國立交通大學

資訊科學系

碩 士 論 文

高 擴 充 性 之 家 電 控 制 平 台

A High Extensible Home Appliance Control Platform

研 究 生：林慧雯

指導教授：袁賢銘　教授

中 華 民 國　九 十 四　年 六 月

高擴充性之家電控制平台

研究生: 林慧雯　　　　指導教授: 袁賢銘

國立交通大學資訊科學研究所

摘要

因為網路的普及，越來越多家庭中的裝置支援連接網路的能力，使用者無論在何處，都可以透過網路控制家裡的裝置，我們將這類的裝置稱為資訊家電。目前資訊家電的應用程式多數針對特定的裝置開發，並且執行在特定的平台；這樣的開發方式使得開發程式的人員必須熟悉各種不同的網路協定(例如 JINI、UPNP)和執行平台(例如 J2ME、.NET Framework)。

為了解決上述的問題，我們提出一個整合 OSGi 平台之高擴充性系統(IAC Platform)，並定義抽象化描述服務的機制，將各種不同網路中的家電裝置服務轉化為 OSGi 服務，並提供一個統一的存取介面，使程式開發人員得以透過 OSGi 標準找尋並使用服務。以 ART 架構為開發及執行的平台可使本系統運行之程式支援不同平台的手持裝置，解決特定應用程式只能在特定平台執行的不便。

IAC Platform 不只整合了家庭網路協定與執行環境，並且提供了撰寫一次應用程式就可以支援各種新服務與新協定的環境。

A High Extensible Home Appliance Control Platform

Student: Hui-wen Lin　　　　　　Advisor: Shyan-Ming Yuan

Department of Computer and Information Science

National Chiao Tung University

Abstract

Since the population of the network technology, more and more home appliances supports the capability of network connection. It means that the house owners could interact with home appliances wherever they are. Nowadays most interacting applications are developed for particular protocols (UPNP, JINI), and executed on special environment (J2ME, .NET Platform).

To solve the problems mentioned above, we proposed a high extensible Platform named IAC Platform. It defines the descriptor that abstracts the services, and transforms services that are in varied protocols into OSGi services. It also provides a universal interface so that the developers could write interacting applications with the same procedure. Besides, we take use of ART Platform to improve the adaptive capability of the platform.

IAC Platform not only integrates the network protocols and execution environments, but also makes the applications support new services and new protocols without be rewritten.

Acknowledgement

首先我要感謝袁賢銘教授兩年來的指導和教誨，讓我所學習的學問更上一層樓；也感謝博士班學長，尤其是葉秉哲、邱繼弘和鄭明俊，謝謝你們這兩年對我的訓練和協助，讓我可以順利完成這篇論文；此外還要感謝蕭存喻、吳瑞祥、高子漢，謝謝你們給我的種種建議，增加此篇論文的思考廣度。

謝謝文如，很幸運可以和妳同學兩年互相勉勵和一起學習；謝謝俊元，感謝你的幫忙；謝謝倫武，那些一起在實驗室奮鬥的日子很值得回憶；以及所有實驗室的其他同學和學弟，謝謝你們一直以來的照顧，協助解決許多課業上的難題，爲實驗室帶來熱絡的研究風氣。

還有我的家人，姝萍、佩瑩和宗勳，謝謝你們平時的鼓勵，以及爲我解決許多生活上的煩惱，讓我可以順利完成學業；以及我的高中好友們，曉蓉、珈倩、佳雯、以萱、乃瑜、怡瑢、貞瑩、佳玲、宛凌，妳們就像是我的姊妹一樣，總是給我完全的支持，謝謝。

最後，我要將這篇論文，獻給我的父母，感謝你們讓我來到這個世界，並且耐心的付出以及細心的教誨，讓我得以完成碩士學業；我會更加努力，面對未來的各項挑戰。

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1. Preface

Controlling the home information appliances should go through many steps. First, the appliances must be connected into the home networks. Then, the application developers have to get the service references provided by service providers, and then they construct the applications to control these appliances. After they finish the developing, the applications are installed into the mobile device, such as PDA and cell phone. At last the house owner could control the appliances with his/her mobile device. However, it is much harder than we think to develop an application to control the home appliance, because the appliance may be connected into home network with varied network protocols, such as JINI [1], UPNP [2], X10, and so forth. Moreover, the application may only be executed correctly on a certain platform, like MIDP platform, while the house owner's mobile device may support another execution platform. To overcome these problems, we have designed a platform, named IAC Platform, which simplifies the procedure of developing applications that control the home appliances.

1.2. Motivation

In these days, the use of information appliances is more and more popular. The appliances could be controlled by not only a certain control box, but the controlling applications that are executed on mobile devices. To take a simple example, the house owner could turn on the air conditioner by his/her cell phone. Further, there are some advanced applications that could manage the interoperability between appliances, for example, when a television in the living room is turned on, the residential gateway would detects the situation and notify the controller of the light in the living room to turn it on.

It is not easy to access all appliances in the house or office with a simple application, because there is no standard protocol for home network now. It means that the home appliances may be connected to different network systems, and the developers have to access these appliances by different protocols. Therefore, we have to design a platform that integrates the services in heterogeneous networks, so that the developers could access them by the unified approach.

We also believe that the house owner would want to control the home appliances with his/her own mobile device, not a particular controller device. To achieve this goal, we need find a solution to handle the adaptive issue between varied execution environments.

1.3. Research Objectives

There are three objectives in IAC Platform: integrating the home networks, improving adaptation ability of the applications, and making the platform extensible to the new protocols and new services.

Integrating the home networks

There are plenty of home network protocols nowadays, while the connection and interaction mechanism of each protocol is quite different. It means that there is no unified approach to access the appliances which are in different network systems. Thus, if a programmer wants to develop applications to control the information appliances, he/she has to be familiar with the every protocol, and implements with the corresponding protocol. It is observable that developing applications with this way will be hard and waste much time. Thus, integrating different home network protocols is a very important issue in IAC Platform.

Improving adaptation ability of the application

On the other side, there are many choices when executing application on mobile device, such like J2ME platform, .NET framework, HTML browser, and WAP browser. In order to let an application be executed on varied platforms, developers have to spend lots of time porting the program to each execution environment. Thus, we want to design a mechanism, which improves the adaptive capability of the program and reduces the porting effort.

Making the platform extensible

The platform is designed to integrate not only the existent home network systems, but the protocols which may be proposed in the future. For the point of programmers, they need not learn the new protocol, but develop applications with procedures that are defined in IAC Platform.

Since the programmers develop applications without considering network protocols, it is possible that the same application could be used to control varied services.

It could be concluded, from what has been said above, that we try to design a platform that has no limit on two sides – home network protocol and execution environment. The platform simplifies the development procedure of the applications. With the platform, developers only have to focus on the program logic. That is to say, developers should not to consider what kind of network protocol the appliance uses and which kind of execution environment the application suits. On the contrary, it is handled by our system, IAC Platform.

With the integration of development environment, the IAC Platform is extensible because it is suitable to new added service and protocol without much modification.

1.4. Organization

In Chapter 2, we introduce the technologies that are used in IAC Platform, and discuss the related research. In Chapter 3, we illustrate the architecture of IAC Platform and the components in it. In Chapter 4, we discuss the design issues of each component, and how these components cooperate to carry out our objectives. In Chapter 5, we turn to developers' perspective. We show that developer could develop an application easily with help of IAC platform. In Chapter 6, we give the conclusions and future works.

Chapter 2 Background

In this chapter, we make a brief introduction of the home network systems which are popular nowadays. We also introduce OSGi Service Platform, which is used to integrate different home network systems, and ART Platform. Besides, related work would also be discussed.

2.1. Home Network System

In the past, the use of the computer network system is mainly in the companies and for the business usage. Nowadays, as the price of the computer is going down and the popularity of the embedded system is growing up, it is often seen that there are more than two systems in a house or an office. These systems may have to share files and peripherals, such as a printer. The requirement of sharing the resources forms the concept of the home network system, which connects shared resources in a small local area. Through the home network system, the devices in the house could communicate with each other.

In general, the bulk of the home network systems use the Ethernet as communication protocol. However, the general users of the home network system may not be familiar with network protocol. Thus, there are some advanced protocols, these advanced protocols need less setting skill, and support the capability of plug-and-play, such as JINI, UPNP, X10; these protocols are the most popular home network systems recently..

2.1.1. JINI Network Technology

JINI network technology, which is based on Java technology, is a distributed communication system. It is an open architecture that enables developers to create network-centric services -- whether implemented in hardware or software -- that are highly adaptive to change. JINI technology can be used to build adaptive networks that are scalable, evolvable and flexible as typically required in dynamic computing environments. The characteristics of the JINI technology are as follows.

1. **Support Plug-and-Play mechanism for network service.**
2. **Provide toolkit for developers to build strong distributed systems.**
3. **Provide an architecture which is based on services.**

The architecture of the JINI technology is indicated by Figure 2-1, and the procedures of finding services could be divided into three steps: Discovery, Join, and Lookup.

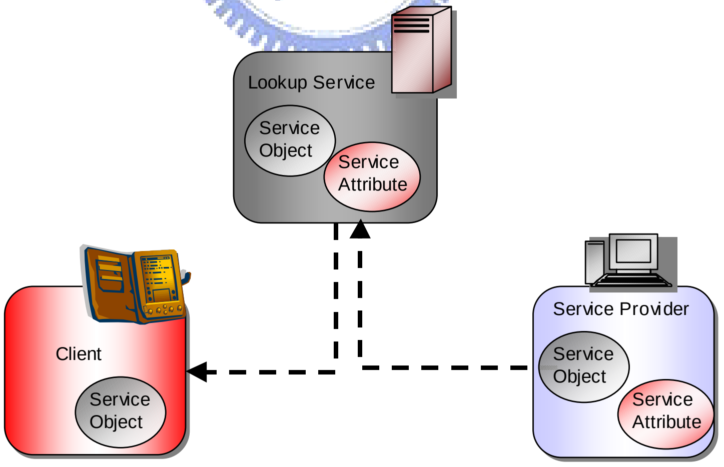


Figure 2-1 The Architecture of JINI

- **Discovery Protocol**

For service providers and clients, the first step of joining JINI network is to search for

Lookup Service, and get the reference of the Lookup Service.

- **Join Protocol**

After a service provider finds the Lookup Service, it registers itself and then uploads the service proxy object to the Lookup Service. It is so called “Join” step in JINI technology.

- **Lookup Protocol**

In the architecture of JINI technology, the client does not connect to service provider and take use of service directly. Instead, it connects to Lookup Service, and search for matched services.

Relying on the above steps, clients could find the required service, and download the service proxy object from the Lookup Service. Then, the clients execute the proxy object directly, or communicate with remote service by the object.

2.1.2.UPnP Technology

UPnP technology is a distributed, open networking architecture that employs TCP/IP and other Internet technologies to enable seamless proximity networking, in addition to control and data transfer among networked devices in the home, office, and public spaces.

The most obvious difference between UPnP and JINI technology is that there is no Lookup Service in the architecture of UPnP technology. The client communicates with the service provider directly. In addition, UPnP technology calls services with remote procedure call, while JINI technology does it with proxy objects.

The processes of searching and controlling services for UPnP technology are explained below.

- **Discovery**

The service provider broadcasts messages, which include the information of the services, such as device name, manufacturer, and etc. On the other hand, the client could send message to search for services. If the service provider gets the query message and held the required services, it would send response to the client with the service information.

- **Description**

After the client finds out the required service, the service provider would send the client a XML document which contains the detail of the service. The content of the document includes the information of devices, services, actions, and etc.

- **Control**

The client sends the action message to execute the action of the service. After finishing the action command, the service provider would send the execution result to the client.

2.1.3.X10 Technology

X10 technology sends signals through home power line. The plug of the home appliance is put in the X10 module socket, and X10 module is connected to home power system. Then we could send signals to control the power state of the appliances.

Considering the above three types of home network protocols, we get the conclusion that there are many different mechanisms in varied protocols. Therefore, it would be a big challenge for programmers to develop applications to control home appliances with different protocols separately.

2.2. Integration of Home Network

2.2.1. OSGi Service Platform

In order to integrate the different home network systems, we chose OSGi Service Platform as the middleware that helps to collect all the services in varied protocols to the same environment.

The OSGi™ specifications [3] define a standardized, component oriented, computing environment for networked services. The OSGi Service Platform provides the capability to manage the life cycle of the software components. Software components can be installed, updated, or removed dynamically. Software components are libraries or applications that can dynamically discover and use other components. The OSGi Alliance has developed many standard component interfaces that are available from common functions like HTTP servers, configuration, logging, security, user administration, XML, and any more. In OSGi 3 specifications, service interface is included. It defines the service interfaces of UPnP and JINI technology to transform these two types of services into the OSGi services. Thus, developers could communicate with the services using OSGi protocol, instead of accessing these services from networks directly. The architecture of the OSGi Service Platform is indicated by Figure 2-2.

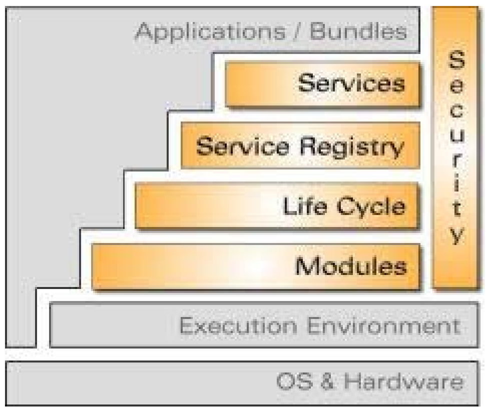


Figure 2-2 The Architecture of OSGi Service Platform

2.3. ART (Adaptive Remote Terminal) Platform

As the computing ability of the mobile devices is poor in the past, we had designed a platform to solve this kind of problem. Adaptive Remote Terminal Platform [4] is a client-server model system, and clients communicate with servers by means of an asynchronous message-delivery mechanism. Figure 3-2 helps to understand the idea.

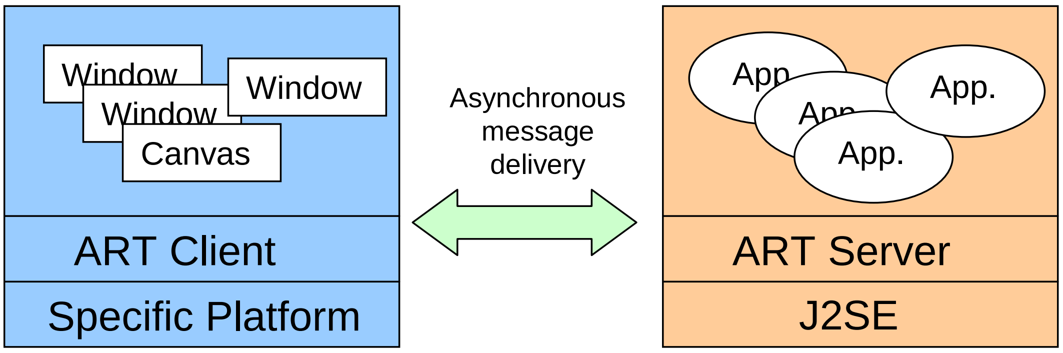


Figure 2-3 The ART Platform

ART has two basic characteristics: "adaptation" and "remote terminal" (remote control and terminal display). To achieve the goals, it separates the UI and the program logic. ART server is responsible for executing program logic, while the ART client is in charge of displaying UI.

The UI of ART platform is described by XUL file, which follows the XML document format. When ART applications are executed on varied platform, ART Platform would generate fit UI. Thus, making use of ART Platform to develop the applications, developers need not consider the execution environment of the mobile device. It means that developers only have to write program once, ART Platform would make the ART application adaptive and suitable for varied execution platform.

2.4. Related work

In the present day, the related research could be divided into three parts: integration of the varied home network systems, the representation capability of OSGi Service Platform on mobile devices, and the solutions of controlling the home appliances.

In aspect of integrating home networks, nowadays most solutions utilize proxy to connect two different home network protocols. Taking such solution, the extensibility of the system is very poor, because it is very hard to put in the third protocol [5-7].

OSGi Service Platform provides a simple HTTP server; it means that the developers could design UI by writing Servlet code. Though almost all the mobile devices support web browser, writing Servlet code is not an efficient approach for designing UI. Besides, the HTML protocol is weak in interaction, because it is one-way client-server model. Only after a client sends requests, the server could send responses back. The server could not send message to the client actively.

One final point is the research on controlling the home appliances. The published result is all limited on a particular home network protocol and execution environment [8-11]. There is no research for two-way opened development environment -- a platform that makes the applications be executed on any mobile device, and control the appliances which are on varied network systems.

Chapter 3 System Architecture

3.1. Overview

First of all, we have to explain the scenario about how to use our system. Taking some simple examples, people may forget to turn off the light when leaving house and want to turn it off with remote controlling application. They may also want to monitor the state of TV when they are in office to see whether children watch TV all the time. Furthermore, when house owner goes traveling for a long time, they may hope to control the appliances as they are at home, so that the thieves will not know that there is no person in the house.

We all know that mobile users could control the household appliances by their mobile devices, such as PDA, cell phone. The devices connect to the home gateway through wireless network. The gateway manages all the appliances at home, including UPNP TV, JINI refrigerator, X10 light, and so on. The scenario is showed as Figure 3-1.

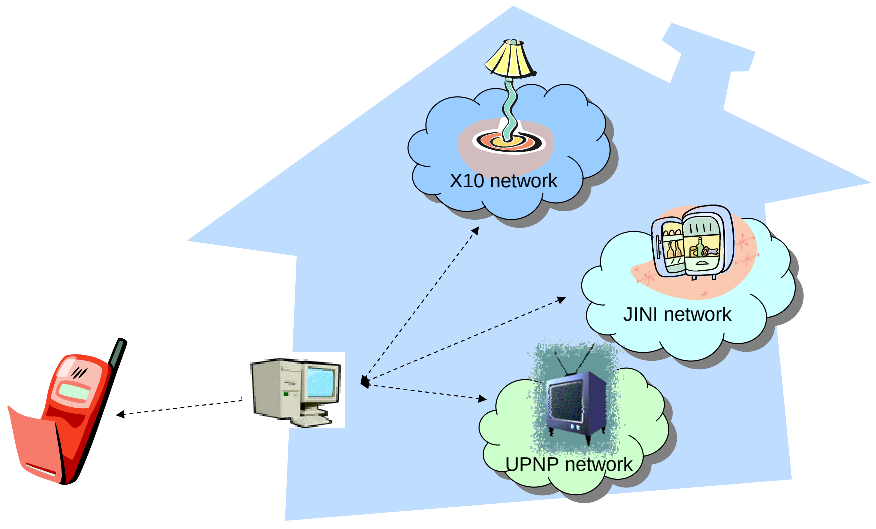


Figure 3-1 The scenario of controlling the home appliances

About the former, the home gateway has to communicate with each network system. It is also a hard work for developers to design controlling applications with these protocols. Thus, we proposed a platform named IAC Platform that provides a universal interface for developers so that they could access the appliances with the same procedure and should not know about what kind of network the appliances belong to. We could say that the IAC platform makes developing the appliance control applications more efficient, because it saves developers' time to study the different specifications of home network protocols.

As there are many kinds of mobile devices in these days, IAC platform also supports varied execution environments. We bring ART Platform into our system because one of the ART Platform's characteristics is "adaptation". Developers only have to write application once, and the ARTApp would be ported to different execution environments automatically.

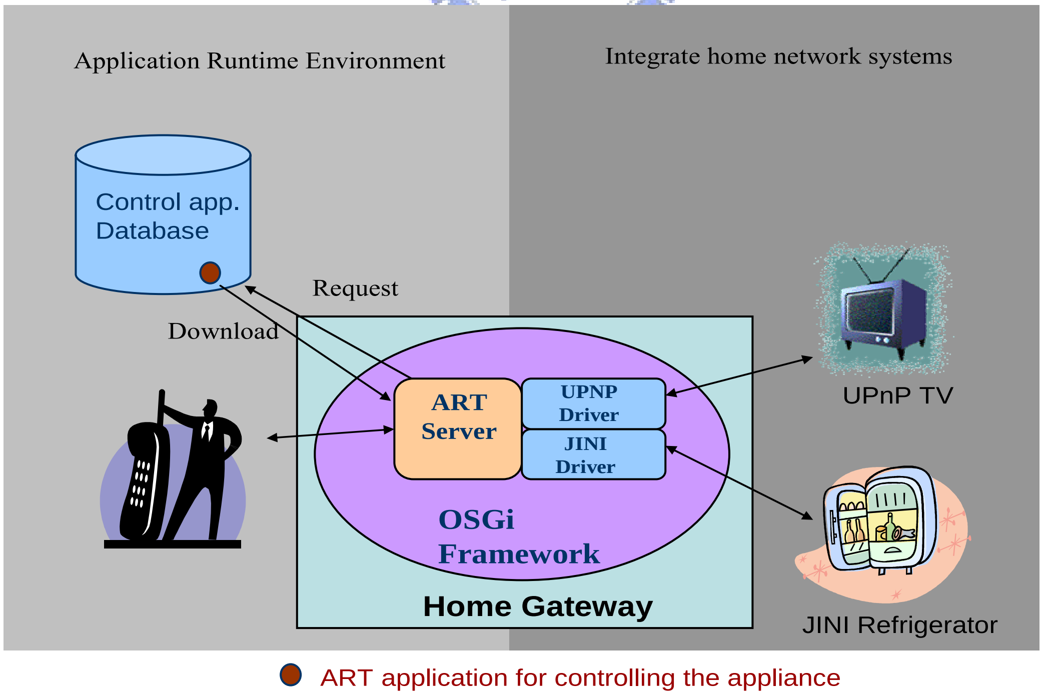


Figure 3-2 The architecture of IAC Platform

The simple architecture of IAC Platform is indicated as Figure 3-2. When a new appliance is detected by gateway, which installs OSGi Service Platform to integrate services, the service driver would transform the service provided by the appliance into an OSGi service. Then the ARTServer would discover the new OSGi service, and tries to download the ARTApp, which is used to control the appliance, from database. Finally the mobile users could connect to ARTServer with ARTClient and interact with the appliance.

3.2. Roles

We define several terms that will be referred. Since IAC Platform brings ART Platform in, we also introduce some ART operations. They are schematized in Figure 3-3.

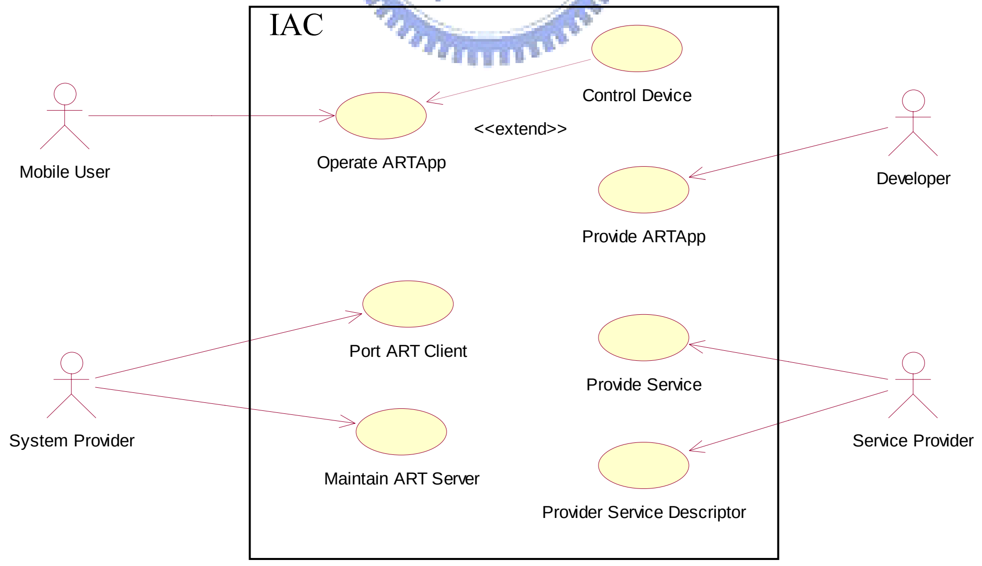


Figure 3-3 Roles of IAC Platform

- Mobile user is house owner who takes use of ARTApps to control the home appliances.

- ARTApps are mobile applications written with ART APIs that we provide. Each of these applications consists of UI and program logic. ARTApps work on the server side and send result to client to show.
- ARTClients are agents which run on mobile devices, and they serve mobile users for interacting with ARTServers.
- ARTServer is to stand for the server body of ART. After an ARTApp comes out, it is put in ARTServer and waits to serve ARTClients.
- Developers we mention below are programmers who write ARTApps to control the household appliances.
- Service providers provide the services and service descriptors.
- Service descriptors are XML documents which describe the information of the services. It is used to abstract services. The detail of the descriptor would be talked in section 5.3.

3.3. Integrate Home Network Systems

One of the most significant contributions of our system is integrating different home network protocols. As the Figure 3-1 indicates, there may be more than two types of network systems at home. If developers write programs for these protocols respectively, they will waste lots of time. Hence, to design a platform which provides a nice development environment, it is clear that integrating home networks is the most important problem that we have to solve.

We divide the integration into two parts: (1) make use of OSGi Service Platform to transform all the services into OSGi service; (2) provide a universal development procedure for developers.

In the first place, we introduce how OSGi service Platform works to transform all the services at networks into OSGi Services. The procedure could be schematized as Figure 3-4. The OSGi specification defines service interfaces, including UPNP and JINI. We could download the implementation of the interface, which is so called UPNP or JINI Base Driver. The Base Driver is composed of importer and exporter. The importer discovers the device in the network and transforms it into an OSGi service; the exporter transforms an OSGi service which is on OSGi Framework into a virtual Device.

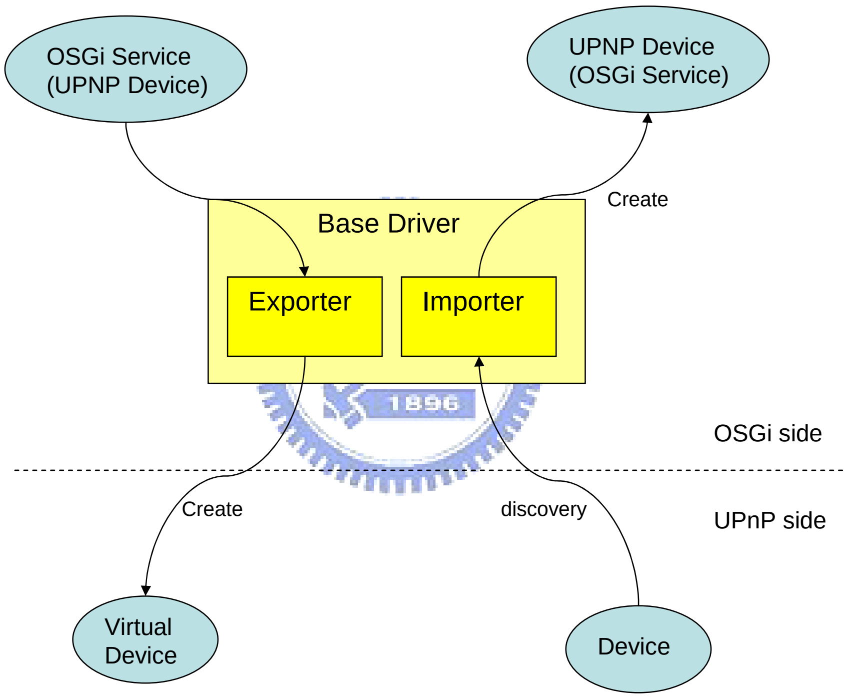


Figure 3-4 The transformation of the UPnP services and OSGi services

In addition to the transformation of services, we also design a universal controller interface named IAController Interface to provide a unified procedure of accessing the OSGi services, and then proceed to control the appliances. With IAController Interface, developers could write programs to interact with the appliances without considering the network protocols. However, it is not easy to design a universal interface since there is much diversity between varied home network protocols. Therefore, we need a descriptor to help us obtaining the information of the service, and then mapping the characteristics of the service into IAController Interface. Figure 3-5 shows this concept. The detail of IAController Interface will be discussed in Section 4.1.1.

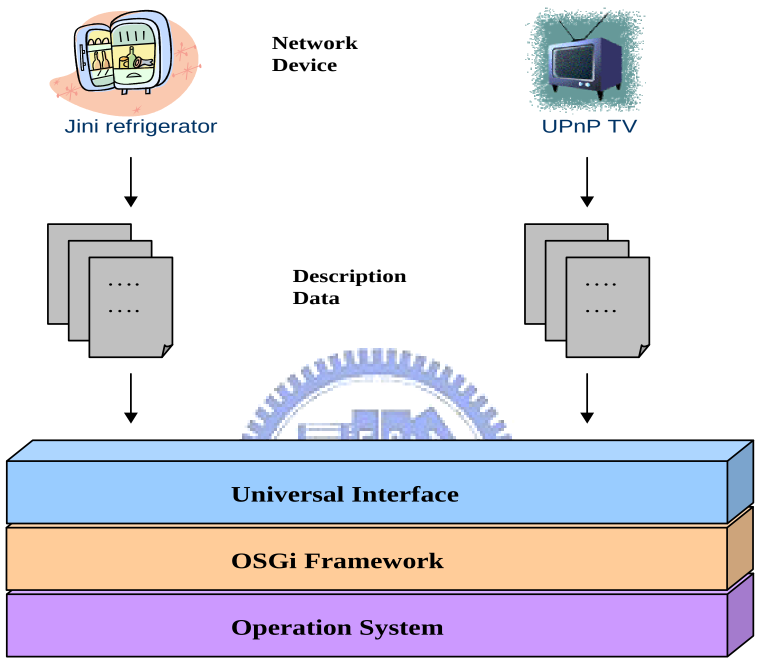


Figure 3-5 Mapping the service to controller interface by descriptors

3.4. Application Runtime Environment

When developers write the controlling applications, using the IAController Interface could avoid communicating with network protocols directly. In the other aspect, to let applications supporting varied execution environment, we chose ART Platform as the development and execution environment. In the Section 2-3, we described the characteristics of ART Platform: ARTApps have adaptive capability on varied execution environments. It could general the suitable code by executing environment.

With ART Platform, developers need not port programs to fit varied execution platforms. However, we have to solve the problem of communication between ART Platform and OSGi Service Platform. The solution we designed is wrapping the ARTServer as an OSGi bundle, so that ART Server becomes a package of the OSGi framework. It could not only export itself to by other bundles, but import other bundles, such as IAController Interface. Thus, ARTApps executing on ARTServer could access the OSGi services, and achieve the goal of controlling the Devices. We could indicate the processes by Figure 3-6.

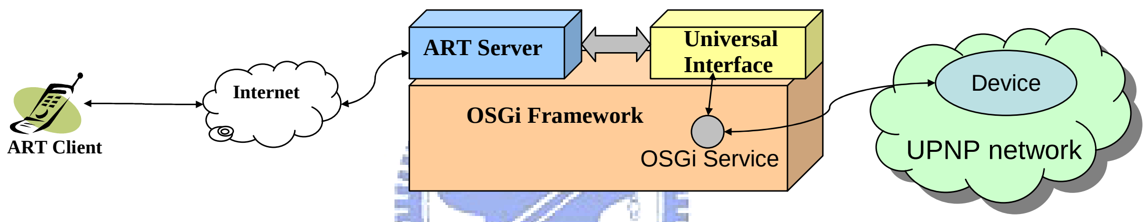
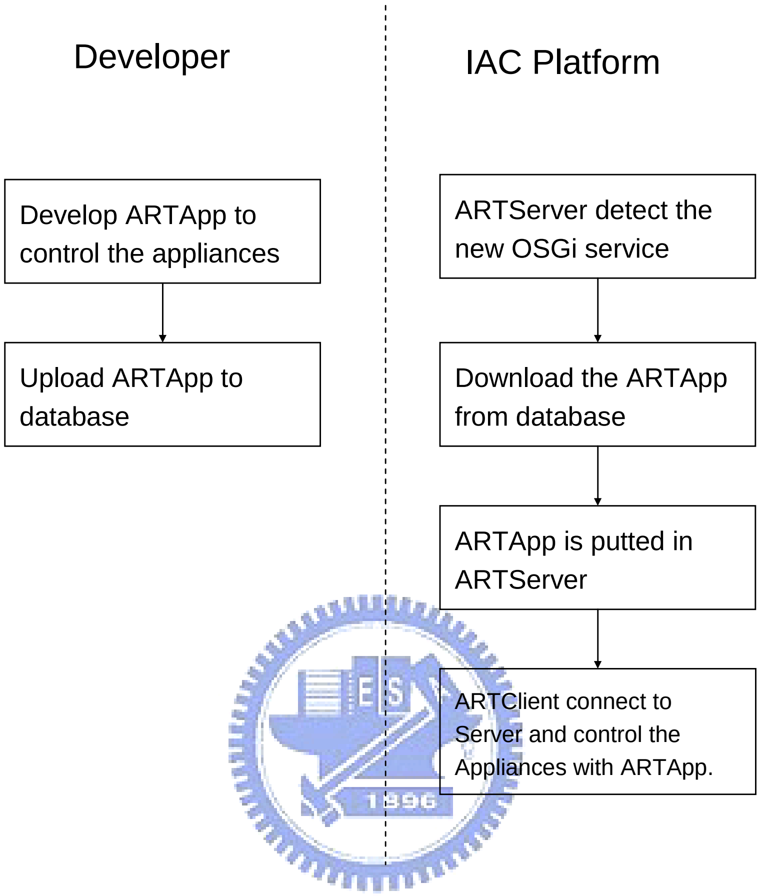


Figure 3-6 The processes of controlling the appliances

So far, we have seen how the ARTClient controls the appliances. Now we discuss how the ARTServer gets correct ARTApps to control the appliances. The issue could be considered in two aspects: developers and IAC platform.

For developers, after developing the ARTApps, they will upload the ARTApps to the database with the service IDs. For IAC platform, the steps of using ARTApp are more complex. First, ARTServer has to listen for the new added OSGi services, and then downloads the corresponding ARTApps from the database with service identities. After the ARTApps are downloaded, they would be putted in ARTServer and wait to serve. When an ARTClient connects to ARTServer, ARTServer will start the ARTApp. Therefore, ARTClient could utilize the ARTApps to control the home appliances. The sequence is showed as Figure 3-7.



Developer
IAC Platform
Develop ARTApp to control the appliances
Upload ARTApp to database
ARTServer detect the new OSGi service
Download the ARTApp from database
ARTApp is putted in ARTServer
ARTClient connect to Server and control the Appliances with ARTApp.

Figure 3-7 The sequence of using ARTApp to control appliances

Chapter 4 Design Issues

In this chapter, we would discuss the design issues of IAC Platform. As mentioned above, the capabilities provided by IAC Platform include integrating varied home network services, and making IAC Platform more flexible and adaptive. There are lots of design issues about IAC platform. For example, in order to integrate home network systems, we have designed a universal controller interface (IAController Interface). With the interface, applications used to interact with home appliances are developed with universal procedure. We have also defined the format of the service descriptor; the descriptor written by service providers abstracts the services so that the actual service in the network could be mapped to abstract service object of IAController Interface. The design concept of descriptor is to help IAC Platform to be extensible for new home network protocol and new service. The design issue of IAController Interface would be discussed in Section 4.1.1, and the content of the descriptor would be explained in Section 4.1.2.

In the aspect of execution platform, we want to choose a platform that support most kinds of ubiquitous devices. The platform must adapt one application to many execution environments.

We also considered the issue that how the components in IAC Platform communicate with each other. At last, we would explain the software management topic in IAC Platform.

4.1. Integration of Home Networks

In order to integrate varied home networks, we have designed a universal interface, which provides a unified procedure to interact with home appliances. Besides, we have defined service descriptors, so that IAC Platform could create abstract service objects according these descriptors. Before introducing detail of design mechanisms, we explain the abstract service and physical service first.

In our design, we separate the services into abstract and physical. All services with varied network protocols would have some common characteristics, and we group these characteristics as the properties of the abstract service. A *Service* object in IAController Interface is designed as a abstract service object. On the other hand, physical service points to the service reference provided by service provider. It may exist in network or be transformed into an OSGi service reference; it is used to control the appliance directly.

4.1.1.A Universal Development Procedure

To provide a universal procedure to control household appliances, we have designed an interface, which is mentioned above as IAController Interface. The interface defines the specification of the abstract objects; it abstracts the service properties, action, and state variable objects and provides mapping information. No matter which original protocol the service belongs to, IAC Platform could map the physical service into the abstract service of the interface. Thus, developers write applications without taking care of the protocol details.

IAController Interface mainly provides the suitable API for programmers to develop interaction application of home appliances. We first define the interfaces of service, action, and state variable, and the relationship of them, and then design how to map these objects to physical services. At last we design a model to handle the event messages. In addition, IAController Interface also regulates the procedures of service discovery and descriptor locating. The issues of descriptor locating and service discovery would be taken up in Section 4.1.1.1.

IAController Interface is used to integrate services. To design flexible interfaces, we have studied several network protocols first, focusing on the similarity of capability and the difference of their implementation. The steps of accessing the home network service could be classified into four parts.

- **Discovering the required Service:**

  Search for the desired service by matching the attributes or capabilities of the service.

- **Getting service information:**

  The service information contains action details, state variable name, and etc.

- **Invoking action:**

  Clients send command to execute functions.

- **Event model:**

  Service sends notification to clients actively, the major goal is to inform the change of the service state.

According to the conclusions we got above, we have designed the class diagram of IAController Interface indicated as Figure 5-1. In the beginning, developers could use *Discovery* object to search for services, and get the required abstract service objects. The abstract service object usually contains two lists, ActionList and StateVariableList, which include *Action* and *StateVariable* objects. These two types of objects are also abstract objects, described in service descriptors. Developers could use above three objects (*Service*, *Action*, *StateVariable*) to invoke basic operations, as the steps described in Section 5.2.2. Now we discuss the concept of each interface.

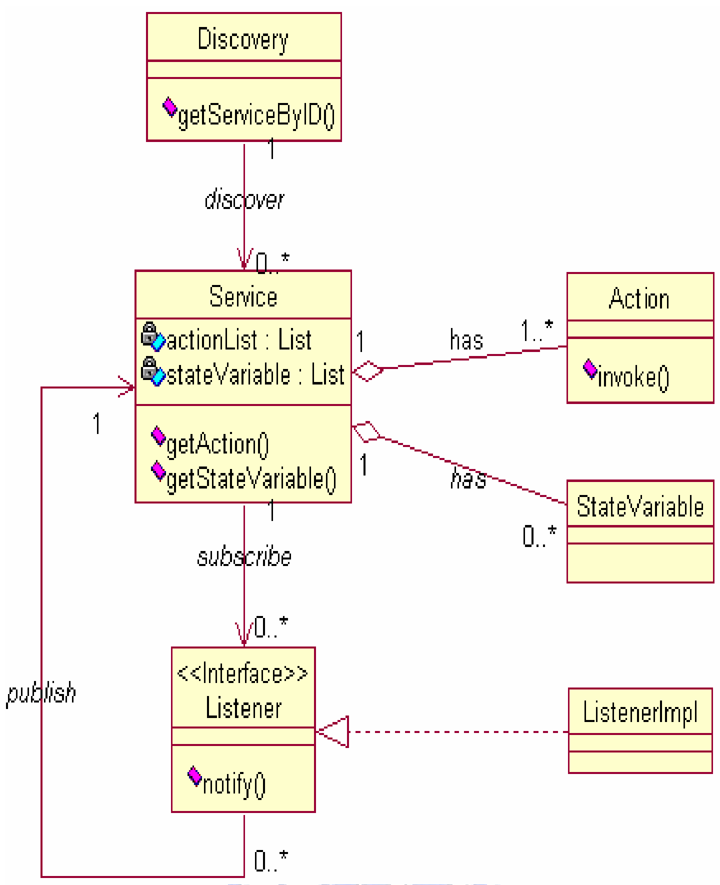


Figure 4-1 Class diagram of IAController Interface

4.1.1.1. Discovery

Now we come to the issues about steps of service discovery. First, we get the service descriptor with descriptor locator. After obtaining the descriptor, IAC Platform would parse it and produce a abstract service object. The procedure of service discovery is illustrated as Figure 4-2.

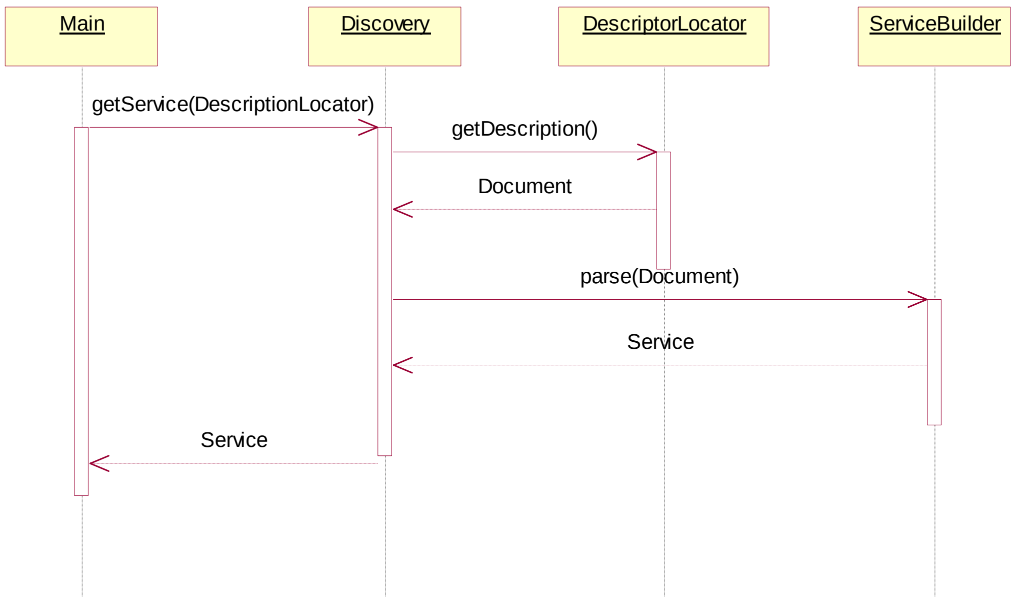


Figure 4-2 Sequence diagram of discovering service

There is a method *getService* in class *Discovery*. The method needs a parameter which's type is *DescriptorLocator*; the class *DescriptorLocator* contains a method named *getDescriptor* which return an *org.w3c.dom.Document* object. After getting the *Document* object, developers could use class *ServiceBuilder* to parse the service descriptor and create abstract objects, and then deliver the service object as return value of method *getService*.

4.1.1.2. Service

Service object contains the abstract information of service properties, such as service ID, service type, and so on. Bedsides, it also has two lists which contain abstract action and state variable objects. The relationship of Action, StateVariable, and Service is indicated as Figure 4-3.

It deserves to be mentioned that we designed *Service* with an abstract concept. The properties of the service mainly come from the service descriptor. With *Service* objects, developers could design application to control appliances without carrying about the implementation of the protocols. Therefore, it could reduce the development effort for the developers. They could use the unified interface to access all appliance services.

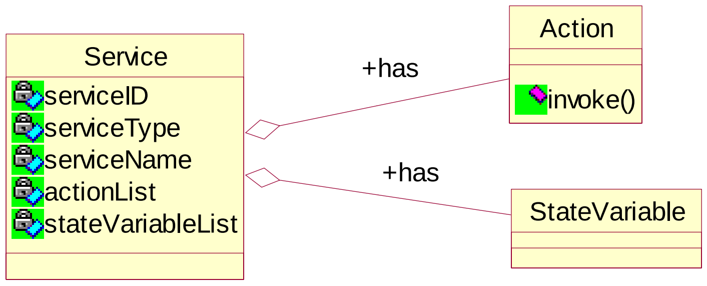


Figure 4-3 The relationship of the Service, Action, and StateVariable

4.1.1.3. Action

The most significant method in class *Action* is *invoke*. It is used to send action command to service in order to interact with home appliances. Before sending an action command, developers have to collect all parameters the action requires, and put these parameters into a *Dictionary* object.

The difference between Service and Action is that the implementations of *Action* would be implemented according to varied protocols. Because the procedure of invoking action depends on protocol mechanisms, each protocol should have its own action implementation. Figure 5-4 indicates the relationship of the interface Action and its implementations.

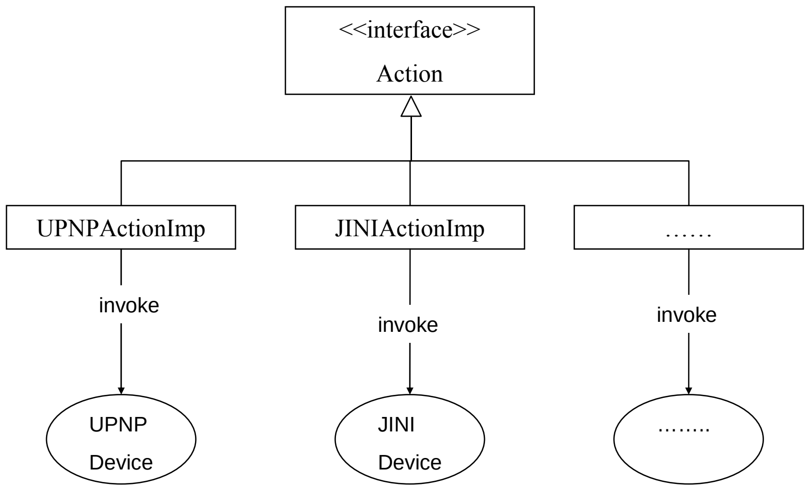


Figure 4-4 The relationship of the Action and its implementation

4.1.1.4. StateVariable

The design of *StateVariable* is similar to *Action*. The main function of *StateVariable* object is getting the current state of the service. Similarly, developers could change the state of the service by setting the state variable value.

4.1.1.5. Listener

We also have to design a mechanism to handle the event messages sent from services. Events are mapped and delivered to ARTApps according to the whiteboard model. The ARTApp interested in receiving the events registers an object implementing the Listener interface (referring to Figure 4-1). A filter could be set to limit the events for which ARTApp is notified.

If an ARTApp is registered with a property named *art.iacontrol.filter* with the value of instance of a *Filter* object, the listener is only notified for matching events.

If an event is generated, the *notify* method is called on all registered ARTApps for which the optional filter matched for that event. If no filter is specified, all events must be delivered. If the filter does not match, the IAC Platform must not notify the ARTApp.

One or multiple events are passed as parameters to *notify (String, Dictionary)* method. The *Dictionary* object holds a pair of *StateVariable* objects that triggered the event and an *Object* for the new value of the state variable.

Event model is an optional mechanism of IAC Platform. It is implemented for those protocols that are suitable for event model.

4.1.2.Mechanism of High Extensible Service Supporting

The main purposes of service descriptor are creating abstract service objects, and defining the configuration properties. Through reasonable mapping, ARTApp could operate the physical service by using the abstract service object which is produced according to descriptor. The concept of configuration is grouping similar operation of different service, so that developers could write an application to control varied appliances.

The general idea of the configuration is indicated as Figure 4-5. Taking light and refrigerator for examples, these two appliances both contain power control operation. The action for light to control the power state is *setPower1 (boolean)*, and for refrigerator is *setPower2 (String)*. In the descriptors of the two services, it is defined clearly what command should be operated when turning on and off the appliances. For light, turning it on should send *setPower1 (true)* command, while turning it off need *setPower1 (false)* command. For refrigerator, setting the power state uses *setPower2 (“on”)* and *setPower2 (“off”).* With the definition of configuration, developers only use *PowerOn* and *PowerOff* operations to set the power state of different appliance, without knowing original action name and parameters

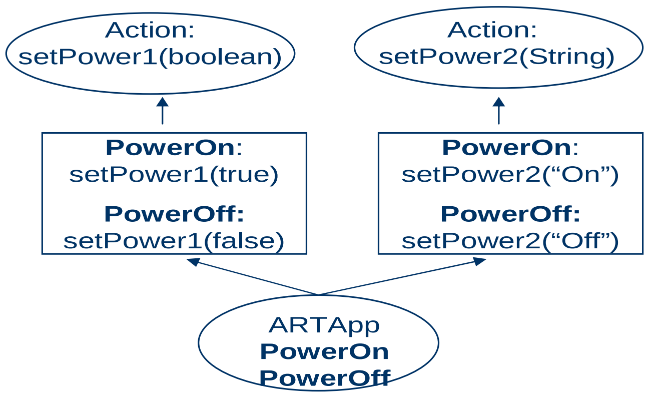


Figure 4-5 The concept of the configuration

There is another advantage of using descriptor; it could increase the extensibility of IAC Platform. Service providers should not rewrite services to fit IAC Platform. Instead, they write descriptors to describe the service information. Similarly, two services with similar operations could use the same ARTApp if they contain descriptors to complete the mapping.

In addition to define the format of the descriptor, it is a design issue to get descriptor, because we hope the access approach of getting descriptor is flexible. The descriptor file may be a local file, a remote file, and even network stream. Thus, we defined the interface *DescriptorLocator*, and the implementation of the *DescriptorLocator* would be replaced to adapt varied descriptor source.

The implementation of the *DescriptorLocator* has to get the descriptor content, and parses it to construct a XML DOM document, which represents the structure of the service, and returns the DOM tree to the originator. After locating the descriptor, the next step is creating abstract service objects for developers. *ServiceBuilder* creates abstract service objects and sets the service attributes with information described in descriptors, such like service id, type, and friendly name. Furthermore, *Action* and *StateVariable* objects are also created and put into *ActionList* and *StateVariableList*.

We design *ServiceBuilder* with a flexible approach, too; if IAC Platform supports a new protocol, and defines the new service descriptor, we only need to extend the *ServiceBuilder* to support the new protocol and service descriptor.

It should be concluded that the steps of producing a abstract service object are as following: (1) locating the descriptor file; (2) parsing the descriptor; (3) creating *Service*, *Action* and *StateVariable* object. (1) and (2) are completed in *DescriptorLocator* implementation, and (3) is in *ServiceBuilder.*

4.1.3. Service Engine

The aim of service engine is charge of mapping the desired OSGi service reference to the abstract service object. Since each protocol has its own properties, it has corresponding service engine, respectively. Taking UPNP for example, UPNP Service Engine catches all devices, actions, state variables from OSGi Framework.

When developers want to access OSGi service reference by abstract service object, service engine will look for matching service reference rapidly. Another advantage is that it could be replaced easily with a newly implementation if we write an efficient service engine, because each engine is written independently. Besides, service engine has cache capability, so that abstract service object could be reused, instead of searching for required service reference every request time.

4.2. Integration of Ubiquitous Devices

Another superiority of IAC Platform is that ARTClient could be executed on varied execution platform, so that house owner could use any mobile device to control the household appliances. To construct smooth communication between OSGi Pltform and ART Platform, we considered all possible approaches and finally decided to wrap the ARTServer into an OSGi bundle and execute on OSGi Platform. Since the design of OSGi Service Platform gives highly capability of bundle management, bundles could interact with each other and share resource. Thus, ARTApp could import package of IAController Interface, and use the operations provided by IAController Interface. Besides, OSGi Platform has excellent ability on managing life cycle of bundles, so the administrator could manage the version and state of ARTServer conveniently. The detail of porting ARTServer would be discussed in Section 5.3.

4.3. Relationship of components in IAC Platform

IAC Platform is an integrated platform. We take many existent standards to compose a home appliance controller platform, and the two most significant platforms of them are OSGi Service Platform and ART Platform.

Before starting explaining how the components interact in IAC Platform, we list the important components in the platform first.

(1) ARTClient

(2) ARTServer

(3) IAController Interface

(4) OSGi Service

(5) Device

The relationship of these components is indicated as Figure 4-6. The communication between ARTClient and ARTServer relies on ART protocol; it uses asynchronous massages to deliver command (client->server) and send display result back (server->client). IAController Interface would be wrapped as an OSGi bundle, so that other bundles in OSGi Framework could use it, such as bundle ARTServer. Thus, the ARTApp executing on ARTServer could use the operations provided by IAController Interface.

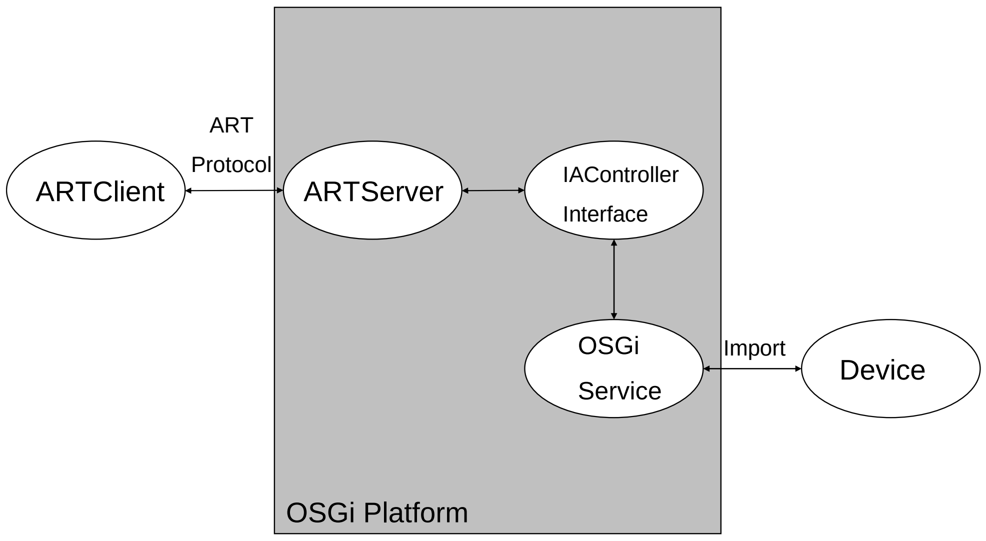


Figure 4-6 The relationship of each component in IAC Platform

IAController Interface plays a key role in IAC Platform, because it bridges the ARTApp and OSGi service reference. It provides developers a universal access approach to access the services. Besides, it searches the required service reference from OSGi Framework by filter description. Therefore, the action and query from ARTApp could be sent to OSGi service reference and get result correctly by help of IAController Interface. Wrapping IAControll Interface in an independent OSGi bundle is convenient to replace the interface with newly implementation in the future.

The last step of controlling appliances is connecting the device service to the OSGi service reference. OSGi Framework is responsible for the transformation. Through the base driver provided by third parties, OSGi Frmawork could transform the appliance services detected in network into OSGi service references. Therefore, the OSGi service reference could be seen as the proxy of the appliance service.

4.4. Software Management

In IAC Platform, we utilize OSGi Service Platform to manage all the components.

There are two components, ARTServer and IAController Interface, wrapped as OSGi bundles and could be replaced dynamically. Furthermore, IAController Interface contains many mechanisms to transform services, such like Service Engine. If IAC Platform supports new protocols or an efficient bundle is adapted to replace old one, OSGi Service Platform provides a perfect solution for updating these bundles dynamically.

Chapter 5 Implementation

After discussing architecture and design issues, we will go into details to implementation. This chapter would introduce the OSGi environment we use; how to write an ARTApp to control appliances; how to write a right service descriptor, and the steps of integrating ART Platoform into OSGi Platform.

5.1. Construct an OSGi Environment

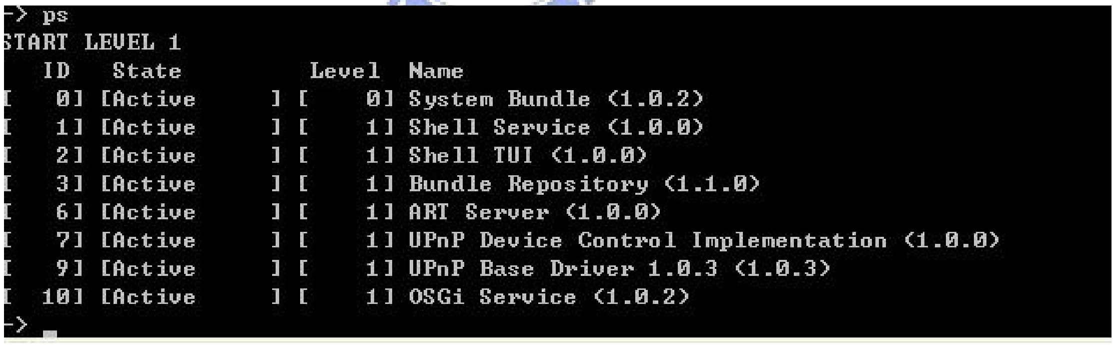
In IAC Platform, we chose OSGi Service Platform as the middleware to integrate network services. Oscar [12] is an open source implementation of the OSGi framework specification, and it is currently compliant with a large portion of the OSGi 3 specifications. Therefore, we could take use of Oscar to construct an OSGi environment. Oscar provides a shell environment, and available command is indicated by Figure 5-1.

```
bundlelevel <level> <id> ... | <id> - set or get bundle start level.
cd [<base-URL>]                     - change or display base URL.
headers [<id> ...]                  - display bundle header properties.
help                                - display shell commands.
install <URL> [<URL> ...]           - install bundle(s).
obr help                            - Oscar bundle repository.
packages [<id> ...]                 - list exported packages.
ps [-l]                             - list installed bundles.
refresh                             - refresh packages.
services [-u] [-a] [<id> ...]       - list registered or used services.
shutdown                            - shutdown Oscar.
```

```
start <id> [<id< <URL> ...]        - start bundle(s).
startlevel [<level>]               - get or set framework start level.
stop <id> [<id> ...]               - stop bundle(s).
uninstall <id> [<id> ...]          - uninstall bundle(s).
update <id> [<URL>]                - update bundle.
version                            - display version of Oscar.
```

Figure 5-1 Available command list of Oscar

Oscar could be executed on embedded system, so it could be installed on a home gateway that does not have powerful capability. After installing Oscar, we could start the OSGi Service Platform and install bundles on it. The bundles used in IAC Platform are ARTServer, IAController Interface, and Base Driver that imports service from network. We use "ps" to see current bundles' state of the OSGi Framework; the result is displayed as Figure 5-2.



```
-> ps
START LEVEL 1
   ID   State         Level  Name
[   0] [Active     ] [    0] System Bundle (1.0.2)
[   1] [Active     ] [    1] Shell Service (1.0.0)
[   2] [Active     ] [    1] Shell TUI (1.0.0)
[   3] [Active     ] [    1] Bundle Repository (1.1.0)
[   6] [Active     ] [    1] ART Server (1.0.0)
[   7] [Active     ] [    1] UPnP Device Control Implementation (1.0.0)
[   9] [Active     ] [    1] UPnP Base Driver 1.0.3 (1.0.3)
[  10] [Active     ] [    1] OSGi Service (1.0.2)
-> _
```

Figure 5-2 Tthe bundle state of OSGi Framwork

5.2. Writing an ARTApp to Control an Appliance

5.2.1. Write an ARTApp

Since IAC Platform chose ART Platform as the development and execution environment, the developers have to learn about how to writing an ARTApp. The steps of developing an ARTApp are described below.

1. State initial UI in XML and save them.
2. Create a new Java file by the name of Main.java, and there must be a class named Main in it.
3. The class Main has to extend from class ARTApp.
4. Developers must implement two method in class Main, *startApp* and *stopApp*.
5. The detailed steps of writing an ARTApp is described in thesis "*An Adptive Mobile Application Development Framework, 2003*".

5.2.2.The procedure of appliance controlling

Then developers start to write the logic part of the application -- controlling the home appliances. Figure 5-3 shows the steps of how an ARTApp gets a service and invokes an action. The sequence begins with ARTApp which sends a request to the *Discovery* object with service ID, and gets the response with a abstract service object. After getting the *Service* object, ARTApp take use of the service object to get the desired action with the action name. At last, ARTApp invokes action according the command which is triggered by ARTClient.

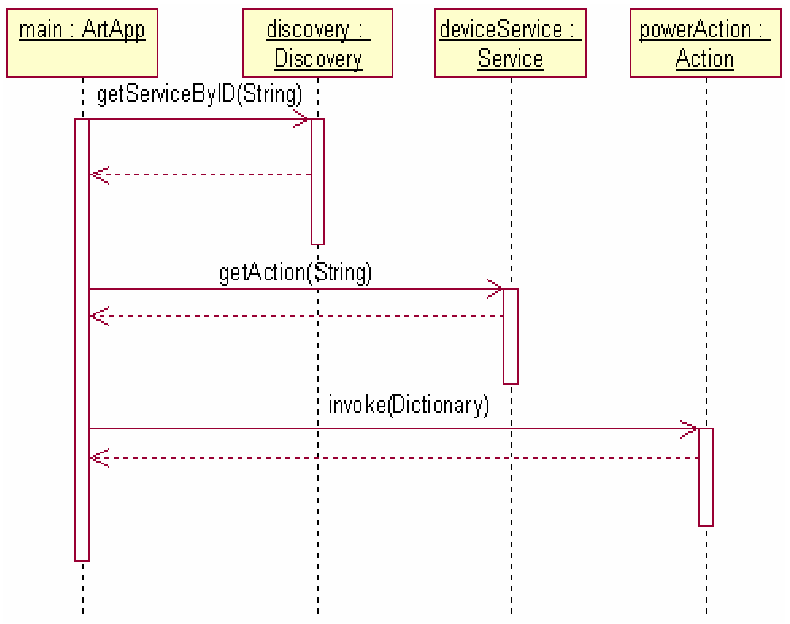


Figure 5-3 The sequence diagram of controlling home appliances

The ARTApp communicates with the abstract service object in IAC Platform, and the abstract service is created according to the service descriptor, so the operation ARTApps provide must be mapped to service descriptor correctly. Taking a simple example, to control a simple light, we write an ARTApp which has two operations, *turnOn* and *turnOff* the light. With the service descriptor, we got the information that there is an action named *SetPower* to control the power state of the light. The *turnOn* method must invoke action *SetPower* with "*true*" parameter, while the *turnOff* method with "*false*" parameter.

Figure 5-4 displays the operation list of the simple Light. If "Power On" is chosen, the UPNP simple Light would be turned on. Otherwise, it would be turned off.

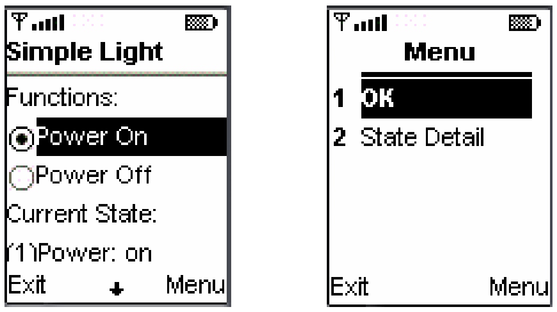


Figure 5-4 The operation list of Simple Light

In this sample, we could observe that developers write ARTApp by following the rules of ART Platform, service descriptor, and IAController Interface, without having knowledge about the network protocols.

5.3. Writing a Descriptor

IAC Platform needs the help of descriptor to gain the service information when detecting a new service reference. The format of the service descriptor is similar to Web Service. The information of the properties, state variables, and actions of the service would be described in the service descriptor.

5.3.1. General case

We take the simple examples to introduce the service descriptor. We choose a simple air conditioner service, which only provides the power control capability. The service has an action named *setPower* and a state variable named *power* with boolean type. The descriptor could be separated into four parts.

In this section we introduce the front three parts; they are mandatory for every descriptor. It is used to abstract the service and map it to the IAController Interface. The first portion contains the service properties such as service ID, type, and etc.

Figure 5-5 indicates this part.

```
<serviceId>DCSLAB-ART-Simple-Airconditoner</serviceId>
<serviceType>upnp</serviceType>
<friendlyName>Simple Airconditoner</friendlyName>
```

Figure 5-5 Service attributes in the descriptor

The second portion defines abstract actions of the service. Developers write ARTApps use the operation provided in this part. Figure 5-6 informs that how many actions developers could invoke. Besides, it also contains the state variable information. The tag *id* (actionId and stateVariableId) is used to map the abstract object to the actual object.

```
<iacDefinition>
    <actionList>
      <action>
        <actionId>art-airconditoner-setPower</actionId>
        <actionName>userSetPower</actionName>
        <argumentList>
          <argument>
            <argumentName>airconditonerPower</argumentName>
            <dataType>boolean</dataType>
            <direction>in</direction>
          </argument>
        </argumentList>
      </action>
    </actionList>
    <stateVariableList>
      <stateVariable>
        <stateVariableId>art-airconditoner-Power</stateVariableId>
        <stateVariableName>userPower</stateVariableName>
        <stateVariableType>boolean</stateVariableType>
      </stateVariable>
    </stateVariableList>
  </iacDefinition>
```

Figure 5-6 The abstract definition in the descriptor

The third portion of the descriptor is about mapping information; it maps the abstract action mentioned above to the actual object, so that the ARTApp could send the command to the actual action object and invoke it. The mapping description of the simple air conditioner is indicated as Figure 5-7. From the description, the action which *id* is *art-sample-power-1* would be map to actual action named *SetPower*, with the service id being *urn:schemas-upnp-org:serviceId:power:1*. The mapping approach of state variable is similar to action.

```
<mapping target="upnp">
    <upnpUdn>uuid:cybergarageAirConDevice</upnpUdn>
    <mapping-actionList>
      <upnp-action targetAction="art-airconditoner-setPower">
        <upnp-serviceId>urn:schemas-upnp-org:serviceId:power:1</upnp-serviceId>
        <upnp-actionName>setPower</upnp-actionName>
        <upnp-argument targetArgument="airconditonerPower">setPower</upnp-argument>
      </upnp-action>
    </mapping-actionList>
    <mapping-stateVariableList>
      <upnp-stateVariable targetStateVariable="art-airconditoner-Power">
        <upnp-serviceId>urn:schemas-upnp-org:serviceId:power:1</upnp-serviceId>
        <upnp-stateVariableName>Power</upnp-stateVariableName>
      </upnp-stateVariable>
    </mapping-stateVariableList>
  </mapping>
```

Figure 5-7 The mapping information of the descriptor

5.3.2.Configuration of Specific Operation

In our design, there is an optional portion in service descriptor. It is used to define specific configuration of the service, such as power control. With this kind of description, one ARTApp could be used by many services, as long as the service descriptors provide the correct configuration. Figure 5-8 shows the configuration information about the power control.

We take power control as the example when explaining the usage of the configuration. It suits for all the appliances having power switch. The description tells the system which abstract action should be used. In this example, the configuration *powerOn* is mapping to *SetPower(true)*,and *powerOff* is mapping to *SetPower(false)*.

```
<configuration>
  <powerService>
    <powerOn>
      <invoked-ationID>art-airconditoner-setPower</invoked-ationID>
      <argument>
        <argumentName>airconditonerPower</argumentName>
        <dataType>booelan</dataType>
        <direction>in</direction>
        <argumentValue>true</argumentValue>
      </argument>
    </powerOn>
    <powerOff>
      <invoked-ationID>art-airconditoner-setPower</invoked-ationID>
      <argument>
        <argumentName>airconditonerPower</argumentName>
        <dataType>boolean</dataType>
        <direction>in</direction>
        <argumentValue>false</argumentValue>
      </argument>
    </powerOff>
  </powerService>
</configuration>
```

Figure 5-8 The configuration definition of the descriptor

5.4. Integrating the ARTServer to an OSGi bundle

There are many advantages to wrap ARTServer as an OSGi bundle. As we discussed before, OSGi Framework is powerful in software management; bundles could share resource with each other, and could be updated dynamically.

A bundle gains access to the OSGi framework using a unique instance of *BundleContext*. In order for a bundle to get its unique bundle context, it must implement the BundleActivator interface; this interface has two methods, *start* and *stop*, that both receive the bundle's context and are called when the bundle is started and stopped, respectively. Figure 5-9 is the source code of *Activator.java*. It is used to produce an ARTServer instance on OSGi Framework. In method *start* an ARTServer is initialed. Figure 5-10 is the manifest file.

```
package com.art.main;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceListener;
import org.osgi.framework.ServiceEvent;

public class Activator
      implements BundleActivator {
   public void start(BundleContext context) {
      System.out.println("Starting ART Server.");
      new ART().init();
   }
   public void stop(BundleContext context) {
       ART.stop();
   }
 }
```

Figure 5-9 Source code of Activator.java

```
Manifest-Version: 1.0
Bundle-Description: ART Server Bundle
Bundle-Name: ART Server
Created-By: DCSLab
Bundle-Activator: com.art.main.ART
Bundle-Classpath: .,lib/xerces.jar,lib/ftp.jar
Bundle-Vendor: DCSLab
Bundle-Version: 1.0.0
```

Figure 5-10 Manifest file of the ARTServer bundle

Then we create an OSGi bundle by wrapping the class files and Manifest.mf to a Jar format file, with the command `jar cfm ARTServer.jar manifest.mf com.`

Chapter 6 Conclusion

6.1. Conclusion

So far, we have described our system, IAC Platform, and how it achieves the objectives mentioned in Chapter 1. The key point of design is on the service descriptor. Through the abstracted service descriptor, developers could write ARTApps to control appliances; the platform could map the original physical service to abstract service and provide a universal development procedure; and most important, the service could configure itself as some kind of device. It means that the service may be controlled by existent ARTApp, instead of writing a new ARTApp. It improves the extensibility of the service supporting. Besides, we bring OSGi Service Platform to integrate varied home networks, and ART Platform to the ubiquitous devices.

The main advantages of using IAC Platform are as follows:

- Service provider need not rewrite service to fit IAC Platform.
- Developers who write application do not consider the network protocols.
- House owner could use his/her familiar ubiquitous device to control appliances.
- The same ARTApp could control varied appliances.
- The descriptor we defined could be extended easily to fit the new type of network and service.

6.2. Comparison

IAC Platform solves many problems of controlling appliances. Thus choosing

IAC Platform as development and execution environment is more suitable than other systems. Table 6-1 is the comparison table.

	IAC Platform	Other System
Integrate Home Network	Using OSGi Framework, the extensible ability is better than others.	Using proxy to transform two kinds of networks, it is complex and not easy to support new protocol.
Integrate Ubiquitous Device	Using ART Platform to solve the adaptation problem. The ARTApps could be executed on many platforms, such like J2ME, .NET, WAP browser, and etc.	The representation of OSGi Framework is similar to Servlet. It is not efficient to write hard code to display the UI.
Extensibility	1. A new network protocol could be included in IAC Platform easily. 2. A new service could be supported by IAC Platform with little effort.	Not support.

Table 6-1 The comparison Table of IAC Platform and other systems

6.3. Future Works

Now we finished the prototype of the IAC Platform, but there are still many jobs that could make the platform more mature. We propose some future works that enhance IAC Platform.

- Support of network protocols

  Now we have finished the implementation of UPNP and X10 network protocols. To make the platform more complete, we must add other protocol, like JINI, as soon as possible.

- Definition of configuration

  Configuration is a significant part of the platform, now we have defined a few, such as power configuration. In the future we will define more kind of configurations. Then, developers would have a better environment when developing application to interact with appliances.

- Optimizing

  To improve the efficiency of IAC Platform, there are some implementations that could be optimized. One example is Service Engine; we could rewrite a Service Engine to increase the speed of searching matching service from OSGi Framework.

Chapter 7 Bibliography

[1] UPnP™ Forum, `http://www.upnp.org/`

[2] Jini.org, `http://www.jini.org/`

[3] OSGi Alliance, `http://www.osgi.org/`

[4] 姚立三, "ART一 可適性的行動應用程式開發平台",國立交通大學電資學院碩士班論文，民 92.

[5] Chau, O.S., Hui, P., and Li, V.O.K., "An Architecture Enabling Bluetooth / JINI Interoperability,"Proc. IEEE PIMRC, Barcelona, Spain, September 2004.

[6] Song Yean Cho, Dae Young Seo, Tai Yun Kim,"Gateway Framework for Home Appliance's Interoperability Based on Heterogeneous Middleware in Residential Networks," Consumer Electronics, 2002. ICCE. 2002 Digest of Technical Papers. International Conference on 18-20 June 2002

[7] J. Allard, V. Chinta, S. Gundala, G. G. Richard III, "JINI Meets UPnP : An Architecture for JINI/UPnP Interoperability," IEEE Computer Society, 2003.

[8] Latvakoski, E.J.; Paakkonen, P., "Remote interaction with networked appliances attached in a mobile personal area network," Communications, 2003. ICC '03. IEEE International Conference, May 2003.

[9] Mariana Nikolova, Frans Meijs and Peter Voorwinden," Remote Mobile Control of Home Appliances," IEEE Transactions on Consumer Electronics, FEBRUARY 2003.

[10] Noriyuki Kushiro, Shigeki Suzuki, Masanori Nakata, Hideki Takahara and Masahiro Inoue," Integrated Residential Gateway Controller for Home Energy Management System engineering," IEEE Transactions on Consumer Electronics, AUGUST 2003.

[11] Zhaohui Ye, Yindong Ji, Shiyuan Yang," Home Automation Network Supporting Plug-and-Play," IEEE Transactions on Consumer Electronics, Vol. 50, No. 1, FEBRUARY 2004.

[12] Oscar Project, http://oscar.objectweb.org/

[13] Domotics Software, http://domoware.isti.cnr.it/