Dynamic Application Frameworks using OSGi and Beanome

Humberto Cervantes, Didier Donsez, Richard Hall

Université Joseph Fourier, IMAG, LSR
Bat. C, 220 rue de la Chimie, Domaine Universitaire
BP 53, 38041 Grenoble Cedex 9, France
Tel : +33 4 76 63 55 49 Fax : +33 4 76 63 55 50
Humberto.Cervantes@imag.fr
Didier.Donsez@imag.fr
Richard.Hall@imag.fr

Abstract. This article discusses how to support the development of applications that exhibit dynamic behavior with respect to extensible functionality, deployment, and administration. The paper begins with a introduction to OSGi and Beanome. It continues with the architecture proposals for an iTV platform, a peer-to-peer collaborative framework and a mobile agent platform.

Keywords. OSGi, Deployment, Interactive TV, Peer-to-Peer, Mobile Agent.

1 Introduction

Application frameworks save software developers time by raising the abstraction level of development for particular problem domains. In the past, for example, many application frameworks focused on graphical user interface (GUI) frameworks, which simplified creating GUI applications by providing standard application structure and event handling capabilities. In the last few years, dynamic application frameworks used a similar approach to support runtime extension of application functionality using techniques such as runtime code loading.

The following three specific domains have either created or could leverage dynamic application frameworks:

- Interactive TV (iTV) - technology that lowers the barrier to online service access via a set-top box (STB) and television equipment; this includes content as well as service access.
- Peer-to-peer systems - technology that allows powerful client computers to form ad-hoc "networks of peers," where each computer is capable of acting as a client or a server in the network; generally, peer-to-peer systems perform a specific task, such as file sharing or work-group collaboration.

- Mobile agent platforms - technology that enables small programs to move autonomously around a network for purposes of reduced network bandwidth consumption; mobile agents are generally simple programs designed to perform a single, small task, such as searching the Web for the lowest price for a piece of computer hardware.

In general, these three domains are either built from scratch or upon hand-coded dynamic application frameworks; for example, DVB-MHP [1] for iTV, JXTA [8] for peer-to-peer, and Aglets [11] for mobile agents. Closer examination of these three technological areas reveals similarities in some of their core requirements. Specifically, all three share these characteristics:

- frequent and regular deployment of new or updated software components,
- coarse- and/or fine-grained extensibility of functionality (via dynamically loadable code),
- potentially non-stop execution,
- the need to limit resource consumption, and
- loosely administered control.

Partial approaches to deal with all of these issues exist in some form, but no framework has focused on providing a solution for the issues that are common for systems that exhibit the characteristics listed above. This paper presents the use of Open Services Gateway Initiative (OSGi) and an extension to it, called Beanome, as a platform for building dynamic frameworks. The combination of OSGi and Beanome provides mechanisms for dealing with the on-going deployment of software components, dynamic code loading, and efficient resource handling.

To illustrate how OSGi and Beanome are useful as the core of a dynamic framework, the three application domains listed above are described in more detail and a proposed architectural framework solution for each is presented that leverages OSGi and Beanome functionality. The next section describes OSGi and Beanome in more detail. The subsequent three sections describe the proposed architectural framework solutions for iTV, peer-to-peer systems, and mobile agent platforms, respectively, followed by the conclusion.

OSGi and Beanome

The Open Services Gateway Initiative (OSGi) [2], established in 1999, is an independent, non-profit corporation working to define and promote open specifications for the delivery of managed broadband services over different types of networks (e.g., HomePnP, HomeRF, CEBus, WiFi, X11, HAVi, etc.). OSGi mainly targets embedded devices with potential memory constraints (set-top boxes, residential gateways, alarm systems, ATMs, healthcare monitors, routers, cable modems, etc.). OSGi has defined the specification for an open services framework, also referred to as a services gateway, for which there currently exist several implementations, such as Sun's JES [17] and OSCAR [3]. OSGi expects services gateway to be widespread in the near future since more and more homes, vehicles, and offices are being equipped with devices and networks that are ideal targets for this technology.

As stated in the specification, "the primary goal of the OSGi service Framework is to use the Java programming language's platform independence and dynamic code-loading capability to make development and dynamic deployment of applications for small-memory devices easier." OSGi also defines the specification of a device access system, which supports automatic detection of attached hardware devices (web cams, smart card readers, etc.). The device access system can automatically download and start or update appropriate device drivers. This allows devices to be plugged and unplugged at any time; the device access system immediately propagates these changes to the registered services.

There are three important concepts in OSGi: *services*, *bundles*, and *contexts*. In OSGi, an application is designed as a set of cooperating services, with each service implementing a part of the application functionality. For instance, a word processor application may rely on a spell-checking service. Services are Java interfaces with associated implementation classes that perform specific functionality. Services are packaged in bundles; a bundle is a Java JAR file and the functional and deployable unit for services along with their associated resources, such as icons files and native libraries. Bundles are downloaded and installed in the OSGi framework. In each bundle, a special *activator* class manages the simple life cycle of the services shipped in the bundle.

Framework management includes installing, updating, starting, stopping and uninstalling the bundles. Management can be done in a local or remote manner and eventually a centralized server may remotely administer the services gateway as shown in figure 1, although this is not explicitly defined in the OSGi specification.

During execution, bundles might change their state and, as a consequence, services might be registered or unregistered. Because of this, clients must listen to events produced by the framework that announce the registering or unregistering of services. When a new service is registered, and if a client is interested in it, the client is notified so that it can ask the framework for a reference to the service. In the case when a service is unregistered and a client is using the service, it will again be notified, but this time it should discard any held reference to the service. Services are registered in the framework using their class name and a set of properties. A simple LDAP-based query mechanism enables bundles to request and use registered services. The bundle context is the execution environment of a bundle in the framework and enables access to the registry; the context is given to a bunde via its activator class at start-up.

The OSGi framework is dynamic in nature for the type of applications it targets, namely home networks in which devices that are connected to the network register services in the framework. These devices might be disconnected at any time and as a consequence bundles may ask the framework to unregister their services, so other bundles should be prepared to handle this situation.

There are two potential downsides to the standard OSGi model. The first one arises from the fact that all the complexity of service registering and connection must be done in the activator, which can be a complex task, and the second one arises from the fact that in OSGi there is no way to express static or dynamic dependencies between services, so it is not possible to describe the structure of a particular framework or application.

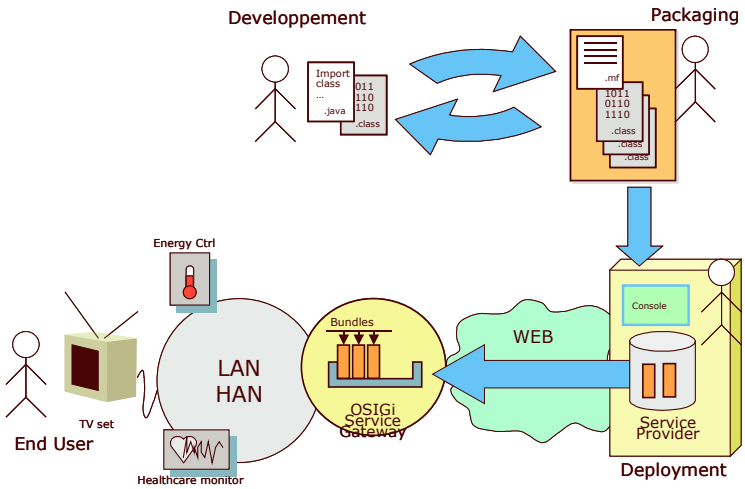

Fig. 1. Life cycle of bundles and services

Beanome [4] adds a layer on top of the standard OSGi model where some of the main concepts found in component models such as CCM [11] and COM [11] are introduced. Components can be described in an abstract way, with provided and required interfaces along with properties. Dynamically extensible frameworks and applications can be built by assembling Beanome components. The Beanome core is delivered itself as an OSGi bundle, and as such it can become the client to services that allow different capabilities to be added to it (for example, the core can make use of a service to render component instances persistent). Beanome and the OSGi framework do not deal with distributed applications.

OSGi has interesting features with respect to deployment and distribution (delivery) of the bundles along with simple (eventually remote) administration of the framework. Beanome completes OSGi by providing a component model to ease dynamic application/framework building. The next three sections describe the use of the OSGi/Beanome tandem to provide a dynamic execution framework for applications widely distributed with loose administration.

Interactive TV Terminals

Context

Interactive TV (iTV) is a new and promising application development field [5][6]. Since TV sets are omnipresent and they are simple to use without much skill, iTV is likely to be one of the major entry points to online services for the masses. iTV terminals, called set-top boxes (STB), can browse and play xHTML documents, Flash presentations and Java XLets. Java XLets is the equivalent to the Java applets for the

desktop HTTP browsers. The network infrastructure for iTV uses mainly one-way broadcast technologies such as cable, satellite, and terrestrial transmissions. The STB is sometimes connected to the network infrastructure via a telephone line to provide a upload link. The upload link enables access to remote servers, such as those of TV interactive games.

The TV viewer can subscribe the Pay TV channels (the stream is scrambled to restrict the access) or receive free and unscrambled channels. In both cases, the content provider may reach millions of STBs. The Pay-TV provider can not administer with accuracy its subscribers STBs, because the one-way broadcast network can not acknowledge installations, updates, and removals of new applications or part of the execution environment. In the case of free channels, the TV viewers buy their own STB and administer it themselves.

In this context, an application (i.e., Java XLet) must be deployed with minimum intervention of the content provider. All its components are downloaded by the broadcast link, installed, and started on-demand. Since the STB memory resources are limited, unused components must be removed when they are not used by any active application.

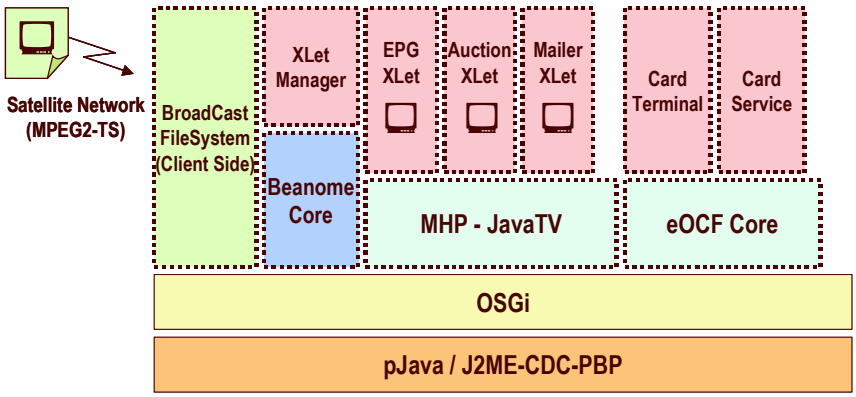

Fig. 2. Interactive TV Terminal Platform

Architecture proposal

The main interest of the tandem OSGi/Beanome is to support autonomous deployment and startup of iTV applications. Moreover, since the application may run for a long time, OSGi/Beanome enables component update and also allows several versions of the same component to coexist in the STB execution environment. Components may be part of dedicated applications or part of the iTV platform (DVB-MHP [1], JavaTV [14], OCF, Personal Profile library [12]) and are packaged as separate bundles (cf figure 2).

The deployment of a XLet can be initiated either by the content provider or by the TV viewer. The content provider pushes a list of applications for deployment through

the broadcast link. An example of these pushed applications is the Electronic Program Guide (EPG), which lists all the channels and iTV services. When the TV viewer selects an iTV service in the EPG panel, the corresponding XLet is deployed. In both cases, the XLet manager retrieves the XLet description in the broadcast link and downloads the bundles of all required components. As soon as all of these bundles are active, the application starts. Figure 2 illustrates three active XLet on the STB. The EPG requires a CardService to gather information preferences and authorization from the subscriber smartcard. When the viewer exists from a completed service, the corresponding XLet is undeployed and all the components are uninstalled if they are not shared by other XLets.

Peer-to-Peer Collaborative Applications

Context

Peer-to-peer computing promotes the maximum usage of all the low-cost individual computing and storage resources available on the Web. Unlike traditional distributed systems, peer-to-peer networks aggregate the millions of computers connected to the Web for purposes of file sharing (Napster, Gnutella [7]) or to leverage wasted machine cycles for supercomputation (SETI, Genome decoding). In a peer-to-peer network, each peer is both a client requesting resources and a server providing resources. Peer-to-peer computing provides various degrees of performance, reliability, scalability, and, in some cases, anonymity. However, in peer-to-peer networks, peers join and leave the network frequently, may not have permanent network addresses, or may be behind a firewall. Therefore, platforms such as Gnutella and JXTA [8] provide communication stacks to overcome these limitations.

This section focuses on one particular application of peer-to-peer: instantaneous collaborative platforms. A general collaborative platform (or groupware) is a software system that supports multiple users working together on a related task, such as editing a shared document, participating in a workflow process, piloting a simulated submarine robot, or playing a role in a multi-player game. The collaborative tool provides mechanisms to help users coordinate and keep track of the on-going collaborative task. The participant (i.e., the user) uses a set of predefined tools (e.g. a drum, a keyboard and a rhythmic box in a collaborative music workbench) to realize his part of the project.

Instantaneous collaborative platforms are collaborative platforms in which not all tools are predefined. The instantaneous platform provides a plug-in mechanism to load new tools on-demand when one of the participants starts to use it. The plug-in is automatically loaded and started on all the participants platforms. The plug-in code can be loaded from its manufacturer's centralized server or directly from the computer of the participant who starts first the tool in the peer-to-peer manner. Then, the replicated plug-ins of the tool cooperate by exchanging peer-to-peer messages (e.g., syn-

chronization events and internal state changes). At the end of the collaboration, the loaded tools may be automatically removed.

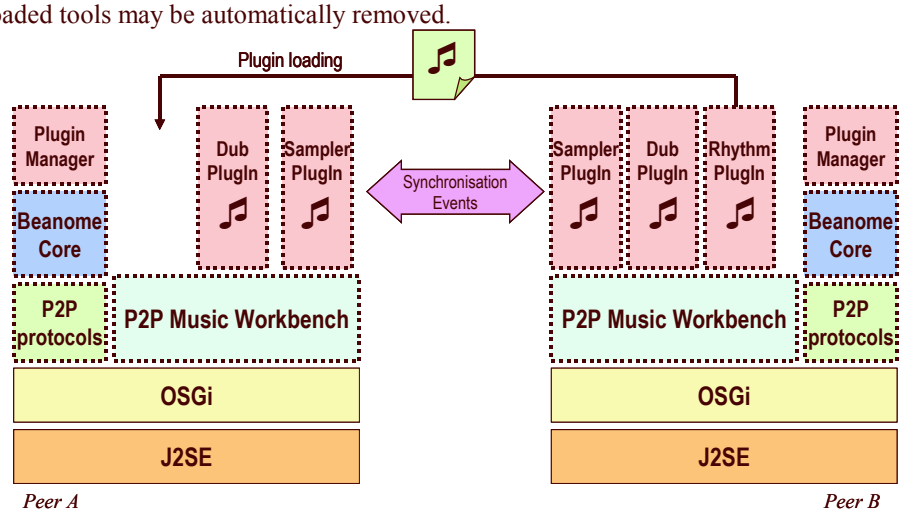

Fig. 3. P2P Collaborative Music Workbench based on OSGi

Architecture proposal

The instantaneous platform framework must load, install, and start plug-ins at runtime. Moreover, the peer-to-peer client should not require restarting when new plug-ins are added or existing plug-ins are freed. A platform built on OSGi/Beanome, benefits from safe on-demand class loading and secure code execution since plug-ins may be downloaded from untrusted peers.

The architecture of the platform is composed of the collaborative environment that represents the shared workspaces, the plug-in manager that downloads and starts the plug-ins, and the plug-ins that implement a specific tool and display the set of commands on the user panel. Plug-ins are designed as an set of services invoked by the collaborative environment. They may depend from the service of others plug-ins. When a participant actives a plug-in installed on his peer, the plug-in manager of a peer notifies all the plug-in managers of the other peers participating to the collaboration session. Then when a plug-in manager receives the notification of new started plug-in, it delegates to Beanome the location and the loading of the required plug-in.

Figure 3 illustrates two peers A and B that execute a collaboration music workbench to compose music. The music workbench is replicated in each peer. But peers may have initially different tools. When the user B starts his rhythmic box plug-in, the plug-in manager B notifies the plug-in manager A of the activation of a new plug-in. Then the plug-in manager A loads and starts the rhythmic box.

The collaborative environment, the plug-in manager and plug-in are packaged in OSGi bundles. Moreover, the communication protocols are also installed as bundles

since the protocols such as peer-to-peer ones are released regularly. This allows a automatic updates of protocols stacks.

Mobile Agents Platforms

Context

A mobile agent paradigm is a computing model used to optimize network resources and to enable fault tolerance in mobile and ubiquitous[9][10]. In the client-server paradigm, the client program sends requests to a server and then processes gathered data to present the results to the user. This incurs a lot network traffic and requires the client to always be connected to the network. A mobile agent is a software program that migrates to the resource's site to process data where the resource resides. Mobile agents are generally designed as autonomous programs that roam from site to site to gather or distribute information, negotiate trade, and finally return back to the initial site to present the result.

A mobile agent platform initiates the creation of new agents for the end-user (terminal site) and accepts immigrant agents for execution (acceptor site). The platform guarantees security by checking the code and/or the signature of the the immigrant agent. The site can make confidential the gathered information by encryption and signing before an emigrant agent roams to another site.

Applications domains of mobile agents are vast and include distributed search for information (named Knowbots or KNOWlegde roBOTs), WorkFlow (active documents), services for telecommunications (in TINA), intelligent networks, intelligent messages, management of domain names DNX, network administration (SMNP), management of the mobility of nomadic users, and e-commerce. A number of mobile agent systems have been designed and implemented in academic institutions and commercial firms. The most well-known systems are the Aglets from IBM-Tokyo, Concordia, and Voyager. Most of agent systems are written in Java, which offers properties such as the code mobility, portability, security, and widespread developer acceptance.

Architecture Proposal

Since the agent code is installed and removed many times at every visited site, the agent execution framework should never stop and must free memory resources each time an agent leave the site. Therefore, OSGi/Beanome is a good candidate to provide a Java-based mobile agent platform. OSGi is natively designed to load and unload external code.

The architecture uses one class loader per agent to load and remove the agent code. Security managers create a sandbox around the agent to protect the site and the other

agents against malicious agents. The bundle packaging securely stores code, resources, and state of the agent during the migration.

The platform may support the execution of agents with different life cycles such as IBM aglets, JMX Mbean, etc. However each kind of agents is managed by a specialized MJAgentManager. This manager invokes the callback methods on the services published by each agent bundles. The migration managers, MJAgentEmigrationManager and MJAgentImmigrationManager, deal with the agent roaming.

Agent State may be saved before emigration or restore after immigration (Java Serialization, Java Externalization or XML Long Persistence Bean introduced in J2SE1.4). The state is stored in an entry of the Bundle JAR file. The new state can be signed by the platform before emigration and confidential data (for instance, trade negotiation data) may be encrypted with the owner public key. Agent and platform securities are based on built-in JAR signature and permission checking.

Migration managers uses HTTP POST requests to send and receive agent bundles to/from another platforms. They use the standard HTTP and Servlet bundles from the OSGi specification. Other protocols could be added such as UDP, UDP Multicast, JavaGroups, JXTA Pipes, and Gnutella Responses to allows firewall passing or network efficiency. As in the previous section with the collaborative environment, the communication stacks can be install from a bundle enabling easy updating and extensible protocols installations.

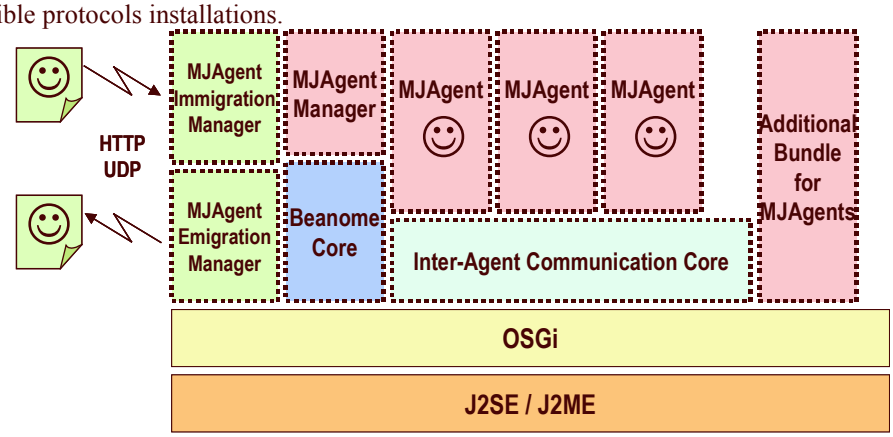

Fig. 4. Mobile Java Agent Platform

The Inter-agent Communication Bundle provides communication layers between agents inside the platforms. Synchronous communications (i.e., method calls) is directly based on the IXC (Inter Xlet Communication) of the J2ME/CDC/Personal Basic Profile. Asynchronous communications provides message producing and consuming. A local version of the JMS message server provides two messaging models. In the point-to-point model, a message is produced by one producer and consumed by one consumer. In the publish-subscribe model, the published message is broadcast to zero or more subscribers. Receivers and subscribers can filter the received messages. Higher semantic communication layers such as KQML use these layers.

User interaction is useful at agent initialization when the agent is configured its journey (the control panel for a file searching). The agent must check for GUI re-

sources of the current platform (classes from javax.swing.*, java.awt.*, javax.microedition.lcdui.*, …) and its permissions on them. Then it can display its interaction panel.

Conclusion

This paper discussed how to use OSGi and Beanome to build dynamic application frameworks. Dynamic application frameworks must support frequent and regular deployment of new or updated software components, coarse- and/or fine-grained extensibility of functionality (via dynamically loadable code), potentially non-stop execution, the need to limit resource consumption, and loosely administered control. The three application domains examined in this paper, iTV, peer-to-peer systems, and mobile agent platforms, require application components to be to deployed, updated, and removed frequently since the machines involved in these applications may have limited resources, run non-stop, or support extensible functionlity.

These three domains benefit from OSGi since it provides sophisticated class loading and security mechanisms for dynamically downloaded code. They also benefit from Beanome, because it provides a component extension to OSGi in which applications are designed as a composition of actives services. Beanome manages the code localization and versioning and resolves the dependencies between services. We are currently experimenting with OSGi to provide iTV on a version of OSCAR running on Personal Java and plan to test it with mobile agents and instantaneous collaboration.

References

1. European Broadcasting Union, Multimedia Home Platform 1.0.2, DVB BlueBook A057 Rev, February 2002.
2. OSGi, Open Service Gateway Specification, version 1.0, May 2000, http://www.osgi.org
3. Hall R., OSCAR, Open Service Container Architecture, http://oscar-osgi.sourceforge.net/
4. Cervantes H., Beanome a component model for extensible environments, http://www-adele.imag.fr/BEANOME
5. O'Driscoll G, The essential guide to Digital Set-Top Boxes and Interactive TV, Pub. Prentice Hall,2000
6. O'Driscoll G., The essential guide to Home Networking technologies, Pub. Prentice Hall, 2001.
7. Ripeanu M., Iamnitchi A., Foster I., Mapping the Gnutella Network, IEEE Internet Computing, January-February 2002, pp50-57
8. JXTA Project web site, http://www.jxta.org
9. Fuggetta A., Picco P.G., Vigna G., Understanding Code Mobility, IEEE Transactions on Software Engeneering, Vol. 24, No 5, Mai 1998, pp342-361
10. Pham V.A., Karmouch A., Mobile Software Agents : An overview, IEEE Communications Magazine, July 1998, Vol. 36 No. 7, pp26-37

11. Aridor Y., Oshima M., Infrastructure for Mobile Agents: Requirements and Design, 2nd International Workshop on Mobile Agents (MA '98), Springer Verlag, September 1998. http://www.trl.ibm.com/aglets/index_e.htm
12. Yang S.C., Inglin D., Ge A., Home Appliance Control Using J2ME Technology on a Wireless PDA, JavaOne2001 Conference, TS 2557,
13. JSR 129, Personal Basic Profile, http://www.jcp.org/jsr/detail/129.jsp
14. JavaTV web site, http://java.sun.com/products/javatv/
15. Box D., Essential COM. Addison Wesley, January 1998
16. Object Management Group. OMG, Corba Component Joint revised submission, August 1999
17. Java Embedded Server web site, http://wwws.sun.com/software/embeddedserver/