Various Extensions for the Ambient OSGi Framework

Stéphane Frénot, University of Lyon, INRIA INSA-Lyon, F-69621, France

Frédéric Le Mouël, University of Lyon, INRIA INSA-Lyon, F-69621, France

Julien Ponge, University of Lyon, INRIA INSA-Lyon, F-69621, France

Guillaume Salagnac, University of Lyon, INRIA INSA-Lyon, F-69621, France

ABSTRACT

*OSGi is a wrapper above the Java Virtual Machine that embraces two concepts: component approach and service-oriented programming. The component approach enables a Java run-time to host several concurrent applications, while the service-oriented programming paradigm allows the decomposition of applications into independent units that are dynamically bound at run-time. Combining component and service-oriented programming greatly simplifies the implementation of highly adaptive, constantly evolving applications. This, in turn, is an ideal match to the requirements and constraints of ambient intelligence computing, such as adaptation to changes associated with context evolution. OSGi particularly fits ambient requirements and constraints by absorbing and adapting to changes associated with context evolution. However, OSGi needs to be finely tuned in order to integrate ambient specific issues. This paper focuses on Zero-configuration architecture, Multi-provider framework, and Limited resource requirements. The authors studied many OSGi improvements that should be taken into account when building OSGi-based gateways. This paper summarizes the INRIA Amazones teamwork (*http://amazones.gforge.inria.fr/*) on extending OSGi specifications and implementations to cope with ambient concerns. This paper references three main concerns: management, isolation, and security.*

Keywords: Ambient Intelligence, Component-Based Software Engineering, OSGi Middleware, Service-Oriented Programming

Introduction

Using OSGi technology in ambient environments requires focusing on specific problems such as footprint of the run-time framework, zero configuration of the application and service provisioning for multi-provider environments. Because ambient intelligence is, and will remain, based on hardware with limited resources, the size and complexity of the framework have to be kept under control. The kind of platform we address is that of middle-sized devices, like smart phones, set-top boxes or automotive embedded systems. They have much more computing resources than tiny embedded systems like micro-controllers, but much less then commodity PCs or traditional servers. We call these platforms gateway devices, since most ofthe time they act as intermediaries between a local network of services and the Internet. As an illustration, the platforms we used in our experimentations were ARM-based devices as the LinkSys NSLU2 (266Mhz CPU, 32MB RAM, 8MB Flash) or sheeva PC plugs (1.2Ghz CPU, 521MB RAM, 512MB flash).

Devices for ambient environment should work in an autonomic way without any user interaction apart from network and electrical connections and hardware factory resets. They should address many kinds of concurrent applications from many providers. Each of them shall have its own running space, where he is able to manage its own local information and interact with local

equipments. This management model is similar to the Apple and Android store model where the end-user has the ability to choose its hosted applications, and where each of them may have its own autonomy. This implies a dynamic architecture where each service provider may have an application life-cycle that is neither bounded nor constrained by the gateway system and hardware life-cycle. Furthermore, various external constraints such as costs and environmental issues distinguish gateway hardware from data-centers. The former has resource constrains both in memory and processing power that are not compliant with full best-effort developed applications.

In this article, we compiled most of our current OSGi-related proposals in order to have a synthetic view of the investigated extensions. The paper is divided in three sections. First we summarize the OSGi framework and focuses on our specific concerns. Next, we present each provided extension as a walkthrough of our various publications. The last section synthesizes our proposed extensions.

OSGi Context

OSGi (*http://www.osgi.org/Main/HomePage*) is a container framework built on top of the Java platform. It hosts deployment units called *bundles*, which contain Java resources such as compiled classes, properties files or dynamically linked native libraries. Each bundle features an *Activator* class, which is the entry point to be notified when the bundle is started or stopped. A descriptor, expressed as a regular Java manifest, details meta-data such as the *Activator* qualified name, or the various requirements the bundle expects, such as the presence of another bundle exposing a specific Java package.

The OSGi platform automatically checks dependencies between bundles and controls the lifecycle of each bundle. One key feature of the OSGi framework is the seamless support of application deployment: new applications can be installed, updated and uninstalled at runtime without requiring a restart of the Java virtual machine itself, thanks to classloaders native isolation. This streamlines administration enables multiple hosted applications installed as independent deployment units.

Bundles are typically materialized as JAR archives that can even be fetched by the OSGi framework from a remote HTTP server. Each bundle is associated with a dedicated Java class loader that provides resource isolation with respect to the other bundles. Unlike the local, closed classloader hierarchies found in standard Java applications, OSGi provides a dynamic hierarchy of classloaders with a fine control of which classes and resources should be exposed or not. At deployment time, the bundle manifest indicates which packages are made public and which packages the bundle depends on.

The OSGi underlying programming model focuses on Service-Oriented Programming (SOP) (*http://openwings.org/*). The driving motivation for SOP is to minimize coupling among software elements of an application. A service is an interface that describes what a software entity provides in terms of API binding requirements. In turn, several implementations can be provided for the same interface and their details remain hidden to the service requesters. Combining dynamic bundle management and SOP awareness enables continuous application evolution at run-time within the same virtual machine instance.

OSGi Extensions

Ambient architectures we studied so-far exposed three main concerns: gateway autonomy, multi-provider application hosting and security. Also OSGi provides many ways to help develop and enforce ambient application in a generic and standard way, it still presents limitations. Since 2004, we provided many extensions to the OSGi framework to cope with these limitations. Most of them focus on the bundle management layer and aim at providing sound answers to the next three questions:

1. How can we improve bundle management and deployment?
2. How can we optimize the run-time environment for bundles?
3. How can we enforce security for the deployed bundles?

Deployment and security concerns influence autonomic behavior. Deployment, Run-time optimization and Security are of high consideration when building a multi-provider architecture. Run-time optimization and Security enforcements directly impacts limited resource equipments where resource control is a matter of importance. In the three next sections, we detail the various OSGi extensions trying to answer those questions.

Bundle Management and Deployment

When dealing with ambient environments we consider that the gateway is remotely managed by providers and end-user and that it performs as many autonomous activities as it can on low cost, low energy, low noise electronic device. Although OSGi enables software management within gateways, it initially has a very primitive way of handling service deployment and management. An initial remote console to interact with bundle was the only provided management solution. Service providers must be able to install new applications on remote gateways and be able to both manage their behaviors and monitor their execution (Royon & Frénot, 2007). In this purpose we provided two extensions: the first is a full implementation of OSGi remote management specification and the second one focuses on an efficient bundle deployment layer.

MOSGi

Management was one of the OSGi specification concerns, which were not implemented in the first place. We developed and donated to the Apache Felix community (*http://felix.apache.org/site/index.html*) the MOSGi framework that mainly consists in a wrapper around the JMX management protocol (Fleury & Frénot, 2003). We provided an end-to-end management layer that covers both the gateway and the client-side part of the architecture. One can find both a run-time for the gateway handling local probes and a run-time manager for the client.

It is worth noting that the manager was also developed as an OSGi bundle and thus provides a really interesting dynamic use-case. When a probe is started on the gateway side, it provides two facets: (1) the MBean facets that provides the management API compliant with the JMX protocol, (2) a specific MOSGi facet that is used by the client side to get a graphical user interface interacting with the remote probe. The client is dynamic: when it connects to the gateway, it requests for the list of installed probes that correspond to the JMX facet integration. Then, for each probe it checks whether it complies with the specific MOSGi facet. If so, it asks the probe for an URL that provides a bundle that brings the graphical user interface. This dynamically discovered graphical bundle is plugged into the manager and starts interacting with the gateway. As an example, the running use-case of the MUSE project (D'Haeseleer, 2008) was that of a user buying a fridge. When the fridge is installed at the home environment, the set-top box gateway downloads a fridge management probe. When the fridge vendor has to remotely access the fridge, its local manager downloads a graphical user interface that can communicate with the fridge probe and provides a dedicated management user interface.

When we developed the MOSGi framework in a multi-provider environment, one issue was to understand the management limits. If any provider can ask for probing information, the management layer can overload the gateway. If the remote manager asks for the fridge temperature every tenth of a second the gateway can fall in some kind of Denial of Service state and not be able to do other things. We developed a monitoring scheduling process for gateway devices (Frénot et al., 2008) through which we wondered if we could anticipate the increase in CPU load when a new probe is inserted into the gateway. The provided approach is a specific bench generator that measures every installed JMX probe and stores its response time. We

showed that the targeted gateway CPU loading curve was regular, and that the combination of the various curves leads to good CPU load schedules. If the equipment provider indicates the probe and the pace he wants to query, we can anticipate the CPU load increment. By way of consequences, we can make a distinction between normal behavior and a probe failure since we can observe and anticipate a homogeneous load activity.

POSGi

Traditionally, OSGi bundles are installed from a remote web server using an HTTP URL such as *start http://www.somehost.com/abundle.jar*. But in ambient environment, simultaneous deployment of the same software component onto a large number of devices may overwhelm the server behind www.somehost.com. The server behind www.somehost.com may host many bundles and can be overwhelmed by a peak in installation requests.

Our Peer-to-Peer (P2P) OSGi extension focuses on efficient and resilient deployment of OSGi bundles. The goal of this system is to substitute the traditional centralized bundle repositories with a P2P architecture. By using a P2P approach, we can avoid server failure during peak installation request period since every gateway can be part of the global, distributed repository. We used Freepastry (*http://www.freepastry.org/*) as a Distributed Hash Table (DHT) overlay network where every gateway and every bundle obtains a specific Freepastry id. DHT yields an efficient routing algorithm between two ids. When sending a new bundle to the network of gateways, it is routed towards the gateway that has the nearest Freepastry id from its own id. When one gateway needs to install a bundle, its request is routed hop-by-hop towards the root gateway that handles the initial archive. On the way back, each gateway on the route can locally store the initial archive. If, later, another gateway needs the same bundle, it can be provided either by the root gateway or by any gateway that handles the bundle on the route. POSGi (Frénot & Royon, 2005) enables URL installation schemes such as *p2p://abundle.jar*. The implementation impacts the OSGi URL scheme handler, adds an alternate local cache where bundles are stored from the Freepastry overlay and a bundle that offers the P2P management.

Discussion

Ambient environments are directly influenced by both deployment and management layers. When dealing with multiple providers acting on the same equipment, each one must have their own management vision of it. If for instance the home gateway manages a specific fridge and a specific vacuum cleaner, each provider may remotely manage these equipments without interfering with others. The MOSGi architecture is a first direction that enables this function. If one of the providers must upgrade to a new version all its bundles a P2P approach such as POSGi presents many advantages in regards to efficiency. For the end-user, an autonomic deployment and management layers are very important, since the end-user equipment should be as simple and standard as other legacy ones. For instance, in a home gateway, there should be no user interface at all. MOSGi and POSGi frameworks bring autonomic and multi-provider support for deploying and managing applications in ambient environments without any user interaction.

This section described the early phases of bundle life-cycle; the next section focuses on the main execution phase through various run-time extensions.

Run-Time Extensions

Java virtual machines combined with the OSGi framework is the operating system solution for handling multi-provider, multi-applications within a heterogeneous hardware environment. Nevertheless Java was not designed with a multi-session philosophy in mind. OSGi is a direction in providing component isolation, but it still has the best-effort scheme imposed by Java. Bundle isolation is not as perfect world as we can imagine, if a bundle does not behave in the proper manner it can stop the virtual machine. By specifications, OSGI, it's all sweetness and light. In a multi-provider and autonomic approach, unless a global manager holds everything, we need to

be much more on our guards. A stopped bundle may be effectively stopped, which implies that al reserved resources during bundle execution should be released when stopped. As it is not the case, Amazones observed this issue through many directions.

OSGi Sandboxing

OSGi sandboxing aims at isolating bundle execution and management one from each other. The sandboxing issues we initially investigated were related to the MUSE multi-provider architecture. Indeed, if we host many bundles from various providers on the same gateway, they should be isolated one from each other as they could be in competition. One bundle should not interfere with the one from the other providers. Sandboxing has many levels of granularity on virtual machines: (1) bundle naming isolation, (2) coding design for proper isolation, and (3) low-level resources isolation within a virtual machine. We investigated all these three level of isolation.

Virtual OSGi VOSGi (Royon et al.,2006) is a rather naive implementation where we provide OSGi itself as a service. The service provides a single start/stop interface that instantiates or nullifies a brand new OSGi implementation within the first one. With this, we can assign a specific provider to an OSGi gateway that virtually runs within another OSGi gateway. We provide a start/stop gateway management shell command available from the main gateway, which is in turn called the *core gateway*. When running multiple OSGi instance in the same management shell, commands are not associated with a specific gateway. In order to interact with each virtual gateway, we developed a specific MOSGi probe that enables to remotely manage the virtual instances.

Although running new OSGi instance within one instance is rather straightforward, difficulties raise when these instances need to communicate with each other. A virtual gateway may use services provided by another gateway, either the core gateway or another virtual gateway. Our current implementation only enables service exchanges from the core gateway to virtual gateways. When starting a virtual gateway, it declares the service interfaces it wishes to access from the core gateway. At virtual gateway bootstrap time, it receives references to the various corresponding implementations and registers each transmitted implementation within its own registry. Hence, each service exchanged between a core and a virtual gateway is declared in both registries and points to the same implementation.

Nevertheless, one remaining problem in the VOSGi approach is that the OSGi "standard" behavior does not enforce "true" bundle isolation: bundles can obtain as many system resources as they want such as network connections, threads, and so on. There is no way to constrain a bundle contract within the framework. Moreover, any class can stop the entire virtual machine through a call to *java.lang.System.exit(0)*. This is a serious issue when hosting multi-provider bundles that may not be "honest". We focused on two developments to improve bundle isolation. The first one is based on development contract conformance enforcement, while the second one is based on a dedicated virtual machine.

Suspend and Resume (Dunklau & Frénot, 2009). This INRIA technical report raises one simple problem linked to the bundle lifecycle. In the OSGi specifications, bundles can be in the following states: *Installed, Resolved* and *Active*. Although this life cycle seems complete, we believe that it lacks a *Suspended* state. Considering that ambient equipment may be paused and resumed and not systematically rebooted, the Suspended state is a real ambient need as bundle may be suspended in order to free its resource, in case of exhaustion. For instance, the gateway manages run-time profiles, such as *"high-priority", "standard",* or *"night supervision"*. It may have to pause bundles when switching to specific profiles. This suspend state necessity arises each time we need to reconfigure partially the gateway. At present, the only way of doing it is by managing an internal state when stopping and starting the bundle. It is impossible to distinguish between rebooting and suspending a bundle. One problem raised by introducing the

Suspended state is that it impacts mostly the run-time model. The Suspended state needs to suspend threads, network connections and all running activities.

When dealing with the programming model, accessing resources is restricted to some kind of dispatcher. It allocates resources to requesters and maintains a whiteboard associating requesters to resources. When suspension time occurs, the dispatcher knows exactly which resource i.e. bundle is associated to each requester. In this approach, every direct call to a specific resource allocation should be transferred to a dispatcher that manages identifications. Our implementation adds a specific Apache Felix shell command that integrates the *suspend #bundleid* action and a specific *BundleSuspendableActivator* Java interface that adds two hooked methods for suspending and resuming a bundle. Our architecture currently focuses on the threading architecture and only provides a dispatcher to allocate threads and suspend them automatically. A similar work should be done for network connections.

Although this architecture is rather simple to put under operation, it is still dependent on the good will of bundle developers: if they provide only standard OSGi bundle, these can still be hosted on the gateway, but may present unmanageable behavior. The only workaround for this issue is to constrain bundle behavior at the virtual machine level.

iJVM The iJVM (Geoffray et al., 2009) project aims at designing a virtual machine for constrained environments that enforces resource control mechanisms. Even though iJVM had security issues in mind, its main goal is to provide an efficient isolation layer for bundle requesting resources while preserving an efficient communication layer between isolated bundles. iJVM provides three main features for OSGi bundles: (1) memory isolation, (2) resource accounting and (3) isolates termination.

Discussion

VOSGi, Suspend and Resume, and iJVM projects target run-time isolation for bundle code. The OSGi bundle paradigm offers the appropriate granularity level to enforce isolation. Depending on the desired isolation guarantees, each approach offers different level ranging from simple naming isolation, to a hard, resource constrained, environment. But the harder the constraints are enforced, the lesser standard and generic the architecture is.

While trying to enforce isolation and since target hardware platforms have limited resources, we also addressed run-time code optimization.

Run-Time Code Optimization

Most of the time bundle life-cycle is human driven either by the end-user or the remote provider. In our run-time optimization investigations, we tried to find more automated ways for handling those updates. Two directions were developed for tackling them: the ROCS architecture and the AxSel framework.

ROCS The Remotely Provisioned OSGi framework for Ambient Systems (Frénot et al., 2010) project leverages a very simple standard Java principle, which is that of remote classes loading. Before starting the activator of a bundle, the OSGi framework performs the following activities. It downloads the bundle from a remote location, extracts the jar file on a local cache, extracts the bundle descriptor to control bundle coherence with the current running environment and, provided everything is right, it loads the activator class and invokes its *start* method. ROCS proposes to manage all these activities through a remote approach. The gateway relies on a remote server that hosts the bundles and responds to gateway remote queries. Exploiting the Java remote class loading is straightforward with the OSGi framework. The usage scenario is the following.

When installing a new bundle, it is downloaded from the repository to a remote server and the gateway obtains a local proxy to interact with the remote server. It asks for the bundle manifest,

downloads the file from the remote Jar and controls the meta-data to see if every dependency is satisfied. If so, it remotely loads into memory the corresponding *BundleActivator* byte-code and starts the bundle. Then, all subsequent required classes are downloaded on-demand through the *RemoteClassLoader.*

This remote architecture has also been applied to the standard Java runtime classes (i.e., *java.** and *javax.** packages) such that it is considered as a plain bundle. We put the necessary classes to bootstrap a minimal OSGi/Java stack into a local classpath classloader. Once started, all remaining standard classes are downloaded from the remote location. Trough our approach, we build an entire OSGi/Java run-time stack that can be hosted on small equipments, less than 8MB of flash, since all loaded classes come from remote locations and are directly brought into memory. ROCS demonstrates that, provided we are connected, we have the same kind of response time as if, we locally cached all bundle classes.

AxSel Whereas ROCS focuses on loading classes contained in remote bundles, AxSel (Ben Hamida et al., 2008) aims at optimizing class loading at the service granularity level. The Service-Oriented Programming feature of OSGi specifications enables the description of an applications as a composition of services. One application can have implementation variations, depending on the environment. For instance, when using a standard OSGi bundle repository, all dependent bundles are automatically downloaded into the gateway, and then started. A local service maintains a dependency graph from *Package-Import* and *Package-Export* statements of the bundle manifest. We exploit the *ImportService* manifest property to also maintain a dependency graph between parts of the application. The AxSel framework holds an internal representation of the run-time and regularly polls remote repositories to find new or alternate implementations. When the gateway is in idle mode, it triggers a reconfiguration process that calculates for every dependent element, bundle or service, if it can find a better implementation. If so, bundles are updated and services are automatically reconfigured.

Discussion

AxSel and ROCS are complementary approaches: the ROCS system enables a minimal run-time where everything is downloaded on-demand within the main memory, and AxSel brings automatic reconfiguration and optimization of the running environment.

Ambient environments need to cope with autonomy, multi-providers and some resource control issues. The various sandboxing frameworks we studied so far aim at empowering application autonomy and multi-provider integration within ambient environments. They enable providers behave as if they were using the system alone. Achieving autonomy is much easier as you can do whatever you want without disturbing your neighbors. Although sandboxing enable application to feel isolated, the underlying system still has to cope with its limited resources. Axsel and ROCS are a first proposal toward dynamic resource management in ambient environment. ROCS focuses on getting the minimal quantity of execution code into memory, whereas Axsel aims at continuously finding better configuration opportunities. They both influence autonomy and resource awareness of ambient systems.

Most of the dynamic features of OSGi framework presented so far, load byte-code into memory from remote locations or from downloaded bundles. We identified many security issues when dealing with isolation. All these concerns led to consider security as specific and generic concern that needs dedicated attention.

Security around OSGi

Most of the time security in ambient environment is a primary concern, and most of the time security in constrained environment is disabled since it has a very high cost. OSGi mostly extends Java security elements without considering their run-time cost. Amazones felt that OSGi

component and service paradigms bring new clues for improving application security while also bringing new ways of disturbing an environment. We started a security activity around OSGi to identify both new breakages and design new ways of handling security.

Security issues in the OSGi/Java stack are spanning across many points. They arise both at low-level (e.g., bundle code signing) and high-level parts of the OSGi stack (e.g., deployment). The overall OSGi picture reveals many points where security needs enforcements. Our overall approach identified the following elements.

Deployment Bundles are downloaded from remote locations. Thus, we need some strong guarantees from those locations regarding the origins and integrity of the bundles. The OSGi specification proposes a signing process that identifies bundle sources and providers, but when we started our study, no implementation was available.

Bundle Downloaded bundles can lead to two kinds of security breaches. Indeed, they can contain code that directly harms the system, or they can contain code that weakens the framework. We need to consider the two issues to decide whether a bundle is a threat to the framework, and whether it opens security flaws that could be locally/remotely exploited.

Framework The OSGi specification does not enforce security constraints, as they are mainly implementation issues. Most of current framework implementations are subject to instability if malevolent bundles are installed.

A preliminary work has been conducted to design a bundle signing architecture. This work has been achieved in the context of the MUSE European project where we designed a tool-chain to sign bundles and verify their validity at deployment time. The architecture is detailed in Parrend and Frénot (2006, 2007b). An implementation is available with the SFelix project (*http://sfelix.gforge.inria.fr/*).

After having designed this tool-chain, we worked on a bundle threats catalog aimed at identifying OSGi security weaknesses. The catalog elements and summary are detailed in Parrend and Frénot (2007a, 2008a, 2008c).

We applied those catalogs to design a hardened OSGi/Java framework. The Parrend & Frénot (2009) article stresses OSGi open-source framework vulnerabilities and expresses guidelines to develop hardened implementations. Yet, some vulnerabilities cannot be controlled at the OSGi framework layer and need to be addressed at the Java Virtual Machine layer. The iJVM (Geoffray et al., 2009) virtual machine for isolating threads was mainly designed for this purpose.

The last track we followed when dealing with security concerns was to be able to statically analyze the code of bundles. We developed an install-time analyzer that introspects bundles compiled code to detect vulnerabilities. CBAC (Parrend & Frénot, 2008b) describes the proposal. The CBAC control is triggered at the same time as the digital verification signature process, thus only inducing an overhead at deployment time. Unlike the standard Java permissions built-in framework, the CBAC security model does not have any runtime overhead.

Discussion

Security improvement within ambient environment is a primary concern. Autonomic computing, multi-provider and limited Resources concerns are all influenced by security issues. When installing, managing, upgrading applications, one has to guarantee that the system still behaves as before, that it does not consume much more resources and that it does not break provider isolation by trying to transfer private data throughout the system. Limited resource architecture influence security architecture since they all come with a non negligible cost and most of the time they need to be disabled in order to have a viable system. Amazones proposals cover many directions to easy security management. They all converge to some kind of benchmark tools to control ambient system security conformity but both metrics and levels still need proper definitions.

Synthesis

The OSGi extensions that we presented address issues such as management, run-time optimization and security. We summarized our contributions in Table 1.

Extension	Domain	References
MOSGi	Management	Royon & Frénot, 2007; Fleury & Frénot, 2003; Frénot et al., 2008
POSGi	Management	Frénot & Royon, 2005
VOSGi	Runtime	Royon et al., 2006
Suspend & Resume	Runtime	Dunklau & Frénot, 2009
iJVM	Runtime	Geoffray et al., 2009
ROCS	Runtime	Frénot et al., 2010
AxSel	Runtime	Ben Hamida et al., 2008
SFelix	Security	Parrend & Frénot, 2006, 2007b
Unsecured Bundle Catalog	Security	Parrend & Frénot, 2007a, 2008a, 2008c
Hardened OSGi	Security	Parrend & Frénot, 2009; Geoffray et al., 2009
CBAC	Security	Parrend & Frénot, 2008b

Table 1. Extensions to the OSGi space

All our extensions have been developed under the Apache Felix project. Some of them, VOSGi, MOSGi, Unsecured Bundle Catalog have also been validated on Concierge (Rellermeyer & Alonso, 2007). They all run on standard Intel-based architectures, but many of them, MOSGi, VOSGi, ROCS, SFelix, have also been validated on a NSLU2 (*http://en.gentoo-wiki.com/wiki/Gentoo/_on/_NSLU2*) ARM board, with a dedicated JamVM/GNU Classpath execution stack. All the code that we developed is available from the INRIA GForge project and can be obtained by sending us an email. All code is provided under either the Apache Software License version 2.0 (*http://www.apache.org/licenses/*), or one of the CeCill licenses (*http://www.cecill.info/*).

The provided extensions impact various elements of the OSGi specifications and implementations as summarized in Table 2.

Specification implementation MOSGi and SFelix propose implementation of OSGi specifications that are not included in the open-source implementations.

Framework patch VOSGi, Suspend & Resume, ROCS, Hardened OSGi need patching some specific version of Apache Felix to operate. Most of the time, the patches correspond to very few lines of code and some extended classes.

Bundle lifecycle modifications Some of our extensions point out the necessity of extending the standard bundle lifecycle (*Resolved, Installed, Active*) through new status. *Suspended* for Suspend & Resume, *Invalid* for SFelix.

Tool Tools are elements that do not interfere with the standard OSGi/Java stack. They provide non-intrusive additional services to the framework.

Since iJVM impacts the underlying virtual machine it does not directly concern the OSGi framework.

OSGi is a very interesting environment. Its simplicity and pragmatism are keys to build efficient run-time systems. The INRIA Amazones team focuses on ambient environments and

Extension	Specification	Patch	Lifecycle	Tool
MOSGi	X	O	O	X
POSGi	O	O	O	X
VOSGi	O	X	O	O
Suspend & Resume	O	X	X	O
iJVM	O	O	O	O
ROCS	O	X	O	O
AxSel	O	O	O	X
SFelix	X	O	X	X
Unsecured Bundle Catalog	O	O	O	X
Hardened OSGi	O	O	O	O
CBAC	O	X	X	O

Table 2. Impacted OSGi elements conclusion

provides extensions that cope with ambient constraints such as limited memory size, remote access, dynamism, context awareness and energy efficiency. Our extensions depend on framework implementations, and as such, they are rather difficult to keep up-to-date with upstream codebase changes. Our current policy is to keep those extensions alive for specific upstream project versions, and to provide upgrades when needed. At present, and due to limited resources within our team, only the MOSGi framework has been donated to the main Apache Felix codebase, under the *mosgi* submodule in the project Subversion repository.

We are still working on various extensions to the framework in different directions. We integrate device mobility concerns in some of our projects and log management features to enhance fault management. However, those projects are still under development and need further validation before publication.

Acknowledgments

Most of this work would not have been made without PhD students and research engineers. PhD works, mostly in French, of Ben Hamida (2010), Ibrahim (2008), Parrend (2008) and Royon (2007) had direct impacts on our extensions and we gratefully thank them all.

References

Ben Hamida, A. (2010). *AxSeL: Un intergiciel pour le déploiement contextuel et autonome de services dans les environnements pervasifs.* Unpublished doctoral dissertation, INSA de Lyon, Villeurbanne, France.

Ben Hamida, A., Le Mouël, F., Frénot, S., & Ben Ahmed, M. (2008). A graph-based approach for contextual service loading in pervasive environments. In R. Meersman & Z. Tari (Eds.), *Proceedings of the Federated International Conferences of On the Move to Meaningful Internet Systems*, Monterrey, Mexico (LNCS 5331, pp. 589-606).

D'Haeseleer, S. (2008). *DB 3.1: Detailed requirement-based functional specification of gateway: Part 1: Private network* (Tech. Rep. No. MUSE IST-6thFP-507295). Brussels, Belgium: MUSE.

Dunklau, R., & Frénot, S. (2009). *Proposal for a suspend/resume extension to the OSGi specification* (Tech. Rep. No. RR-7060). Le Chesnay, France: INRIA.

Fleury, E., & Frénot, S. (2003). *Building a JMX management interface inside OSGi* (Tech. Rep. No. RR-5025). Le Chesnay, France: INRIA.

Frénot, S., Ibrahim, N., Le Mouël, F., Ben Hamida, A., Ponge, J., Chantrel, M., et al. (2010). ROCS: A remotely provisioned OSGi framework for ambient systems. In *Proceedings of the Symposium on Network Operations and Management*, Osaka Japan (pp. 503-510). Washington, DC: IEEE Computer Society.

Frénot, S., & Royon, Y. (2005). Component deployment using a peer-to-peer overlay. In A. Dearle & S. Eisenbach (Eds.), *Proceedings of the Third International Working Conference on Component Deployment*, Grenoble, France (LNCS 3798, pp. 33-36).

Frénot, S., Royon, Y., Parrend, P., & Beras, D. (2008). Monitoring scheduling for home gateways. In *Proceedings of the Symposium on Network Operations and Management*, Salvador de Bahia, Brazil (pp. 411-416). Washington, DC: IEEE Computer Society.

Geoffray, N., Thomas, G., Muller, G., Parrend, P., Frénot, S., & Folliot, B. (2009). I-JVM: A Java virtual machine for component isolation in OSGi. In *Proceedings of the International Conference on Dependable Systems and Networks*, Estoril, Portugal (pp. 544-553).

Ibrahim, N. (2008). *Spontaneous integration of services in pervasive environments.* Unpublished doctoral dissertation, INSA de Lyon, Villeurbanne, France.

Parrend, P. (2008). *Modèles de Sécurité logicielle pour les plates-formes à composants de service (SOP).* Unpublished doctoral dissertation, INSA de Lyon, Villeurbanne, France.

Parrend, P., & Frénot, S. (2006). *Secure component deployment in the OSGi(tm) release 4 platform* (Tech. Rep. No. RT-0323). Le Chesnay, France: INRIA.

Parrend, P., & Frénot, S. (2007a). *Java components vulnerabilities an experimental classification targeted at the OSGi platform* (Tech. Rep. No. RR-6231). Le Chesnay, France: INRIA.

Parrend, P., & Frénot, S. (2007b). Supporting the secure deployment of OSGi bundles. In *Proceedings of the International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Helsinki, Finland (pp. 1-6). Washington, DC: IEEE Computer Society.

Parrend, P., & Frénot, S. (2008a). Classification of component vulnerabilities in Java service oriented programming (sop) platforms. In M. R. V. Chaudron, C. Szyperski, & R. Reussner (Eds.), *Proceedings of the 11th International Symposium on Component-Based Software Engineering*, Karlsruhe, Germany (LNCS 5282, pp. 80-96).

Parrend, P., & Frénot, S. (2008b). Component-based access control: Secure software composition through static analysis. In C. Pautasso & E. Tanter (Eds.), *Proceedings of the 7th International Symposium on Software Composition*, Budapest, Hungary (LNCS 4954, pp. 68-83).

Parrend, P., & Frénot, S. (2008c). More vulnerabilities in the Java/OSGi platform: A focus on bundle interactions (Tech. Rep. No. RR-6649). Le Chesnay, France: INRIA.

Parrend, P., & Frénot, S. (2009). Security benchmarks of OSGi platforms: Towards hardened OSGi. Journal of Software Practice and Experience, 39(5), 471–499. doi:10.1002/spe.906

Rellermeyer, J. S., & Alonso, G. (2007). Concierge: A service platform for resource-constrained devices. In *Proceedings of the European Conference on Computer Systems* (pp. 245-258). New York, NY: ACM Press.

Royon, Y. (2007). *Environnements d'exécution pour passerelles domestiques.* Unpublished doctoral dissertation, INSA de Lyon, Villeurbanne, France.

Royon, Y., & Frénot, S. (2007). Multiservice home gateways: Business model, execution environment, management infrastructure. *IEEE Communications Magazine,* 45(10), 122–128. doi:10.1109/MCOM.2007.4342834

Royon, Y., Frénot, S., & Le Mouël, F. (2006). Virtualization of service gateways in multi-provider environments. In I. Gorton, G. T. Heineman, I. Crnkovic, H. W. Schmidt, J. A. Stafford, C. Syzperski et al. (Eds.), *Proceedings of the 9th International Symposium on Component-Based Software Engineering*, Stockholm, Sweden (LNCS 4063, pp. 385-392).

Stéphane Frénot is associate professor at the Center for Innovation in Telecommunication and Integration of Services (CITI Lab.), Telecommunications Department of the National Institute for Applied Sciences of Lyon (INSA Lyon, France). He leads the Amazones (AMbient Architectures: Service-Oriented, Networked, Efficient and Secured) INRIA team. His main interests are middlewares, component instrumentation, component security, deployment and management, constrained devices and specially home gateways.

Frédéric Le Mouël is associate professor in the National Institute for Applied Sciences of Lyon (INSA Lyon, France), Telecommunications Department, Center for Innovation in Telecommunication and Integration of Services (CITI Lab.) (2004-). He is also member of the AMbient Architectures: Service-Oriented, Networked, Efficient, Secure research group at INRIA (Amazones Team). He holds a master's degree in Languages and Operating Systems (1997) and a Ph.D. degree in Computer Science and Telecommunications (2003) from the University of Rennes 1, France. His dissertation focused on an adaptive environment for distributed executions of applications in a mobile computing context. His main interests are service-oriented architectures and middleware, specifically in the fields of dynamic and autonomic configuration, adaptation, composition, orchestration of services. He is specially studying these topics in the domains of constrained and mobile devices in ambient intelligent environments.

Julien Ponge is an Associate Professor in Computer Science and Engineering at INSA de Lyon, a leading engineering school in France. He teaches in the Department of Telecommunications, Services and Usages while he does his research activities as part of the CITI Laboratory/INRIA in the Amazones group. Prior to joining INSA-Lyon, he obtained a PhD in Computer Science and Engineering from Université Blaise Pascal, Clermont-Ferrand, France, as well as a PhD under cotutelle agreements from the University of New South Wales, Sydney, Australia. He was also a temporary lecturer at ISIMA, the computer science and engineering school of Université Blaise Pascal. His current research interests cover middlewares, service-oriented architectures, next-generation software distribution and deployment.

Guillaume Salagnac is an associate professor in the Information Technology department of INSA-Lyon University, and a member of the CITI research laboratory, in the INRIA Amazones research group. He holds a PhD (2008) from Grenoble University, where he studied real-time automatic memory management for embedded Java applications. After that, he was a post-doctoral researcher at the CSIRO ICT Centre in Brisbane, Australia, where he worked on high-level programming for Wireless Sensor networks. His current research interests lie in the area of programming languages and operating systems for embedded platforms.