Chapter 7.12
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 runtime. 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.

DOI: 10.4018/978-1-61350-456-7.ch7.12

INTRODUCTION

Using OSGi technology in ambient environments requires focusing on specific problems such 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 of the 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 life-cycle 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