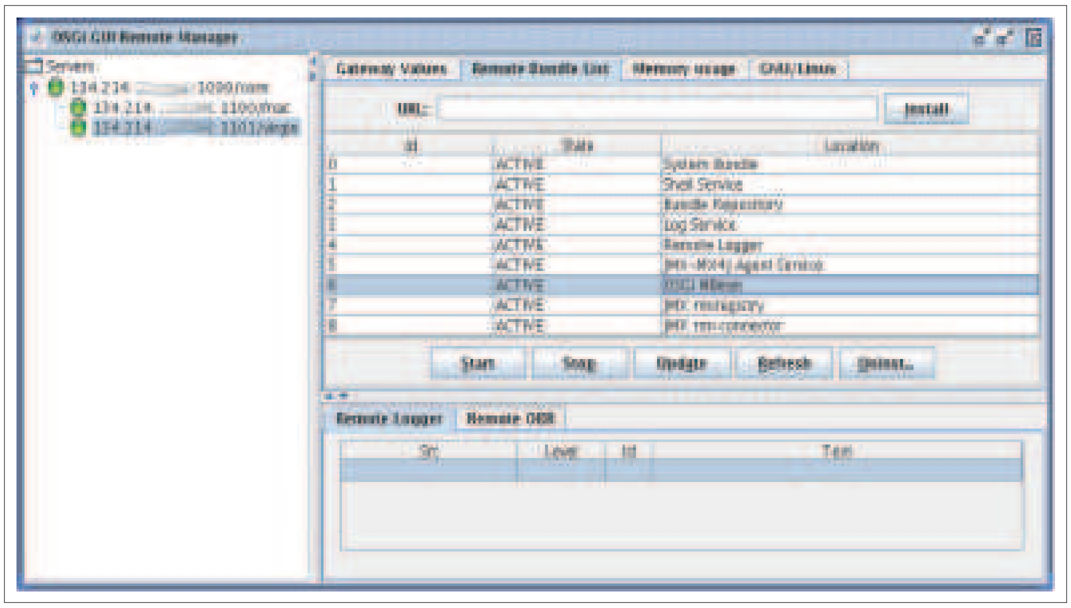
Our approach is to embed OSGi as an OSGi bundle. A "core" OSGi instance runs multiple OSGi instances, all sharing the same Java virtual Machine (JVM). Then the scalability factor is only the price of virtualizing (or embedding) OSGi.

The aim here is to give each service provider a separate execution environment, as depicted in Fig. 2. A virtual gateway is operated by one service provider, who sees it as a standard OSGi service platform.

The core service gateway is responsible for launching virtual gateways. It is managed by a gateway operator, who can also be the owner of the gateway or the access provider. Operations related to virtual gateway life cycle are subject to contracts between a service provider and a gateway operator.

We now have an execution environment where each actor around the home gateway gets a little privacy. Service providers are only aware of services running in their own virtual gateway. The gateway operator ensures that allowed virtu-

Our default implementation uses JMX. For scalability reasons, each JMX agent needs to be lightweight in terms of memory consumption. We use a modified version of MX4J that we stripped down and split using Service-Oriented Programming.



Id	State	Location
0	ACTIVE	System Bundle
1	ACTIVE	Shell Service
2	ACTIVE	Bundle Repository
3	ACTIVE	Log Service
4	ACTIVE	Remote Logger
5	ACTIVE	JMX-MX4J Agent Service
6	ACTIVE	OSGi MBean
7	ACTIVE	JMX rmiregistry
8	ACTIVE	JMX rmi-connector

Src	Level	Id	Text

Figure 3. *Monitoring console.*

al gateways are up and running, but cannot access their contents.

After ensuring isolation, we need to add a controlled way to share code and services. Two cases are considered. The first is services common to all actors on the home gateway. This could be useful for an embedded Web server or a generic fault logger. These common services are hosted by the core gateway, and explicitly exported to all virtual gateways. When a virtual gateway is launched, it adds this collection of shared services to its internal service registry. The second scenario is for sharing code, such as libraries. For example, a single instance of multimedia codecs can be hosted by the core gateway and shared with all virtual gateways. It is similar to the previous case, except for the import/export mechanism in OSGi (Java package vs. OSGi service).

So far, we have described an execution environment that supports the multiservice model. The following describes the management framework that runs on top of it.

AN END-TO-END JMX-BASED MANAGEMENT SYSTEM

Multiparty access to a gateway can be handled using different approaches. One is to create a specific entity naming scheme for each provider. This means that each provider may access only a subset of the whole management data. We chose to define a virtual execution environment for each service provider; thus, a service provider sees a gateway as if it was the only one using it. Each virtual gateway hosts its own management agent, with the technology the service provider chooses.

Our default implementation uses JMX. For scalability reasons, each JMX agent needs to be lightweight in terms of memory consumption. We use a modified version of MX4J that we stripped down and split using SOP. Management data in JMX is represented through MBeans, which are Java interfaces. They follow the JavaBeans model: get an attribute, set an attribute, execute a method. Queries on an MBean are redirected to a set of probes; we have implemented probes that give access to:

- The OSGi framework: version number, current running profile, and so on
- Bundles life cycle: start, stop, update, and list bundles inside the current gateway
- OBR, the Bundle Repository: allows triggering installation of a bundle on a remote gateway
- Memory usage inside the JVM
- The underlying OS metrics: global CPU, memory, swap, and disk usage

Core gateways also have an MBean to start, stop, and list running virtual gateways.

Any bundle can come with its own MBean, for instance, to represent management data for a specific device or OSGi application. In this case the MBean registers itself to the OSGi framework as an OSGi service. The JMX agent, listening to service-related activities, automatically registers the MBean; this effortless approach is called the *whiteboard* pattern [8].

Finally, a remote logger implements the OSGi Log service. It notifies a remote manager of each event that occurs on the gateway.

The JMX agent and all MBeans are located on the gateway. For remote interactions, queries on MBeans are sent through connectors. In our cas RMI and XML/HTTP are used.

We have developed a monitoring console (Fig. 3) that dynamically discovers the list of bundles on a gateway. Their associated MBeans are gathered following the *visitor* design pattern [9], then displayed in graphical tabs. This allows management to be specialized depending on the user's service subscriptions, and reuse of the console for each of the three realms presented in Fig. 1.

Results

Memory Performance

Figure 4 shows the amount of memory used on our test system,[3] cumulatively by the OS, Java, OSGi, and applications. On the *x* axis, we run sample services that each allocate 1 Mbyte of memory.[4] On the first curve, all services run in the same gateway (single-user mode). The other curves run each service in a separate virtual gateway, with and without separate management agents.

Our current implementation, which is not yet optimized for size, shows a 3 MB memory overhead per service provider using OSGi virtualization and a JMX agent. There are no significant CPU and disk overheads.

The memory overhead is not negligible, so there is still room for improvement. However, the advantage of this proposal is to enable OSGi's service-oriented programming between multiple virtual gateways, without modifying the underlying OS and JVM. Other options suffer from either scalability, programming model or availability. The first alternative is to launch one JVM per user; scalability is one order of magnitude worse than OSGi virtualization, and the OSGi framework would need heavy modifications to support SOP between JVMs. The second alternative is to use a multi-task JVM. This is the best compromise between scalability and resource isolation; however, the only implementation that we know of [10] runs only on big SPARC/Solaris systems. A last alternative is to use lower-level isolation such as Xen. This offers the best resource isolation available; however, service oriented programming between Xen instances is yet to be achieved. With current implementations, we would need to run one JVM per Xen instance, with the downsides cited above.

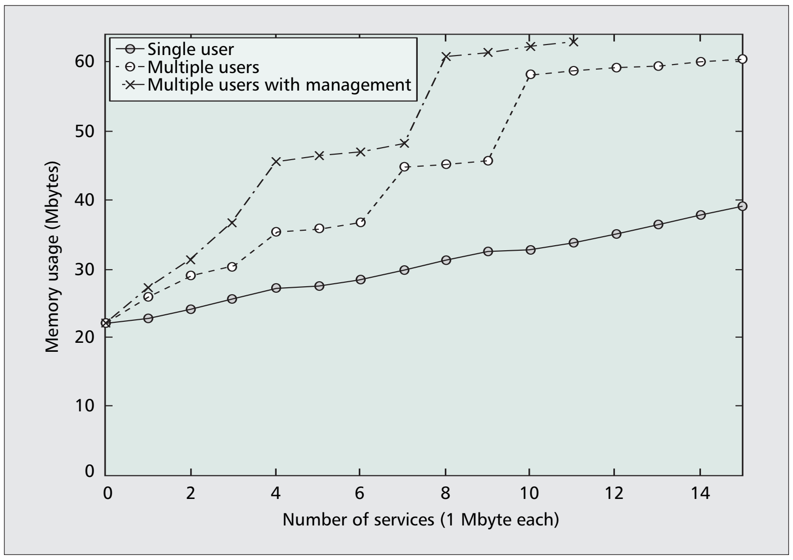


Figure 4. *Memory performance on a 64 MB machine.*

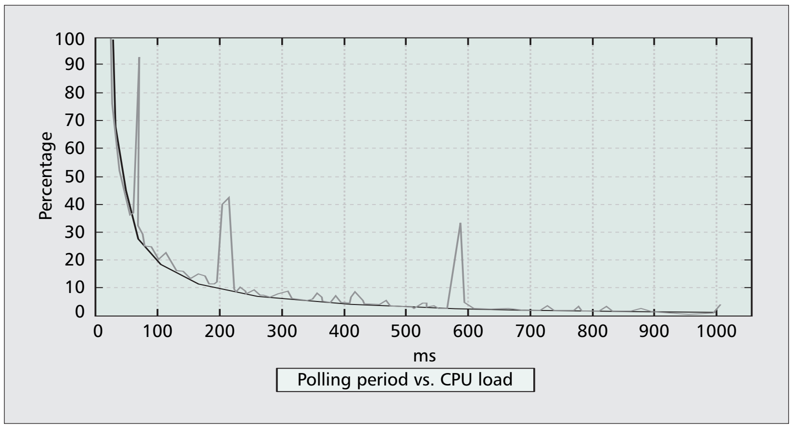


Figure 5. *Load induced by management probes on a 133 MHz machine.*

Management Performance

The management layer is evaluated by measuring response times. On Fig. 5, the thin black curve is the pseudotheoretical load induced by a simple probe: getCPU(). Its shape is $y = \alpha/x + \beta$. α can be seen at y = 100 percent CPU; it is the response time for `getCPU()` with zero network delay. β is the residual load when the gateway is not running any particular application; this is also called system noise. The curve is obtained by measuring two arbitrary points, e.g., at 100 and 400 ms period.

The thick green curve is the experimental load. It calls the `getCPU()` probe at various periods, ranging from every 1000 ms to every 10 ms. The peaks on the curve correspond to garbage collections on the system under scrutiny.

The figure shows that, if we allow the management system to use no more than 5 percent CPU time, the `getCPU()` probe can be polled every 500 ms. With around 10 users on the gateway, each Service Provider can poll a simple probe every 5 seconds without encumbering the CPU.

Summary

Looking back at the requirements expressed earlier, we have addressed some of them, and left others open. Here is the status of what works and what needs improvement.

Technical Definition of Users — Users (i.e., service providers) are defined as managers of virtual gateways. Each user has its own management agent and management data in its own virtual gateway accessible via its own port number. Thus, users are isolated from a management point of view. Moreover, each user chooses its own technology; we use JMX, but SNMP bundles are available on the OSGi Bundle Repository (http://www2.osgi.org/Repository).

We have not yet addressed user authentication and secure session. We plan to use SSL-enhanced connectors for this.

The last requirement in this category is that the root user be able to take coercive measures against users that use too much resources. This needs further work, since resource control for Java-based embedded systems is still a hot topic. We plan to use a feature similar to Linux's Out-Of-Memory OOM)-killer: when too much memory (or CPU) is used system-wide, the root locates the most consuming thread and kills its virtual gateway. If the thread belongs to the

3 An average home gateway has a processor around 266 MHz, 64 Mbytes of memory, and 16 Mbytes of storage. We have run performance tests on an Epia (1 GHz Nehemiah CPU, 64 Mbytes of RAM) running a Sun JDK 1.5 with default parameters, with the Felix OSGi implementation.

4 1 Mbyte is enough to host, say, several small games, embedded applications, and tiny Web servers. Our own UPnP/audio-video control point uses 600 kbytes of memory.

We propose a lightweight, OSGi-level isolation of the execution environment, where we launch "virtual" gateways (one per Service Provider) inside a "core" gateway (controlled by the operator). Each core and virtual gateway runs a separate management agent.

root, the related bundle should be killed.

Isolation and Sharing of Modules — We enforce namespace isolation between software modules (OSGi bundles) that belong to different service providers. This is weaker than resource isolation, but has the advantage of being OS- and JVM-independent, and it is production-ready.

Service sharing has been implemented statically. When the root user creates a new virtual gateway, the core gateway declares a list of services to export. In the future we plan to export this list dynamically.

Remote Access for Management —Our management bundles contain connectors for RMI and HTTP. Connectors for other remoting protocols are possible.

Local Storage for Preferences and Configuration — We use a straightforward scheme to isolate local storage among users. The Felix OSGi implementation uses profiles to cache bundles and data related to bundles. Each virtual gateway has its own profile, with its own cache directory. Bundles usually access files via the `bundleContext.getDataFile()` OSGi primitive, which only allows them to reach this private directory. We still need to stop malicious bundles that try to open `FileInputStreams` on other users' cache; Java permissions allow this.

Our code is available under open source licenses. Management-related projects are integrated within the Apache Felix project [11], distributed under the ASL license. The code for OSGi virtualization is also written for Felix, and can be obtained on demand under the CeCILL license.

CONCLUSIONS

Business models around home gateways are evolving, and will inevitably open the door to new actors and new services. The immediate conclusion is that the execution environment on the gateway must be split into isolated areas dedicated to different service providers, and that each actor needs autonomy and choice in terms of management solutions.

We propose a lightweight OSGi-level isolation of the execution environment, where we launch "virtual" gateways (one per service provider) inside a "core" gateway (controlled by the operator). Each core and virtual gateway runs a separate management agent. This enables namespace isolation; it is weaker than Xen's or VServer's isolation mechanism, but it scales reasonably on resource-constrained devices and allows OSGi services to be shared between core and virtual gateways. On top of this, we provide an end-to-end management infrastructure (probes, agent, and monitoring console) that take virtual gateways and multiservice constraints into account.

We show that a simple implementation allows hosting on the order of 10 service providers on a typical home gateway, with enough memory to run, say, a UPnP/audio-video control point. Each service provider can query simple probes on the order of 10 times per minute without hindering the CPU.

REFERENCES

[1] N. Fournel, A. Fraboulet, and P. Feautrier, "Booting and Porting Linux and uClinux on A New Platform," Research rep. 206- 08, LIP / ENS Lyon, Feb. 2006.
[2] G. Bieber and J. Carpenter, "Introduction to Service Oriented Programming," 2001; http://openwings.org/download/specs/ServiceOrientedIntroduction.pdf
[3] J. C. Dueñas, J. L. Ruiz, and M. Santillán, "An End-to-End Service Provisioning Scenario for the Residential Environment," *IEEE Commun. Mag.*, vol. 43, no. 9, Sept. 2005, pp. 94–100.
[4] HGI, "Public Deliverable v1.0," July 2006; http://www.homegatewayinitiative.org/publis/HGI V1.0.pdf
[5] Y. Royon, S. Frénot, and F. Le Mouël, "Virtualization of Service Gateways in Multi-provider Environments," *Proc. 9th Int'l. SIGSOFT Symp.Component-Based Software Eng.*, June 2006, LNCS 4063, pp. 385–92.
[6] P. Barham *et al.*, "Xen and the Art of Virtualization," *Proc. ACM Symp. Op. Sys. Principles*, Oct. 2003.
[7] S. Soltesz *et al.*, "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors," *Proc. EuroSys Conf.*, Mar. 2007.
[8] OSGi Alliance, "Listener Pattern Considered Harmful: The Whiteboard Pattern," 2nd rev., 2004, http://www.osgi.org/documents/osgi_technology/whiteboard.pdf
[9] E. Gamma *et al.*, *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.
[10] G. Czajkowski, L. Daynés, and B. Titzer, "A Multi-User Virtual Machine," *Proc. USENIX 2003 Annual Tech. Conf.*, pp. 85–98.
[11] Apache Software Foundation, "Felix OSGi R4 Service Platform Implementation"; http://felix.apache.org/

BIOGRAPHIES

YVAN ROYON (yvan.royon@insa-lyon.fr) received his M.Sc. and M.Eng. degrees from INSA Lyon, France, in 2004. He is now a Ph.D. candidate, jointly at the CITI laboratory and on the INRIA Ares team. He is also a teaching assistant in the Telecommunications and Networking Department. His research interests include multi-user middleware, P2P deployment, and service management in GNU/Linux.

STÉPHANE FRÉNOT (stephane.frenot@insa-lyon.fr) is an associate professor at INSA Lyon, France, and a researcher at the CITI laboratory and with the INRIA Ares team. He received his Ph.D. in computer science in 1998 from the University of Lyon I, France, on the topic of distributed systems for medical care. Since then he has worked on several projects in the areas of application servers management and distributed middleware. His current research interests include service-oriented programming, middleware instrumentation and management, Java-based middleware for resource-constrained devices, and security in component-based middleware.