3rd International Conference on Sensing Technology, Nov. 30 – Dec. 3, 2008, Tainan, Taiwan

# OSGi-Based Smart Home Architecture for Heterogeneous Network

Red-Tom Lin, Chin-Shun Hsu,
Networks and Multimedia Institute,
Institute for Information Industry, Taiwan.
ling9028@iii.org.tw

Tee Yuen Chun and Sheng-Tzong Cheng,
Department of Computer Science,
National Cheng Kung University, Taiwan.
stcheng@mail.ncku.edu.tw

***Abstract*—With the development of home network and service applications, different protocols and transmission modes are proposed. More digital devices and home appliance compliance to the protocols in the development. The proposed protocols are usually unable to communicate with each other; we design and implement a Service-Oriented Smart-Home Architecture to integrate popular protocols such as UPnP, Jini, DPWS on OSGi framework and collaborating Tmote, Zigbee and Bluetooth to converge various service oriented applications. Furthermore, with the well-developed Tmote, Zigbee and Bluetooth technology, majority of devices developed with these technologies supported, we propose the three new base drivers to integrate different devices communication on our platform. Additionally, we propose a Service Resolving Bundle to complement the drawbacks of OSGi mechanisms. This architecture with service-oriented mechanisms accommodates applications implemented across different domains and allows system components to interact with one another.***

***Keywords: OSGi, DPWS, Bluetooth, Smart-Home, SOA, Tmote, Zigbee***

## I. INTRODUCTION

The home network is realizing the vision that Mark Weiser described years ago [1]: our environment is more and more filled with networked devices. More services are expected to be provided to the future home. Currently the home networking technology is still in a middle of development, there are various protocols available to accommodate the communication between devices and appliances such as Universal Plug and Play (UPnP), Java for Intelligent Network (Jini), Devices Profile for Web Services (DPWS), Digital Living Network Alliance (DLNA) and etc. This situation results non-conformity of general available home network applications and inextensibility for users to set up their digital home environment.[1]

Besides dealing with the communication protocol issue, home network is also unable to avoid the problem of dynamically varying home environment and distributed connection. Since there are mobile devices connected to the home network with distributing, it often happens that the devices were disconnected dynamically, which means that the services configuration in the smart home are usually dynamic. Moreover, smart home should not only execute the commands received from the residents, but also collecting context information actively from surrounding environment. With the context information collected, system will infer the current user scenario and makes suitable reaction to provide comfortable environment for resident. To deal with the whole operation mentioned, system needs to control and communicate with all the devices distributed in the home network. Therefore designers of innovative home applications are facing three main challenges in this pervasive environment: dynamicity, heterogeneity and distribution.

In the light of the requirements mentioned above, a smart home should comply with emerging computer technologies and appropriately fulfill human requirements. Regarding to the mass potential consumer market, we develop smart home with conforming to open standards [2]. Here we propose a OSGi-based [3] smart home architecture for heterogeneous network with the integration of UPnP, DPWS, Bluetooth, Zigbee and Tmote.

## II. RELATED WORK

### *A. OSGi*

The Open Services Gateway Initiative OSGi is an independent, non-profit corporation working to define and promote open specifications for the delivery of managed services to networked environments, such as homes and automobiles. These specifications define the OSGi Service Platform, which consists of two pieces: the OSGi framework and a set of standard service definitions. The OSGi framework, which sits on top of a Java virtual machine, is the execution environment for services. The OSGi framework was originally conceived to be used inside restricted environments, such as the set-top box in a smart home. OSGi can however be used in other domains, as for example, an infrastructure to support underlying release 3.0 of the eclipse IDE..

## III. ARCHITECTURE

### *A. System Overview*

With the capabilities of OSGi, home applications can be maintained under a generic environment which comprising several heterogeneous frameworks. We now discuss how OSGi accommodate this heterogeneous framework to allow the maintenance of the applications from different domain.

[1] This research was supported by the Applied Information Services Development & Integration project of Institute for Information Industry and sponsored by MOEA ,R.O.C.

With the modular architecture mentioned above, we are able to integrate the devices and services from different domain into our own OSGi framework. The modular design enables the remote devices and services to appear as local entities on our OSGi framework. When a client wants to request for these services, they first request the relevant service bundle on our OSGi framework. Next, the client sends commands via the bundle which forward them to the services.

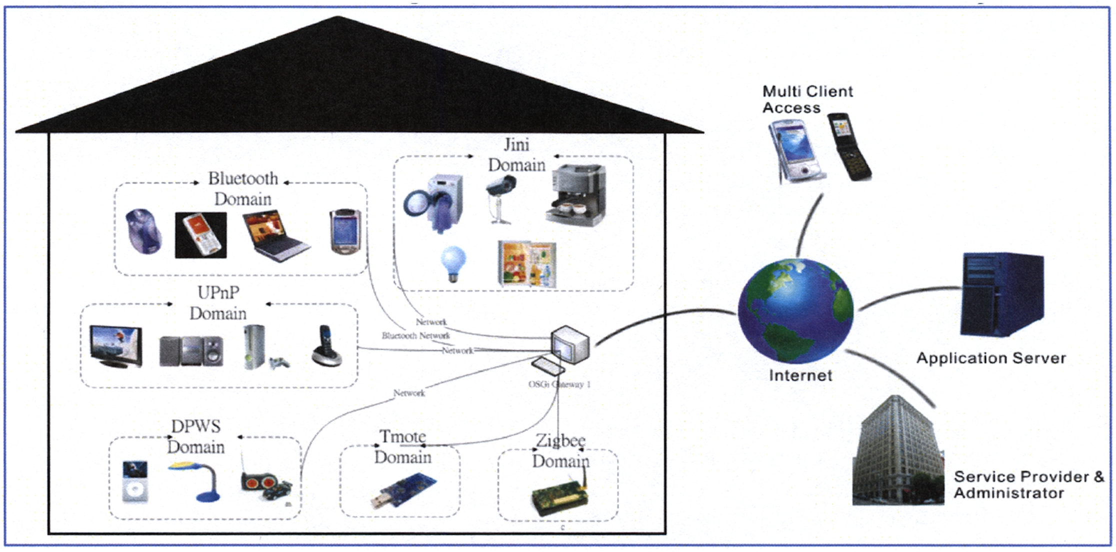

Figure 1. Our OSGi home network architecture

*B. System Architecture*

Figure 1 depicts our OSGi home network architecture. The OSGi gateway serves as the central coordination point for managing the home network, spanning multiple heterogeneous communication technologies. The OSGi specifications enable service providers and application developers to deploy and manage in-home network services and devices.

With the modular architecture mentioned above, we are able to integrate the devices and services from different domain into our own OSGi framework. The modular design enables the remote devices and services to appear as local entities on our OSGi framework. When a client wants to request for these services, they first request the relevant service bundle on our OSGi framework. Next, the client sends commands via the bundle which forward them to the services.

*C. Base Driver*

Generally, the architecture of base drivers is same but works corresponding to different domain. Here we discuss Tmote base driver as an example to show how it works.

*D. Tmote Base Driver*

We have developing a Tmote base driver to ease the usage of Tmote devices in OSGi environment. This Tmote base driver is composed by two parts: Importer and Exporter as same as other base drivers. Importer is responsible to listen to the advertisements sent on the Tmote networks by the devices and services. Then it will import all the discovered services and devices into OSGi environment, making them appear as valid OSGi entities and fully accessible by other OSGi entities (Figure 2). On the other hand, Exporter is registered as ServiceListener with the OSGi framework and it responsible to export the OSGi entities which compliant to the Tmote profile to the Tmote environment automatically, making them appear as virtual Tmote services and devices and full accessible by other Tmote entities.

Even it is not required by the OSGi specification, the imported and exported devices and services are registered with the registration property TMOTE_IMPORT and TMOTE_EXPORT correspond to their situation.

In order for Tmote devices to communicate properly, we developing this Tmote base driver with conform to the Tmote specification. The Tmote specification, like any other spec, defines the standard that a Tmote device should adhere to, as well as rules that need to be enforced when communicating. The Tmote protocol stack and profiles together comprise the Tmote specification.

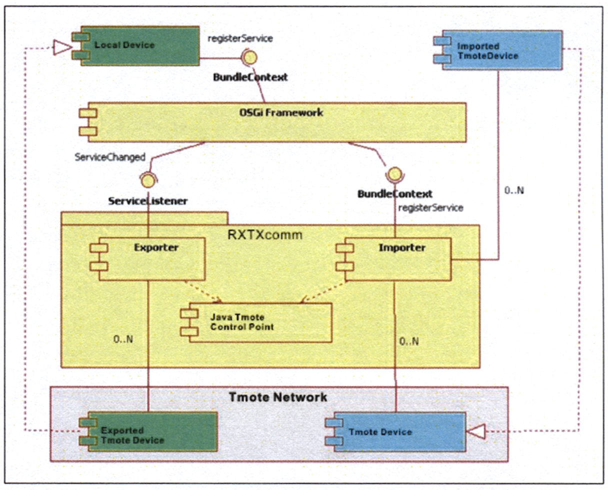

Figure 2. Tmote Base Driver Architecture

*1) Tmote to OSGi Transformation*

As an example of how a service gateway benefits from the Tmote to OSGi transformation, consider a Tmote network connected by Tmote base driver on OSGi framework, user is able to get access to control the Tmote lamp and Hifi audio player through the OGSi service provided on the OSGi framework.

Firstly, the Tmote base driver bundle discovers the Tmote Lamp and audio player in the Tmote network. Then it discovers the lighting service provided by the lamp and music service by the audio player, after that it registers the services of the both device onto the OSGi service registry.

This makes the Tmote services and devices appear to be local entities in the OSGi framework and accessible by other OSGi entities.

### 2) *OSGi to Tmote Transformation*

The OSGi to Tmote transformation in the Tmote base driver offers convenience to services running in the OSGi framework: these services need not to register Tmote service; instead, the Tmote base driver do so. This significantly reduces the complexity in definition and registration of Tmote services.

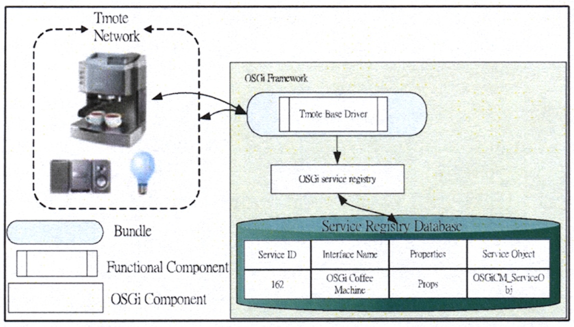

Figure 3. An Example of OSGi to Tmote Transformation

Our OSGi framework stores the registration of a service in XML form in our service registry. When a service is registered in the framework, the Tmote base driver inspects this property. If it is a Tmote compliant service, the driver registers it in its internal Tmote wireless sensor node list

Consider the following example, illustrated in Figure 3: assume that a service that controls a coffee machine available in the OSGi framework and it is Tmote compliance. The Tmote base driver detects the presence of the service and exports it to the Tmote network. If a Tmote client wants to use the coffee machine service, it contacts the Tmote Control Point in our Tmote base driver and retrieves the coffee machine service proxy. Next, the client sends command via the proxy to the original coffee machine service, which forwards them to the coffee machine.

## IV. A XML TRANSPORTING SERVICE IN OUR OSGI PLATFORM

### A. *Drawbacks of OSGi Mechanisms*

**Discovery**: OSGi's current discovery mechanism uses syntactic matchmaking. So the discovery phase can only retrieve service references that exactly match the service interface name in the query or whose properties include the same syntactic keywords when using a filter. This mechanism has several problems with synonyms (semantically similar but syntactically different services) and homonyms (syntactically equivalent but semantically different services). For example, a search for a Printing service might miss a service registered under the service name print.

**Selection**: OSGi discovery might result in a list of matched services—for instance, when several home printers register under the Printing service interface name. Adding filtering information at the discovery phase can aid the selection process. However, OSGi filtering is again based on syntactic comparisons.

**Invocation**: OSGi prevents unaware bundles (bundles without prior knowledge of the service interface) from dynamically invoking the service [7].

### B. *Service Resolving Bundle*

Service Resolving Bundle plays an essential role in the OSGi framework because it provides a more appropriate description of OSGi services additionally. However, you must relate any OSGi service with constructs of additional description stored in our proposed Service Resolving XML (SRXML). Also, the service user may at least know the way to invoke the service of this bundle to get additional information of the aimed services.

This Service Resolving Bundle is registered with interface name called "SInfo" at OSGi service registry and listening to the request on the network.

#### 1) *Registering OSGi services*

The Service Resolving Bundle is default set to inspect whether there is any bundle performing registration. If it does, the bundle will request the SRXML respect to the Service Identifier of the new service in the OSGi service registry. Each time our Service Resolving Bundle received a file, it will records the interface name of the corresponding service provider.

#### 2) *Discovering OSGi services*

Because obtaining OSGi services requires the requesting bundle to know too much information in advance, we propose using the knowledge stored in the SRXML files to facilitate OSGi service discovery. This fact is reflected in the querying issue, which we provide two method which available on our Service Resolving Bundle to support more flexible queries. Consequently, drawing an analogy with current OSGi methods, we propose providing the following methods as the service of Supplement bundle:

- supGetServiceReferences( String SupplementURI )
- supGetServiceReferences( String SupplementURI, String actionFilter )

The first parameter provides a flexible service discovery. The requesting bundle is able to use any synonyms, homonyms and profile correspondence keyword to specify the kind of service needed. The `actionFilter` parameter in the second method defines filtering information to aid the search process.

Similarly to the current OSGi specification, both provided method returns an array of service reference objects satisfying not only the search criterion but also the selection filter which is specified in `actionFilter` parameter.

### *3) Invoking the OSGi services*

The Service Resolving Bundle is used to manage the addition service description provided by the bundle developer storing in XML file. The Service Resolving XML (SRXML) is responsible to record the synonyms, homonyms and profile correspondence keyword of a serv ice. This SRXML is design based on UPnP Service Template [4] to describe the profile correspondence keyword of a service and adding additional tag to describe the service's synonyms and homonyms.

### *4) Synonyms and Homonyms*

Table 1 Synonyms and Homonyms for SRXML

| Variable Name | Req. or Opt. | Data Type | Allowed Value | Default Value | Eng. Units |
|---|---|---|---|---|---|
| Synomyms | O | String | [A-Z,0-9] | R | Null |
| Homonyms | O | String | [A-Z,0-9] | R | Null |

With adding the synonyms and homonyms tag in SRXML will allow bundle developers to describe themselves.

## V. Analysis and Discussion

### *A. Performance*

We assume that there are $n$ clients requesting service at the same time; each service involves $m$ Service Providers (SPs) of $y$ Control Points (CPs) and costs each provider $p$ computation time units. Each service has to interact with the CP with $j$ messages, and it costs $h$ network traffic units to transmit one message, each CP has to interact with the server with $z$ messages and it costs e network traffic units to transmit one message. Each client sends request to the server and costs $i$ network traffic units to transmit one message.

To perform a service invoke task in a conventional smart home architecture mentioned above, each client must send service request to the server and then the server interacts with these y CPs by sending $z$ messages to accomplish theses service requests. Each CP then sends $j$ messages to the SP to invoke the requested services. After receiving data from different SP passed by CPs, the server processes these data, integrates them and returns the final results to the clients. Analysis is shown in Table 2, in which CP stands for "Control Point" and SP stands for "Service Provider", (*n), (*y) and (*m) represent the possible numbers of the associated entities and 1~n in the CP and SP columns means each of them accepts 1~n service requests, thus changing the workload of its network traffic and computation.

As for the performance analysis of our architecture, clients query the service directory first to retrieve information about SP and obtain the service reference of the corresponding SP. Then the client interacts with m SPs remotely by sending j messages to each. When a client receives data that are returned from m SPs, it processes them and costing q computation time units for each and then integrates them. Performance analysis is shown in Table 3.

Table 4 shows the performance analysis of the P2P SOA with MA [8]. First, the clients query the service directory to retrieve information about SPs, and then create Mobile Agents (MAs) embedded with interaction logic based on service requirements. After that, every client sends its MA to SP, and since MA represents the client, the SP is able to interact with the MA locally and directly.

Table 2 Performance Analysis of Conventional Smart Home Architecture

| Conventional Architecture | Network Traffic | | | | Computation Load | |
|---|---|---|---|---|---|---|
| Scenario | Client (*n) | Server | CP (*y) 1~n | SP (*m) 1~n | Server | SP(*m) 1~n |
| Client Request Server | i | i*n | | | | |
| Server Request CP | | e*z*y*n | e*z~e*z*n | | | |
| CP request SP | | | h*j*m~h*j*m*n | h*j~h*j*n | | |
| SP Start Service | | | | | | p~p*n |
| SP Return Data | | | h*j*m~h*j*m*n | h*j~h*j*n | | |
| CP return data | | e*z*y*n | e*z~e*z*n | | | |
| Server processes data | | | | | q*y*n | |
| Server integrates data | | | | | q*n | |
| Server returns results | i | i*n | | | | |
| total | 2*i | 2*n *(i+e*z*y) | 2*n *(e*z+h*j*m) | 2*h*j*n | q*n*(1+y) | p~p*n |
| | 2*(2*n*(i+e*z*y)+h*j*m*n*(y+1)) | | | | n*(q*(y+1)+p*m) | |

Observing the computation load part in these three tables, the computation load is the same in our architecture and P2P SOA with MA model. It is a bit heavier for the computation load in the conventional architecture because the server needs to process and integrates data from different CPs. In the P2P SOA with MA model, the server load is distributed to SPs. Although this kind of distribution seems effective, it is not suitable for SPs whom lack computation power. Based on our architecture, the server computation load is distributed among $n$ clients and this kind of distribution seems fair.

As for the network part, the client handles more traffic in our architecture because it needs to get service reference from the supplemental bundle additionally. It is not easy to compare network traffic in our architecture and in P2P SOA with MA model because there are different types of network traffic units. Note that our architecture with the base driver helps to pass by the interaction between the client and CPs and hence the decrease of the server network traffic comparing with the conventional smart home architecture.

Comparing network traffic of the client of our architecture with other model, the total traffic of the client of our model is greater because of getting service reference. But it provides a facility in the architecture which the other does not has.

Table 3 Performance analysis of Our Architecture

| Our Architecture | Network Traffic | | | | Computation Load | |
|---|---|---|---|---|---|---|
| Senario | Client (*n) | Service Registry | Supplemen Bundle | SP (*m) 1~n | Client (*n) | SP(*m) 1~n |
| Client queries Service Registry | i | i*n | | | | |
| Client queries Service Reference | i | | i*n | | | |
| Client requests SP | i*j*m | | | i*j~i*j*n | | |
| SP Start Service | | | | | | p~p*n |
| SP Return Data | i*j*m | | | i*j~i*j*n | | |
| Client processes data | | | | | q*m | |
| Client integrates information | | | | | q | |
| total | 2*i*(1+j*m) | i*n | i*n | 2*(i*j~i*j*n) | q*(1+m) | p~p*n |
| | 2*i*n*(3+2*j*m) | | | | n*(q+m*(p+q)) | |

Table 4 Performance Analysis P2P SOA with MA

| P2P SOA With MA | Network Trafic | | | Computation Load | |
|---|---|---|---|---|---|
| Senario | Client (*n) | Service Directory | SP (*m) 1~n | Client (*n) | SP(*m) 1~n |
| Client queries Service Directory | i | i*n | | | |
| Client sends MA to SP | k | | k~k*n | | |
| SP Start Service | | | | | p~p*n |
| SP Process Data | | | | | q~q*n |
| SP returns MA with data | i+k | | (i+k)~(i+k)*n | | |
| Client integrates results | | | | q | |
| total | 2*(i+k) | i*n | (i+2k)~(i+2k)*n | q | (p+q)~(p+q)*n |
| | 3*i*n+i*m*n+2*k*m*n+2*k*n | | | n*(q+m*(p+q)) | |

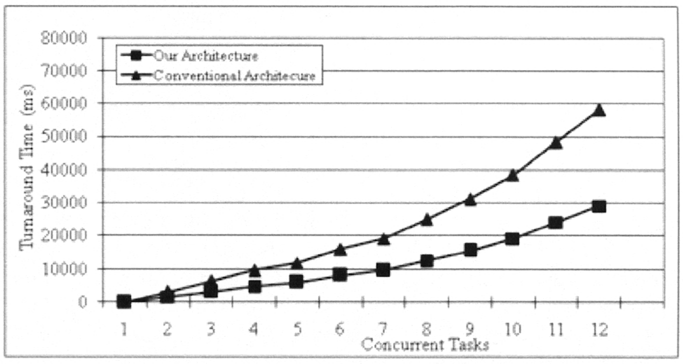

Figure 4. Performance evaluations of different models

## VI. System Evaluation

### A. Application Scenario and Prototype Implementation

To implement our proposal, we selected the open source OSGi Service Platform, Knopflerfish version 2.0.5 which led and maintained by Makewave [5]. It is an open software implementation of the OSGi framework. We have set up the framework and implement it at the National Cheng Kung University Aspire Home for Quality Life. The Jini base driver is supported since OSGi specification version R3 and UPnP base driver is adopted. Currently, we are using Jini version 1.2.1, Odonata DPWS base driver 1.0.6 [6], Domoware UPnP Base Driver version 3.0.2 and the prototype has implemented each Control Points and Light devices correspond to DPWS, Jini and UPnP. Also, we have developed the Control Point managed in the base drivers of Bluetooth, Tmote and Zigbee. We have connected the Tmote audio player, Tmote coffee machine and Zigbee light devices in the Aspire Home and all the connected devices are controlled by Bluetooth enabled hand phone through a media center as graphical user interface.

### B. Performance Evaluations

To study the performance of different interaction models (conventional smart home architecture and our architecture), we have conducted experiments with our prototype implementations. We use a task controller to initialize the execution of a new task and to measure the turnaround time of execution in each experiment. A task is defined as a sequence of executable procedures on devices distributed over the network. In our experiment, the OSGi platforms have to collect data from devices as and integrate the data as well as to perform inferring to produce the result upon receiving new task.

Our experiments are described below. In the first scenario (conventional smart home architecture), there is a single OSGi platform connected with five devices through CPs from different domain via Ethernet locally as we shown in Figure 1: Tmote device, Zigbee device, Bluetooth device, DPWS device and UPnP device. Each task is defined as sequentially gathering data from five devices and then performs integrating and reasoning procedures in OSGi platform. The time to integrate and reasoning and the data retrieval latency are both set as 100ms.

Figure 4 shows the experiment results. Compared to conventional smart home architecture, the turnaround time is decreased drastically with our architecture. The performance gained from the direct interaction between client and the proxy of service provider and the decreased number of interaction between client and server. Because the services are invoked locally in our architecture, most of the costs come from the data transferring latency. Consequently, the experiment result shows us our architecture helps to reduce the task execution by local execution of services.

## VII. Conclusions

The device and service discovery mechanisms of OSGi enable the creation of advanced home networking services. As illustrated in the preceding usage scenario, OSGi can serve as an effective bridge across disparate networking technologies, making new device interactions possible and leading to services within the home that can fully leverage the growing diversity of devices and better serve the needs of end users.

## VIII. Reference

[1] M. Weiser, “The computer for the 21st century”, Scientific American, 265(3):66-75, September 1991.

[2] K. Wacks, “The successes and failures of standardization in home systems,” in Proc. 2nd IEEE Conf. Standardization Innovation Inf. Technol., Boulder, CO, pp.77-88, Oct. 2001.

[3] OSGi alliance [Online]. Available: http://www.osgi.org

[4] UPnP Service Template Version 1.01[Online]. Available: http://www.upnp.org/resources/documents/Service-Template-1.01_000.doc

[5] Knopflerfish OSGi [Online]. Available: http://www.knopflerfish.org/

[6] InfraGforge: Amigo [Online]. Available: http://gforge.inria.fr/frs/?group_id=160&release_id=1804

[7] Rebeca P. Díaz Redondo, Ana Fernández Vilas, Duque, and Alberto Gil Solla, "Enhancing Residential Gateways: A Semantic OSGi Platform," IEEE Intelligent Systems, pp. 32-40, vol. 23, no. 1, 2008.

[8] Chao-Lin Wu, Chun-Feng Liao, Li-Chen Fu, "Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology", Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions, March 20