Service-Oriented Smart-Home Architecture Based on OSGi and Mobile-Agent Technology

Chao-Lin Wu, *Member, IEEE*, Chun-Feng Liao, *Member, IEEE*, and Li-Chen Fu, *Fellow, IEEE*

***Abstract*—The architecture of a conventional smart home is usually server-centric and thus causes many problems. Mobile devices and dynamic services affect a dynamically changing environment, which can result in very difficult interaction. In addition, how to provide services efficiently and appropriately is always an important issue for a smart home. To solve the problems caused by traditional architectures, to deal with the dynamic environment, and to provide appropriate services, we propose a service-oriented architecture (SOA) for smart-home environments, based on Open Services Gateway Initiative (OSGi) and mobile-agent (MA) technology. This architecture is a peer-to-peer (P2P) model based on multiple OSGi platforms, in which service-oriented mechanisms are used for system components to interact with one another, and MA technology is applied to augment the interaction mechanisms.**

***Index Terms*—Agent-based system, mobile agent (MA), multiagent system, Open Services Gateway Initiative (OSGi), smart home, service-oriented architecture (SOA).**

I. Introduction

A TRADITIONAL smart home is often implemented under a centralized architecture, as shown in Fig. 1. The home appliances are connected by the home network and controlled by the home gateway, which is the platform for service providers (SPs) to provide services to residents. However, as information appliances with computing capabilities and intelligent services are ubiquitous, the system can naturally distribute its functionalities over these devices in the home environment, which thus leads to a possible architecture designed based on ubiquitous computing devices. Another critical problem within smart homes is the dynamically varying home environment. Mobile devices are not always connected to the system, and the services configuration in the smart home is usually dynamic. Therefore, dealing with the interaction between the system and these mobile resources is not an easy task. In addition, the system should not only wait for the execution of commands given by the residents, but also collect information from the surrounding environment, infer the current situation based on the collected information, and react based on the scenario it actively infers. In the light of the requirements mentioned above, it turns out that the architecture of a smart home should be changed, thus, to comply with emerging computer technologies and to appropriately fulfill human requirements. To build such a smart home, this paper proposes a service-oriented architecture (SOA) based on Open Services Gateway Initiative (OSGi) [1] and mobile-agent (MA) technology.

To develop a smart home, it is important to conform to open standards. Otherwise, a variety of proprietary systems will cause incompatibility, which can be harmful to the mass potential consumer market [2]. In this regard, OSGi, which is an emerging open standard for deploying services to smart-home environments, appears to be a good choice. The OSGi standard is essentially a service-oriented component model. Managed software components deployed in the OSGi platform are called "bundles," and the bundles can be installed, updated, or removed on the fly without having to disrupt the operation of the device. Bundles are libraries or applications that can dynamically discover other services from the service directory or can be used by other bundles [3].

The architecture of a smart home based on OSGi is usually designed according to the client–server paradigm, as shown in Fig. 1 [4]. However, since this is a server-centric model, there is a risk of single point of failure in the home gateway. Some research has pointed out that the architecture of MAs is superior to the traditional client–server architecture in distributed computing [5], and others have suggested combining the MA architecture with the traditional smart home to solve problems caused by the client–server paradigm [6], [7], [42]–[45]. According to the requirements or different situations, an MA can dynamically migrate to ubiquitous devices in the home environment and use the associated resources to accomplish its assigned tasks, thus taking full advantage of ubiquitous computing.

SOA is a style of distributed computing that helps organizations share logic and data among multiple applications and usage modes. The basic architecture of SOA is shown in Fig. 2. There are three components interacting with one another, namely, the service consumer, the SP, and the service directory. By using the service-oriented approach, the system will have a flexible infrastructure, which can easily adapt to user requirements. Together with OSGi and the MA technology mentioned above, SOA completes our architecture. SOA coordinates these platforms and treats every interaction among them as a service requesting/provision, thus playing the role of the middleware so that these platforms can communicate with one another. As a consequence, these OSGi platforms can access system information, gather environment status, use services provided by others, and dynamically create MAs to perform tasks.

Manuscript received September 5, 2005; revised January 23, 2006 and May 5, 2006. This work was supported by the National Science Council under Project NSC93-2752-E-002-007-PAE. This paper was recommended by Guest Editor F. Wang.

C.-L. Wu and C.-F. Liao are with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei 10617, Taiwan, R.O.C. (e-mail: f89922042@ntu.edu.tw; d93922006@ntu.edu.tw).

L.-C. Fu is with the Department of Computer Science and Information Engineering and the Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan, R.O.C. (e-mail: lichen@ntu.edu.tw).

Digital Object Identifier 10.1109/TSMCC.2006.886997

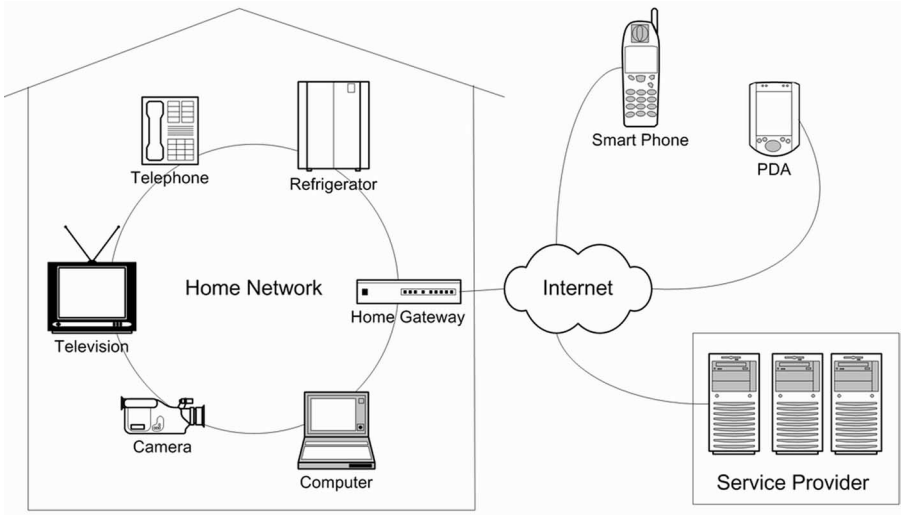

Fig. 1. Typical architecture of traditional smart home.

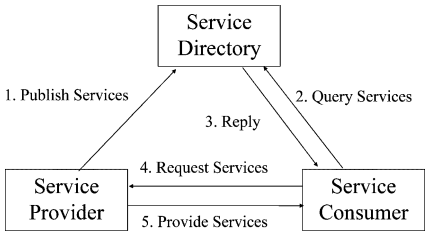

Fig. 2. Service-oriented architecture.

The rest of this paper is organized as follows. Section II overviews related works, and Section III describes the overall system architecture. Section IV will address issues regarding the design of the basic system platform, and a more thorough system examination will be presented in Section V. In Section VI, the system is evaluated by demonstrating an application scenario and the experimental results. Conclusions are made and some future works are suggested in Section VII.

II. Related Works

A. OSGi-Related Works

In recent years, there has been a lot of research related to the construction of middleware based on the OSGi standard [8]–[12]. However, the initial plan of OSGi is to develop home gateways, which makes the models adopted by these researches server-centric. As a result, there is only one OSGi platform in the proposed architectures, so that all the services and computation tasks are managed at the gateway in a centralized manner, and all the other information appliances can only be controlled by the home gateway instead of being operated autonomously. However, there will be more and more OSGi-compliant devices besides the OSGi gateway in the home environment in the future such as the cell phones, automobiles, and so on [13]. The server-centric architecture cannot take full advantage of the full capabilities of the ubiquitous computing devices and is not suitable for ubiquitous computing environments.

B. MAs

MAs were first proposed by White at General Magic in 1994 [14]. The concept of an MA was implemented as the Telescript, while a runtime environment was also constructed for the execution and migration of MAs. The Telescript was patented in the U.S. in 1997 [15]. Chess *et al.* [5] authors proposed "Itinerant Agents" in 1995, which are agents that are generated and dispatched by computers and that execute some management tasks over the network. The platforms of MAs consist of the runtime environments and development application program interfaces (APIs), and many different platforms were proposed after 1997 [16]–[19]. The most well known among them is the "Aglets" platform [18] developed by the IBM Tokyo Research Center [20], and the most recent one is Tracy proposed by Braun *et al.* [19]. Plant automation based on distributed systems (PABADIS) [21] is a project that aims to provide a plug-and-participate environment, based on a platform with multiple MAs, in production plants. In a typical PABADIS system, a product agent (PA) is an MA that represents a work piece being produced, and it migrates to several cooperative manufacturing units (CMUs) to perform specific tasks. Because there are a variety of heterogonous platforms of MAs, some efforts toward standardization, like [22] and [23], have been attempted, but none of them have been widely accepted.

To apply MA technology in a dynamic environment like a smart home, there remain some problems. First of all, a prerequisite for MAs is an agent host, the purpose of which is to let every MA in an environment run its assigned tasks; but in fact it is difficult to request that each device have an agent host embedded in it in advance. Second, most of the current MA systems lack interoperability, because their implementations are based on a specific customized protocol. The third problem is the tight coupling of MAs and the services they use. This means developers of MAs have to be aware of remote services and bind the

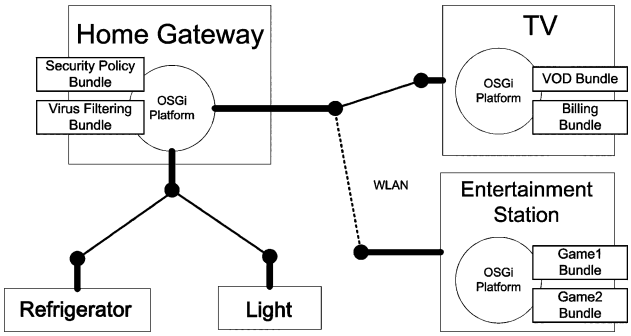

Fig. 3. P2P model of a smart-home environment.

agent to these services at development time. This tight coupling is usually not desirable since it makes MAs lack robustness. For example, if a remote host happens to remove some services which are exactly those to which MAs are bound to, the MAs will crash because of the abrupt exception.

III. Architecture

The OSGi framework was originally designed for a home gateway. However, the OSGi specification does not preclude the use of multiple platforms in an environment. Recent progress in the computing power of embedded systems has made it possible to embed the OSGi platform inside intelligent appliances such as the Interactive Television or home entertainment stations. The OSGi whitepaper [13] also pointed out that in addition to home gateways, more and more OSGi-based devices will appear in our living spaces, such as cell phones and automobiles. Consequently, instead of the traditional client–server architecture, we propose in this paper to use a more distributed peer-to-peer (P2P) model [24] for intelligent appliances (see Fig. 3), where the OSGi platform appears in several devices, thus distributing device-dependent services over several devices. The loading of the home gateway is reduced due to the distributed deployment of the bundles, and these local device-dependent bundles still operate normally if the gateway crashes.

Based on this P2P model, the MA technology can be very useful. When some OSGi service bundles can only be accessed locally or certain data cannot be processed remotely, the OSGi platform can embed the processing schedule into an MA and send it to the target OSGi platform to perform tasks. The MA can also dynamically choose its behavior according to pre-embedded logic if the remote execution environment changes the status, thus avoiding some erroneous actions and enhancing the system's capabilities.

Our proposed system architecture for smart-home environments is shown in Fig. 4. There are several components in this architecture, and each component is an OSGi bundle. According to our P2P model for OSGi, these components are distributed over multiple OSGi platforms in the home environment. These distributed platforms can communicate with one another based on the service-oriented approach. As for the MA technology, it facilitates the cooperation among these components. Detailed information is given in Sections III-A–I, including the functionalities of each component, and how they interact with others.

A. Interface Agent

1) Functionalities: This agent is responsible for interaction between the user and other components. The status of the smart home is, after being determined by this agent, presented to the user, who then manages the smart home according to such collected information. Next, this agent interacts with other agents relevant to the management requests.

2) Interaction: At first, the interface agent queries the agent directory to retrieve information about all the registered services in the smart home. After that, when the user requests services, the interface agent will: 1) call bundles directly if they are located at the same platform; 2) send requests to the remote platform like calling a web service if services are provided by remote bundles; or 3) use an MA generator (MAG) to create an MA which will migrate to a remote platform to consume the services provided by bundles at a remote platform. With the help of the MAG, a user can package some semantic scenario, a series of actions, into an MA. As for the preference agent, a user can use the interface agent to set his/her inference rules or adjust his/her preference model, and subsequently the interface agent will receive the service recommendation provided by the preference agent according to the related setting and context.

B. Device Agent

1) Functionalities: This agent is the only one connected to the devices and is responsible for direct communication with all the devices. Usually, it will be a device-dependent bundle, whose tasks are to control the devices, to monitor their status, and to cooperate with other components, such as to receive their service requests and/or to report to them the device status. It will also monitor the status of the environment and transform the observed phenomena into the raw data.

2) Interaction: The device agent will register itself at the agent directory to provide the information about how other bundles can interact with it, and it will receive service requests from other components. Raw data will be sent to a context agent for some inference purpose.

3) Devices: The devices controlled by the device agent can be classified into two categories: sensor and appliance. The difference between these two kinds of devices is that the latter can be controlled to serve users, whereas the former can only be used to collect data.

C. Service Agent

1) Functionalities: The service agent is usually a device-independent bundle consisting of predefined basic functions. It performs high-level control management by controlling several device agents, or transforms raw data to high-level semantic data, which are much more meaningful to human beings, by integrating data extracted from Device Agents. Every service agent is responsible for a specific function, like security management, energy saving, etc. Note that although both web

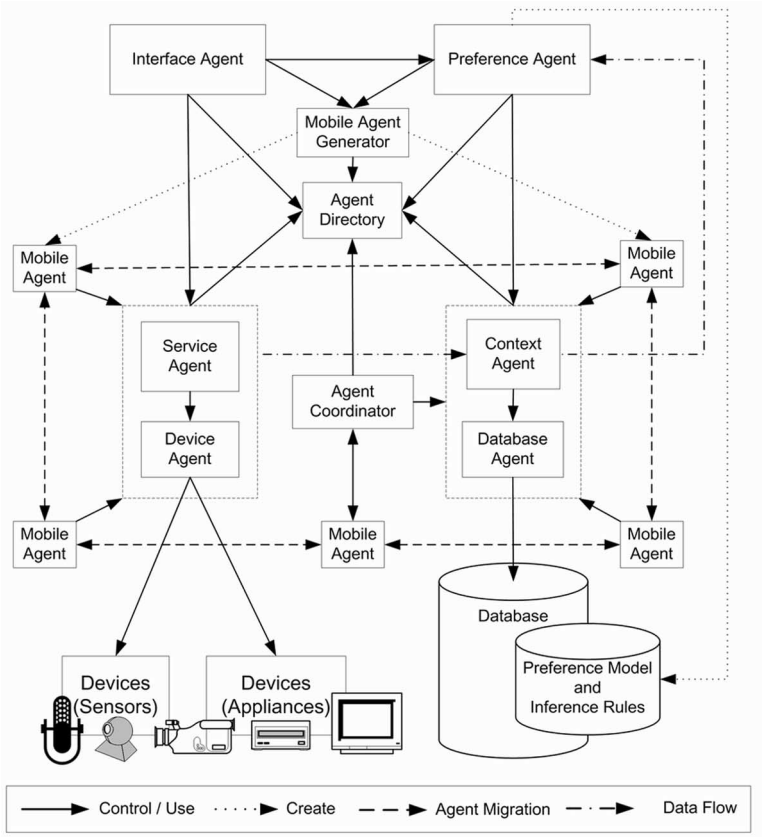

Fig. 4. System architecture.

services bundles and agent host bundles both belong to the service agent, to simplify the presentation they are not explicitly shown in Fig. 4, as they exist in almost every communication.

2) Interaction: The service agent also registers itself at the agent directory to provide interaction information and receive service requests from other components. Through the agent directory, every service agent retrieves information on related Device Agents, and thus knows how to interact with them to perform the specified services. High-level semantic data are sent to the context agent for inference purposes also.

D. Agent Directory

1) Functionalities: This component is a registry of service bundles, which is the service directory in the SOA. The interaction information of public bundles, including where they are, what services they provide, and what information is necessary for other bundles to request their services, are stored in the agent directory. In our architecture, this component is a universal descripition, discovery, and integration (UDDI) service in the home domain.

2) Interaction: Any service bundle managing to provide services to remote clients publishes itself as an agent to the agent directory. Specifically, the bundle first registers itself on the registry of the web services bundle at its own OSGi platform, and then the web services bundle publishes these registries to the agent directory. The agent directory also provides information for the agent coordinator to solve problems.

E. MAG

1) Functionalities: According to related information, this component creates an MA by specifying trips and tasks in the schedule of the MA based on the document type definition (DTD) of the MA specification markup language (MASML), which is be described in the following section.

2) Interaction: Based on the semantic scenario from the interface agent or the preference agent, the MAG queries the interaction information from the agent directory, and then transforms the semantic scenario into an MASML document. After that, the MAG returns this document, which is an MA, to the agent requiring it, for the agent to dispatch this MA to perform its tasks.

3) MA: An MA migrates to agent hosts in the smart home to perform tasks according to its schedule. These tasks include requesting services from bundles and processing data retrieved from previous tasks. When an MA finds that the task conflicts with some task of another MA, or that ambiguous information exists in the schedule, it seeks help from the agent coordinator, which solves conflicts or ambiguity by providing the current status of the system.

F. Context Agent

1) Functionalities: This component plays the role of the data aggregator. It transforms raw data and semantic data into context data, provides context data for related queries, creates events based on these data, and delivers these data-related events.

2) Interaction: The context agent registers itself in the agent directory for other components to find it. Based on the information provided by the agent directory, the context agent knows how to interact with the Device Agent and the service agent in the smart home, and then gathers data from these agents. Subsequently, these data are transformed into context data based on the preference model and inference rules, and are then sent to the database agent for storage. When interpreting context data, related messages are created and sent to the preference agent to trigger events. Context data are provided to an MA to perform computations, or to the agent coordinator to solve problems caused by dynamically changing contexts.

G. Database Agent

1) Functionalities: This is the only agent connected to the database, and its purpose is to manage all kinds of data in the smart-home environment, including raw data, semantic data, and context data. It also provides the interface for other components to store or access data in the database.

2) Interaction: The database agent registers itself in the agent directory for other components, especially the context agent and the preference agent, to find it. It receives data from the context agent and stores it, provides data query services for an MA dispatched by other components, and provides the preference agent with all the necessary data for learning purposes.

3) Database: All kinds of data are stored here, including raw data, semantic data, and context data.

H. Preference Agent

1) Functionalities: A home environment usually accommodates several residents and each may have their preferences toward different kinds of services. This agent preserves individual preferences for various services. The preference agent creates a preference model for each resident, customizes this model through learning procedures, and updates the model whenever possible. The preference agent also sets Inference Rules according to requests from each resident. Moreover, according to events as well as the preference model and inference rules, the preference agent reasons about semantic scenarios and then performs related tasks or recommends service to residents [25].

2) Interaction: The preference agent publishes itself in the agent directory for the Interface Agent to find it and sends requests to update the Inference Rules. It retrieves all the necessary data from the database agent and learns the optimal preference model. Based on the messages from the context agent, the preference agent looks up the preference model and inference rules to create semantic scenarios if necessary. If there is an existing scenario, the preference agent requests the MAG to create an MA to perform tasks based on this scenario.

3) Preference Model and Inference Rules: The model and rules are built and set for each resident. In general, the Inference Rules are usually static and set by the user, whereas, the preference model is often dynamic and frequently updated. In our system, the preference model is created and trained based on the Bayesian belief network, and it builds a relationship among the raw data, semantic data, and context data [25]. This model is adopted by the context agent to check whether an event has to be triggered after certain data changes and is used by the preference agent to decide if a service has to be provided.

I. Agent Coordinator

1) Functionalities: This component is mainly used for solving problems that arise from MAs. Since there are two sources that create MAs, namely the Interface Agent and the preference agent, the tasks of one MA may conflict with those of another MA if these two sources have different requirements, hence causing a feature interaction problem [26], [27]. The agent coordinator can help solve this kind of conflict by checking the current status of the environment. In addition, the environment status triggering MA may have changed by the time the MA is ready to fulfill its assigned mission, thus requiring the MA to change its behavior accordingly. If the change in the situation is predictable, embedded logic can help the MA to solve this problem. However, the change is not always predictable. So, the MAG may need to generate a flexible schedule for the MA and allow the MA to consult the agent coordinator for a resolution. If the problem cannot be solved, the agent coordinator will not do anything, thus leaving the problem to be solved amongst the residents. Because the MA carries error messages back to the user, the user can always directly interact with the smart home via the Interface Agent.

2) Interaction: The agent coordinator registers itself in the agent directory for MAs to find it. It also queries the agent directory, context agent, and database agent to retrieve related information to solve problems for MAs as mentioned in the previous section.

IV. Interoperable MA Migration Mechanism

The MA is a crucial component in our architecture, but the OSGi specification does not explicitly define standard inter-OSGi communicating mechanisms, thus causing agent hosts deployed by each individual OSGi platform to be isolated across different platforms. In order to deal with the problem related to MAs mentioned above, we propose a new architecture for MAs, in which their migration mechanisms are based on extensible markup language (XML) and web services.

The core idea is to encode the plans, behaviors, and states of MAs into XML-based scripting documents, the MASML, which can be pulled or pushed with the web services and transmitted over the hypertext transfer protocol (HTTP) and simple object access protocol (SOAP) [28]. For example, an agent host can initialize an MA by acquiring the MASML from a web service. The agent host then starts to interpret the MASML and executes several tasks or invokes external services, after which the results or unexpected errors such as network failure are written back to the document. Finally, this agent host is either directed to let the MA migrate to the next agent host or returns the above-mentioned document back to the original agent host after the MA has finished all of its assigned tasks. Detailed design issues are described in the later sections.

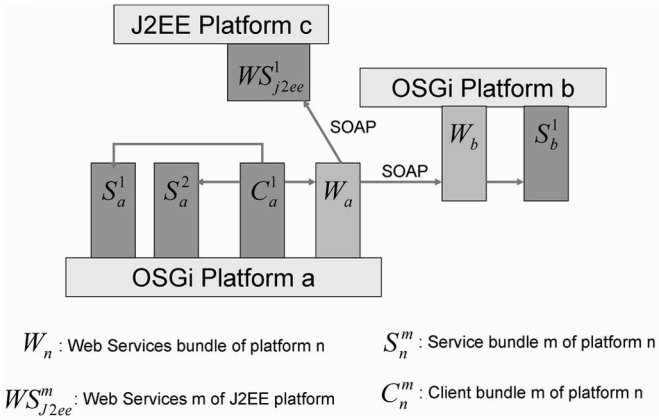

Fig. 5. Inter-OSGi interaction via web services bundles.

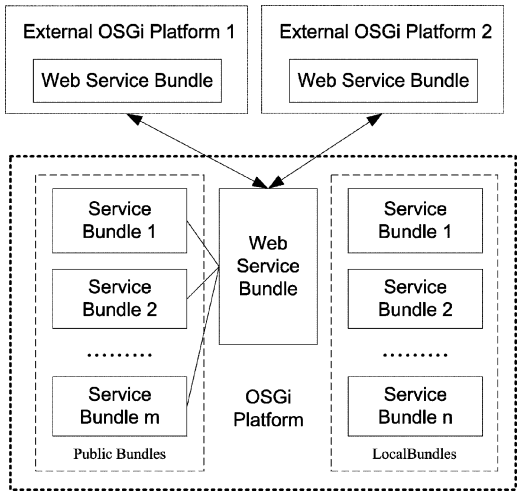

Fig. 6. OSGi platforms connecting with the web services mechanism.

A. Inter-OSGi Interaction with Web Services

As mentioned earlier, the major issue in our P2P model for smart-home environments is the communicating mechanisms, or protocol among OSGi platforms. The service-oriented approach we proposed to deal with this issue is that, in each OSGi service platform, we deploy a web service gateway bundle. By doing so, each OSGi platform can publish its services, which makes other OSGi platforms aware of and able to utilize those services. Each OSGi service that registers itself to this bundle is able to interact with other web services through the SOAP, which is a web services standard proposed by the World Wide Web Consortium (W3C).

The major benefit of using the web services standard is that it provides interoperability between OSGi and non-OSGi platforms. For example, in Fig. 5, the OSGi platforms can publish their local services as public web services via a web service bundle, and consume the web services that are published by other OSGi or J2EE [29] SPs.

However, due to security reasons and data issues, a service bundle may publish itself, but cannot provide services to remote clients, or it will limit access only to bundles at local platforms. Under this circumstance, system environments with multiple OSGi platforms, which are connected by web services mechanisms, are in a configuration as shown in Fig. 6. It then follows that any client located at a remote platform cannot request those constrained services directly, and hence, the MA technology, which is discussed in next section, is used to solve this problem.

B. MA Architecture for OSGi Platforms

Based on OSGi specifications, an agent host can be implemented as a service bundle, and each OSGi-compliant device can dynamically download it if necessary, thus fulfilling requirements to set up execution environments for the MAs. As a result, by incorporating the inter-OSGi interaction mechanism, the agent host cannot only be deployed on the OSGi platform, but also interact with an external environment via web services. Note that the proposed approach here is based on open standards,

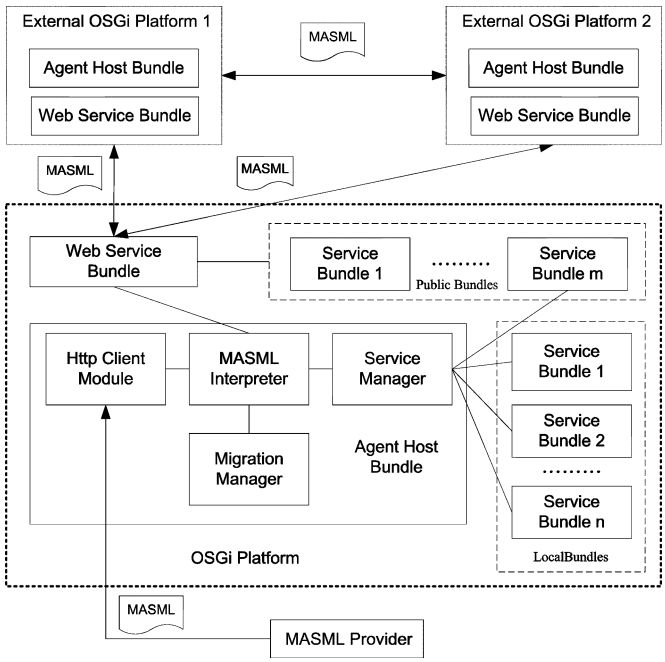

Fig. 7. MA architecture for an OSGi platform.

and the services can be discovered and bound to the caller at runtime in a typical OSGi environment.

Fig. 7 shows the proposed MA architecture for the OSGi platform. In this architecture, the MAs can migrate to a number of agent hosts deployed on different OSGi platforms and can carry out some services provided by the target platform. By doing this, nonpublic local service bundles can be called by MAs via agent hosts, thus solving the problem mentioned at the end of the previous section. From a technical point of view, the migrating behavior of an MA is actually the transportation of MASML documents over web services mechanisms, and the service-execution behavior of an MA is actually realized by the service manager component, which is instructed by the interpreter according to the MASML document.

The web services bundle is the gateway to facilitate communication with external OSGi platforms. The agent hosts publish their services by registering themselves on the registry of the web services bundle. When the web services bundle receives a request for the registered services, it delegates control of the execution to the associated agent host bundle. Note that an agent host bundle is composed of four components, namely, the MASML interpreter, the agent specification, the service manager, and the migration manager. The MASML interpreter analyzes and executes the incoming MASML documents according to the agent specification interpretation algorithm (ASIA), the details of which are discussed later. When the interpreter finds that services have to be executed, the service manager is responsible for looking up the references for these services and invoking them. Finally, if the MA needs to migrate to another host, the migration manager performs the migration service by calling the target agent host bundle registered as a remote web service.

The architecture supports both the pull and push model on the acquirement of MASML. In the push model, the agent host bundle plays a more passive role and waits for MASML documents to be executed. The syntax of MASML also supports the hyperlinked feature, which is the nature of web documents. For example, in order to enhance reusability, an MASML document may import one or more remote documents by providing the uniform resource locator (URL) as a reference. The HTTP client module is responsible for fetching this type of remote document from an MASML provider, which is usually a web server or an application server responsible for hosting or generating MASML documents. Note that the pull model also implies that MASML can be dynamically generated at runtime by server-side scripting languages provided by the web servers, e.g., the hypertext preprocessor (PHP) provided by Apache web servers or active server pages (ASP) provided by the internet information server (IIS) web server. The server-side scripting technologies greatly increase the flexibility of MASML because the servers can produce personalized documents according to different requests.

C. MASML

The core concepts of MASML can be expressed via a conceptual object model in the unified modeling language (UML) class diagram (see Fig. 8). In this model, the behaviors and states of an MA are specified through the MA specification. Here, an MA is created for fulfilling a specific request by migrating through a series of agent hosts. We call this type of migration process a "trip," and within each agent host, a number of tasks have to be carried out. Accordingly, an MA specification consists of one or more Trips, and each Trip consists of one or more tasks. There are two types of Task in the MASML object model, namely, "logic" and "service." A "service" means an external function call to OSGi services, whereas "logic" stands for an inference procedure for internal data processing in an agent.

According to the object model, we propose an XML-based markup language, MASML, to reify this conceptual object model. The DTD below also provides a formal definition of this markup language (see Fig. 9).

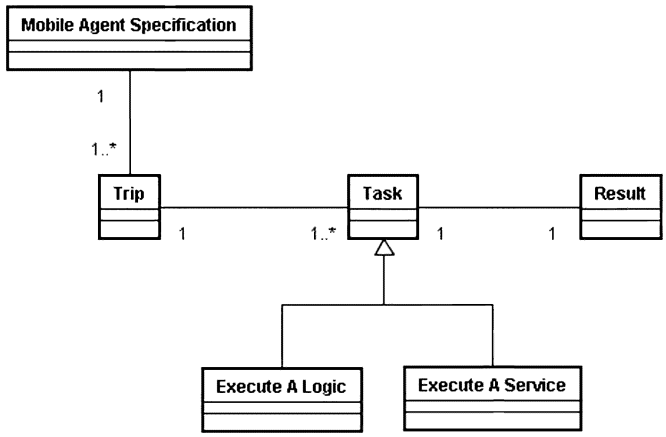

Fig. 8. Conceptual object model of a mobile-agent specification.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT masml (agent-id, agent-home, current-trip, agent-trip*)>
<!ELEMENT agent-id (#PCDATA)>
<!ELEMENT agent-home EMPTY>
<!ATTLIST agent-home
wsurl CDATA #REQUIRED
method CDATA #REQUIRED
>
<!ELEMENT current-trip EMPTY>
<!ATTLIST current-trip
seq CDATA #REQUIRED
>
<!ELEMENT agent-trip (service | logic)*>
<!ATTLIST agent-trip
seq CDATA #REQUIRED
wsurl CDATA #REQUIRED
method CDATA #REQUIRED
>
<!ELEMENT logic (#PCDATA|EMPTY)>
<!ATTLIST logic
seq CDATA #REQUIRED
src CDATA "null"
>
<!ELEMENT service (param*, on-error, result?)>
<!ATTLIST service
seq CDATA #REQUIRED
service CDATA #REQUIRED
method CDATA #REQUIRED
>
<!ELEMENT param (param-name, param-value)>
<!ELEMENT param-name (#PCDATA)>
<!ELEMENT param-value (#PCDATA)>
<!ELEMENT on-error EMPTY>
<!ATTLIST on-error
mechanism (ignore | goHome) "ignore"
>
<!ELEMENT result (#PCDATA|EMPTY)>
<!ATTLIST result
(logic | text) CDATA "text"
>
```

Fig. 9. DTD of the MASML.

Fig. 10 is an example of a valid MASML document, which represents a "turn on lamp" MA. We do not include the headers and DTD declarations of a standard XML file so that only the major content of the MASML is focused upon. In this document, the agent's ID and URL are provided whereby the MA can return to the original agent host. In this example, the MA migrates from

```
<masml>
    <agent-id>SDK38D-0001</agent-id>
    <agent-home wsurl="http://localhost:8080/axis/services/agentHost0"
        method="acceptBack"/>
    <current-trip seq="0"/>
    <agent-trip seq="0"
        wsurl="http://localhost:8081/axis/services/agentHost1"
        method="accept">
        <service seq="0" service="lamp" method="control">
            <param>
                <param-name>command</param-name>
                <param-value>on</param-value>
            </param>
            <on-error mechanism="ignore"/>
        </service>
    </agent-trip>
</masml>
```

Fig. 10. MASML of a "turn on lamp" mobile agent.

```
<masml>
    <agent-id>SDK38D-0002</agent-id>
    <agent-home wsurl="http://localhost:8080/axis/services/agentHost0"
        method="acceptBack"/>
    <current-trip seq="0"/>
    <agent-trip seq="0"
        wsurl="http://localhost:8081/axis/services/agentHost1"
        method="accept">
        <service seq="0" service="lamp" method="control">
            <param>
                <param-name>command</param-name>
                <param-value>on</param-value>
            </param>
            <on-error mechanism="ignore"/>
            <result/>
        </service>
        <logic seq="1"><![CDATA[[
            if (result[0] == 'fail')
            {
              log.info("task failed");
            }
            ]]></logic>
    </agent-trip>
</masml>
```

Fig. 11. MASML of a "turn on lamp" mobile agent with failure handling.

agentHost0 to agentHost1 in order to execute a lamp control service. MA returns to agentHost0 after the task to turn on the lamp is accomplished. Since the tasks are executed serially, the "current-trip" element stores the current state of the whole journey of an MA. For example, if there are five trips in this document, the "current-trip" element will point out which trip the MA is currently located in.

Sometimes an agent has to process data or react to the results of service executions. We provide this type of agent logic by the European Computer Manufacturers Association (ECMA) script [30]. ECMA is an organization facilitating the creation of information and communications technology (ICT) and consumer electronics (CE) standards. In Fig. 11, an example of a MASML with an embedded ECMA script is given. In this example, an additional logic element is added that requires logging into the system database in order to react to the failure condition. The script is surrounded by character data (CDATA)

```
<masml>
    <agent-id>SDK38D-0003</agent-id>
    <agent-home wsurl="http://localhost:8080/axis/services/agentHost0"
        method="acceptBack"/>
    <current-trip seq="0"/>
    <agent-trip seq="0"
        wsurl="http://localhost:8081/axis/services/agentHost1"
        method="accept">
        <service seq="0"
          filter="(&(place=livingroom)(light=*))"
          service="lamp" method="control">
            <param>
                <param-name>command</param-name>
                <param-value>on</param-value>
            </param>
            <on-error mechanism="ignore"/>
            <result/>
        </service>
    </agent-trip>
</masml>
```

Fig. 12. MASML of a "turn on lamp" mobile agent with service filtering.

to ensure that the special characters, which may conflict with XML syntax, are escaped. The embedded script is processed and interpreted by the Rhino scripting engine, an open-source project hosted in Mozilla [31].

To provide more flexible services, the MA is allowed to select and bind the services dynamically. To attain this, the lightweight directory access protocol (LDAP) search filter [31], as suggested by the OSGi, is used. Fig. 12 shows a "turn on lamp" agent using the service filter. To use a service filter, service consumers have to pass a "filter" parameter when calling the service. In this example, the symbol "&" means AND relationship, and "$= *$" means an existing test. As a consequence, "(&(place=livingroom) (light$= *$))" means that the service will only be selected only if the lamp is in the living room and the light is turned on. A more complete syntax of LDAP search filters can be found in [32].

D. ASIA

ASIA is intended to provide a behavioral view of the agent hosts that is capable of interpreting MASML. The communications between agent hosts are based on web services, which implies that an agent host does not always have to be implemented on an OSGi platform. As a result, an MA can migrate to any agent host that can interpret the MASML with ASIA. On the other hand, if the service platform is not OSGi-based, the MASML interpreter has to be designed so that it can process the service filtering appropriately.

Fig. 13 shows the Agent Specification Interpretation Algorithm. UPDATE_TRIP_COUNT means that the trip count is increased by one after each migration is done in order to keep track of the current trip section that the interpreter is executing. After the current trip section, indexed by the trip count (T[i]), is found, its registered tasks are interpreted sequentially, and in turn some services or logics are carried out according to the agent specification, EXECUTE_TASK(T[i]). Next, the execution result is processed with HANDLE_ERROR() and updated with APPEND_RESULT(). Finally, after the tasks are all

```
Procedure ASIA ( ∑ : MASML document)
Variable  H : Location of the original agent host
          H[k] : Location of agent host at trip k
          C : Current trip count
          T[n] : Task n
          R : Result

 C ← C +1
 UPDATE_TRIP_COUNT( C , ∑ )

 for i ← 0 to n
 do
      R ← EXECUTE_TASK( T[i] )
      If  R is an error result
                HANDLE_ERROR( ∑ )
      APPEND_RESULT(R, ∑ )
 end

 if T[C +1] = null then
      GO_HOME(H, ∑ )
 else
      MIGRATE( H[C +1], ∑ )
```

Fig. 13. Agent specification interpretation algorithm.

finished, the MA either returns to the original agent host by calling the GO_HOME() or migrates again to its next destination by calling the MIGRATE().

V. Analysis and Discussion

In this section, we first analyze system performance under three different models and then discuss fault tolerance issues.

A. Performance

Assume that there are n clients requesting services at the same time; each service involves m SPs and costs each provider p computation time units. Each service has to interact with the service client with j messages, and it costs i network traffic units to transmit one message, whereas, every migration of an MA requires k network traffic units. Each SP produces data for a client, and the underlying process costs q computation time units. Finally, to simplify the representation, we assume that it also costs q computation time units to integrate all these data returned from each SP.

For a conventional server-centric architecture, to perform the scenario mentioned above, each client must send service requests to the server, and then the server interacts with these m SPs by sending j messages to accomplish these service requests. After receiving data from these SPs, the server processes these data, integrates them, and returns the final results to the clients. Analysis is shown in Table I, in which SP stands for "service provider," $(*\,\mathrm{n})$ and $(*\,\mathrm{m})$ represent the possible numbers of the associated entities, and $1\sim n$ in the two SP columns means each SP accepts $1\sim n$ service requests, thus changing the workload of its network traffic and computation.

As for the P2P SOA, clients query the service directory first to retrieve information about SPs, and then interact with m SPs remotely by sending j messages to each. When a client receives data that are returned from m SPs, it processes them, hence costing q computation time units for each one, and then integrates them. Performance analysis is shown in Table II.

Table III shows the performance analysis of our proposed model. Here clients also query the service directory first to retrieve information about SPs, and then create MAs embedded with interaction logic based on service requirements. After that, every client sends its MA to SP, and since the MA represents the client, the SP can directly interact with the MA locally, thus avoiding remote interaction that creates a lot of network traffic usage. Besides that, the MA uses the resources of the SP to process data based on its previously embedded logic, thus reducing the computation load of clients in P2P models.

TABLE I
PERFORMANCE ANALYSIS OF THE SERVER-CENTRIC ARCHITECTURE

Server-Centric Architecture	Network Traffic			Computation Load	
Scenario	**Client (* n)**	**Server**	**SP (* m) 1~n**	**Server**	**SP (* m) 1~n**
Client requests Server	*i*	*i*n*			
Server requests SP		*i*j*m*n*	*i*j~i*j*n*		
SP starts services					*p~p*n*
SP returns data		*i*j*m*n*	*i*j~i*j*n*		
Server processes data				*q*n*m*	
Server integrates data				*q*n*	
Server returns results	*i*	*i*n*			
Total	*2*i*	*2*i*n *(1+j*m)*	*2*i*j~ 2*i*j*n*	*q*n *(1+m)*	*p~p*n*
	*4*i*n+4*i*j*m*n*			*n*(q+m*(p+q))*	

TABLE II
PERFORMANCE ANALYSIS OF THE P2P SOA

P2P SOA	Network Traffic			Computation Load	
Scenario	**Client (* n)**	**Service Directory**	**SP (* m) 1~n**	**Client (* n)**	**SP (* m) 1~n**
Client queries Service Dir	*i*	*i*n*			
Client requests SP	*i*j*m*		*i*j~i*j*n*		
SP starts services					*p~p*n*
SP returns data	*i*j*m*		*i*j~i*j*n*		
Client processes data				*q*m*	
Client integrates results				*q*	
Total	*i+ 2*i*j*m*	*i*n*	*2*i*j~ 2*i*j*n*	*q*(1+m)*	*p~p*n*
	*2*i*n+4*i*j*m*n*			*n*(q+m*(p+q))*	

TABLE III
PERFORMANCE ANALYSIS OF P2P SOA WITH MA

P2P SOA with MA	Network Traffic			Computation Load	
Scenario	**Client (* n)**	**Service Directory**	**SP (* m) 1~n**	**Client (* n)**	**SP (* m) 1~n**
Client queries Service Dir	*i*	*i*n*			
Client sends MA to SP	*k*		*k~k*n*		
SP starts services					*p~p*n*
SP processes 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))*	

However, to embed logic into the MA increases its size, thus costing $(k - i)$ additional network traffic units relative to that for normal messages. Finally, when the MA returns back to the client, it carries the processed data for the client to perform the final integration. Note that we assume the result carried back by the MA only increases the size of the MA by as much as that for a normal message.

Observing the computation load part in these three tables, we can see: 1) that the total computation load is always the same; 2) the load of the SP does not change much; and 3) the main difference lies in the client part. In the P2P model, the server load in server-centric architecture is distributed among n clients. Although this kind of distribution seems fair, it is not suitable for clients who lack computation power, especially for mobile devices carried by people. Based on our proposed architecture, with the help of MAs, the computation load of a client who lacks computing power can be more widely distributed to more SPs, and the distributed load is not transferred to one single server only, which is the situation in the server-centric model.

As for the network part, from a server-centric architecture to a P2P SOA, the total traffic is decreased while each client handles more, thus distributing the network traffic. It is not easy to compare network traffic in P2P SOA and those with MAs, because there are different types of network traffic units which cannot be compared directly, and further analysis will be discussed later. Note that an MA helps to decrease the number of remote interactions between a client and an SP by sending $j \times m$ messages in total to transmitting one MA instead, but such seemingly simplified network traffic handled by a client actually increases, since the size of the MA is greater than that of a normal message. However, for a current mobile device which has a dynamic network connection, the MA can drastically decrease the number of interactions, and if the MA can be sent within a short period of time, the time the mobile device needs to maintain the network connection is also decreased drastically from the entire service period to the period of time needed to send the MA, and then receive it.

Comparing network traffic in P2P SOAs and those with MAs, the network traffic of a client is increased by $i + 2(k - ijm)$, so if the size of an MA is equal to, or even less than, the size of all the interaction messages, the network traffic does not cost much. As for the SP, the network traffic is increased by $i + 2(k - ij)$ for each service. Based on the earlier assumption, this increased amount is roughly $i + 2ij(m - 1)$, which shows that any SP receiving an MA has to pay an additional network traffic cost for the data needed for other SPs. However, since SPs usually stay at permanent networks, this additional traffic does not cost much.

B. Fault Tolerance

In this section, for each component, we discuss the situations in which it may fail.

1) Interface Agent: If this agent fails, the user cannot know the system's status, and also loses the ability to directly interact with the system. However, if the other components still work, the preference agent continues to provide services to the user based on the preference model and inference rules, and the user can still interact with the system manually in a conventional lifestyle.

2) Device Agent: Since our architecture is based on the P2P model, failure of one device does not affect the other ones. Therefore, the whole system can still function well, except that the services provided by the failed Device Agent are no longer available.

3) Service Agent: The situation resulting from a failed service agent is the same as that of a failed Device Agent.

4) Agent Directory: Without the agent directory, all system components can neither publish their own services to others nor update the information telling others how to interact with it. In other words, system components do not know whether a new service has been added to the system or a previously existing service has been removed. Due to the lack of dynamic information provided by the agent directory, the agent coordinator is not able to solve problems arising from MAs, and the MAG is also not able to generate MAs with dynamic schedules. However, the relationships established before the agent directory fails still exists, so: 1) the user can still directly use the service agent and control the Device Agent; 2) the context agent still gathers system data; and 3) the preference agent keeps providing static services. Note that by federating several UDDI services, information stored in the agent directory can be replicated, hence preventing the aforementioned situation.

5) MAG: If this component fails, new MAs cannot be created in the system. However, this component can also be duplicated, so that there are multiple ones in the system. Moreover, the MAG can even be embedded into the Interface Agent or preference agent to prevent loss of help from the potential failure of the MAG.

6) Context Agent: Without help from the context agent, context data are not interpreted, and the system's status is not updated into the database, either. The former would mean that no messages could be sent to the preference agent to trigger events, and the latter affects the updating of the preference model through learning, due to the lack of new preference data. Although the system is not able to provide context-aware services, the user can still interact with the smart home via the Interface Agent.

7) Database Agent: If the database agent fails, the system's status cannot be updated into the database, therefore stalling the updating of the preference model due to the lack of new preference data. Although the preference model cannot be updated, the user can still interact with the system, and context-aware services can still be provided.

8) Preference Agent: The failure of this agent voids context-aware services. However, the user can still interact with the smart home, and the context agent continues to update the database, so the preference model can resume updating once the preference agent is recovered.

9) Agent Coordinator: Without the help of the agent coordinator, MAs are not able to solve conflicts or to determine dynamic schedules. However, other components are not affected, and MAs carry error messages back to the user, thus leaving the user to solve the problem manually.

VI. System Evaluation

This section evaluates our proposed architecture. The significance of our proposed work lies mainly in the P2P model for OSGi and the MA mechanism. The former achieves interoperable multiplatforms, while the later automates semantic scenario via logic and schedules. We first describe the application scenario to address those two significant features, and then present the prototype implementation as well as the results of the experiments.

A. Application Scenario

Alice leaves her office for home at 17:30, with a smart phone loading her unfinished documents. On her way home, while Alice is using the smart phone to write e-mails to arrange a meeting for the next day, she receives a phone call from her boyfriend, Bob, who sends her a new MP3-formatted song as a gift and asks her out for a dinner and an opera. Alice gladly updates the calendar in her smart phone with this 19:30 date and stores the song in her smart phone.

When Alice returns home at 18:30, her home gateway sends an MA to perform tasks as usual. The MA migrates to her smart phone to retrieve Alice's unfinished works, including unsent e-mails and unfinished documents. The MA also checks Alice's calendar and finds that Alice is going out on a date. The MA then migrates to her entertainment station to play Alice's favorite music, and then to inform the entertainment station that Alice is not going to stay at home that night. After that, the MA migrates to her PC to store Alice's documents and send her e-mails. Finally, the MA migrates back to the home gateway. Instead of heating water at the usual 21:00 time, the MA asks the home gateway to heat water for Alice to bathe immediately, as Alice must prepare for her date.

As the MA from the home gateway performs its tasks, the smart phone also sends an MA, scheduled by Alice on her way home, to perform other tasks. This MA first migrates to the entertainment station where it stores the new song from Bob and also plays it. However, the last action causes a feature interaction problem because this MA wants to play a music selection different than the one previously played by the MA from the home gateway, and hence, the agent coordinator is asked to solve this feature interaction problem. After that, the MA migrates to the PC to get information for the date that night, including the ideal route, restaurant menu, and opera. Finally, the MA returns to the smart phone with the information.

When the entertainment station is notified that Alice is not going to stay at home, it automatically schedules the recording of the television shows Alice usually watches at 20:00, and also sends out a third MA to perform new tasks. Originally, the entertainment station had planned to inform Alice at dinner time that a DVD in which she was intererested was released that day. Since Alice is not going to eat dinner at home, the MA from the entertainment station first migrates to the PC to check DVD shops in the area where Alice's date is going to take place, and then migrates to the smart phone to store this information about the newly released DVD where Alice will see it.

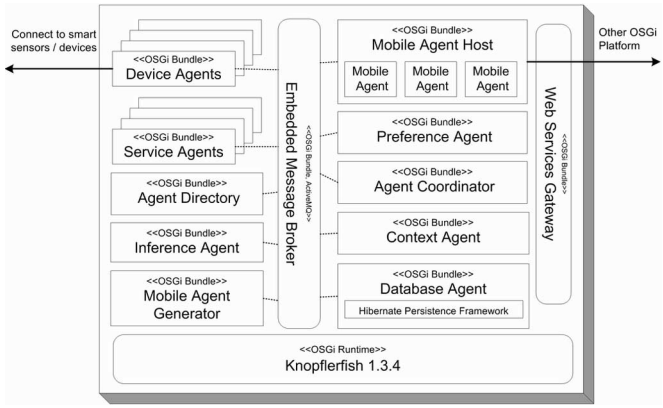

Fig. 14. Experimental prototype.

After showering, Alice goes out for her date with the smart phone, which has stored the information about the date and DVD. With this information, Alice can easily go to the restaurant, order from the menu, discuss the opera with Bob over dinner, and purchase the DVD at a shop near the theater.

Without the MA mechanisms, many tasks would not be automated, especially those tasks performed by the MA from the smart phone, and things would not be so easy and comfortable for Alice. In addition, if our P2P model for OSGi is not used, the performance of the home gateway suffers, and the capabilities of the various computing devices are not fully utilized. This issue is addressed in the next section.

B. Prototype Implementation

In order to evaluate the effectiveness of the proposed architecture, we have built an experimental prototype, as shown in Fig. 14. The agents are realized as OSGi bundles and are deployed on Knopflerfish 1.3.4 [33], which is an open source OSGi R3 implementation. These OSGi platforms are hosted on P3/600 MHz mini-PC with 512 MB RAM.

To enhance the flexibility of this prototype, the inter-agent communications are implemented according to a publish-subscribe messaging style with an embedded message broker bundle, which is developed, based on an open-source lightweight embeddable message queue called ActiveMQ [34]. The web services functionalities are provided by Apache Axis 1.1 bundle service that shipped with Knopflerfish OSGi implementation, and the MySQL 4.0.18 is used as the repository of context and preference information.

C. Performance Evaluations

To study the performance of three different interaction models (server-centric, P2P, and P2P + MA), we have conducted experiments with our prototype implementations. In each experiment, we use a task controller to initialize the execution of a new task and to measure the turnaround time of execution. A task is a sequence of executable procedures on devices distributed over the network. Upon receiving a new task, the OSGi platforms have to collect data from devices, and then integrate the data as well as perform reasoning to produce the results.

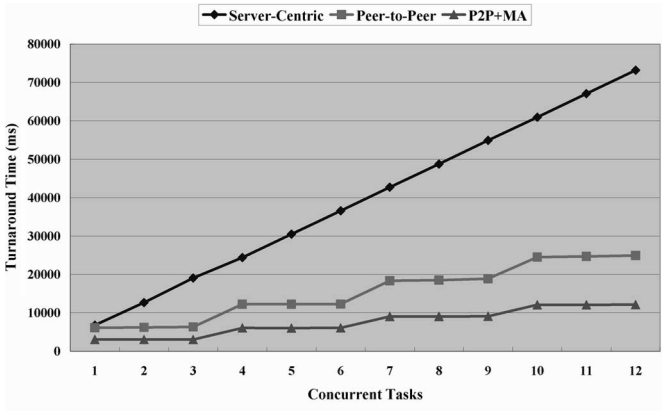

Fig. 15. Performance evaluations of different interaction models.

Experimental environments are described below. In the first scenario (server-centric), there is a single OSGi platform connected with six devices. As for the second and third scenarios (P2P and P2P + MA), there are three OSGi platforms interconnecting with one another via Ethernet locally, and each platform is connected with two devices. Each task is defined as sequentially gathering data from six devices and then performing integrating/reasoning procedures in OSGi platforms. The time to integrate/reason and the data retrieval latency are both set as 100 ms.

Fig. 15 shows the results of the experiments. Compared to the server-centric interaction model, the turnaround time is decreased drastically with either P2P or P2P + MA one. As pointed out in Section V, the performance gains come from the parallel processing of concurrent tasks on several OSGi platforms. In the P2P + MA interaction model, the performance is even better. Because the services are invoked locally in the P2P + MA model, most of the costs come from the migration latency. Consequently, the experiment results show that the P2P + MA model helps to reduce the task execution time by parallel processing of tasks and local execution of services.

VII. Conclusion and Future Works

In this paper, we have proposed an SOA for a smart-home environment, and this architecture is based on OSGi and MA technology. We have proposed a P2P interaction model with multiple OSGi platforms instead of a conventional server-centric model, thus taking advantage of all the resources in the smart-home environment by distributing the working load over the system. With OSGi technology, an agent host, which is a prerequisite to apply MA technology, can be dynamically downloaded to the platform, by creating an environment suitable for a MA. MAs are used in our architecture to deal with dynamic situations and further distribute loading from client to multiple SPs. To preserve interoperability between agent hosts, we propose a web services-based migration mechanism for MAs. MAs are encoded with MASML, which can then migrate over and consume services within several agent hosts that support ASIA. The service-oriented approach is applied in our architecture so that all the components can dynamically join/leave the system, and the SOA coordinates the interactions between these platforms.

At this point, we focus on the service integration and mobility within the home environment. A possible extension of our work is to export the OSGi-hosted services to the external environments. Moyer *et al*. [35], [36] proposed an extended version of a session initiation protocol (SIP) [37], which aims to export the home services of networked appliances to the mobile environments. SIP is originally a call-setup and session management protocol widely used in the internet protocol telephony environment, and it has many attractive features such as asynchronous notification and session control, which are not addressed in the conventional HTTP. Recently, these features can be found in the emerging second generation of web services standards proposed by OASIS [38]. Due to the ubiquity of HTTP and the advantages of XML, the approach based on web services is a competitive alternative.

In the future, we will extend the capabilities of the MASML DTD to make MAs more flexible. We will also use the ontology web language (OWL) [39] to model our context after the current preference model in this paper and also on human behavior analysis. In addition, techniques from [26] and [27] will be studied and applied with MAs, e.g., embedding priority information into an MA for locking or restricting the access of the resources, to solve the details of feature interaction problems in such a multiagent environment. The security issues within agent migration, services installation, and service providing will also be addressed. Based on existing open web services security specifications such as the security assertion markup language [40], and extensible access control markup language [41], the platform will ideally be able to provide transparent security mechanisms more easily. We will also try to evaluate the implementation complexity and cost of the proposed system. Finally, we will attempt to efficiently integrate, find, and use multiple resources, to distribute the working load even further, and to enhance the fault tolerance of the smart-home environment.

References

[1] *OSGi alliance* [Online]. Available: http://www.osgi.org
[2] K. Wacks, "The successes and failures of standardization in home systems," in *Proc. 2nd IEEE Conf. Standardization Innovation Inf. Technol.*, Boulder, CO, Oct. 2001, pp. 77–88.
[3] *OSGi service platform R3* [Online]. Available: http://www.osgi.org/documents
[4] R. S. Hall and H. Cervantes, "Challenges in building service-oriented applications for OSGi," *IEEE Commun. Mag.*, vol. 42, no. 5, pp. 144–149, May 2004.
[5] D. M. Chess, B. Grosof, C. G. Harrison, D. Levine, C. Paris, and G. Tsudik, "Itinerant agents for mobile computing," IBM T. J. Watson Res. Center, Yorktown, NY, Tech. Rep. RC 20010, Oct. 1995.
[6] C. L. Wu, W. C. Wang, and L. C. Fu, "Mobile agent based integrated control architecture for home automation system," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, vol. 4, Sep. 28–Oct. 2, 2004, pp. 3668–3673.
[7] J. J. Yoo and D. I. Lee, "Scalable home network interaction model based on mobile agents," in *Proc. 1st IEEE Int. Conf. Pervasive Comput. Commun. (PerCom 2003)*, Mar. 23–26, 2003, pp. 543–546.
[8] X. Li and W. Zhang, "The design and implementation of home network system using OSGi compliant middleware," *IEEE Trans. Consum. Electron.*, vol. 50, no. 2, pp. 528–534, May 2004.

[9] H. Ishikawa, Y. Ogata, K. Adachi, and T. Nakajima, "Building smart appliance integration middleware on the OSGi framework," in *Proc. 7th IEEE Int. Symp. Object-Oriented Real-Time Distrib. Comput.*, May 2004, pp. 139–146.
[10] D. Valtchev and I. Frankov, "Service gateway architecture for a smart home," *IEEE Commun. Mag.*, vol. 40, no. 4, pp. 126–132, Apr. 2002.
[11] C. Lee, D. Nordstedt, and S. Helal, "Enabling smart spaces with OSGi," *IEEE Pervasive Comput.*, vol. 2, no. 3, pp. 89–94, Jul.–Sep. 2003.
[12] T. Gu, H. K. Pung, and D. Q. Zhang, "Toward an OSGi-based infrastructure for context-aware applications," *IEEE Pervasive Comput.*, vol. 3, no. 4, pp. 66–74, Oct.–Dec. 2004.
[13] *OSGi technical whitepaper* [Online]. Available: http://www.osgi.org/documents/
[14] J. White, "Mobile agents white paper," General Magic, Inc.. Sunnyvale, CA, White Paper, 1996.
[15] J. E. White, C. S. Helgeson, and D. A. Steedman, "System and method for distributed computation based upon the movement, execution, and interaction of processes in a network," U.S. Pat. Off., Washington, DC, U.S. Patent 5,603,031, 1997.
[16] J. Baumann, F. Hohl, K. Rothermel, M. Schwehm, and M. Straber, "Mole 3.0: A middleware for Java-based mobile software agents," in *Proc. Middleware 1998: IFIP Int. Conf. Distrib. Syst. Platforms Open Distrib. Process.*, 1998, pp. 355–372.
[17] D. Johansen, "Mobile agent applicability," in *Proc. 2nd Int. Workshop Mobile Agents*, Sep. 1998, pp. 80–98.
[18] *Aglets* [Online]. Available: http://aglets.sourceforge.net/
[19] P. Braun and W. Rossak, *Mobile Agents—Basic Concepts, Mobility Models, and the Tracy Toolkit*. San Mateo, CA: Morgan Kaufmann, 2005.
[20] D. B. Lange and M. Oshima, *Programming and Deploying Java Mobile Agents with Agelets*. Reading, MA: Addison-Wesley, 1998.
[21] *PABADIS project*, [Online]. Available: http://www.pabadis.org/
[22] Object Management Group. *Mobile agent system interoperability facilities specification* [Online]. Available: http://www.omg.org/docs/orbos/97-10-05.pdf
[23] FIPA Agent Management Support for Mobility Specification, *FIPA Specifications 00087, Foundation for Intelligent Physical Agents*, [Online]. Available: http://www.fipa.org/specs/fipa00087/
[24] N. Minar and M. Hedlund, "A network of peers: Peer-to-peer models through the history of the Internet," *Peer-To-Peer: Harnessing the Power of Disruptive Technologies*. Sebastopol, CA: O'Reilly, Mar. 2001.
[25] L. M. Chen, C. L. Wu, and L. C. Fu, "Automatic personal preference learning system in intelligent e-home," Presented at 8th Int. Conf. Autom. Technol., Taichung, Taiwan, May 2005.
[26] M. Kolberg, E. H. Magill, and M. Wilson, "Compatibility issues between services supporting networked appliances," *IEEE Commun. Mag.*, vol. 41, no. 11, pp. 136–147, Nov. 2003.
[27] M. Wilson, E. H. Magill, and M. Kolberg, "An online approach for the service interaction problem in home automation," in *Proc. 2nd IEEE Consum. Commun. Netw. Conf.*, Jan. 3–6, 2005, pp. 251–256.
[28] *XML Protocol. W3C Recommendation* [Online]. Available: http://www.w3.org/2000/xp/Group/
[29] *J2EE* [Online]. Available: http://java.sun.com/j2ee/
[30] *ECMA-262* [Online]. Available: http://www.ecma-international.org/
[31] *Mozilla java script project* [Online]. Available: http://www.mozilla.org/js/
[32] T. Howes, *A String Representation of LDAP Search Filters*, IETF RFC 1960, 1996.
[33] *Knopflerfish 1.3.4* [Online]. Available: http://www.knopflerfish.org/
[34] *ActiveMQ*, [Online]. Available: http://www.activemq.org/
[35] S. Moyer, D. Marples, and S. Tsang, "A protocol for wide area, secure networked appliance communication," *IEEE Commun. Mag.*, vol. 39, no. 10, pp. 52–59, Oct. 2001.
[36] S. Moyer, D. Maples, S. Tsang, and A. Ghosh, "Service portability of network appliances," *IEEE Commun. Mag.*, vol. 40, no. 1, pp. 116–121, Jan. 2000.
[37] M. Handley, *SIP: Session Initiation Protocol*, IETF RFC 2543, Mar. 1999.
[38] OASIS. *Organization for the Advancement of Structured Information Standards*. [Online]. Available: http://www.oasis-open.org/
[39] *OWL web ontology language overview* [Online]. Available: http://www.w3.org/TR/owl-features/
[40] *Security assertion markup language* [Online]. Available: http://www.oasis-open.org/specs/index.php
[41] *eXtensible access control markup language* [Online]. Available: http://www.oasis-open.org/specs/index.php
[42] F-Y. Wang and C.-H. Wang, "Agent-based control systems for operation and management of intelligent network-enabled devices," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, vol. 5, Oct. 2003, pp. 5028–5033.
[43] F.-Y. Wang, "Agent-based control for fuzzy behavior programming in robotic excavation," *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 4, pp. 540–548, Aug. 2004.
[44] ——, "Agent-based control for networked traffic management systems," *IEEE Intell. Syst.*, vol. 20, no. 5, pp. 92–96, Sep./Oct. 2005.
[45] F.-Y. Wang and G. N. Saridis, "A coordination theory for intelligent machines," *Automatica*, vol. 26, no. 5, pp. 833–844, 1990.

Chao-Lin Wu (M'03) received the B.S. degree in industrial technology education from the National Taiwan Normal University, Taipei, Taiwan, R.O.C., in 1996. Currently, he is working toward the Ph.D. degree at the National Taiwan University, Taipei.

His research interests include smart homes, smart environments, intelligent spaces, context-aware technologies, intelligent agents, and topics related to them.

Chun-Feng Liao (M'06) received the B.S. and M.S. degrees in computer science from the National Cheng-Chi University, Taipei, Taiwan, R.O.C., in 1998 and 2004, respectively. Currently, he is working toward the Ph.D. degree at the National Taiwan University, Taipei.

His research interests include intelligent systems, context-aware middleware, and service-oriented software architecture in smart living spaces.

Li-Chen Fu (M'84–SM'94–F'04) received the B.S. degree from the National Taiwan University, Taipei, Taiwan, R.O.C., in 1981, and the M.S. and Ph.D. degrees from the University of California, Berkeley, in 1985 and 1987, respectively.

Since 1987, he has been a Faculty Member in the Department of Electrical Engineering and the Department of Computer Science and Information Engineering, National Taiwan University, where he currently is a Professor. His research interests include robotics, flexible manufacturing systems scheduling, shop floor control, home automation, visual detection and tracking, e-commerce, and control theory and applications.

He is a Senior Member of the IEEE Robotics and Automation Society and the IEEE Automatic Control Society.