JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 25, 1337-1356 (2009)

Classification and Evaluation of Middleware Collaboration Architectures for Converging MHP and OSGi in a Smart Home*

CHENG-LIANG LIN, PANG-CHIEH WANG AND TING-WEI HOU
Department of Engineering Science
National Cheng Kung University
Tainan, 701 Taiwan

If Interactive Digital Television (IDTV) and Residential Service Gateway (RG) converge, *i.e*. to share services and resources, add-on values or new services could be created. We assume that a user will have an IDTV instance and an RG instance. These two can be implemented on a single machine or each on a separate machine connected by a network. Collaboration architectures of IDTV and RG support the converged IDTV and RG services. We first make a classification of the collaboration architectures to be: RG based on IDTV, IDTV based on RG, or networked IDTV-RG implementations. Secondly, we propose the use of Proxy design pattern for collaboration between IDTV and RG. Thirdly, we implemented the Proxy design pattern for all collaboration architectures. Finally, we evaluated (1) their required efforts in lines of code modified/enhanced, (2) quantitative performance metrics, such as memory usage, system startup time, object registry time and method invocation time, and (3) qualitative metrics, such as bilateral call, dynamic upgrade, cohesion, and coupling. The experimental target IDTV middleware was DVB/MHP Java Profile, and the Residential Gateway middleware was Open Service Gateway initiative (OSGi).

Keywords: DVB/MHP, OSGi, convergence, interactive, middleware

1. INTRODUCTION

Network services have moved beyond offices and public places into homes, providing mobilized, diversified, residential and personalized digital services. Among digital home devices, a Residential (Service) Gateway (RG) serves as an intermediate device between internal and external communications for applications/services such as home security, health care, remote control of appliances, *etc*.

Compared with traditional analog TV, interactive digital TV (IDTV) both improves the quality of video/audio and introduces new interaction models between TV users and content providers. European DVB, American ATSC and Japanese ISDB are IDTV standards. These standards require middleware for enhancing communication protocols, quality of video/audio, enabling interaction between users and services [1] and for customization. It is inevitable that IDTV and RG will jointly enter a digital home [2], which will certainly make daily life more convenient. Currently, a TV set and a RG set have their own dedicated functions/applications, which restricts their scope of applications.

Received October 31, 2007; accepted June 3, 2008.
Communicated by Jonathan Lee, Wei-Tek Tsai and Yau-Hwang Kuo.
* This paper was partially supported by the National Science Council of Taiwan, R.O.C., under grant No. NSC 95-2221-E-006-231-MY3.

Additionally the middleware of these two cannot mutually access each other's resources, and they are not cooperative, neither collaborative.

As IDTV and RG both enter a home, we believe that the services they control respectively, such as information services, entertainment services and control automation services, are to integrate in a converged home network (Fig. 1). The integration process requires the convergence and the collaboration of the IDTV middleware and RG middleware to make one's resources available to the other and further to create add-on new services on top of the IDTV-RG services and resources. We classify the collaboration architectures (configurations) according to the software architecture into three types: (1) IDTV functions based on RG middleware, (2) RG functions based on an IDTV middleware, and (3) IDTV middleware and RG middleware are isolated. In this paper, only the software architectures for the collaboration between an instance of IDTV middleware and an instance of RG middleware are discussed. The collaboration between multiple instances of IDTV middleware and RG middleware can be built on single-instance collaboration models, so they are not discussed.

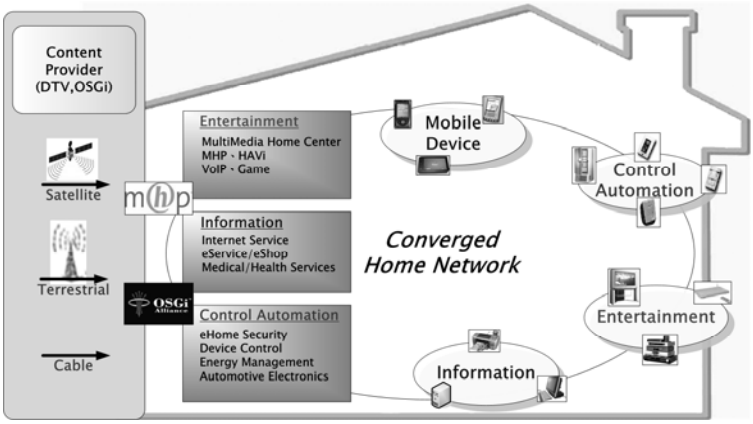

Fig. 1. MHP and OSGi to converge future home network.

We chose DVB/MHP (Multimedia Home Platform) and OSGi (Open Service Gateway initiatives) as experimental targets for IDTV middleware and RG middleware, respectively, because they both are Java-based and have open source implementations. In this paper, we are to evaluate their collaborative architectures from a practical view. We used a Proxy design pattern. The model was then implemented according to the three types of software architectures. We evaluated efforts for implementing the collaborative architectures in line-of-code modified/enhanced, and evaluated the performance of them in terms of memory usage, system startup time, object registry time and method invocation time. Note that because Java Virtual Machine (JVM) is required, type (3) is further extended to three kinds of implementations, which are addressed in section 2.

Scenarios for the services that require the convergence of MHP middleware and OSGi middleware can be found in [3-7, 24]. They are helpful in motivating this research. For instance, the visitor scenario is an UPnP camera controlled by OSGi middleware at the front door. The images captured by the camera is displayed on a MHP TV screen via

MHP-OSGi collaboration. When a visitor appears at the front door, the TV watcher sees the visitor's image popped on the screen [8].

The contributions of the paper are three folds: (1) we make a classification of the collaboration architectures of IDTV and RG; (2) we use the Proxy design pattern to design, implement, and evaluate all the collaboration architectures; (3) qualitative and quantitative characteristics are evaluated. The evaluation would serve as a sound base for manufacturers, vendors, system integrators and service providers who consider making IDTV and RG collaborative. This paper is organized as follows. Section 2 introduces MHP and OSGi platforms and the collaboration architectures of MHP and OSGi respectively. Section 3 is the system analysis and the designs of software architectures. Section 4 shows their performance measurements and the development efforts. Section 5 concludes with discussions on future work.

2. CLASSIFICATION AND RELATED WORK

An MHP application program is called an Xlet. MHP middleware includes Application Manager, Common Java Package, options, JVM, native APIs as mandatory components, and device drivers. JavaTV, JMF, Havi-UI, DAVIC and an XML parser are required for MHP middleware for controlling services, streamed media, user interface and XML functions. Application Manager gets the application list, locates and executes applications, controls each application's lifecycle and authorizes applications to access resources. The architecture of MHP middleware and the lifecycle of an MHP Xlet are shown in Figs. 2 (a) and (b). Each Xlet has its own classloader, and it cannot invoke instances of other Xlets. But MHP 1.1.x specification [9] allows Inter-Xlet Communication (IXC). IXC uses Java RMI to allow each Xlet to be accessible by other Xlet(s) loaded by another classloader.

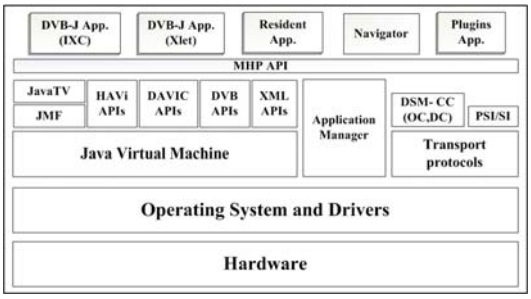

Fig. 2. (a) MHP architecture.

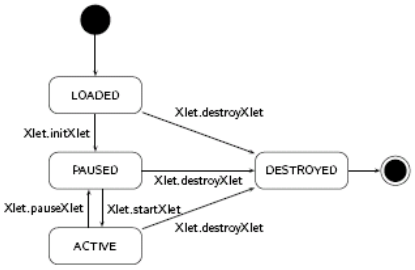

Fig. 2. (b) MHP life cycle control.

The architecture of the OSGi framework and the lifecycle of an OSGi bundle [10] are shown in Figs. 3 (a) and (b). A bundle implements a service or part of a service. A bundle is a Java JAR file. The OSGi BundleContext is the execution environment of a bundle in the OSGi framework, and it enables the bundle to access the registry. The BundleContext is given to a bundle by its activator at start-up. In the following, we shall use MHP and OSGi for the abbreviation of MHP middleware and OSGi middleware (frame-

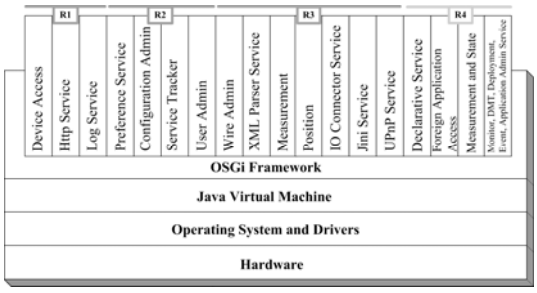

Fig. 3. (a) OSGi architecture.

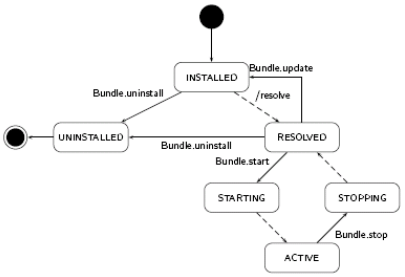

Fig. 3. (b) Life cycle control.

work), respectively. OSGi manages all the activities and creates bindings between bundles dynamically. The binding process includes service publication, discovery and the availability of services.

2.1 Classification

To make MHP and OSGi collaborative implies that MHP Xlets and OSGi bundles could cross-access one or more of each other's objects. Following the classification in section 1, with MHP and OSGi as the target middleware for IDTV and RG, we classify the collaboration architectures into the three types: MbO, ObM and MnO, as summarized in Table 1.

Table 1. Classification of MHP-OSGi collaboration architectures.

Architectures	Hardware: integrated on a single platform	Hardware: networked and dedicated platforms
MbO	MHP based on OSGi and a JVM	None
ObM	OSGi based on MHP and a JVM	None
MnO	MnO_{1j1p}: MHP neighbors OSGi on a single JVM	MnO_{2j2p}: MHP neighbors OSGi but they are on two JVMs distributed on two networked physical machines
	MnO_{2j1p}: MHP neighbors OSGi but they are on two isolated JVMs	

The three types, with their subtypes are:

- **MbO** (**M**HP **b**ased on **O**SGi) architecture: OSGi serves as the base and an MHP Application Manager is wrapped as a bundle on OSGi. Objects are cross-passed by reference. Since the MHP Application Manager is wrapped as a bundle, the MHP Application Manager can be dynamically upgraded. The OSGi framework, by specification, can only be restarted, not dynamically upgraded.
- **ObM** (**O**SGi **b**ased on **M**HP) architecture: MHP serves as the base. OSGi is wrapped as an Xlet. Objects are also cross-passed by reference. OSGi can be delivered and upgraded as an Xlet.
- **MnO** (**M**HP **n**eighbors **O**SGi) architecture: MHP and OSGi are not wrapped as in MbO or ObM. They are isolated. MnO can be further classified into three subtypes

according to their implementations: MnO_{1j1p} (MHP and OSGi are built on **a** **J**VM on **a** single platform), MnO_{2j1p} (MHP and OSGi are built on **two** **J**VMs on **a** single platform) and MnO_{2j2p} (MHP and OSGi are built on **two** **J**VMs on **two** platforms connected by a home network).

2.2 Related Work

The MbO architecture is as shown in Fig. 4. Underlying OSGi must support part of the functions of the MHP, including Xlet lifecycle management, DSM-CC carousel control, AIT control, HAVi-UI, JavaTV, JMF API and static libraries. MbO was initially proposed by H. Cervantes, *et al*. in [11]. They proposed that a specific software component should be loaded to integrate other middleware on OSGi. For MbO, the specific software component should manage Xlets. However, they did not name the component, and they did not implement the component for the MbO architecture. We implemented this architecture [24].

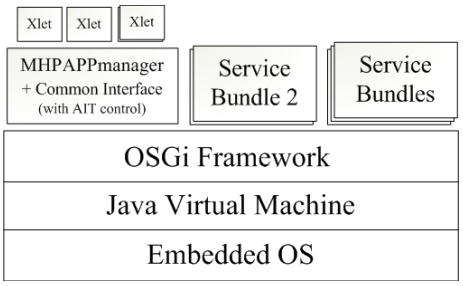

Fig. 4. MbO collaboration architecture.

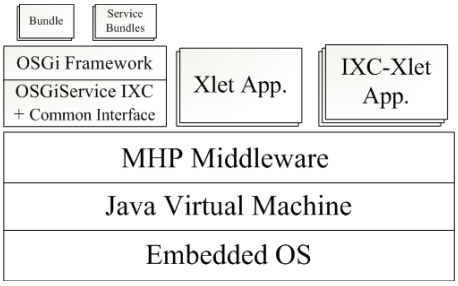

Fig. 5. ObM collaboration architecture.

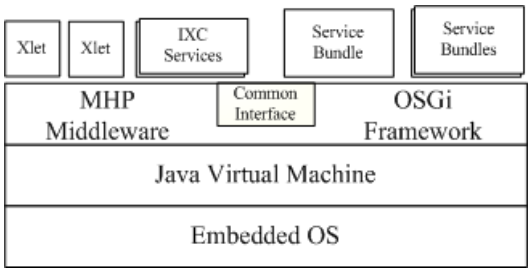

Fig. 6. MnO_{1j1p}: MHP-OSGi neighbors with a single JVM.

The ObM architecture is shown in Fig. 5, where OSGi is wrapped as an Xlet, named as OSGiService IXC. The OSGService IXC contains an interface to collaborate MHP and OSGi. To do this, mechanisms like methods overriding must be devised, and the interface has to launch OSGi first.

The MnO_{1j1p} architecture is shown in Fig. 6. A common interface is required. There are two alternative implementations [3, 6]. In [3], the XbundLET, which is both an Xlet and a bundle, is to "glue" MHP and OSGi. The drawback of this approach is that an application developer needs to understand the XbundLET. It is not easy to update, because

it is not easy to distinguish if an XbundLET is an Xlet of the MHP or a bundle of the OSGi. In [6], a bridge enables MHP and OSGi to collaborate by embedding a static java library into a Java virtual machine. The idea is new and it requires a JVM embedded with the bridge. The library style overcomes the drawback of the XbundLET.

In the MnO_{2j1p} model, MHP and OSGi are on two distinct JVMs but running on the same physical platform, such as a set-top-box. MnO_{2j1p} is logically the same as the MnO_{2j2p}, but its physical configuration is implemented on a single platform. The MnO_{2j2p} architecture indicates that there are an MHP set-top-box (or TV set) and an OSGi service gateway connected by a home network. A simple way to enable them to communicate by pre-defined commands is to establish a socket to connection. Tkachenko and Kornet [7, 12] propose a method that through broadcasting, an Xlet, wrapped as a bundle, can be automatically mounted to OSGi; the Xlet awaits the request for connection from MHP. But they only support a limited number of commands. In our proposed approach, Remote Method Invocation (RMI) enables the transmission and communication of objects to be flexible, dynamic and portable. As shown in Fig. 7, Xlets of MHP and bundles of OSGi communicate, and share resources and services of both ends by the support of MHP2OSGI IXC service and OSGi2MHP bundle, respectively.

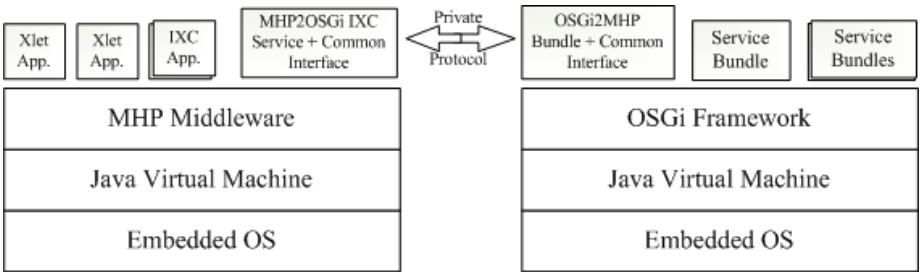

Fig. 7. MnO_{2j2p} combines MHP and OSGi by a home network.

3. SYSTEM ARCHITECTURE

The OSGi and MHP collaboration architectures require a mechanism to make the OSGi framework and the MHP middleware know each other's services and share them. We chose the Proxy design pattern [13, 14] to model and implement the mechanism. Consequently, there is a common interface to enable each Xlet/bundle to access bundle/ Xlet services. The lifecycle management of cross-access Xlets and bundles must be taken care of. In this section, the proposed solutions to fulfill the three requirements are described.

3.1 MbO

In MbO, the core of MHP is wrapped as an OSGi bundle, called MHPAPPmanager, shown in Fig. 4. MHPAPPmanager loads and invokes Xlets. Active Xlets can invoke OSGi service bundles, and OSGi service bundles can also invoke these Xlets through MHPAPPmanager. Finally, MHPAPPmanager stops or pauses an Xlet, upon requests (signaling) from Application Information Table (AIT) or OSGi. Since the MHPAPP-

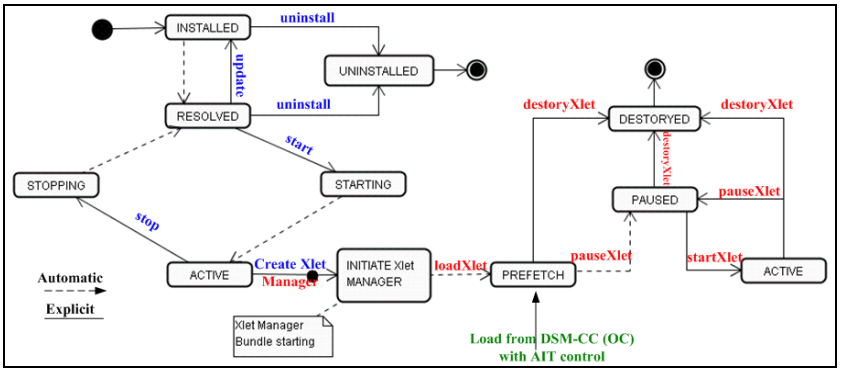

Fig. 8. MHPAPPmanager bundle coordination with OSGi lifecycle.

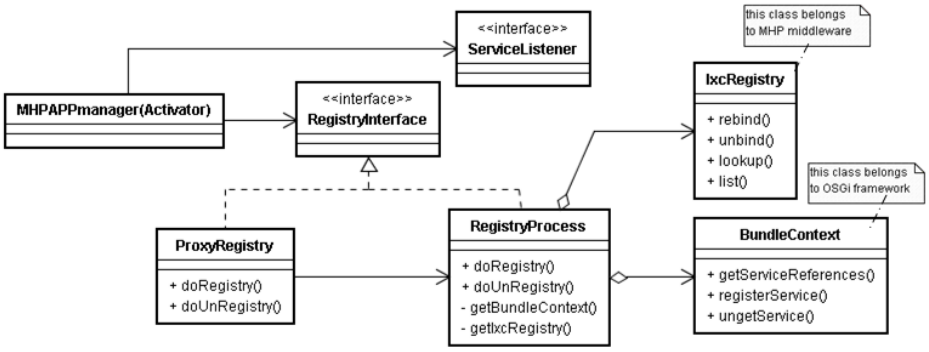

Fig. 9. The MbO class diagram.

manager is a bundle, it must first be subject to OSGi lifecycle management. As shown at the left side of Fig. 8, its OSGi state starts from “installed” to “resolved”, “starting” and “active”. As it becomes active, as at the right side of Fig. 8, MHPAPPmanager starts to manage the lifecycles of Xlets.

After entering the PREFETCH state, MHPAPPmanager handles the Xlets. The Application Information Table (AIT) can be retrieved by JNI (Java Native Interface). By gathering AIT signaling, MHPAPPmanager performs operations such as loadXlet, pasuseXlet or destoryXlet to control the lifecycle of the Xlets. The proposed MHPAPPmanger’s lifecycle is shown in Fig. 8. An abstract wrapper class is devised for a RegistryProcess object, which is responsible for registering or unregistering OSGi service objects into the MHP IxcRegistry. Fig. 9 is a class diagram which illustrates the relationship between the MbO related classes.

A proxy object with its interface as RegistryInterface is implemented by ProxyRegistry, whose real subject is RegistryProcess. The RegistryProcess implementation is responsible for registering or unregistering OSGi service objects into MHP IxcRegistry by using a “doRegistry” or “doUnRegistry” method when MHPAPPmanager bundle is active. Vice versa, RegistryProcess is also responsible for registering or unregistering IXC (Inter-Xlet Communication) service objects into OSGi. It makes OSGi be able to access Xlets. The “BundleContext.getServiceReferences” method is used to get OSGi service

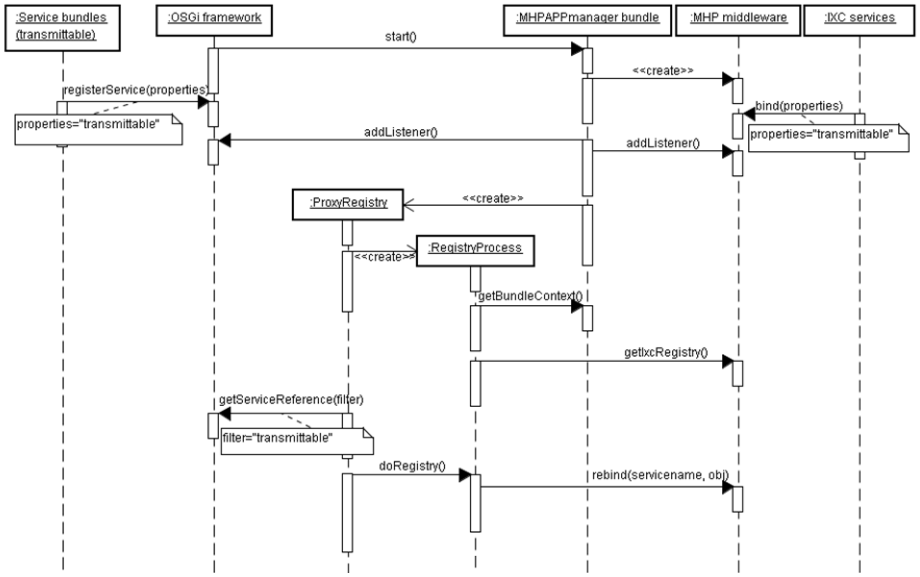

Fig. 10. The MbO sequence diagram.

references. These service references are of an array type, which are sent IxcRegistry to register into the MHP Application Manager. The ServiceListener is an OSGi listener interface. It is responsible for notifying which service's status is changed. A sequence diagram, as shown in Fig. 10, illustrates the sequence of starting the MHP Application Manger, the registry process and the access operations. Fig. 10 shows:

1. When OSGi is ready, MHPAPPmanager bundle can be installed and started. Then transmittable service bundles also can be installed and started by OSGi. (The transmittable property is described in section 3.3.)
2. MHPAPPmanager creates an instance of the MHP Application Manager. (In the implementation, the XletView, an MHP simulator [15], is launched.
3. MHPAPPmanager creates ProxyRegistry.
4. RegistryProcess gets the OSGi's BundleContext reference using the getBundleContext method. Then ProxyRegistry waits and listens to those service bundles whose properties are transmittable using the getServiceReference method. Similarly, a service bundle on OSGi is able to get references of these IXC service by RegistryProcess, too.
5. RegistryProcess registers the service references in the MHP Application Manager using the rebind method of the IxcRegistry class.

3.2 ObM

In ObM, OSGi, together with OSGi Service IXC and the common interface is wrapped as an Xlet, as in Fig. 5. IXC enables OSGi to communicate with other Xlets. The lifecycle control and the common interface resemble that of the MbO. When OSGiService IXC is active, the common interface can be loaded and activated by the OSGi-

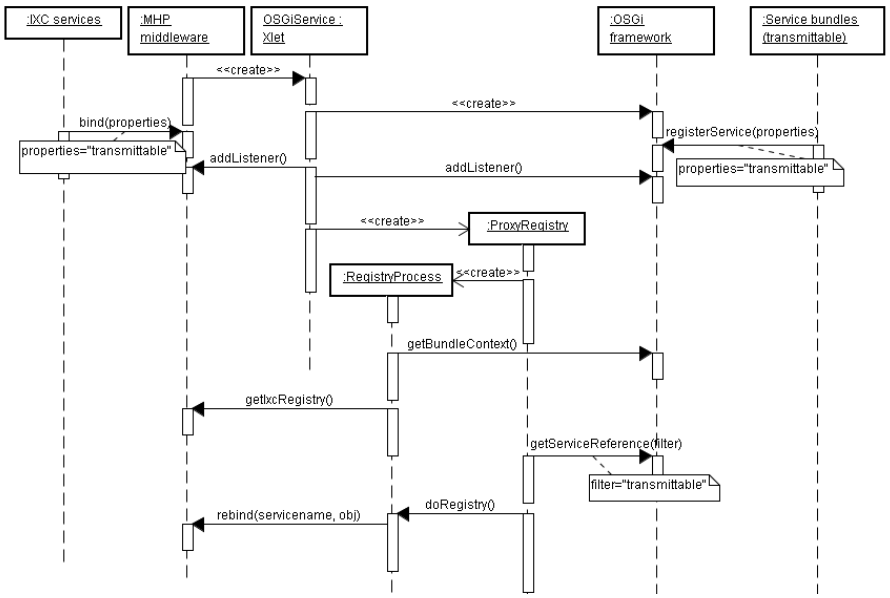

Fig. 11. The ObM sequence diagram.

Service IXC. Hence, MHP middleware can access OSGi service bundles through the common interface, and vice versa. The interface also refers to the Proxy design pattern. The ObM sequence diagram is in Fig. 11. The scenario is as follows:

1. OSGiService IXC is installed and started by the MHP Application Manager.
2. As OSGiService IXC is started, it creates an instance of OSGi. (In our implementation, Oscar, an open-source of OSGi, is launched.)
3. OSGiService IXC creates ProxyRegistry.
4. ProxyRegistry invokes getServiceReferences from OSGi, which returns an array type with active OSGi bundles' references.
5. OSGiService IXC registers the service references into MHP using rebind method.

3.3 MnO

MnO differs from MbO and ObM in that MnO does not have a 'based-on" relationship between MHP and OSGi. Because of implementation considerations, MnO is subdivided into three subclasses: MnO_{1j1p}, MnO_{2j1p}, and MnO_{2j2p}.

3.3.1 MnO_{1j1p}

Fig. 6 shows that MHP and OSGi are loaded and executed on a Java virtual machine of a physical machine. One issue is on accessing private methods. This implies that MHP internal classes can not create or access Oscar's internal classes directly. To solve this problem, we can either create a new public class outside of the Oscar or use reflection API [16] to modify the runtime behavior of internal classes or applications. We chose

to create a new public class, an AgentClass, responsible for overriding getBundles, getService and getServiceListener methods from OSGi's BundleImp class. These three functions must be invokeable out of the scope of OSGi to get the services object or services listener. Therefore MHP and OSGi can mutually communicate with each other. MHP uses IXC and prescribes a common interface to connect OSGi.

The transmittable property is necessary for two reasons. First, it unifies MbO, ObM, MnO_{1j1p}, MnO_{2j1p}, and MnO_{2j2p} architecture service bundles. Since MnO_{2j1p} and MnO_{2j2p} use RMI, such a service bundle is different from a normal service bundle. Second, among MbO, ObM, and MnO_{1j1p}, not all of the OSGi services should be available to MHP Xlets. Therefore, each transmittable bundle is labeled as props.put ("Bundle-Category", transmittable). In this way, ServiceListener can monitor these labeled bundles. The MHP cooperation interface (inside the MHP2OSGi IXC) listens to the events of OSGi cooperation interface (inside the OSGi2MHP bundle) and stores the bundle objects that conform to the transmittable condition. The difference between the transmittable bundles of MnO_{2j1p} and MnO_{2j2p} and is that in MnO_{2j1p} the bundle is only labeled, but in MnO_{2j2p}, a transmittable bundle is both labeled, and in an RMI style. Fig. 12 shows the sequence diagram for the scenario as follows:

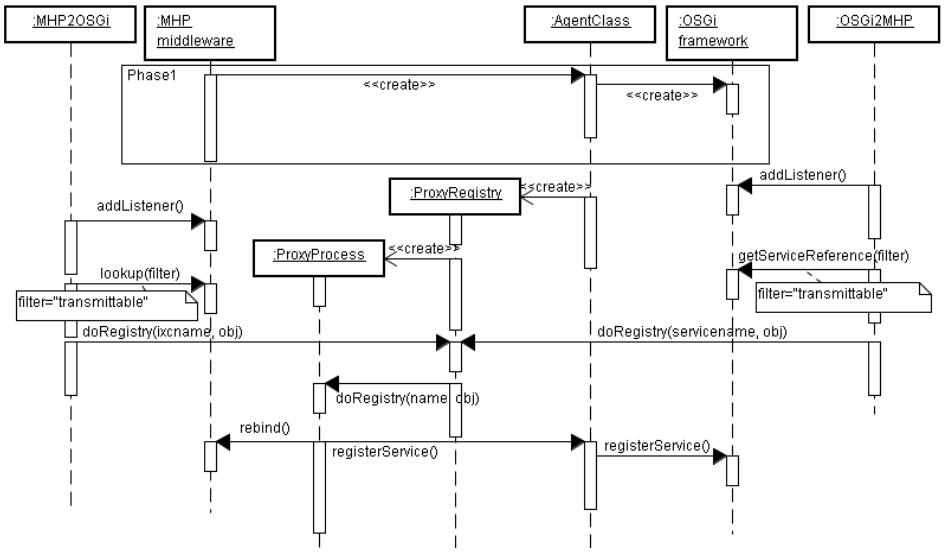

Fig. 12. The MnO_{1j1p} sequence diagram.

1. In the initial phase (phase1), MHP creates AgentClass to start OSGi. The MHP2OSGi IXC and OSGi2MHP bundle are installed and started by MHP and OSGi.
2. AgentClass creates a class, whose name is ProxyRegistry.
3. MHP2OSGi subscribes a service listener from MHP. MHP2OSGi waits and listens to those IXC-services whose properties are transmittable using the lookup method. Similarly, OSGi2MHP is waiting and listening to get references of those service bundles, too.
4. If MHP2OSGi finds a transmittable service, it invokes ProxyRegistry by doRegistry

method. Then, ProxyProcess invokes registerService method and passes the object reference to AgentClass.

5. AgentClass registers the object reference into OSGi by registerService method.

3.3.2 MnO_{2j1p} and MnO_{2j2p} (network communication)

As in Fig. 7, an MHP TV set and an OSGi gateway set are distinct and network connected. A communication protocol is needed for MHP and OSGi to cooperate. Among many communication protocols, such as socket based, RMI [17], XMLRPC [18], SOAP [19], and JXTA [20], Java RMI is chosen because the implementations of OSGi and MHP are Java-based.

An OSGi service can be easily transmitted to MHP by RMI. A service bundle, ServiceTracker is added to OSGi to collect services for MHP. When a MHP client connects to OSGi2MHP bundle, OSGiServiceCollector collects the OSGi service bundles and transmits a stub object to the client by RMI. OSGiJoint of MHP side is an IXC-Service, and it includes the RMI mechanism. When an instance of OSGiJoint is added to the OSGiServiceCollector, a remote object (remote object extends java.rmi.Remote) can be received and stored at the IXC repository through IxcRegistry. The stubHandle collects the classes exported by the remote service bundle, and the codebases of the classes are registered to an OSGi Http Service form http://ipaddress:port/codebase/.

OSGi2MHP bundle and MHP2OSGi IXC are the major components for communication. The sequence diagrams for them are in Figs. 13 and 14. In Fig. 13, OSGiServiceCollector object has three main functions. The first is to wait for the OSGiJoint object and to act as an MHP contact window. The second is to activate an RMIServiceTracker object, when an object is activated. (An RMIServiceTracker object collects all remote service objects in the OSGiRegistry). The third is to activate a common codebase controller, named stubHandle, to facilitate the remote services object's knowledge of which code base its own stub class comes from.

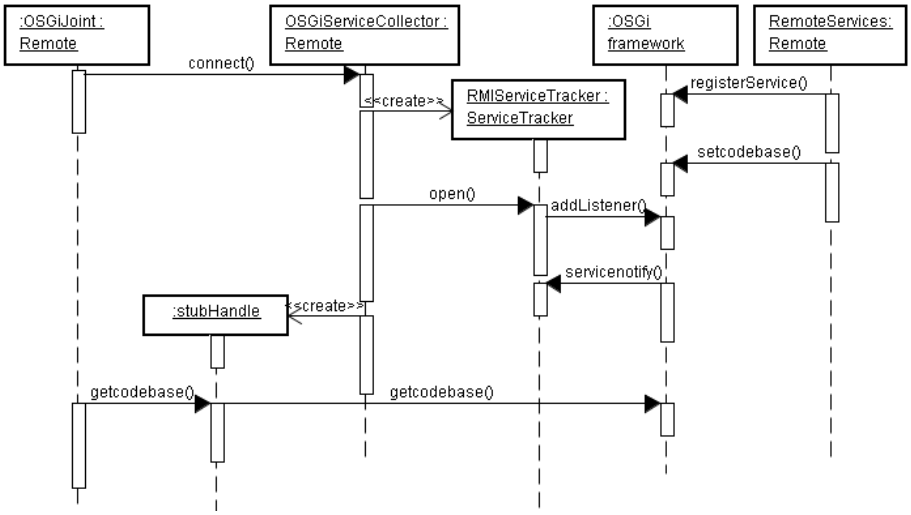

Fig. 13. The sequence diagram on the OSGi2MHP bundle side.

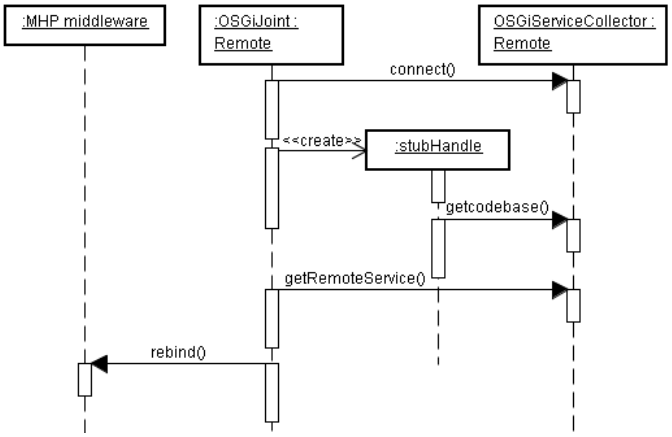

Fig. 14. The sequence diagram on the MHP2OSGi IXC side.

In Fig. 14, OSGiJoint object has three main functions. The first is to connect OSGiServiceCollector object and to get codebase URL. The second is to accept the collected remote object sent from OSGiServiceCollector object via OSGiJoint. When the OSGiJoint receives the remote object (stub class), it registers to MHP IxcRegistry using the IXC mechanism.

4. IMPLEMENTATION AND EVALUATION

We implemented the architectures to converge MHP and OSGi. The computers were equipped with an AMD Athlon 32 processors 2500+ running at 1.8 GHz, with 512MB RAM with only required services. They were free of other traffic and were restarted before each test to ensure the same initial conditions. Every test was repeated ten times with the JVM garbage collector disabled, according to the strategy in [21]. Debian 2.4.27 and Java 2 Platform Standard Edition version 1.4.1.07 were used. MHP was Xletview 3.6, with our IXC implementation. OSGi implementation was Oscar 1.5.

4.1 Measures

We evaluate the cost of sharing services between MHP and OSGi in terms of (1) Lines of code modified, (2) performance overhead. Lines-of-code is a software engineering metric to estimating the size of a software product. The performance overhead includes: memory usage and the time for system startup, transmittable bundle registry, Xlet-invoke-bundle, and upgrade. We first summarize the modifications to the source codes of Xletview and Oscar in Table 2. Because MnO_{2j1p} is like MnO_{2j2p}, MnO_{2j1p} is omitted. MnO_{2j2p} modified more lines, in average 3.2% than those of the MbO, ObM and MnO_{1j1p}, because of network communications.

We summarize their performances in Table 3. The average memory usage of the networked machines is 1.54% more than the single machine. MnO_{2j2p} register object time for a service bundle increases 48.15%, and MnO_{2j1p} just increases 1.65% more than the single machine. The reason is that the latter does not need to convert before access; they

Table 2. Lines-of-code for the proposed approaches.

Summarily	MbO	ObM	MnO_{1j1p}	MnO_{2j2p}
Common interface	3 classes, 112 lines modified	3 classes, 115 lines modified	5 classes, 156 lines modified	
IXC		4 classes, 332 lines modified	2 classes, 158 lines modified	6 classes, 339 lines modified
Bundle	5 classes, 312 lines modified		2 classes, 147 lines modified	7 classes, 1050 lines modified
Total	8 classes, 424 lines	7 classes, 447 lines	9 classes, 461 lines	13 classes, 1389 lines

Table 3. Performance measures.

Summarily	MbO	ObM	MnO_{1j1p}	MnO_{2j2p}
Memory Usage (Details in Appendix 1)	35,968KB	36,470KB	34,162KB	21,632KB (OSGi) 34,630KB (MHP)
Time of System Startup (Details in Appendix 2)	2782ms	2801ms	2328ms	3624ms
Time of a transmittable bundle registry into IxcRegistry (Details in Appendix 2)	20ms	20ms	20ms	963ms
Time of an xlet invoking a bundle (Details in Appendix 2)	1.5ms	1.5ms	1.5ms	3ms
Upgrade time of wrap and/or common interface (Details in Appendix 3)	350ms	152ms		356ms (OSGi) 149ms (MHP)
Upgrade time of middleware (Details in Appendix 3)	2787ms	2806ms	2332ms	1347ms

can directly invoke the objects' references. For the upgrade time of wrap and common interface, MbO is slower than ObM and MnO_{2j1p}/MnO_{2j2p} by 2.3%. However, the upgrade time of the entire middleware depends on the time of their system startup. The extensibility of the MnO_{2j1p}/MnO_{2j2p} not only can dynamically upgrade their common interface for new functionalities, but their operations are also independent, meaning they are not influenced by each other if one machine should crash. MbO and ObM also can dynamically upgrade their common interface, but they are wrapped in an Xlet or a bundle. The extensibility of MnO_{1j1p} is not good. When it needs to upgrade new functionalities, it needs to shutdown and then restart again. Therefore, we suggest that the MbO, ObM and MnO_{2j1p}/MnO_{2j2p} are more suitable for future applications.

4.2 Comparisons and Discussions

The design of MbO, ObM and MnO_{1j1p} are similar in terms of functions. Objects can be directly accessed through pass-by-reference. Inter-communication API can be directly invoked at both ends. In terms of interface coupling, MnO_{1j1p} adopts an interface. This means that MHP must initialize OSGi and the interface at the start up time. After-

wards, the common interface is still in the environment. The framework interface coupling is therefore regarded as tight, and it cannot be updated at runtime. However, MbO supports dynamic upgrading bundles. Since MbO is upgradeable, its interface coupling is considered as loose. ObM is like MbO, and its interface coupling is loose, too.

Regarding the networked case, there are more limitations. Though networked middleware cannot support pass-by-reference, it still can achieve the collaborative operations by RMI, SOAP, UPnP, *etc*. We implemented and demonstrated that the same approach can be applied to integrate OSGi to MHP or OSGi to OSGi, both by RMI. Networked architecture has a dynamically upgradeable feature. However, the biggest disadvantage to the use of networked architecture is that counterparty API cannot be directly invoked which may bring difficulties in development and integration. A summary of qualitative comparison of the convergence approaches are in Table 4.

Table 4. Qualitative comparison between MHP and OSGi convergence approaches.

	MbO	ObM	MnO_{1j1p}	MnO_{2j1p}/MnO_{2j2p}
Communication Method	Pass by Reference	Pass by Reference	Pass by Reference	Pass by Message
Bilateral API Call Directly	Yes	Yes	Yes	No
Run-time Upgrade Interface	Yes	Yes	No	Yes
Dynamic Upgrade Middleware or Wrap	MHP only	OSGi only	None	Both
Interface Cohesion	Sequential Cohesion (High)	Sequential Cohesion (High)	Logical Cohesion (Low)	Sequential Cohesion (High)
Interface Coupling	Loose	Loose	Tight	Loose
Extensibility	Medium	Medium	Below Medium	High

In Table 4, cohesion is addressed. Sequential cohesion is that when parts of a module are grouped because the output from one part is the input to another part like an assembly line. Logical cohesion is when parts of a module are grouped because they logically are categorized to do the same thing, even if they are different by nature [23].

The responsibility of the common interface of the MbO and ObM is to execute a register/unregister procedure. In addition to the procedure, there are no other related tasks. Hence, the common interface is marked as a sequential cohesion. Considering MnO_{2j1p}/MnO_{2j2p}, the common interface is separated from MHP and OSGi. These two components, OSGiServiceCollector and OSGiJoint, have a relation, though, they are separated. Hence, the interface is a sequential cohesion. As for MnO_{1j1p}, the register/unregister procedure, performed by the common interface, has to access AgentClass, otherwise it cannot be executed. The common interface is a logical cohesion.

The advantage for the MHP and OSGi on a single platform is direct access of mutual APIs. In terms of integration, a single platform is truly appropriate. The MnO_{2j2p} does not seem to be as good as MbO or ObM intuitively. However, the performance of MnO_{2j2p} is acceptable in the visitor-scenario in [3-7, 24]. In the visitor-scenario, the media content of the video-doorphone cannot be directly displayed on the Picture in Picture

(PIP) of the TV screen. The reason is that the networked collaboration (MnO_{2j2p} or MnO_{2j1p}) cannot directly access mutual APIs (MHP, OSGi) to get that media content. But the media content of the video-doorphone could be transmitted by the real-time multimedia streaming protocol (RTP, RTSP, *etc*.) and displayed on PIP of the TV.

The evaluation results show that the performances of all the collaboration architectures are acceptable, if there is no real-time requirement. For a practical choice of these architectures, we wonder consider from an MHP vendor, an OSGi vendor, and a system integrator's perspective. An MHP vendor, who owns MHP middleware, could apply ObM architecture to transmit an OSGiService IXC wrapper through the broadcasting end. Hence a (high-level) TV or set-top-box could have the option that it installs OSGi to control home appliances. For an OSGi provider, who supplies a software environment or hardware equipment, one can choose the MbO architecture and download a MHPAPP-manager bundle wrapper as a service. Then, then there is a (high level) RG which is also a set-top-box. Certainly, it should have the hardware and/or software for handling digital TV signals.

A system integrator has all the other choices beside MbO and ObM, depending on one's policy. MnO_{1j1p} has the best performance except upgrade time. It could be adopted if one does not have an exiting MHP or OSGi product while performance is an important factor. MnO_{2j1p} and MnO_{2j2p} are quite alike. The point for choosing MnO_{2j1p} is that there is already a MnO_{2j2p} solution and the hardware platform is capable of executing MnO_{2j1p}. For a service provider, if MHP return channel (internet connection) is ready, and there is a home network that connects MHP and OSGi, ObM, MbO and MnO_{2j2p} are also feasible, depending on the underlying hardware support of the MHP and OSGi platforms. However, if the service provider does not belong to either the party of MHP or OSGi providers, we suggest choosing the MnO_{2j2p}, which has less dependency on the platforms.

5. CONCLUSIONS

This research investigates collaboration architectures of IDTV (using MHP as experimental target) and RG (using OSGi as experimental target). We classify the collaboration architectures into three classes: IDTV based on RG (MbO), RG based on IDTV (ObM), and networked IDTV-RG (MnO). We used the Proxy design pattern to implement a common interface with necessary enhancements to 'glue" the MHP middleware and the OSGi framework so that MHP Xlets can access OSGi service bundles and vice versa. Based on this collaboration, add-on-value services can lead to the sharing of information and/or services between the OSGi framework and the MHP middleware. We also evaluate (1) the development efforts; (2) qualitative characteristics, and (3) quantitative performance measures. The evaluation results of the collaboration architectures show that the overhead are acceptable to users, if there is no real-time demand. It is up to the vendors, manufacturers, system developers, service providers and researchers to decide to adopt which collaboration architecture(s) depending on their requirements, respectively. Based on the collaboration architectures, service integration between IDTV and RG could be supported. Currently there are no killer applications (scenarios) on service integrations and there are no de facto benchmarking programs. Scenarios on home-care applications and benchmarking could be the future research directions.

REFERENCES

1. Y. S. Bae, B. J. Oh, K. D. Moon, and S. W. Kim, "Architecture for interoperability of services between an ACAP receiver and home networked devices," *IEEE Transactions on Consumer Electronics*, Vol. 52, 2006, pp. 123-128.
2. T. Zahariadis and K. Pramataris, "Multimedia home networks: Standards and interfaces," *Computer Standards and Interfaces*, Vol. 24, 2002, pp. 425-435.
3. M. R. Cabrer, R. P. D. Redondo, A. F. Vilas, J. J. P. Arias, and J. G. Duque, "Controlling the smart home from TV," *IEEE Transactions on Consumer Electronics*, Vol. 52, 2006, pp. 421-429.
4. A. F. Vilas, R. P. D. Redondo, M. R. Cabrer, J. J. P. Arias, A. G. Solla, J. G. Duque, M. L. Nores, and Y. B. Fernández, "MHP-OSGi convergence: A new model for open residential gateways," *Software: Practice and Experience*, Vol. 36, 2006, pp. 1421-1442.
5. M. R. Cabrer, R. P. D. Redondo, A. F. Vilas, and J. J. P. Arias, "Controlling the smart home from TV," in *Proceedings of IEEE International Conference on Consumer Electronics*, 2006, pp. 255-256.
6. M. C. Yang, N. Sheng, B. Huang, and J. Tu, "Collaboration of set-top box and residential gateway platforms," *IEEE Transactions on Consumer Electronics*, Vol. 53, 2007, pp. 905-910.
7. A. Kaplan, D. Tkachenko, and N. Kornet, "Convergence of iDTV and home network platforms," in *Proceedings of IEEE Conference on Consumer Communications and Networking*, 2004, pp. 624-626.
8. W. Z. Huang, C. L. Lin, T. W. Hou, and C. Y. Chang, "Integration and combination of MHP and OSGi," *Symposium on Digital Life Technologies – Building a Safe, Secured and Sound Living Environment*, 2006, pp. 6-10.
9. Multimedia Home Platform (MHP) Specification 1.1.1, Digital Video Broadcasting (DVB), June 2003, http://www.mhp.org/.
10. OSGi Service Platform, Release 4, OSGi Alliance, http://www.osgi.org/.
11. H. Cervantes, D. Donsez, and R. S. Hall, "Dynamic application frameworks using OSGi and Beanome," in *Proceedings of the International Symposium and School on Advanced Distributed Systems*, 2002, http://www.humbertocervantes.net/papers/ISADS2002.pdf.
12. D. Tkachenko, N. Kornet, A. Dodson, L. Li, and R. Khandelwal, "A framework supporting interaction of iDTV applications and CE devices in home network," in *Proceedings of IEEE Conference on Consumer Communications and Networking*, 2005, pp. 605-607.
13. U. Zdun, C. Hentrich, and W. M. P. van der Aalst, "A survey of patterns for service-oriented architectures," *International Journal of Internet Protocol Technology*, Vol. 1, 2006, pp. 132-143.
14. E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, 1995.
15. XleTView 0.3.6, http://xletview.sourceforge.net/.
16. The Reflection API, http://java.sun.com/docs/books/tutorial/reflect/index.html.
17. Remote Method Invocation Specification, http://java.sun.com/.
18. XML-RPC Specification, http://www.xmlrpc.com/spec/.
19. SOAP Specification, http://www.w3.org/TR/soap/.

20. JXTA Project, http://jxta-rmi.jxta.org/.
21. S. Wilson and J. Kesselman, *Java Platform Performance: Strategies and Tactics*, Addison-Wesley, USA, 2000.
22. Oscar: An OSGi Framework Implementation, http://oscar.objectweb.org/.
23. W. Stevens, G. Myers, and L. Constantine, "Structured design," *IBM Systems Journal*, Vol. 13, 1974, pp. 115-139.
24. C. L. Lin, P. C. Wang, and T. W. Hou, "A wrapper and broker model for collaboration between a set-top box and home service gateway," *IEEE Transactions on Consumer Electronics*, Vol. 54, 2008, pp. 1123-1129.

APPENDIX 1. MEMORY CONSUMPTION

Memory usages are summarized in Table 5 for the middleware built on a single machine and on Table 6 for two networked sets. They show that the MnO_{1j1p} memory usage is the smallest. We also compared one bundle versus five bundles. The memory consumption is not linear because of code reuse.

Table 5. Memory usage on a single platform.

	A Single Set					
	MbO		ObM		MnO_{1j1p}	
	OSGi	MHP	OSGi	MHP	OSGi	MHP
Initialization	14,814KB		31,798KB		33,264KB	
Correlation services*	498KB	none	498KB	none	498KB	none
wrap and/or common interface	20,520KB		4,034 KB		273KB	
A transmittable bundle	136KB		140KB		127KB	
Total Usage	35,968KB		36,470KB		34,162KB	

* Correlation services includes CM service and log service

Table 6. Memory usage for two networked sets.

	Two Networked Sets			
	MnO_{2j1p}		MnO_{2j2p}	
	OSGi	MHP	OSGi	MHP
Initialization	14,874KB	31,798KB	14,858KB	31,826KB
Correlation Services**	3,465KB	none	3,479KB	none
common interface	3,164KB	2,519KB	3,182KB	2,774KB
A transmittable bundle	112KB	30KB	113KB	30KB
Total Usage	21,615KB	4,347KB	21,632KB	34,630KB

** Correlation services includes OSGi Util, http service, CM service and log service

APPENDIX 2. OVERHEAD

The running overheads are shown in Tables 7 and 8. In a single set, the total execution time of MnO_{1j1p} is less than others, and the start time of wrap and common interface

Table 7. System startup, object registry and method invocation time of a set.

	A Single set		
	MbO	ObM	MnO_{1j1p}
Initialization	680ms (OSGi)	1563ms (MHP)	1952ms
wrap + common interface	1559ms (MHP) 543ms (interface)	685ms (OSGi) 553ms (interface)	376ms (interface)
Total startup	2782ms	2801ms	2328ms
A transmittable bundle registry	20ms	20ms	20ms
An xlet invokes a bundle	1.5ms	1.5ms	1.5ms

Table 8. System startup, object registry and method invocation time of two sets.

	Two networked sets			
	MnO_{2j1p}		MnO_{2j2p}	
	OSGi	MHP	OSGi	MHP
Initialization	696ms	1212ms	696ms	1224ms
wrap and/or common interface	1341ms	360ms	1342ms	362ms
Total startup	3609ms		3624ms	
a transmittable bundle registry	33ms		963ms	
An xlet invoking a bundle	3ms		3ms	

of MnO_{1j1p} is shorter than others. This is because the MHP middleware and the OSGi framework are independent. They do not require the 'wrap' overhead. In addition, the interface is not an Xlet or a bundle, so the access (translation) time is reduced. But MbO, ObM and MnO_{1j1p} invocation time are the same. In networked sets, double time is required to perform access translation as compared with the single set.

APPENDIX 3. UPGRADE TIME

The measurements of upgrade time are shown in Table 9. The OSGi upgrade takes more procedures to handle bundles' upgrade control as compared with MHP. In MbO and ObM, the wrapper contains the core and interface. We extracted the application manager, net.beiker.xletview.*, from the xletview as the to-be-upgraded MHP core. The upgrade of the OSGi is constructed using oscar.jar and moduleloader.jar. We added a procedure to handle the upgrade, which was to kill the current version after the download new middleware component, had been installed. For MnO_{2j2p} only the interfaces (gw2mhp bundle and mhp2osgi ixc) were updated. So the upgrade time of MbO architecture is 2.3% slower than ObM and MnO_{2j1p}/MnO_{2j2p}, but the upgrade time of the entire middleware will depend on their system startup time. We set up an http server in our laboratory to provide the MHP core, OSGi core and the common interface for downloading. The effective network transmission rate measured was 4Mbps, and the network transmission rate of the receiver side was 2Mbps.

Table 9. Upgrade time.

	A Single set		
	MbO	ObM	MnO_{1j1p}
wrap and/or common interface	350ms (for MHP and the common interface)	152ms (for OSGi and the common interface)	
middleware	2787ms (OSGi core + MHP bundle and the common interface)	2806ms (MHP core + OSGi Xlet + common interface)	2332ms (OSGi core + MHP core + common interface)

	Two networked sets			
	MnO_{2j1p}		MnO_{2j2p}	
	OSGi	MHP	OSGi	MHP
wrap and/or common interface	353ms	147ms	356ms	149ms
middleware	1346ms (OSGi core + the common interface)	365ms (MHP core + the common interface)	1347ms (OSGi core + the common interface)	367ms (MHP core + the common interface)

Cheng-Liang Lin (林政良) received the B.S. and M.S. degree in Computer Science and Information Engineering from Shu-Te University, Taiwan, in 2003 and 2005. And now he is a Ph.D. candidate in Engineering Science, National Cheng Kung University. His major research is in software methodologies, middleware collaboration, services design for ubiquitous computing environments, interactive digital TV and Java related technology.

Pang-Chieh Wang (王邦傑) received the B.S. degree in Engineering Science from National Cheng Kung University and the M.S. degree in Engineering Science from National Cheng Kung University, Taiwan, in 2000 and 2002. And now he is a Ph.D. candidate in Engineering Science, National Cheng Kung University. His major research is in sensor network, ad hoc network, multi-media cryptography and Java related technology.

Ting-Wei Hou (侯廷偉) received B.S., M.S., and Ph.D. degrees all in Electrical Engineering, National Cheng Kung University, Taiwan, in 1983, 1985, and 1990 separately. He has been an Associate Professor in Department of Engineering Science, National Cheng Kung University since 1990. He was a visiting scholar of CSRD of University of Illinois at Urbana-Champaign, Illinois, U.S.A., during 1993-1994. His major research is in embedded systems and system integration. He is currently working on Java based embedded systems, such as Java Virtual Machines, Java obfuscators, MHP, OSGi, Java card applications, and RFID applications.