2005

Generic framework for the personal omni-remote controller using M2MI

Chih-Yu Tang

Department of Computer Science

Rochester Institute of Technology

MS Thesis

A Generic Framework for the Personal Omni-Remote Controller Using M2MI

Date: April 13, 2005

Author: Chih-Yu Tang (cxt6239@cs.rit.edu)

Approvals:

_____

Chair: Prof. Alan Kaminsky Date

_____

Reader: Prof. Hans-Peter Bischof Date

_____

Observer: Prof. Ankur Teredesai Date

Abstract

A Generic Framework for the Personal Omni-Remote Controller Using M2MI is a master's thesis outlining a generic framework for the wireless omni-remote controller that controls neighboring appliances by using Many-to-Many Invocation (M2MI). M2MI is an object-oriented abstraction of broadcast communication. First, this paper introduces the history of remote controllers and analyzes omni-remote controller projects made by other researchers in this area, such as the Pebbles PDA project at Carnegie Mellon University and HP's COOLTOWN project. Second, this paper depicts a generic framework of the personal omni-remote controller system including architecture, type hierarchy, and service discovery. In this framework, a module approach and a decentralized dual-mode service discovery scheme are introduced. When users request a certain type of service, their omni-remote controller application will first discover the available appliances in the vicinity and then bring up the corresponding control module for the target appliance. Thus, users can control the appliance through the User Interface of the control module. To join the omni-remote controller system, servers and clients need to follow the type hierarchy convention of the system. Finally, several implementations are given to show the control of different appliances with different capabilities. These appliances include thermostats, TVs with parental control, and washing machines.

Table of Contents

1 Introduction

A Generic Framework for the Personal Omni-Remote Controller Using M2MI is a master's thesis outlining a generic framework for wireless omni-remote controller. The controller controls nearby appliances by using Many-to-Many Invocation (M2MI) [1, 2]. This article first provides the development history of remote controllers and gives a detailed description and analysis of related researches in this field, such as the CMU's Pebbles PDA project [3] and the HP's COOLTOWN project [9]. A summary of these related projects is given at the end of Section 2. Section 3 of this article depicts a generic framework of the personal omni-remote controller system that includes three main components—architecture, type hierarchy, and service discovery. Section 4 is the demonstration part of the personal omni-remote controller system, in which the programming projects are designed to show the control of different appliances with different capabilities. The programming projects include the control of thermostats, TVs with parental control, and washing machines. A user's manual on how to execute and edit the source code is provided in Section 5. Future work such as an agent system for the omni-remote controller system and the security problem is discussed in Section 6. Finally, in Section 7, a conclusion of this research paper is given.

2 Background

Finding a remote controller or getting the right one from a pile of them becomes a MISSION IMPOSSIBLE. Most people remember the moment when they search every corner of the house to look for a remote controller or struggle with picking the right one from a couple of them. Fortunately, the idea of the omni-remote controller was introduced to provide all the remote control functions in one single device. However, the so-called omni-remote controller is not truly universally adaptable due to lack of hardware and software technologies, such as bidirectional communication and cross-platform software. With the advance of new technologies, the idea of the omni-remote controller becomes more of a possibility.

2.1 History

Before the invention of the remote controller, users have had to operate on the built-in, static user interface (UI) of appliances. For example, in order to change the channel, people would have had to use the TV directly. These days we would consider this to be very inconvenient. Because of the remote controller, people are no longer bound to the appliance itself. Wired remote controllers were the first step towards separating an appliance from its controls. However, wired remote controllers are still not that convenient due to the limitation of wires.

During World War I, radio-controlled motorboats were invented for military use. This was the first attempt at designing a controlling appliance without wires. In World War II, radio-controlled bombs were used. Based on this technology, garage door openers and

ultrasonic TV remote controllers, invented by Dr. Robert Adler[1], became popular for domestic purposes after the war. Ultrasonic remote controlling technology began an era of wireless appliance-controlling that lasted for the twenty-five years beginning in 1956. However, the nightmare of piled-up remote controllers also begins.

Several attempts to solve the nightmare by combining remote control functions in one device did not work well until recently. Based on my research in this area, I found that the omni-remote controllers can be categorized into the following three generations.

First Generation Omni-remote Controller

The first generation of the omni-remote controllers is called the Universal Remote Controller (URC). The communication medium in this generation is primarily infrared frequency (IR). The following picture in Figure 2-1-1 is an example of the first generation of omni-remote controllers.

Figure 2-1-1: An example of the first generation universal remote controller retrieved from *http://www.dipol.com.pl/epilot.htm*

A URC is mainly used in controlling entertainment appliances such as TVs, DVD players, and stereos because they are among the most commonly-used household appliances. A URC works by having the codes of the various models of the appliances preinstalled. After buying a URC, users need to configure the controller in order to control an appliance. This is done by entering a predefined appliance code. Then, the URC is set up to emit the corresponding control signal when one of the built-in buttons is clicked on.

This attempt at integrating all of the controlling functions was a milestone in the development history of remote controller. However if the model of an entertainment appliance is newer than the model of a URC, the URC will not work for the appliance because the URC, which is not upgradeable, does not "know" the appliance code. To solve this problem, the user has to replace the old URC with a new one that "knows" the new appliance code.

The UI of a URC is another drawback. A URC's interface, consisting of a number of built-in buttons, typically has all the common control functions of an entertainment

[1] Dr. Robert Adler is best known as the "Father of the TV Remote Control."

appliance. For example, a portion of the buttons is devoted to TV-like appliances while another portion is for VCR-like appliances. Thus, it is harder for a user to find the correct button (out of 50 buttons for example) to do the right job. In addition, a study by Jeffrey Nichols and Brad A. Myers shows that clicking on a wrong button is twice as likely to occur on a built-in control panel as on a touch screen display in their paper "Studying the Use of Handhelds to Control Smart Appliances." [6]

Second Generation Omni-remote Controller

In the second generation of omni-remote controllers, a portable computing device such as a Personal Digital Assistant (PDA) is often used to provide the omni-remote controlling function. The picture in Figure 2-1-2 shows an example of the second generation omni-remote controller.

Figure 2-1-2: An example of the second generation omni-remote controller retrieved from *http://www.pacificneotek.com/omnisw.htm*

Just like the first generation, the second generation omni-remote controller uses IR and performs one-way communication, in which control signals are sent out by a user's request, but there is no feedback from the appliances. However, the second generation is better than the first generation in many ways.

First, a PDA has a touch screen display that is easier for users to use than the built-in buttons of the first generation. Embedded with a large LCD, the PDA provides a Graphic User Interface (GUI), with which a user can set up a preferred interface (control panel) for each appliance. This changes the UI from a fixed-concrete form (in the first generation) into a flexible, user-centered model (in the second generation). Second, a PDA has more complex functions because it is able to learn control signals. For example, it can be trained to recognize which IR is used for a certain appliance function, and thus the trainable PDA truly becomes an omni-remote controller. That is to say, as long as it can be trained, a PDA is able to control all infrared-controlled appliances. When training a PDA, a user needs an appliance's standard remote controller and then assigns a certain function of this controller to a PDA's button. That is, a user points the infrared emitter of the standard controller to the PDA's infrared receiver and clicks on a standard controller's button to emit the control signal to the receiver. The training process is repeated for each desired controlling function.

Third Generation Omni-remote Controller System

Using IR is not the best choice because it can cover only a short range of area and the signal cannot penetrate obstacles. For instance, if the infrared receiver of an appliance is behind an object, the control signals will not be able to reach the appliance.

Just like the second generation, the third generation omni-remote controller has a user-friendly UI and can perform all functions that the second generation can. The difference is that the third generation omni-remote controller performs two-way communication, in which control signals are sent out to appliances and appliances give feedback to controllers.

The third generation controller must act correspondingly based on the state of a target appliance. For example, if a TV is on, the third generation controller will change the state of the power button in the GUI, so pushing the button will turn off the TV. This guarantees that a user cannot make a redundant command.

Because there is a series of two-way communications between the controller and controllee, in which the controller and the controllee can "talk" to each other, more sophisticated operations become possible. The third generation omni-remote controller is under heavy investigation by many research groups. For example, researchers at Carnegie Mellon University (initiators of the Pebbles project) and MAYA Design worked together in a project called "Generating Remote Control Interfaces for Complex Appliances." [4] Their design pattern is that a service-provider appliance, such as an application in a PC, sends control parameters to a PDA, and the PDA generates an appropriate control panel (customizing the size of buttons, language, left or right hand control, etc., to the user's preferences) to control the appliance. A series of two-way communications between the PDA and the PC are involved in the process. As mentioned in the Pebbles project, "all participants' PDAs are in continuous two-way communication with each other, and with the main computer which is often projected on a screen to serve as the focal point of the discussion." [3] In order to make this scenario work, the appliance on the server side must be "smart." In other words, it should be able to communicate with the controller and also do some computations. Unfortunately, most current appliances are not that "smart."

Summary

Omni-remote controllers evolved from simple Universal Remote Controllers to the most recent complex omni-remote controller systems. The first generation omni-remote controllers are not upgradeable once produced. That is, a URC used to control the old model of appliances must be replaced when a user wants to control a new model of the appliance because the appliance codes need to be installed in order to control the new appliance.

Different from the built-in buttons in a URC, the second generation omni-remote controller uses a touch screen—a GUI—to control appliances. Most importantly, the second generation system can be trained to recognize which IR is used for what appliance

function. However, IR is not the best medium for delivering signals because it has a short coverage area and cannot penetrate obstacles.

The most important difference between third generation omni-remote controllers and the first two generations lies in its two-way communication approach. In the one-way communication approach, a controller sends out control signals but no response is sent back from the appliances. However, in a two-way communication, there is a series of communications between the controller and controllee. For example, when appliances receive control signals, they will respond with some information, such as the current appliance status. By a series of two-way communications, more complicated functions can be realized. Detailed discussion about the three generations of omni-remote controllers will be given in the following section of this paper.

2.2 Related Project

Several research groups and major corporations are actively working on the topic of device interaction, especially wireless devices. Success on this topic means not only a great jump in the technology but also enormous commercial profits. But, advances in a smart-device interaction have been a real challenge. Although researchers and corporations put a lot of effort in creating a real personal omni-remote controller, they have yet to yield any truly universal results. Current academic researches and commercial products can be categorized into: low ambition projects, medium ambition projects, and high ambition projects.

2.2.1 Low Ambition Projects

Low ambition projects aim to solve small integration problems which are relatively easy to solve. An one-to-one client-server architecture is often used. The problem of security is out of their research scope and therefore is not covered in most of these projects. Table 2-2-1 is a partial list of current low ambition projects.

Project Title	Institute	Paper Date
An Optical Pointer for Infrared Remote Controllers [10]	AirMouse Remote Controls	1995
Light Widgets: Interacting in Every-day Spaces [11]	Brigham Young University	2002
Magic Wand: The True Universal Remote Control [12]	Brigham Young University	2002
OmniRemote Springboard Module [13]	Pacific Neo-tek	Commercial product
InVoca 12-in-1 Touch Screen Universal Remote [14]	InVoca	Commercial product
ProntoPro [15]	Philips	Commercial product
Harmony-Internet Powered Universal Remote Controls [16]	Intrigue Technology	Commercial product

Table 2-2-1: A partial list of the low ambition projects

2.2.1.1 An Optical Pointer for Infrared Remote Controllers

2.2.1.1.1 Introduction

"An Optical Pointer for Infrared Remote Controllers" [10] is an optical pointing system that controls a TV by directing a laser at its screen. The idea of the system is simple: A special TV screen embedded with optical detector detects a light spot created by a laser beam emitted from a laser pointer. Initially the TV screen displays a predefined menu page, on which there are several control blocks. Depending on the position of the laser beam on the screen, the optical system determines the x-y coordinate of the light spot and translates it into a corresponding command. For instance, when a user points to the "volume up" section of the TV control page, the volume is turned up.

The interactive TV application needs a simple lens and four photo-detectors in the TV screen to measure the angle of incidence of light received from a reference beam. That is the TV screen has an optical detector to determine the x-y coordinate of any landed light spots. Based on the coordinate of beam, the optical detector's microprocessor reports the event to the base-unit microprocessor. The base-unit microprocessor then executes the corresponding command.

The article focuses on how the hardware is designed to gather more light. A laser beam fades with distance, so it is important for the hardware to capture as much light as possible in order to increase range. The author introduces a modified aperture-based camera that uses an optical window or "light pipe" in front of the optical detector to gather more light. Figure 2-2-1 shows the design of the "light pipe" camera.

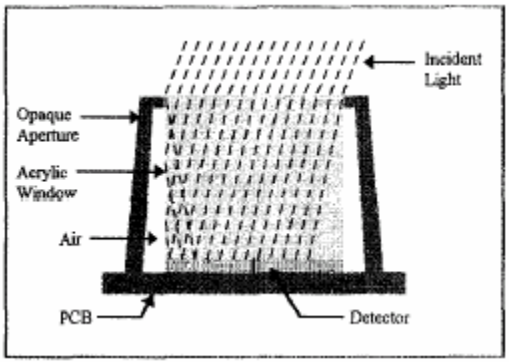


Figure 2-2-1: A "light pipe" camera [10]

In prior designs, when incident lights land on the optical detector, some of the lights will bounce back and get lost in the air. However, the author uses an optical window to catch reflected light. As we can see in the figure, incident light coming from a laser pointer goes into the light window on a TV surface. Most of the light is received by the optical detector while some light are reflected. Then the reflected light is bounced back to the optical detector by the acrylic-air interfaces in the walls. Through the use of a light window, the light incident to the detector surface is a summation of unreflected and reflected light.

2.2.1.1.2 Analysis

In this project, a laser pointer is used as a remote controller. A user points a reference beam to predefined areas on TV screen. The TV's optical detector system calculates the landed light spot and interprets it into a control command. The optical pointer system is simple enough that people of all ages can learn how to use it. However, if a user is too far away from the screen, the fading infrared reference beam may not land on the screen or the weakening reference beam may not be detected by the optical detector even with an optical window. A similar project called *MagicWand*, using a laser pointer as an input device, will be discussed in Section 2.2.1.3.

The paper gives an idea of using laser pointers to control appliances. Even though the controller of the system is a simple laser pointer, the system is not expandable because it is designed only for controlling TVs.

Pros:

- The system's controller is simple—a laser pointer.
- The control system is easy to use for people of all ages.

Cons:

- The control system only targets TVs that can support the optical detector.
- A user might not point to the correct control block or might accidentally point to the nearby block that will cause a false command.
- The system cannot handle complex controls because the predefined control area is limited due to the physical boundary on a TV screen.

2.2.1.2 Light Widgets

2.2.1.2.1 Introduction

Light Widgets [11] uses the idea of cameras capturing the operations on a visible surface to determine what type of actions the user performs. This system works similar to the motion sensor lights that people use in their yards. Two preinstalled cameras and a user's hands work as the front end of the control system. When a user's hand moves over a predefined area (they call it an environment tag) in respect to the cameras' view such as the top-right corner of the table, the cameras capture the motion and trigger the command associated with this predefined area. Figure 2-2-2 shows the idea of the project.

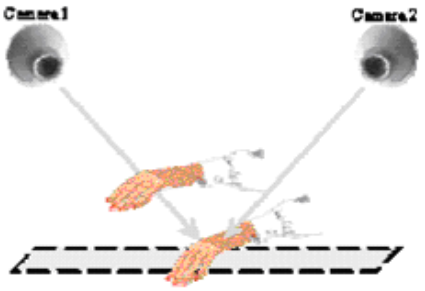


Figure 2-2-2: Multi-camera detection [11]

To prepare the *Light Widgets* control system, a user first installs two cameras in a target room. After setting up the cameras, the user defines control blocks in the operational area through the backend system. To do so, the user needs to draw control blocks on the picture of the room taken by the cameras and associates each control block with a certain function. A control block can be specified as either a boolean value or a continuous value. When defining a control block as a boolean value, the value is either on or off. When defining a control block as a continuous value, the value is like a GUI slider. For example, the top-left and bottom-left corners of the desk can be set up to turn on and off TV while the right edge of the desk can be set up as a GUI slider for changing a TV's volume. After control blocks are defined, the system is ready to go. When the user's hand touches or moves over a predefined block, the cameras capture the motion and report the event to the backend system. The backend system then executes the corresponding command. To be noticed, the control blocks must be defined by the views of the two cameras. To make an input to the system, a user needs to position his or her hand on a control block in respect to both cameras' views. As we can see in Figure 2-2-2, the lower hand triggers a command while the upper hand does not because the upper hand is not placed in the correct place from Camera2's angle.

To provide feedback from the controls, *Light Widgets* integrates with the XWeb cross-modal interaction platform [18]. XWeb is a subscription service that reports data changes to subscribed clients. For example, a client can subscribe the status of a TV in living room. Whenever the status of the TV has changed, XWeb reports the event to the subscribed clients. XWeb project will be analyzed in Section 2.2.2.2.

The authors mention that the cameras should only process the image locally to maintain privacy. That is, the microcontroller-powered cameras process the images locally and only return the predefined block tag (called the light widget identifier) and its approximated selected block where the user's hand is. The authors claim that the project is low cost and the user will not need to wear or carry any physical device. As they say "we are not trying to augment the world with information, but integrate interactivity into the physical world."

2.2.1.2.2 Analysis

The system defines control blocks based on the cameras' views of an area. When a user's hand moves over the control blocks, the system performs the associated action. However, if a user moves the furniture of the room and changes the view of the area, the system needs to be re-configured. Also, since the controller is a hand, any hand-like item could confuse the system. In this paper, the authors give an example showing how *Light Widgets* can help facilitate people's lives. They say the system is advantageous when a mechanic wants to adjust the height of a vehicle. However, I disagree that the system would be advantageous in that situation. Accidents could occur if a bird with skin-like feathers flies over the control block and causes the car to move down while the mechanic is under the car. Furthermore, this system cannot handle complex tasks since the space of a room is limited for associating with commands. Finally, it won't work if there isn't enough lighting in the room; the cameras would not be able to detect any images.

This project presents the possibility of using cameras to help control appliances. It also emphasizes the importance of an appliance feedback which is an important topic for me while implementing my personal omni-remote controller system. Even though the system claims to be simple and cheap, the *Light Widgets* system does not provide a universal interaction between a user and appliances because it is impossible to set up cameras everywhere.

Pros:

- The system's controller is simple— A user's hand.
- The system is easy and cheap.
- Users do not need to wear or carry any device.
- Feedback is possible through XWeb.

Cons:

- If the view of a target area is changed, the system needs to be re-configured.
- Accidents might happen when a hand-like object moves over a control block.
- The system needs enough lighting to capture images in a targeted area.
- The system cannot handle complex controls because the predefined control block of a room is limited.

2.2.1.3 MagicWand Project

2.2.1.3.1 Introduction

The *MagicWand* project [12] is a descendent of the *Light Widgets* project described previously. The difference between these two projects lies on their input techniques. That is *Light Widgets* uses a hand as the controller while *MagicWand* uses a laser pointer's light spot as the input of the system. In *MagicWand* project, a simple laser pointer with a camera works as the front end of a universal remote controller system. When a user points a laser point to a predefined block in the camera's view, the system's camera captures the light and triggers an associated command.

To prepare the *MagicWand* control system, a user first installs one camera in a target room. The camera is like the eye of the system and all the controls can only happen under the view of this camera. After a user sets up the camera, he or she can define control blocks of the room in the backend system. To do so, a user can draw control blocks on the camera's two-dimension photo of this room and associates each control block with a certain functionality such as turning a light on or off. It does not matter where a user defines the control area on the image. However, what function a control block is associated with is the most important thing. For example, a user can draw a control block on the wall for turning on a TV and another control block on the ground for turning it off.

A control block can be specified as either a boolean value or a continuous value. When defining a control block as a boolean value, it is either on or off. When defining a

control block for a continuous value, it is like a GUI slider. Figure 2-2-3 shows an image of the setup blocks.

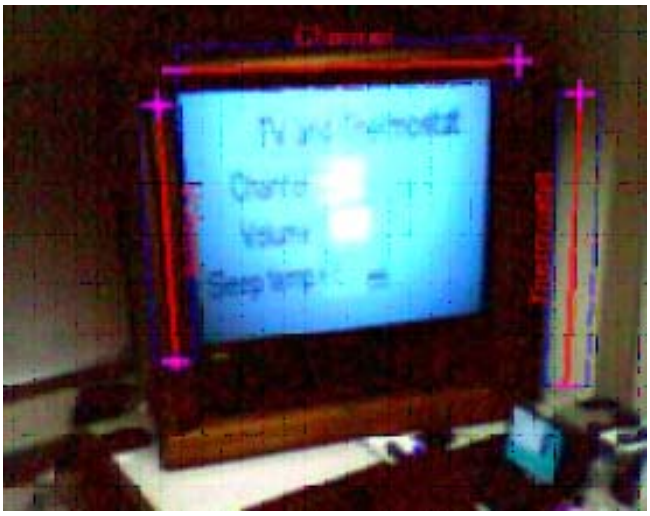


Figure 2-2-3: Setting up MagicWand blocks [12]

As shown in the picture, the red, linear line on top of the image is like a GUI slider that is used for changing the channel, and the red line on the left-hand side of the image is for volume control. Note, the control blocks can be defined at any place in the image. There is no difference when a user defines the channel control on top of a desk. After setting up the control blocks, a user can start controlling TV by pointing a laser pointer at the control block. The camera unit that captures the motion reports the event back to the backend system, and the associated function is performed accordingly.

2.2.1.3.2 Analysis

The *MagicWand* system defines control blocks based on a camera's view of an area. The system uses a laser pointer as the controller of the system. When a laser point lands on one of the predefined areas, the camera sees it and reports the event back to the backend system. The system then performs a corresponding function. Since the *MagicWand* project is a variation of the *Light Widgets* project, it inherits most of the drawbacks from *Light Widgets*.

The *MagicWand* system and the optical pointer system described previously both use laser pointers as the system's controller. However, the former uses a camera to catch a pointing action, while the latter uses an embedded optical detector. The *MagicWand*'s approach is better than the optical pointer system's in that it covers larger area. That is, the control block of the optical pointer system is on TV, while the control block of *MagicWand* is in a room. Also, *MagicWand* provides feedback through XWeb, but the optical pointer system does not support feedback from an appliance.

Pros:

- The system's controller is simple—laser pointer.

- The system is easy and cheap.
- Users do not need to wear or carry any device.
- Feedback is possible through XWeb.

Cons:

- If the view of a target area is changed, the system needs to be re-configured.
- Need enough lighting to capture images in a targeted area.
- The system cannot handle complex control because the predefined control block of a room is limited.

2.2.1.4 Commercial Productions

2.2.1.4.1 Operational Description

The commercial products of omni-remote controllers belong to consumer electronics. These products work by remembering the control code of appliances of many brands and models. When users want to control a certain appliance, they can configure their omni-remote controller by entering the appliance code. Then, the controller is set up to emit the corresponding control signal when a user clicks on the buttons.

The commercial products that I have found have as many similarities. I summarize the operational functions of these commercial products as follows.

User Interface:

The UI of commercial product consists of several built-in buttons and a touch screen LCD. When a user selects an appliance, the appliance's GUI control buttons show up on the controller's screen. A user can click on the buttons to perform desired functions. Furthermore, a customized GUI is often supported in commercial products. These functions include positioning, resizing, and coloring buttons. Users can configure the buttons as they wish. Finally, some commercial products such as InVoca^{TM} [14] support voice control. A user of this controller can record a voice to associate with a command and then speak to the controller to control appliances.

Code Training:

There are certain cases when an omni-remote controller cannot control an appliance, such as when an appliance is not supported by or is newer than an omni-remote controller. In this case, if the target appliance has a corresponding infrared remote controller, a user can train the omni-remote remote controller to learn the IR commands of the appliance.

Macro Functions:

Macro functions help users to perform multi-step operations that combine a series of operations into one click. For example, a macro function may work like this: turn on the TV → turn on the DVD player → switch to video channel → set the TV stereo to movie stereo → play the movie.

Communication Medium:

The commercial omni-remote controllers use IR as the primary communication medium. When IR is used, an IR emitter in the controller points to a target appliance's IR receiver to send a control signal. Some other products use Radio Frequency (RF) to compensate for the appliances that are not in line of sight. For example, if a target appliance is in a cabinet or in another room, a user can place an RF extender in front of the target appliance to receive the RF signal from a controller. The RF extender then sends the corresponding IR signal to the target appliance.

2.2.1.4.2 Product Example

User Interface:

The touch screen LCD of the ProntoPro TSU7000 consists of two components. First is a 3.8" TFT color LCD with 320x240 pixels and 65,536 colors. Second is a high-sensitivity touch screen. Figure 2-2-4 is the product's image.

Figure 2-2-4: An image of ProntoPro TSU7000 retrieved from http://store.premiumelectronics.biz/100-phitsu7000.html

Code Training:

A Harmony Internet Powered Remote Controller [16] called *Intrigue Harmony H745* is one of the commercial products that supports the code learning function. The learning distance of this controller is from 1 inch to 1 foot, and the learning frequency is up to 60 KHz. Figure 2-2-5 shows one of the Harmony remote controllers.

Figure 2-2-5: An image of Harmony SST-659 remote controllers retrieved from http://reviews.designtechnica.com/review1809.html

Macro Function:

InVoca 12-in-1 Touch Screen Universal Remote [14] is one of the commercial products that supports a macro function. A user can assign a macro button with a sequence of operations. To use the macro function, a user just needs to click on the button. Figure 2-2-6 shows a product's image.

Figure 2-2-6: A product image of InVoca 12-in-1 Touch Screen Universal Remote retrieved from http://www.dayafterindia.com/apr1/hot_tips.html

Communication Medium:

OmniRemote Springboard Module [13] is a commercial product that uses IR as its communication medium. Another commercial system called ProntoPro [15] uses both IR and RF as its communication medium. When controlling an appliance in its line of sight, ProntoPro uses IR. When controlling an appliance not in its line of sight, ProntoPro sends out an RF signal, toward an RF extender placed in front of the appliance receives the signal, translates the command into IR code, and sends the code out to the target appliance.

2.2.1.4.3 Analysis

Basically, all the commercial products adopt a one-way communication scheme. That is, only the omni-remote controllers send out control signals, and no feedback is received from the target appliance. Because there is no feedback from a target appliance, a user may click on a button that does not change to suit the state of a server. For instance, it is redundant that a user turns up the volume while the current state of a target appliance's volume reaches the top. If a system does feedback, the user will know the state of a target appliance and thus the above example will not happen.

Pros:

- The function of code training helps the omni-remote controllers to extend their capability onto controlling new appliances.
- A customized GUI gives users full flexibility for customizing the display of the UI.
- RF extenders solve the problem of existing appliances that do not support RF.

Cons:

- The commercial omni-remote controller systems adopt the one-way communication scheme that does not provide feedback from a target appliance.
- The systems cannot work together for cooperative tasks. For instance, if a group of washing machines can work collaboratively, a person, instead of checking each washer's availability, can ask the group if there is a washing machine available. The washers, working as a group, will tell the user which washer is available, if any.

In my proposed system, the appliances report their status periodically to the omni-remote controllers. Thus, the controller can provide a user with an updated device status and GUI. Also, my proposed system, adopting a Many-to-Many (M2M) structure, is able to work collaboratively because the client and server devices talk to each other. Thus, based on my framework, the manufacturers can design their system to perform cooperative tasks if they want to.

2.2.2 Medium Ambition Projects

Medium ambition projects aim to solve a medium-size problem, which is usually more complicated than projects in the last section. These projects often involve control over multiple devices from a single client. Table 2-2-2 shows the projects belonging to this category.

Project Title	Institute	Paper Date
A Remote Controller for Home and Office Appliances by Telephone [17]	Ankara/Turkey	1998
Cross-modal Interaction using XWeb [18]	Brigham Young University	2000
An Agent-based Bidirectional Intelligent Remote Controller [19]	Shizuoka University, Mitsubishi Electric Crop	2001
An Implementation of Intelligent Remote Controller Based on Mobile-Agent Model [20]	Shizuoka university	2002
Autonomous and Asynchronous Operation of Networked Appliances with Mobile Agent [21]	Keio University, Japan	2002
FReCon: a fluid remote controller for a Freely connected world in a ubiquitous environment [22]	Doshisha University/Hitachi	2003
The Information Home Appliance Control System – A Bluetooth Universal Type Remote Controller [23]	National Cheng Kung University	2004

Table 2-2-2: A partial list of the medium ambition projects

2.2.2.1 A Remote Controller for Home and Office Appliances by Telephone

In 1998, a paper named “A Remote Controller for Home and Office Appliances by Telephone” [17] depicted a phone-based remote controller system that sends commands to appliances through the existing telephone line.

2.2.2.1.1 Introduction

All sorts of technology advances facilitate the information exchange between homes and offices. The purpose of this remote controller system includes the following:

1. To find a solution for some prevailing problems. For example, equipment that is supposed to be left “On” or “Off” at home or in the office such as a thermostat
2. To make our lives easier. That is, people can easily turns the devices on or off remotely through the remote controller system
3. To create an economic remote controller system through the existing telephone lines. The authors thought that since the telecommunication network reaches almost everywhere around the world, it would be great if we could use the current telecommunication infrastructure

This remote controller system uses an additional control unit (a digital circuit) connecting to a local telephone to check the audio signal coming from the other end. A telephone keypad works as an input device for the system. “By pressing the ‘1’ key from DTMF (Dual Tone Multiple Frequency) keypad, it sends a high level potential and the LED (Light Emitting Diode) as a load turns on. By pressing the key ‘2’, the D type F/F (Flip-Flop) receives a low level potential, resets input and Q goes to low. As the result of this, LED turns off.” [17] This control unit allows a user to remotely turn on or off the device.

The authors claimed that the system has three advantages:

1. A solution for some prevailing problems: if someone forgets to turn the device on or off, he or she can still do it at office.
2. Convenience: a user can operate the devices at home while working. For example, heating up the room before going back home.
3. Economic: Telephone line reaches almost everywhere. We should make more use of it instead of just for verbal communication. The existing infrastructure reduces the cost of implementing this system.

The authors believe that this remote controller system is useful because it does not require a PC, which is good for the places where PC is not available. The authors also think that by elaborating and adding additional functions to this system, automation of household and office applications become possible. For instance, in the case of an electric outage, this system can connect itself to a battery and informs the user.

In the matter of security, the signal receiver at home uses a caller ID provided by the phone company to identify the controller (telephone). Only authorized caller IDs are allowed to control the system.

2.2.2.1.2 Analysis

Using the existing telephone line as the medium of the remote controller system is quite unique. However, the current design of this system only allows the control of TURN ON or TURN OFF. The system can be further extended to take full advantage of what can be done by using a telephone keypad and voice control. For example, it is possible to use different keys to control the different functions of an appliance and the system can provide voice feedback. Furthermore, the security scheme in this project is too simple. Anyone who knows or uses the authorized telephone number can access the user's appliances. Thus, this security approach is not practical.

Pros:

- This remote controller system utilizes the existing telecommunication network as its communication medium to lower the cost for developing it.
- This system takes advantage of the existing equipment and thus lowers the cost of the system.
- This system gives a convenient way of turning a device on or off remotely.

Cons:

- This system only allows a device to be turned on or off and complex control functions are not shown.
- The input device of this system—a telephone keypad—limits the user's ability to give commands. Only digits, a pound sign, and an asterisk are the possible choices for the input of this system.
- The security of this system is not practical.
- No feedback is provided by this system.

2.2.2.2 Cross-modal Interaction using XWeb

"Cross-modal Interaction using XWeb" [18] is a World Wide Web (WWW) based application that addresses the problems of interaction in various platforms. The goal of this paper is to let any interactive client connects and accesses any service. The authors claim that by using XWeb, the service creators do not need to be concerned with the details of interacting platforms or devices. An XWeb client can access an XWeb server's webpage using the underlining XWeb Transport Protocol (XTP).

2.2.2.2.1 Introduction

The rapid growth in computing capability and the popularity of the Internet impose a fundamental, interactive challenge that the traditional UI architecture cannot meet. These days, many aspects of human activity involve electronic devices and these devices need to share information in order to perform interactive tasks. Thus, there is an enormous potential for a universal model that supports heterogeneous systems. A successful model should support large and diverse interactive systems. However, the model cannot succeed

if it requires each computational device to install its own interactive hardware and software. Thus, the authors propose an interactive solution—XWeb. XWeb is a client or server model supporting various interactive platforms without requiring the participants to have their own interactive hardware and software.

The authors believe that the reason that we cannot have an effective, built-in UI is due to the cost of hardware. For example, in order to have a successful VCR interface, the cost of a product will almost double. When a remote access to the device is available, both the cost and the hassle can be minimized. For instance, if a vending machine's current state is accessible remotely, many visits by the vendor or their customers can be avoided. Thus, an interactive service protocol is necessary for clients to access a service.

XWeb is based on the WWW architecture with additional interaction and collaboration functions. An XWeb server is implemented independently from a user's interactive platform and provides responses to the XWeb Transport Protocol (XTP) which is a network interaction protocol. The reason that the XWeb user interface architecture is elastic is because an XWeb client and server can choose their own independent implementation strategies as long as they conform to the XTP. When a user works in the XWeb world via interactive clients, the XWeb server's functions are tuned to meet the capability of a particular client. For example, if a client only has two buttons that turn a device on and off, the server will only provide the on and off functions necessary to meet the capability of the client. Thus, a user can pick up a particular client to control an XWeb server that suits the user's need. Furthermore, because that type of client is all the user needs, he or she can learn how to operate only that type of client without bothering to study more complex clients.

XTP network interaction protocol uses a GET method to retrieve a server's information and a CHANGE method to edit a server's data. When retrieving information from an XWeb server, a client types in a URL of the site in a format like "xweb://domainname:port/pathname." It is similar to an HTTP URL and the port is optional as with a URL. The pathnames "consist of the indices or identifiers of the child object starting from the root of the site." For example, "xweb://automate.home/lights/livingroom/" gives a description of the lights in the living room at home, and "xweb://automate.home/lights/livingroom/1/@brightness" retrieves the brightness attribute of the second light (starting from 0) in the living room at home. When editing a server's data, a user can add, modify, and delete an attribute's value on an XWeb server.

All XWeb transport data is represented in XML, however XML data does not have to be the internal representation of a server or a client. When an XWeb client receives the XML data representation of an XWeb server, it interprets the XML data and generates a corresponding UI. A simple example of a UI is shown in Figure 2-2-7.

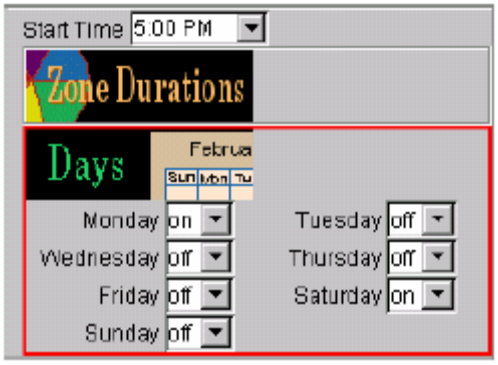


Figure 2-2-7: A simple example of a UI [18]

2.2.2.2.2 Analysis

First of all, the authors do not discuss about how a user finds out the domain name of an XWeb server. Some sort of service discovery is needed for this system to work. Each device has an XWeb page available for the XWeb clients. Even though the authors did not mention the communication scheme, I believe there is a series of two-way communications between the client and the server. Thus, in order to report a server's status, the system needs to send out many copies of the same message to the subscribers.

The authors say that "every interactive platform will have its own XWeb client implementation," which means that each client has its own way to interact with the server. This matches the idea of a control module in my thesis. For the screen layout, the authors use a general layout solution for all the servers. They believe that providing a general layout solution can simplify the development of new clients and have advantages for different screen-size clients such as shrinking or stretching the layout to a large- or small-screen client. However, I think this solution would not work well because not every client has the same characteristics. For example, a touch-screen client is different from a sound-controlled client in the way it interacts with the user and therefore it is hard to give a general solution to different clients. Furthermore, an auto-generated GUI can only abide by the syntactic meaning of a content description of a server, but it can hardly comply with any semantic rules.

Like my project, XWeb's users can choose a particular client that is suited to their needs, learn how to use that type of client, and yet interact with all possible services. For a remote accessible system, an advanced security scheme is critical because it is undesirable for others to be able to turn off our light remotely. However, the security issues are not mentioned in the paper.

2.2.2.3 An Agent-based Bidirectional Intelligent Remote Controller

"An Agent-based Bidirectional Intelligent Remote Controller" [19] uses "XML and agent-based macros to realize advanced interoperations among home appliances."

2.2.2.3.1 Introduction

Currently, remote controllers play an important role in improving the usability of appliances. However, an increasing number of built-in buttons and inflexible macro functions reduce the usability. Also, because each appliance has its own remote controller, scattered remote controllers become a problem. In order to solve these problems, the authors introduce an agent-based bidirectional remote controller that controls appliances with an XML-based touch screen LCD.

The authors categorized the contemporary remote controllers into two classes—the Learning Remote Controller and the Bidirectional Remote Controller. The Learning Remote Controller learns control codes for appliances from an appliance's remote controller and assigns the learned functions to its button. The authors give two drawbacks for this kind of remote controller. The first one is that the interface is difficult to use due to the redundant buttons. Plus, some buttons that have no use to users often cause confusion while inputting commands. The second drawback is that macro functions will not stop once they start and unexpected accidents might occur. For example, when recording a TV show, the macro function would still turn on a TV and proceed even if no tape is put in VCR.

The Bidirectional Remote Controller with a touch screen panel aims to improve the usability by retrieving the status of an appliance. When the remote controller knows the state of appliance, it can provide a user-friendly control panel such as only showing the buttons that are necessary for changing the state of appliance. The bidirectional remote controller also has the ability to discover new appliances to control. With the bidirectional communication ability and a touch screen panel, the bidirectional remote controller provides better usability than the learning remote controller.

To enhance the bidirectional remote controller, the authors propose an agent-based scheme that generates a UI based on the appliance's XML description and has agents to perform tasks that are sequential operations among appliances. Before controlling an appliance, the remote controller downloads the appliance's functions in XML format and interprets the context for generating UI. Each of the functions has its own X-Y layout coordinate. The interpreted descriptions are stored in a structured table in Table 2-2-3.

CAPTION	ACTION	CODE	X	Y
PLAY	PLAY	0x0322	300	150
PROGRAM	PROGRAM	0x0328	550	550
STOP	STOP	0x038c	50	350
FORWARD	W	0x0321	550	250
REWIND	REW	0x0323	50	250

Table 2-2-3: a structured table storing XML data [19]

The stored data describes the function and the button layout of each appliance's control in a control panel. As shown in the table, CAPTION is the label of a button. ACTION holds the name of a standardized function that is used for subsequent filtering. The CODE field is a combination of custom code and data code in which a custom code identifies the

manufacturer and the type of appliance while a data code provides a method for controlling the appliance. The X and Y columns are used for generating a GUI. In this system, a user-defined layout is possible.

Once the UI is generated, a user can initiate a task. In this system, an agent model is used to execute a series of commands. It is similar to an advanced macro function with error checking.

The state of the server is retrieved by the client through the bidirectional communication scheme. By knowing a server's state, an agent is able to ensure the process sequence. That is, an agent checks the status flag of the appliances to make sure each step is completed correctly. For example, the steps of duplicating a tape include:

Step 1: VCR2→ Record
Step 2: VCR2→ Pause
Step 3: VCR1→ Play
Step 4: VCR1→ Pause
Step 5: VCR2→ Start
Step 6: VCR1→ Start

In each step, if the state of an appliance is not what the agent expected, the agent will stop the process and deal with the error according to the predefined rules.

The authors describe an implementation in a two-page paper named "An Implementation of Intelligent Remote Controller Based on Mobile-Agent Model." [20] The main goal of the agent-based model is "to perform a complex inter-appliance operation with one button." [20] In this implementation, the authors make the agent "mobile," that is, the agent script is embedded in both the remote controller and the appliances. By having a mobile agent, the remote controller is not needed once the task has started because each appliance knows the overall steps of the task and they know what they should perform next. Different from the mobile agent approach used in this paper, "Autonomous and Asynchronous Operation of Networked Appliances with Mobile Agent," [21] which will be discussed in Section 2.2.2.4, describes a mobile agent that migrates from one appliance's agent platform to another appliance's agent platform.

2.2.2.3.2 Analysis

This system is based on the bidirectional communication between the controller and the appliance, by which the client is able to know the state of the server. This is an important scheme for designing an omni-remote controller system. In order to provide an accurate UI and to give a user feedback about his or her command, the state of target appliance must be known. The agent-based system makes life easier by checking and implementing a series of predefined steps. An agent is actually a software process that achieves its goal by interacting with appliances. For example, a "tape it" agent records a TV show by interacting with both TV and VCR.

The agent system helps a user to execute and monitor a sequence of commands without the user's attention. However, putting copies of an agent script among the

appliances might be a little tricky. One agent script is for one task. If there are many different tasks, the duplications of the same agent script scatter among the appliances. When one agent script is updated, the system needs to renew all the other copies among the appliances.

This system generates a UI with a structured table that describes each of the functions. The auto GUI generation scheme is similar to the one in XWeb project. However, when the control function of an appliance gets complex, an auto generated UI results in a non-user-friendly interface because the semantic meaning of each individual appliance cannot be expressed by the syntactical UI interpreter. Furthermore, since each controller has different characteristics such as the different sizes of touch panels, auto generated GUIs are not suitable. In my system, each type of UI is implemented independently to ensure the GUI matches different types of client devices.

2.2.2.4 Autonomous and Asynchronous Operation of Networked Appliances with Mobile Agent

In the paper of "Autonomous and Asynchronous Operation of Networked Appliances with Mobile Agent," [21] the authors introduce a Smart Operation of Networked Appliances (SONA) system that realizes the autonomous and asynchronous operation through a mobile agent. This project is a variation of "An Agent-based Bidirectional Intelligent Remote Controller." [19]

2.2.2.4.1 Introduction

Our environment is filled with computation and network-capable appliances. Even though advanced appliances enrich our lives, the complexity and time to control these devices reduce the benefit they deliver. Therefore, the authors suggest the Smart Operation of Networked Appliances (SONA), an agent-based system, to help users control appliances.

The existing operation systems of appliances, such as remote controller systems, require users to input their commands synchronously with controllees. For example, in order to watch movie, a user needs to turn on a TV and DVD device and change them into the movie setting. To facilitate a user's operations, the authors introduce an agent-based system called the SONA system in which an agent is used to perform a task for the user. When using a regular remote controller system, a user has to push the buttons one after another to complete a complex task. However, with the SONA system, a user only needs to notify the SONA system for the task he or she wants to perform. Then the SONA system generates an operation plan of the task. When the user clicks on the button associated with the task, an agent of the task starts to work on the job.

An agent residing in a user's controller device helps the user to perform a task. If a task involves many appliances, the controller device will communicate with the controllees. To make the system more efficient, the authors suggest mobilizing the agent, that is, a mobile agent migrates from one device to another and performs one step of the

task in each appliance. For example, when a user wants to watch a movie, he or she clicks one button to initialize a mobile agent that is defined to turn on the TV and set the TV to a movie setting, and then turn on a DVD player to play the movie. When the user clicks on the button, the mobile agent knows its first step and migrates from the user's controller device to the TV. While in the TV, the agent turns on the TV and sets it up to a movie setting. After performing the first step, the mobile agent migrates from the TV to the DVD player and performs the second step.

Three main components of the SONA system are described in the paper. They are the appliance lookup service, operation plan composer, and mobile agent system. A lookup server acts as a bootstrap server for appliances, in which all the appliances need to register themselves to the lookup server. Thus, controllers can find all the available appliances with the lookup service. The operation plan is created by the operation plan composer based on the lookup service and the user's demands. In order to execute the plan correctly, the mobile agent needs to check the appliances' sensors in order to get their current state. A mobile agent is defined by the authors as a software component with four characteristics: autonomy, social ability, reactivity, and pro-activeness. Any existing mobile agent system that matches these four characteristics is eligible for the SONA system. The software architecture of the SONA system is shown in Figure 2-2-8.

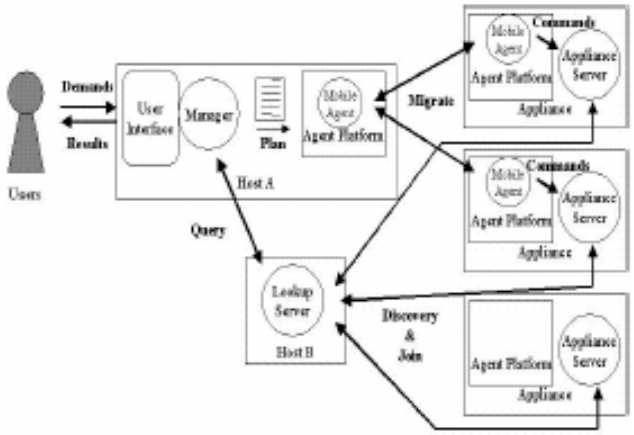


Figure 2-2-8: Software Architecture of SONA [21]

As we can see, in the SONA system, controllers and appliances have an embedded agent platform. After a user sets up the demand for a task, the operation plan composer generates the corresponding operation plan. The operation plan can be associated with a button. When a user clicks on this button, a mobile agent is initiated. The agent then works its way through the appliances by migrating from one agent platform to another agent platform depending on the need and the operation plan.

The SONA system is based on the client-server architecture, in which a one-to-one communication is applied. In this system, a client and server are connected through the Internet architecture in an asynchronous manner. Thus, we can say that the SONA system is an Internet-based remote controller system. The underlying security of the SONA

system is supported by whatever mobile agent platform is used while implementing the system. The SONA system itself does not support application-level security.

2.2.2.4.2 Analysis

This project is a variation of "An Agent-based Bidirectional Intelligent Remote Controller." [19] In [19], each agent script is predefined and the same script is placed among appliances. In "Autonomous and Asynchronous Operation of Networked Appliances with Mobile Agent," an agent script is generated by the operation plan composer and the agent migrates from one appliance's agent platform to another appliance's agent platform.

The most significant improvement in this project is the development of the mobile agent system. All the participating appliances must install the mobile agent platform which enables the migration of the agent code. However, since the existing mobile agent platform requires a lot of system resources, an additional computation device is needed for each of the appliances.

The authors stated that "the agents migrate into the appliances and operate them according to the given plan." This helps to reduce the communication time needed between the client and the server. However, they do not discuss the topic of fault tolerance of the system in this paper. Therefore, more research is necessary to answer questions such as "what if the appliance hosting an agent fails?"

The SONA system is the Internet-based system built on top of the client-server architecture. If the agent needs to report its status to other mobile agent platforms, the same status message will be sent out several times. This problem can be solved by using the M2M architecture that reports an agent's state in one broadcast.

2.2.2.5 FReCon

"FReCon: a fluid remote controller for a Freely connected world in a ubiquitous environment" [22] is a general-purpose remote controller. Every appliance has its own website and a client uses the IP address of an appliance to gain access.

2.2.2.5.1 Introduction

Computer-embedded systems enrich our lives. However, not everybody is able to enjoy the convenience they bring to us due to the complicated UI of these appliances. The authors propose a system called Fluid Remote Controller (FReCon) that aims to control appliances through the use of portable device such as a PDA. The authors define Fluid Remote Controller as "a universal, general-purpose remote controller, that is to say a unique portable device capable of controlling a wide range of appliances."

In this system, each appliance has its own website that is available for one client at once. A user can first find the desired appliance by pointing the controller at it. Then, the

user uses his or her controller to interact with the appliance through the appliance's website. Initially, the user "naturally" points the controller to the desired appliance and the appliance is able to sense the "sensible" object embedded in the controller device. Depending on its status, "Free" or "Busy," the appliance will either grant or deny the requester the right to access its webpage. Figure 2-2-9 shows the process of accessing an appliance.

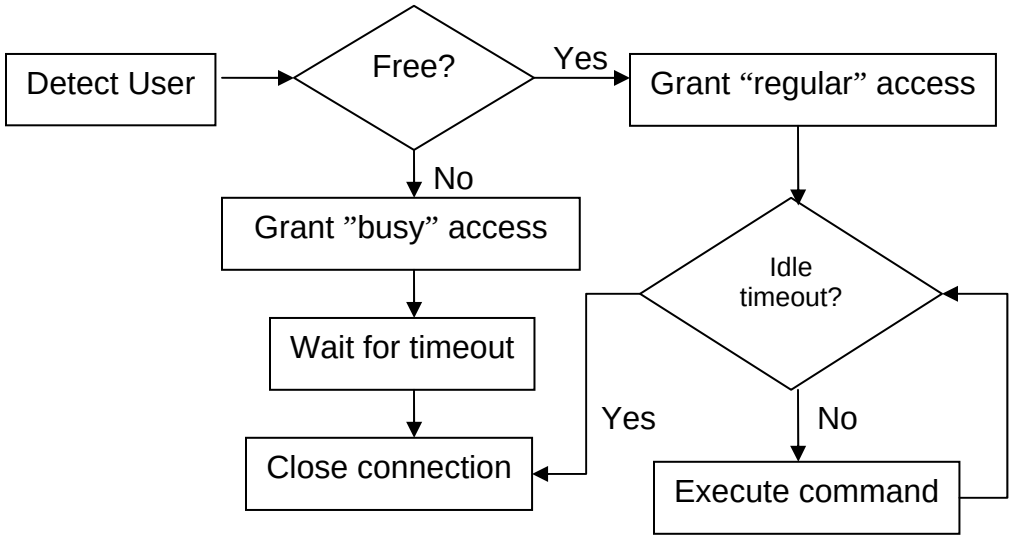


Figure 2-2-9: the process of accessing an appliance [22]

If the appliance grants the access right to this controller it will transmit its IP address to the client device. The service discovery stage is all done through the use of IR.

After the service discovery phase, the communication medium switches to RF. If the controller gets the right to access the appliance, it uses the appliance's IP address to locate the appliance's website and control the appliance through the webpage. Tomcat, the Java server of the appliance, is responsible for interacting with the authorized client device. If the controller does not get the permission to access the appliance because the appliance is "already connected," a busy message is shown on the client's screen. Because only one client can control an appliance at once, a client is considered "timed out" if it does not gain access to the appliance within a given amount of time. The appliance is then available to serve the next client.

If a user wants to control an appliance while still controlling a different appliance, he or she can just point the controller to the desired appliance to begin another service discovery process. The previously controlled appliance will eventually time out and become available for the next user.

2.2.2.5.2 Analysis

In this project, IR is engaged in the service discovery stage. The authors assume that it is a natural action to point the controller to a desired appliance. However, if the appliance is behind an object or in another room, it will not be found by controller at all.

For example, if the thermostat device is located in the living room, a user in the bedroom is not able to control the thermostat. Also, if there are many appliances in one place, an IR signal collision may occur. A signal collision problem could occur when signals from multiple appliances come in contact, causing an error signal reading of this signal.

When a controller switches its control from one appliance to another appliance, the previous appliance can become available only when a timeout happens. I believe it is better for the controller to notify the previous appliance in order to make the system work effectively. Furthermore, the authors announce that they take issues related to a multi-user environment into consideration by granting the access right to one user at once. However, this is actually not a real multi-user application. What it does is restrict an appliance to one user at once. Thus, the appliances of this system do not support multiple users. To improve the system, an appliance should allow many clients to have the "READ" (monitor) access at the same time, and only one client to have "WRITE" (control) access at once.

2.2.2.6 The Information Home Appliance Control System – A Bluetooth Universal Type Remote Controller

"The Information Home Appliance Control System – A Bluetooth Universal Type Remote Controller" [23] is an environment controller using the Bluetooth technology. This two-page paper, published in 2004, gives readers a broad view of the control system but a limited amount of detailed information.

2.2.2.6.1 Introduction

Advanced appliances are now used in almost every household; however, a unified model or style for controlling these devices is not yet established. That is to say, each appliance has its own remote controller. The authors propose an environment controller built on top of the Bluetooth technology via a radio wave that helps to penetrate space barriers. A Bluetooth universal type remote controller aims to substitute remote controllers of various appliances and to provide an integrated control interface for them. When all appliances have the Bluetooth module, the system can form a remote control environment with each device having a transmitting end and a receiving end. The transmitting end consists of an input device, a microcontroller, and a Bluetooth module, and the receiving end is an appliance with a microcontroller and a Bluetooth module.

A communication cycle of the control system includes: 1) a user inputs a command from the keyboard of a client device, 2) the client device sends out this command through its Bluetooth module, 3) the message is received by the appliance's Bluetooth module, and 4) the command is performed in the appliance.

The authors give an example by applying the framework to an existing telephone system. They install a Bluetooth module into a home telephone set and use a Bluetooth remote controller to receive phone calls. The implementation contains more hardware description than software. The system is controlled by a microprocessor based on IC

(Integrated Circuit) control logic. The result of this experiment shows that the voice received by the client Bluetooth remote controller is very clear and stable. However, one drawback from this design is mentioned: that is, the system consumes more energy because many digital ICs are used in the hardware design.

2.2.2.6.2 Analysis

The omni-remote controller system depicted in this paper leans more toward hardware design, but my system is more of software design. However, we do have some shared ideas. First, we all anticipate that advanced technologies will bring smart appliances into our lives soon. So far, a refrigerator embedded with an LCD that can get online or a digital box that can play previously broadcast TV shows is not uncommon. Second, we all believe that the use of radio waves as a communication medium of remote controller improves the accessibility of the system. Even if an appliance is behind an object or wall, controller and controllee can still communicate with each other in a close range via radio waves. No more detailed information is provided by this two-page paper.

2.2.3 High Ambition Projects

The high ambition projects aim to solve the heterogeneous device integration problem, and this type of project is complicated in nature. A partial list of these projects are in Table 2-2-4.

Project Title	Institute	Paper Date
Composable Ad-hoc Mobile Services for Universal Interaction [24]	UC Berkeley	1997
A Universal Information Appliance (using TSpaces) [25]	IBM	1999
Development of Bi-directional Remote Controller Protocol and Systems for Domestic Appliances [26]	RHA Information System Committee and UC R&D Project Tokyo	2000
Integrating Information Appliances into an Interactive Workspace (iRoom) [27]	Stanford University	2000
ICrafter: A Service Framework for Ubiquitous Computing Environments [28]	Stanford University	2001
HP-COOLTOWN project [9]	HP	2002
Pebbles project - Universal Personal Controller [3, 4, 5, 6, 7, 8]	CMU	2002
Autonomous and Universal Remote Control Scheme [29]	National Cheng Kung University, Taiwan	2002

Table 2-2-4: A partial list of the high ambition projects

The projects in this category often use a dedicated server for managing their system. A bottleneck (performance) problem happens when a lot of clients and servers try to use

the resources provided by a server and the server reaches a limit on its ability to provide services. The second problem associated with a central server is the single-point-of-failure. When a system depends on a dedicated server, a failure of the server means a total breakdown of the system. Finally, for the security of a system using a dedicated server, the denial-of-service (Dos) attack becomes an issue. A single server often posts a threat of attack and hackers can overload the server. Thus, the server will lose its ability to serve in a normal manner.

2.2.3.1 Composable Ad-hoc Mobile Services for Universal Interaction

In 1997, "Composable Ad-hoc Mobile Services for Universal Interaction" [24] depicted a new open service architecture that supports heterogeneous client and server devices with minimal assumptions about standard interfaces and control protocols. By using this system, when a client device moves into a new environment, it will adapt its functionality to exploit discovered services.

2.2.3.1.1 Introduction

An electronic device with wireless communication ability will soon become a norm rather than an exception. However, the environment for supporting this type of device to roam is not yet mature. The key point for continuously available among different networks is to be adaptable. That is, a device should be able to interact with the environment when it enters the area. Thus, a sense of a universal interaction is required for mobile devices to roam.

To achieve a universal interaction, three issues ought to be taken into consideration. They are the network connectivity in each environment, the ability of each client, and the variability of available services in each network, all of which vary significantly and thus make device roaming difficult. To solve this problem, the authors "describe an architecture for adaptive network services allowing users and their devices to control their environment." The goal of the project is to let clients discover what they can do in a new environment with a few assumptions about the control protocols and standard interfaces.

To find out the services in an area, beaconing is necessary. A "beacon" is an aid for a client to discover a server. A sender's beacon often includes enough information, thus a receiver is able to reach the sender. There are two kinds of beacon—*client beacon* and *server beacon.* A server (base station) beacon has the following benefits:

- Service providers (servers) are often powered through the wall, and controller devices (clients) are mostly mobile and battery-powered. Thus, by adopting a server beacon scheme, clients will consume less power.
- When a client beacons, it requires two messages: one is for a client requesting and another is for a server responding. When a server beacons, it requires only one message: the server's beaconing information is broadcast and the client can pick up the message. Thus, a single message is enough for service discovery process if adopting a server beacon.

- There is less traffic if a server beacon is used. When a server beacons, the message is for every client. When a client beacons, there will be a responding message for the requester.

This system uses server beaconing because of the above listed reasons, such as power for mobile devices is more important and *server beaconing* can save more power than *client beaconing*.

To make device roaming possible, there are three key elements in the Universal Interaction architecture. They are an augmented mobility beacon, an IDL, and client interfaces that maintain a layer of indirection. An augmented mobility beacon provides device location information along with security enhancement. A beacon encoding, shown in Figure 2-2-10, includes a server's bootstrap information, a *nonce* to make the scope of service local, and a dynamically configurable application-specific payload.

Header	SIP IP addr	port	nonce	Application-specific payload …

Figure 2-2-10: A service beacon encoding [24]

The server's location information includes a header for routing, a Service Interaction Proxy (SIP) IP address, and a port number. In addition to the standard cryptography-based security and public-key encryption, a *nonce* is added for security reasons. Because this is an IP-based system and a message can be sent through the networks across the world, "a person across the country can turn off our light" must be avoided. Thus, a server would periodically broadcast or multicast a "new" *nonce* to local clients only. The local clients with the correct *nonce* are able to interact with this server. Because remote clients do not know this *nonce*, they are not allowed to use this server's services.

The field of application-specific payload, a variable length, is needed to support thin client interfaces. A thin client interface holds up different realization on different hardware for different clients because each client has its own display and interface, such as a different size of screen and different types of input devices. For example, a PDA client uses a server's application-specific payload to generate the PDA's UI and the control of the server.

As a part of the realization of the application-specific payload, an *Interface Definition Language* (IDL) paired with hierarchical components of abstract UIs is developed. For example, an application-specific payload containing an abstract UI of TV, described in IDL, can be interpreted by a client device to generate a suitable UI for the user. In other words, the IDL allows exported server interfaces to be mapped to a client device's control interface. By using IDL, a client is able to perform a remote procedure call (RPC) of a server.

Furthermore, to make a universal interaction system compatible with other reference language implementations, the IDL is used in addition to other interface languages and

coexists with them. In the authors' words, "upon discovery of a service, the client device checks to see if a language implementation is available that it can support, and if not, uses the IDL file to learn the RPC (remote procedure call) calls and parameters that can be used to access the service." That is to say, a reference language for a server's UI is not necessary to be IDL.

Figure 2-2-11 shows the project's operating environment.

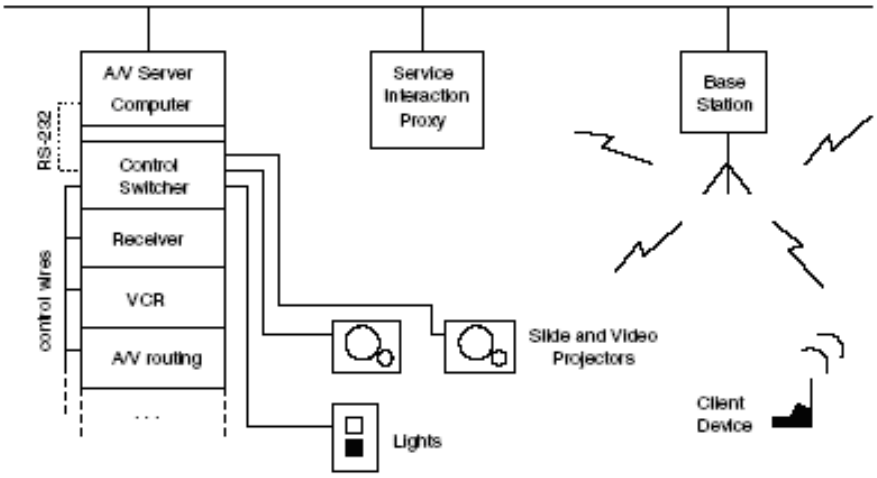


Figure 2-2-11: Project operating environment [24]

When a client device enters a universal interaction environment, it receives a beacon sent by a beaconing daemon in the base station. This beacon contains the bootstrap address and port number of the SIP. The client uses this information to register itself with the SIP and then to establish IP connectivity. After establishing IP connectivity, the client requests a list of the available services in the region and, based on the contents of the reply, the client displays these services for the user. When the user selects a particular service, the client software will bring up the interface of the requested application if the interface is in the local cache. If the interface is not yet generated, the client will ask the SIP for the service's "properties." The properties include a list of available interface descriptions and/or implementations. Then, the client chooses to either download a particular interface implementation (such as a Java applet) or the generic interface description of the service that can be used by the client to generate a UI. After the above process, the user is able to access the service accordingly.

2.2.3.1.2 Analysis

Universal Interaction is an IP-based system that supplies IP-level transparency. A device is assigned a Service Interaction Proxy IP, by which it can roam, discover the local services, and be controlled by use of the application-specific payload in server beaconing.

This system adopts a server beaconing scheme. In my personal omni-remote controller system, I use a dual-mode service discovery scheme—the use of both *server*

beacon and *client beacon*. When a user enters a service area, his or her control system can discover the available services by either waiting for servers' beacons (*server beacon*) or requesting servers' beacons (*client beacon*) depending on the needs.

The system allows heterogeneous clients as well as servers. Any client can try to interact with any server. The coexistence in the Universal Interaction is a great ambition. However, without an interacting standard (a standard type hierarchy) like TCP/IP in the network, it is not guaranteed that clients can interact with any other service providers. That is why in the remote controller world, Sony's TV remote controller is not able to control Philips' TV, which is an obstacle hard to overcome right now.

Since it is an IP-based system, the authors introduce a *nonce* to solve the problem of unauthorized remote control—to avoid messages getting routed to a destination across the world. If adopting the local broadcasting, the broadcast message is not routed. Thus, we can remove the *nonce* from the beacon.

2.2.3.2 A Universal Information Appliance Using TSpaces

"A Universal Information Appliance" (UIA) [25] is "a personal device, such as a PDA or a wearable computer that can interact with any application, access any information store, or remotely operate any electronic device." This system is built on top of TSpaces [34] that is a network communication middleware. TSpaces uses a shared whiteboard model where all clients can post tuples and see the tuples in the same global message board. A tuple is "a fixed-length list containing elements that need not have the same type. It can be, and often is, used as a key-value pair[2]."

2.2.3.2.1 Introduction

We are quite familiar with the built-in interfaces appearing on different appliances, such as the interface of a TV, coffeemaker, or thermostat. Because of our experience with the physical world, when developing software, we tend to design static interfaces as well. Applications with a dynamic interface (varying at each run-time instance) and a context-sensitive interface (changing under different contexts) are not very common. However, if a single device has a flexible interface, it will be able to represent an unbounded number of appliances. For example, we will want to have our own personalized interfaces to an ATM (Automatic teller machine). Thus, the authors introduce a Universal Information Appliance (UIA) that aims to create an environment in which a mobile device serves as a bridge between human and electric devices.

The authors give a scenario that describing the functionality of an UIA.

> "BLARP, BLARP, BLARP . . ." You wake up with a start to the loud ringing of your universal information appliance in your apartment. During the night, the overseas sales office scheduled an emergency meeting first thing in the morning. They checked your schedule and

[2] See: http://c2.com/cgi/wiki?TupleSpace

> found that you had an open slot. (You gave them authorization to do this.) By scheduling the early slot in your day, your UIA subsequently changed your wake-up time appropriately (but within the permitted boundaries). Glancing at the time, you pick up your UIA, turn off its alarm, and place the UIA in your pocket. While grabbing a quick breakfast in your kitchen, you remember that you want to set the VCR to record a program in the evening, so you take a quick jaunt into the living room. As you retrieve your UIA from your pocket, you realize that the "top level" kitchen applications have been replaced with living room and entertainment applications. As you select VCR, the UIA quickly retrieves your saved preferences. You add the program that interests you to the list of programs to record, and then submit your new settings. As you leave the apartment building, the household applications disappear from the set of active applications on the UIA, and new applications appear. You click on the transportation application and find that your bus will be late (a flat tire has been reported). You open the alternatives menu and find that you can still catch a taxi to the subway stop. Selecting "taxi" in the menu, you get instant confirmation that the taxi will arrive in two minutes at your location, determined by a quick location fix courtesy of the built-in Global Positioning System (GPS) [25].

In order to realize these functions, the system needs a communication software infrastructure and a wireless communication link between these UIAs and the communication infrastructure. The UIA is a two-way wireless communication system and uses IBM's TSpaces as its communication software. TSpaces is a network communication buffer providing "group communication services, database services, URL-based file transfer services, and event notification services." [30] Furthermore, since TSpaces is implemented in Java, it is automatically platform-independent by Java's cross-platform characteristic.

An UIA can be broadly divided into three main areas:

1. *A user's interface to the digital domain*: The UIA is focused on translating the common expression of a UI into a platform-specific user interface. Depending on a client's ability, such as a PDA's screen size, the UIA generates a suitable UI, which means each type of client will have a devoted UI.

2. *A wireless infrastructure keeping the UIA connected to network*: The UIA uses a wireless communication with both continuous and intermittent modes of operation. The continuous link is for fixed or near fixed areas, such as in a house or office, whereas the intermittent link is for users moving around in a wide area, such as a downtown location.

3. *Communication middleware connecting the UIA to services and information*: The primary functionality of the UIA is to provide a UI of a required service which is done through a network middleware. In the authors' words, a network middleware provides "a uniform mechanism for accessing devices or services across heterogeneous platforms that is open to application and language-specific data types and allows the UIA both to trigger and to be

triggered by events in other entities." Communication middleware also performs service discovery and reports to UIAs accordingly.

When a user, equipped with an UIA, enters a "smart room," the UIA will pick up a broadcast message indicating that this is an UIA-enabled environment. The UIA responds to the "smart room" with its ID for registering itself to the smart room and then queries the room for the information of the other appliances in order to interact with them. The "smart room" has a server running TSpaces system and an event application. The event application is responsible for obtaining events from UIs and sending the events back to event subscribers. The event application also serves as a bootstrap server in the area.

The authors develop *a reference platform* to define the common elements of service providers' functions. When implementing a function in the way that is required by the reference platform, a service provider becomes a valid system member. When an UIA registers itself to the local communication middleware, it will submit events to the middleware. Then, the middleware will forward the events to the subscribers because it has all the information of the UIAs in the area.

2.2.3.2.2 Analysis

The UIA project aims to provide an environment where all the devices can interact with each other. However, for a device to join this platform, it must implement the common set of functions from the reference platform. By the common interfaces, the device will be recognized by its peers. The communication middleware, TSpaces, plays an important role in this project. The UIA uses the utility of TSpaces to operate. Furthermore, service discovery is also achieved through the use of the middleware.

An UIA generates a suitable UI according to the type of a client. However, when the functions of a targeted device become complex, a well-structured UI cannot be guaranteed by the auto-generation schema. For instance, if there are hundreds of functions provided by a copy, fax, and scan (three-in-one) machine, it will be difficult for the auto-UI generator to lay out the controls correctly. In my personal omni-remote controller system, I use a control module approach. A control module is an appliance's interface in the controller. Thus, for each service provider, there is a corresponding control module for each type of client. In the above example, there will be a devoted control module (a three-in-one machine's interface), possibly created by the device manufacturer, for each type of client such as a PDA client.

Since all the functions are done through the local middleware, the issues of the single-point-of-failure, a bottleneck problem, and the denial-of-service attack become evident. As the authors described in the paper, the communication middleware knows wherever the UIAs are, which means the user's location is known too. Therefore, future research needs to be conducted on the personal privacy problem.

2.2.3.3 Development of Bi-directional Remote Controller Protocol and Systems for Domestic Appliances

"Development of Bi-directional Remote Controller Protocol and Systems for Domestic Appliances" [26] proposed the Bi-directional Controller Protocol (BCP) that allows the devices to communicate with each other regardless of their different manufacturers.

2.2.3.3.1 Introduction

Remote Controller (RC) has been a valuable tool for a long time. However, different types of remote controllers may cause the signal collision problem; that is, multiple devices try to access at the same time. To solve this problem, in 1987, the Association for Electric Home Appliances (AEHA) had a meeting and introduced with a standardized carrier frequency unification of infrared remote controllers. This standard regulates the infrared control signal used in remote controllers and tries to prevent the signal collision problem.

One topic left undiscussed in this 1987 meeting was the standardization of functions and codes for devices' mutual communication. Thus, the committees gathered together again in the year 2000 and studied the issues of the human interface design of the networked home appliances. Issues discussed in the meeting included:

- Usable with all appliances in full-compatibility by the common Remote Controller (RC), regardless of different manufacturers
- Usable with new products by the common RC without any change
- Controllable from anywhere in the house
- Easy to operate by user-friendly interface on a screen
- Usable and expandable to networked functions for energy saving applications

In order to provide an answer to these issues, AEHA organized a study committee consisting of 9 leading companies to draft a RC specification and the system was evaluated in Japan.

The researchers in the committee proposed a Bi-directional Controller Protocol (BCP)/Equipment Control Profile (ECP) architecture targeting domestic appliances. Figure 2-2-12 shows the system model.

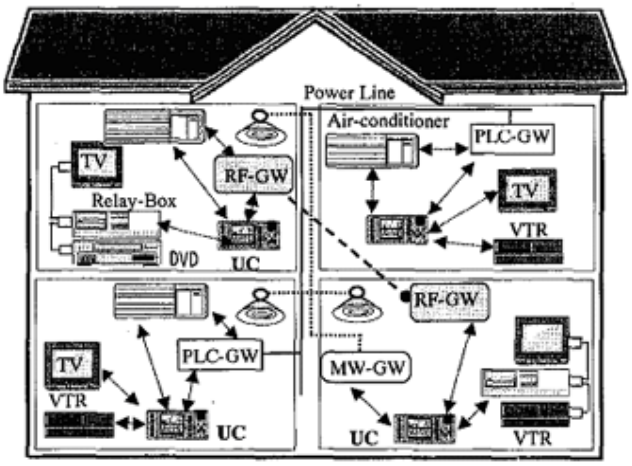


Figure 2-2-12: Universal Remote Controller featured System Model [26]

As we can see in the figure, each room in the house has a Universal Remote Controller (UC), a Gateway Controller (GW), and several appliances. UC communicates with appliances via a bidirectional communication scheme. Also, a gateway controller in each room is a part of an infrastructure of peer-to-peer connection throughout the house. UC is able to control the appliances in another room via the use of gateway controllers that transmit control signals from one room to another. The primary communication medium in this UC system is the IrDA (Infrared Data Association); however, the researchers also suggest that RF is a possible candidate of communication medium when the cost of RF equipment is lowered in the near future.

Initially, when a user clicks on the equipment-selection-button on a UC's LCD screen, the UC's lower layer will begin a search of the appliances via the IrDA. The appliances reply their icon data back to the UC and the UC displays the icons on its LCD screen. When a user selects an appliance, the device control menu is downloaded from the target appliance's ECP (Equipment Control Profile). In order to give a standard display of the device control panel, the UC screen is divided into Cells, Units, and Aspects. Each aspect is equivalent to a screen display. Figure 2-2-13 depicts the design.

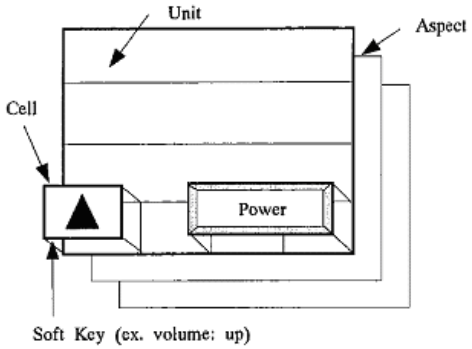


Figure 2-2-13: Cell, Unit, and Aspect on LCD [26]

For instance, each unit (row) can have many cells (boxes) and each aspect (screen) can have many units (rows). As shown in the figure, we have 3 aspects. Each aspect has 4 units and each unit has 4 cells. The volume-up button (a soft key) uses one cell whereas the power button occupies 2 cells. Once the aspect of a device is displayed on the UC, the user can control the target appliance accordingly.

After the UC decodes the data into GUI and operational commands on the UC's screen, a user can start controlling the target device. When clicking on a button, the entered information is translated into control command and the command is then sent to the target appliance via infrared rays. To control another appliance, the user just needs to repeat the equipment-selection process to obtain the GUI of the target appliance. The researchers address that the UC is designed to have a memory to cache up to 10 screen data (aspects) downloaded from ECP before. Thus, the data can be reused immediately if needed without a new download process.

2.2.3.3.2 Analysis

Briefly, the researchers state that the control system as "a model of peer-to-peer connection through wall barriers via Gateway Controllers (G/Ws)." Since it is in a household setting, if the system uses the RF instead of the IR, the coverage area would increase without the help of G/Ws. However, when in a large area, the G/Ws are needed for ensuring the quality of connections.

In the demonstration of the system, only one type of client with the same screen size is engaged and the appliance's ECP (Equipment Control Profile) only maintains control panel's data for one type. In reality, there are many different types of clients and an ECP should contain various control panel data of different types.

Another problem in this project is that the appliance does not give any feedback when a user initiates a control. The only way to know if the control was completed successfully is for the user to see the physical state change of the appliance. Because there is no state reporting from an appliance, a "high-quality" GUI, that is able to provide appliance's status and grey-out inappropriate buttons, becomes impossible. The lack of feedback and state reporting decreases the usability of this UC system.

2.2.3.4 Interactive Workspaces: iRoom

The authors believe that "Integrating Information Appliances into an Interactive Workspace" [27] is a "robust, infrastructure-centric, and platform-independent approach to integrating information appliances into the iRoom, our interactive workspace." The system is based on a middleware called IBM's TSpaces system [30]. TSpaces is a network communication buffer with database capabilities that provides group communication services, database services, event notification services, and URL-based file transfer services.

2.2.3.4.1 Introduction

The technology-rich environment and the "smart" devices are all around us nowadays; however, there are a few integrated multi-device computer environments existing. These multi-device computer environments are highly specialized and tend to target some specific applications. The researchers at Stanford were exploring a new way of integrating heterogeneous devices and systems in technology-rich spaces.

The authors were trying to construct a higher level operating system for the ubiquitous computing world. They came out with the framework for the Interactive Room (iRoom) infrastructure physically constructed during the summer of 1999. In iRoom, a user equipped with a PDA is able to browse the appliance's information and use the services of an appliance by a web browser application. The connection can be either wired (Ethernet taps and serial cables) or wireless via IEEE 802.11 base station. All communications are done via TCP/IP protocols.

In iRoom as shown in Figure 2-2-14, there are several screens for displaying information. A Java-based *room display manager* shows all the targeted devices' URLs on one of the large screens. A user can access the web content of these devices by clicking on their URL through a user's input device, such as a PDA. Unlike a general browser using the same screen for following a link, the authors introduce a *multibrowsing* technique that allows a webpage to be displayed on any of the large screens in the iRoom. To do so, for example, when a user clicks on a link specifying which screen is used for visiting this link, a *multibrowsing* event is generated and post to the system's event heap (based on TSpaces middleware). A simple daemon of the screen subscribing to this *multibrowsing* event is noticed and causes the screen's browser to visit a URL that is encoded in the event. When the target device's webpage is displayed on the screen via the HTTP technology, a user can see the information of the device and control the device by clicking on control functions. The control message is sent to the device through the HTTP GET and POST operations.

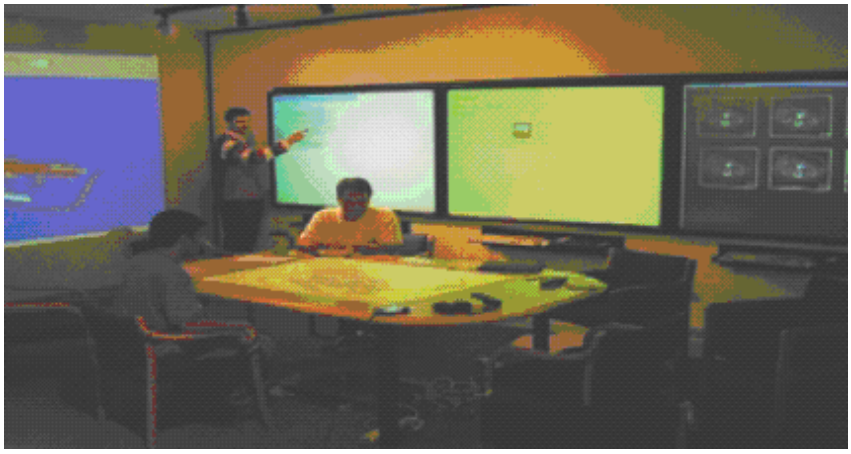


Figure 2-2-14: The Interactive Workroom (iRoom) [27]

One of the most important ideas in the iRoom design is the event heap, a software abstraction, based on TSpaces middleware. The event heap functions as a tuple-space-

like application. The authors think “an event heap resembles a traditional GUI event queue” but with some distinctions. These differences are:

- Multiple entities can subscribe to a given event stream. When informing the event to the subscribers, a multicast-style communication can be used. Thus, the event heap enables multicast-style groupware applications.
- If an event is not consumed, it will expire automatically and is eventually garbage-collected from the event heap. Furthermore, the event senders can send the event without knowing if the event receiver(s) is present.
- The event data structure is largely self-describing and extensible. Therefore, events can be subclassed without explicitly informing all entities about the changes of the event fields.

The key of the event heap design is that an iRoom application can post its events to the event heap and subscribe to a given event stream to be aware of other iRoom entities.

The paths through the event heap are shown in Figure 2-2-15.

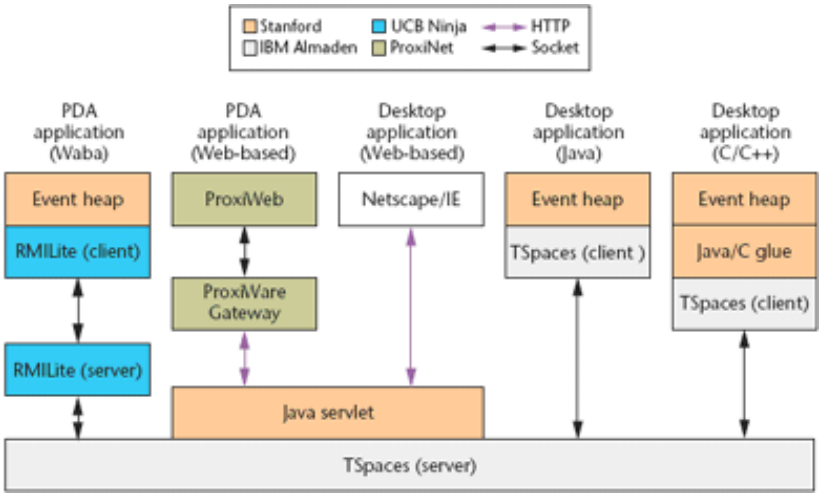


Figure 2-2-15: The various paths through the event heap [27]

Built on top of the TSpaces communication middleware, the authors implement different clients such a PDA or a desktop computer using HTTP and socket connections. Since TSpaces is a java-based middleware system, different types of clients are able to invoke TSpaces’ functions to get the event passing around. These implementations aim to demonstrate the usability of the event heap approach in iRoom. Furthermore, there is a workstation in each iRoom acting as a central server of the system on which TSpaces and event heap software run.

Since the project is event based, the event heap server can perform a multicasting and send the event at once to the subscribers. In the authors’ words, “the event heap enables the broadcast-style communication required for many group applications.” Moreover, because the communications among the devices (including controllers and service providers) are handled by the event heap application, iRoom engineers do not need to implement the group communication in their systems.

Finally, the event heap application does not implement any kind of authentication scheme and the authors suggest that a security system can be enforced in the application layer. Future work is needed to embed the security into this system.

2.2.3.4.2 Analysis

The iRoom project is an IP-based system that communicates via wired and wireless connections. Each iRoom has an event heap built on top of the TSpaces network middleware. An event heap application receives the events from iRoom entities and reports the events to the event subscribers, which means that each iRoom system must have a workstation running an event heap application. An iRoom system will not work if there is lack of a central server in the area. That is to say, if the server is not available or has failed, the iRoom environment will be ceased. Obviously, a single-point-of-failure becomes a problem in this system.

The authors created a technology called *multibrowsing* which "lets users construct Web sites in which following links can cause the destination page to appear on any of the iRoom's screens, not just the screen displaying the page containing the link." The *multibrowsing* technology actually is like having many frames to display HTML webpage and each frame means one large screen. When a user clicks on a hyperlink, the webpage will be opened in a target frame (screen).

By using an event heap, the system can multicast an event to all the event subscribers at once, which belongs to a broadcast-style communication. My personal omni-remote controller system is built on the M2MI/M2MP (Many-to-Many Protocol) layer and, therefore, broadcasting is the default the communication mechanism.

This system does not talk about the feedback of commands. Since the WWW protocols are used, I would think the system can constantly renew the webpage of the target server in order to get the server's current state.

2.2.3.5 Interactive Workspaces: ICrafter

"ICrafter: A Service Framework for Ubiquitous Computing Environments" [28] is "a framework for services and their UIs in a class of ubiquitous computing environments." The main goal of this system is to provide a flexible environment for users to interact with local services by using a variety of modalities and input devices. ICrafter is a framework in iRoom, discussed previously in the *interactive workspaces* [27] project.

2.2.3.5.1 Introduction

An interactive workspace refers to a technology-rich, physical location with wired or wireless interconnected computers (desktop, handhelds, etc.), utility devices (printers, scanners, etc.), and input/output (I/O) devices (microphones, speakers, etc.). People equipped with the portal devices meet in an interactive workspace to perform collaborative activities, such as a speech or slide presentation. The design goals of the ICrafter project are *adaptability*, *deployability*, and *aggregation*.

Due to the characteristics of an interactive workspace, heterogeneity can occur in both the device level and the workspace level. An ideal framework should facilitate the adaptation of the appliances and workspaces.

To achieve *deployability*, the system must be flexible in language, operating system, and UI toolkit. When a new service is added, an ideal system should be accessible without developers having to manually design a new UI for a client. That is, UI should be available whenever a user wants to access a service.

For *aggregation*, the framework needs to "facilitate the creation of user interfaces for combinations of services." [28] An ideal system is able to generate control of multiple services without manually designing a new UI, because a user often needs to control multiple devices simultaneously. For example, in the case of a slide presentation, it will be very convenient if the system can aggregate the lighting controls with the slide show controls. That is, a combined function can turn off the light and then start a projector instantly. Another example is that an ideal system should be able to tell whether or not a camera and a printer are compatible, and if they are, pictures in the camera can be sent to the printer for printing. However, to make all the functions of the devices match is impractical because of the number of the combinations of the possible functions. Therefore, in this project, the authors focus on the service aggregation that creates UIs without the creation of composite services.

Initially, each service provider beacons for its presence and the beaconing information, which is a *service description* (such as the services supported by a server), is stored in an infrastructure-centric component: *Interface Manager* (IM) located at the central server of an interactive workplace. A user (client) device can request the UIs for service(s) from IM. Let us look at Figure 2-2-16.

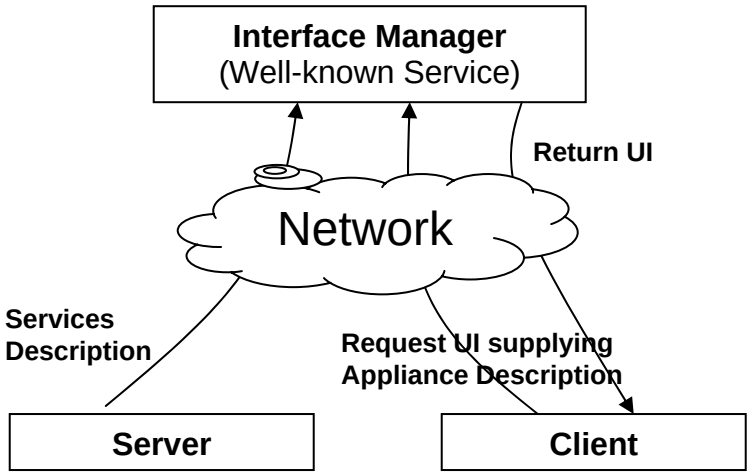


Figure 2-2-16: ICrafter Architecture [28]

As shown in the figure, a client requests the UI of one or many services from an IM along with the client's *appliance description*. The appliance description describes the

specification of a client, such as the screen size. When the IM receives the request, it selects an appropriate UI generator based on the user's appliance description and required services. Then, the UI generator is executed and has access to the user's *appliance description*, the *service description*, and the *context memory*, which is the information about the workspace context. (These data are represented by using XML.) Then, the generator creates the UIs and sends them back to the client (requester). Thus, the client can use a web browser to access these services as in iRoom [27].

As we discussed before, an EventHeap that is similar to a traditional GUI event queue is an event-based communication system used by all iRoom applications [27]. Clients and servers can post and get events from the shared EventHeap.

Initially, all servers post a short-lived bootstrap event to the EventHeap, which serves as a bootstrap server, to indicate their presence. An IM that queries all these bootstrap events is aware of these available local services. When a client enters an iRoom, it first requests the IM's UI that allows a user to select one or more services and requests the UI for them. After the user selects the wanted services, the request of a UI is sent to the IM. The IM then uses the known information to generate a corresponding UI as described previously. After a while, the UI of these services is sent back, and the client displays it on a web browser. Thus, the user can use the wanted services via the UI.

2.2.3.5.2 Analysis

The ICrafter project is an IP-based system that communicates via wired and wireless connections. In an iRoom with an ICrafter system, a critical technology is an EventHeap that is built on top of TSpaces, a network middleware. An EventHeap application receives events from ICrafter entities and reports the events to the event subscribers. Thus, in each ICrafter environment, there must be a workstation to host an event heap application. As we mentioned in the iRoom project, when the workstation is not available or has failed, the ICrafter system will definitely stop working. Therefore, a single-point-of-failure problem will become a problem in this system.

The authors state that "adding a new appliance must not force writing UIs for that appliance for all existing services." Thus, fully or partially automatic UI generation is desirable. However, as I described before, automatic generation of UIs will cause problems when the service functions become complex.

An *Interface Manager* is used to generate the specific UI of services for a requesting client. In this way, different types of clients will have a UI devoted to a certain client. My project uses a module approach, by which for each type of client, there will be a devoted UI. Thus, the UI of a certain type of client is guaranteed to be compatible and graceful with the UI design. Furthermore, in this project a central server is required to host the *Interface Manager*, and the quality of the generated UI may not be guaranteed. It becomes difficult when an *Interface Manager* needs to support various types of client devices, such as vocal clients.

Similarly to the iRoom project, this system can use a broadcast-style communication, in which an event can be multicast to all the subscribers at once by the EventHeap technology. In contrast to these two projects, my system uses broadcast-style communication by default because it is built on top of the M2MI/M2MP layer.

2.2.3.6 HP COOLTOWN Project

The HP-COOLTOWN project describes "a vision of a technology future where everything – people, places, and things – has a web presence." [9] The authors believe that in the future everything will have a web presence and people will use mobile devices, such as a PDA, to access services from their websites via the WWW protocols. The creation of this mobile service platform will bring a new revenue stream for the industry of e-service, infrastructure, and appliances.

2.2.3.6.1 Introduction

Currently, the convergence of increasing availability of highly functional portable devices, deployment of wireless networking options, and the explosion in the number of services offered over the WWW make the online population larger and larger. Through a mobile browser, people can enjoy the services and convenience brought by the WWW no matter where they are. This trend inspires the HP-COOLTOWN project, in which everything is connected to the Web through wired or wireless connections. The researchers link the physical world and the digital world together, and their main idea is "any device can become a resource or a service to anything else by simply connecting to the Web."

HP saw the trend of the popularity of portable devices, the wireless network, and the WWW. In the COOLTOWN project, researchers assume that people, places, and things all have their web presence. Wherever the mobile devices are, they can always get access to the web presence of the physical unit. For example, when a user with a mobile device enters a bookstore, he or she not only can enjoy reading the physically presented books but also enjoy the services provided by the bookstore's webpage, such as searching for the location of a certain book or retrieving the book summary. In order to realize the notion of everything-has-a-web-presence, a standard infrastructure is essential.

In COOLTOWN, every place has *a place manager* that is an infrastructure component providing a web representation of the place. A place owner uses the place manager to create a custom webpage and to manage all the virtual services provided by the place for their customers. In the bookstore example, when a book is out of stock, it can be ordered by the customer through its website and shipped immediately without customers waiting in line for a store clerk to help.

A place manager actually is a combination of a resource inventory manager and a web server that manages the location of customers and staff, the current time of the store, the identity of the customers accessing the place, and the capability of the customers' interactive devices. A place manager with *place directory* contains the *entity descriptions*

of people (staff and customers) and of things (services). Before an entity can register itself to the place directory, it must know how to access the place manager. In the COOLTOWN project, the most commonly-used method for service discovery is to have the place manager beacon its homepage's URL periodically. When a device enters the place, it will pick up the beacon that contains the place manager's URL.

Once a device knows the URL of the place manager, it can register its entity description. To do so, an entity needs to invoke an HTTP Get operation to the place manager and supply the entity's URL of the XML description as an argument. Then, the place manager uses the entity's URL to retrieve the entity's description and caches the description in the place directory. The current state of the place directory will always reflect the current state of the place at any given moment in time. Furthermore, the entity can delete itself from the place directory as well.

Some services are publicly accessible, and some should only be limited to certain people. For example, the services of providing bus information in a bus station should definitely be public. However, bank services should be limited to registered bank customers. To protect the security of non-public information, an advanced security scheme is critical. Because each place may enforce different security policies on how its website should be accessed, it is the place owner's responsibility to make the rules while creating the webpage.

2.2.3.6.2 Analysis

Everything in COOLTOWN has a web presence. To use a certain service, a user can browse the appliance's website and interact with the webpage accordingly. This is the only UI supported by the COOLTOWN project. However, using browser is only one type of UI. A comprehensive system should be able to support different types of UIs, such as a vocal control.

This IP-based project uses WWW protocols to communicate, and RF is the communication medium. Thus, this system has to handle the person-across-the-world-turns-off-the-light problem as discussed in the Universal Interaction project. For devices to communicate with each other, there will be a series of two-way communications. Compared to a one-to-one communication, the M2M architecture is much more efficient. For example, in a bus station, the place manager needs to notify all the clients of the updated information about bus locations. When using a one-to-one communication, the place manager has to send the same information to each client. If the M2M architecture is used, the place manager broadcasts the updated message once in a while, and all the nearby clients will get the information.

In the project, the web representation of a place will automatically show up on the browsing devices at the time when people with a client device step in the place. Instead of automatically offering the place's webpage, I believe it should be changed to "available upon request." The website shows up only when customers request this service. Otherwise, it will be just like the popup ads that annoy so many people.

The place manager is like a central server responsible for all activities in the place, including client authentication. If the place manager fails, the whole system becomes unusable. Also, as the number of clients increases, the client-server design will face a bottleneck problem, which is when the server is either unable to handle, or slowly handles, the huge amount of work. Another problem is that a central server is often a target for hacking.

This system uses server beacon for service discovery. A client entering the place will receive the place manager's beacon and then is able to register its URL to the place directory. In my project, I use both a *client beacon* and a *server beacon*, which makes use of both advantages of the two approaches. The researchers do not mention the feedback design in the COOLTOWN project. Since WWW protocols are used, I would think the system can constantly renew the webpage of the target server in order to obtain the server's current state. The system collaboration issue is also not discussed in this system.

2.2.3.7 Pebbles

The CMU's Pebbles project [3], as described on their project website, involves "exploring how handheld devices, such as PDAs including devices running PalmOS or Pocket PCs, and mobile phones, can be used when they are communicating with a 'regular' personal computer (PC), with other handhelds, and with computerized appliances such as telephones, radios, microwave ovens, automobiles, and factory equipment." This is a huge project, in which on of its subdivisions called Personal Universal Controller (PUC) [4, 5, 6, 7, 8] is closely related with my thesis and will be discussed in the following section.

2.2.3.7.1 Introduction

Currently, computerized appliances and portable hand-held devices are pervasive in our daily life. The more computerized these devices are, the more complicated their interfaces become. If one built-in button is associated with one appliance's function, hundreds of buttons are needed on those appliances with complicated functions. Thus, the built-in, static interfaces on a complex appliance will not be an efficient design. Fortunately, with the invention of the touch screen and GUI, complex appliances can still be very easy to use. It is foreseeable that everyone with a portable device, such as a PDA, will be able to interact with the rest of the electronic world in the near future. In this project, the authors proposed a Personal Universal Controller (PUC) system that supports the interaction between portable devices and computerized appliances.

In "Generating Remote Control Interfaces for Complex Appliances," [4] the authors introduce the Personal Universal Controller (PUC) that is used to control all kinds of equipment at home, in the office, or in factories. The PUC system is designed to be more user-friendly, compared to the built-in buttons, by providing a user with an intermediary graphical or speech interface to control appliances. This is a two-way communication system, and both infrared and RF are used as the communication medium. The basic idea

of the system is when a user, via infrared, points his or her PDA at a target device, such as a light switch or photocopier, the target device will send its control parameters in XML format back to the PDA. The PDA uses this information to generate a UI of the target device and to display the GUI on the PDA's screen for the user to easily control the target device. Furthermore, GUI is not the only option for a user to control a device. In *Universal Speech Interfaces project*, the control parameters of a device are used to generate the vocal interfaces.

The architecture of the Personal Universal Controller (PUC) is shown in Figure 2-2-17.

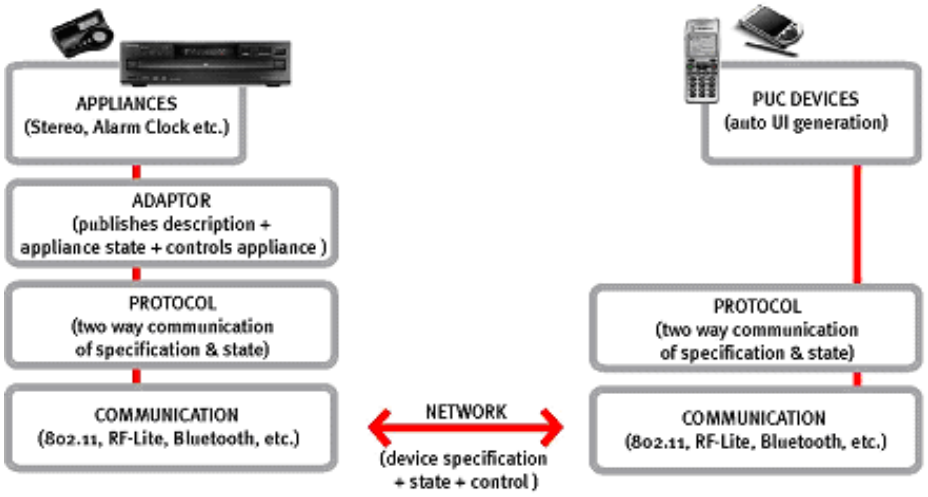


Figure 2-2-17: An architectural diagram of the PUC system showing one connection (multiple connections are allowed at both ends) [4].

In the PUC system, an asynchronous and bidirectional communication scheme is adopted, which allows a PDA to update an appliance's interface and to provide feedback to the user. For example, if a TV is currently on, the TURN ON buttons shown on the interface of a user's PDA will gray out because the TURN ON command does not conform to the state of the TV. This utility is achieved through *the dependency information*. Each function interacting with the appliance has its own dependency information relative to the state of the appliance. That is, graying out a button of GUI is possible by having the feedback from the appliance and knowing the dependency information of the function. For example, the TURN ON function of an appliance has its dependency information indicating that TURN ON is available when the appliance is currently off, and TURN ON is not available when the appliance is currently on. By having the dependency information, the appliance's GUI becomes more functional and usable.

In order to group related functions together while generating GUI, the authors create *a group tree* that is a hierarchical grouping of all appliance functions. The group tree groups together similar elements of the appliance's functions. For example, the Play button, Pause button, and Stop button of a VCR are grouped together in a VCR's group tree. While generating the GUI, these functions are able to be placed together by the grouping information.

The authors create a XML-based, PUC specification language that includes the information of appliance description, dependency information, and group tree. While generating the GUI on a PDA to control an appliance, a graphic interface generator takes a description of a function at once and uses the group tree and the dependency information to create UI.

The researchers in the Pebbles project found that "people using an appliance interface presented on a handheld computer performed the same set of tasks in half the time while making half the errors as compared to using the appliance's built-in control panel." [6] GUI will definitely increase the usability of appliances.

2.2.3.7.2 Analysis

To control an appliance, a user needs to point his or her Personal Universal Controller (PUC) to the target device to download the device description via Infrared signal. The same approach is found in the project called "Autonomous and Universal Remote Control Scheme," [29] which we introduced and analyzed previously. We all know that infrared, as a communication medium, is not the best choice when the target device is behind an object or sits in another room. We want to be able to find appliances in the near vicinity without the hassle of pointing our PUC around. Fortunately, a target device can be found in the vicinity without this hassle by using RF. Since the system uses RF for control, I think RF should also be used for service discovery as well.

In this project, a series of two-way communications is used for the controller and the appliance to talk, which is better than those systems performing one-way communication, such as the systems in [14, 15]. However, issues such as communication overhead and system inefficiency are presented. For example, if a TV is controlled by three controllers at the same time, there will be three two-way communication channels between the TV and the controllers. When reporting the state of the TV to the three controllers, the same status message has to be sent out three times. In my system, instead of two-way communication, the M2M architecture is used as the underlying communication scheme, which will require less communication. The M2M scheme becomes more efficient while working on multi-user or system collaboration topics.

In order to provide a "high-quality" GUI that conforms to the appliance state, the system uses the dependency information of a function to update the GUI. That is, the dependency information is used to decide if greying-out a button is necessary or not. In my system, instead of having the dependency information, the controller will execute its device logic, each time upon the arrival of appliance's status, to update the GUI and to ensure the consistency of UI.

The authors think that "the device (equipment) will not need to dedicate much processing power, hardware, or cost to the UI, since it will only need to contain a description of its capabilities and storage for the current settings, along with hardware for wireless communication." Since the device has all the characteristics needed for a smart

device, instead of considering how we can minimize the equipment on the appliance, we should try to maximize the functionality without caring too much about the hardware.

2.2.3.8 Autonomous and Universal Remote Control Scheme

"Autonomous and Universal Remote Control Scheme" [29] describes an autonomous and generic remote control system between information appliances and remote controllers.

2.2.3.8.1 Introduction

Information Appliances (IAs) now become an inseparable part of our lives. The further development of IA technologies and their competitive price will greatly increase people's dependence on IAs. However, as we mentioned before, if one IA comes with one remote controller, many remote controllers for various appliances will not bring us more convenience, but rather trouble. To solve this problem, the authors proposed the autonomous and generic remote control scheme that describes a framework for the interactions of remote controllers and IAs.

This universal remote control scheme contains Information Appliances and Generic Remote Controller (GRC). Information Appliances is installed with Generic Embedded Controller (GEC) that is able to communicate with other appliances via RF and infrared rays. Generic Remote Controller is equipped with a LCD touch screen, a wireless communication component, and an IR.

Via infrared, the GEC of an information appliance periodically sends out Unified Identification (UID) that contains the device's identification and IP address, such as "JavaTV140.116.086.177." When a user with a GRC walks to the front of this appliance, the GRC picks up the appliance's UID via infrared, and the GRC knows the appliance's ID and IP address. The GRC can handle many UIDs from various IAs, and it is up to the user to decide which IA to control. So far, the GRC still does not yet know how to control IA. After the user selects an IA, the user's GRC interacts with the IA's GEC to retrieve the IA's context—a specific interface for interoperation between GEC and GRC. This step is done through distributed computing via wireless network, in which the GRC performs a remote procedure call (RPC) for retrieving the context through GEC's wireless interface. In this system, they use XML for the appliance's context description. After receiving the request, GEC publishes its context via wireless interface, and the context is received by the GRC.

The GRC has a context translator implementing an interpreting function by using compiler technology. It uses a context translator to transform the XML context and to display the GUI of the context on its LCD screen. When a user clicks a button on the screen, the GEC performs a remote procedure call of the command via a wireless network. The IA's GEC receives the call and performs the command. After finishing the action of the command, the GEC sends back a reply signal to the caller. The GRC receives the feedback and displays the message on the screen.

The authors developed several prototype systems to demonstrate the generic framework, and the systems were able to show the viability of the autonomous and universal remote control scheme. That is, when a GRC moves to the front of different servers, the GRC gets the appliance's UID, downloads the XML context, and displays the UI of the appliance on the GRC's screen.

2.2.3.8.2 Analysis

The authors believe that information appliances will sooner or later dominate people's lives. However, as we all know, one remote controller for one appliance is certainly not the best solution. Thus, a personal omni-remote controller system is essential for people to enjoy the convenience brought by information appliances. In this project, the authors use infrared as the medium of service discovery. However, as I described previously regarding the limitation of IR, I believe RF as communication medium will be the better choice. Since each information appliance has a Generic Embedded Controller (GEC) with a wireless component, wireless communication should be used for both service discovery and appliance control in this project.

In order to discover an appliance by this control system, a user needs to bring his or her omni-remote controller to the front of the appliance to receive the appliance's UID, which is apparently not very convenient. In my project, I use RF as the communication medium for service discovery. Thus, a controller is able to find all the desired appliances in the vicinity without any trouble.

In this system, when GEC receives GRC's command, GEC sends the control signal to the hardware and sends the replay signal back to GRC as feedback. As compared to the feedback design in this project, the system in my project does not explicitly send feedback to a command; instead, the server of the control system has a daemon thread broadcasting its current status periodically in a short time interval. By checking the current state of a server, a client is able to know if its command is in effect or not. By that, a multi-user system becomes possible and a device's status shown on a client's controller will not be out of synchronization due to another user's command.

In this system, the GUI of GRC is automatically generated from the IA's XML context. However, as the IA's control function gets more sophisticated, the GUI generation might not be able to lay out correctly. Also, because different client devices may have different characteristics, the auto-generated GUI can hardly match their different specifications. Another problem of this system is that the omni-remote controller would not know the state of the IA due to the lack of a state reporting system of IA in their design, which may cause some problems or confusion, such as sending out the "turning-on" command while the appliance is already on. This is an IP-based system, and its communication architecture adopts a one-to-one scheme.

2.3 Summary

With the popularity of portable devices, the advance of computing equipment, and the improvement in wireless technology, researchers aim to merge the ubiquitous systems that will bring people into a technology-encompassed world. A future picture will be one in which everyone carries a personal device that does all the functions needed for everyday tasks. An example is the smartphone device that combines the functions of a cell-phone and a PDA. Based on my literature search, I have summarized the characteristics of the omni-remote controller systems and grouped them into four areas: Service Discovery, Architecture, Type Hierarchy, and Security.

Service Discovery

There are two major approaches for the service discovery: *with bootstrap server* or *without bootstrap server*. When the system is designed to work with a (local) bootstrap server such as [19, 21, 25, 28], all the service providers have to register themselves to the bootstrap server with their bootstrap information. Before a service provider can register itself, the bootstrap server might check for the provider's identity. When the bootstrap server accepts the service provider's bootstrap, the service provider's information is kept in the bootstrap server for a certain time period. If the service provider wants to have its bootstrap entry continuously stay in the bootstrap server, it has to renew its bootstrap information periodically. From the bootstrap server, the clients can find out the service type and the desired service provider to interact with. Or, the bootstrap server can notify the subscribers when the desired service becomes available.

When the system is designed to use without a bootstrap server such as [24, 29], a client is either passively waiting for a service provider's beacon or actively requesting a service provider's beacon. When discovering a service actively, a client points its infrared transmitter to the target device to begin downloading the service provider's bootstrap information. When discovering a service passively, a client periodically receives the bootstrap from a service provider's beacon while passing by the provider device.

My omni-remote controller system is designed to perform service discovery without a bootstrap server, because, I want to design a system in which a client and a server can interact without the presence of the third unit. (If a bootstrap server is required, an omni-remote controller system will become less flexible.) Also, if there is no central bootstrap server, a control system will be excluded from the following issues:

1. The bootstrap server failure (the single-point-of-failure problem)
2. The overcrowded bootstrap server (a bottleneck problem)
3. A perpetrator's attack such as "denial-of-service."

My system adopts both *client beacon* and *server beacon* techniques. That is, clients can receive servers' bootstrap information by either actively asking for it (*client beacon*) or passively waiting for it (*server beacon*).

Architecture

As we can see in many of the low ambition projects, the architecture is quite straightforward. One command from the controller triggers one action in the server. As the ambition of the project grows, the design of the system becomes more complex

because these projects try to integrate heterogeneous systems by providing universal standards. Fortunately, the advance in hardware and software technologies really helps a lot while designing the omni-remote controller system. For example, in order to join the omni-remote controller system, the appliance has to be a smart device that is able to compute instructions, save its states, and communicate wirelessly in addition to its original functions. Thus, an extra hardware embedded in an appliance is needed. Furthermore, Java cross-platform ability provides us with a universal platform to build an omni-remote controller system on top of it.

Communication

The communication method includes Infrared, wired, and wireless. The use of one or more of these methods has been found in the research projects. When IR is used, it is mostly for its well-developed technology and competitive price. When wired is used, it is primarily for connecting the system's infrastructure such as [26]. When wireless is used, it is aiming to use the properties of the radio waves such as the covering range. For instance, in "Autonomous and Universal Remote Control Scheme," the authors use IR for the service discovery phase and RF for downloading the server's description and later control [29]. Furthermore, Internet-IP or domain-name-based systems seem to be very popular among the omni-remote controller systems such as [9, 18, 19, 24]. I believe this is because of the existing TCP/IP infrastructure. Most of the researchers want to take advantage of the existing architecture and avoid writing their own routing protocol.

User Interface

UI plays an important role between a user and a service provider in the omni-remote controller system. The omni-remote controller system should be designed flexible enough to adapt to various types of input systems such as static button, touch screen panel, voice recognition application, and eye tracking. Furthermore, when GUI is used, depending on the approach the researchers used, the GUI display can be 1) user configurable [16], 2) automated generation [3, 18, 19, 21], or 3) downloaded from a third party such as the *Interface Manager* [28].

Designing a flexible system that is compatible with the ubiquitous input devices is the goal of the researchers. The same thing applies to my framework. Since my system is built on top of the Java platform, as long as the appliance has a Java platform installed, it should be able to interact with the omni-remote controller application. For the GUI part, different from the other projects, my system uses a module approach that preinstalls the control module in the client device. When a user wishes to control a type of the device, the control module of that type is brought up. (See Section 3 for details.)

Agent-based System

There are researchers focusing on the subject of the *Agent-based system* [19, 21]. This type of omni-remote controller system aims to simplify a user's work by taking his or her one input (click) and based on the semantic meaning of the input completes the whole task, such as recording a TV show. The process often involves several different devices. For my system, the agent enhancement is something that needs to be investigated in future work.

Server's Description

Before a client can interact with a server, the client needs to know how. There are two ways to do so. One is *the webpage-based control* and another is *the context-downloading-based control*. When it is a webpage-based system, each server has its own web representation; HP-COOLTOWN [9] is one of these projects. The client can access the service through the server's website. When it is a context-downloading-based project, the system would have a server's *description language* that depicts the functionalities of a server. Based on the information describing the service, the client would know how to interact with the server. Many systems use Extensible Markup Language (XML) as their server's *description language*, such as [3, 18, 19]. For others, they would create their own *description language* to specify the functions of the device. More advanced projects define the device function in a hierarchical way, that is, specifying the common elements of the devices [4, 24, 25]. In my project, the system defines the common elements of the devices in a hierarchical way. See the description in the following Type Hierarchy section.

Type Hierarchy

There are a few projects that use a hierarchical structure to represent the common elements of the appliances. The Universal Interaction [24] uses the *Interface Definition Language* (IDL), the Universal Interaction Appliance [25] uses the reference platform, and the Pebbles project [4] uses the group tree. All of them are trying to define the common functions of the devices. By doing so, the devices at least have a common structure that can be used to identify each other. Moreover, the controller can use the description to generate the UI for the user.

In my system, I use an object-oriented type hierarchy approach and build a hierarchical structure of the devices that I have implemented. For all the clients and servers to join my omni-remote controller system, they have either to implement the common interface or to extend the existing interface and then implement it. For example, a TV and its remote controller (client-server pair) can either implement the existing TV server and client interface respectively or extend the existing TV server and client interface to form a new client-server pair such as the Sony-TV server interface and the Sony-TV client interface. My programming project was implemented in Java which provides the function of the Object-oriented type hierarchy. More detailed discussion is in Section 3.

Security

Security is the part that hasn't been discussed a lot in most of the projects I have found. I suppose the reason is because people are still in the initial phase, finding out how to make the omni-remote controller work in the first place. When the projects mention security, they often say that the standard cryptography-based security of encryption and authentication can be used such as "Composable Ad-hoc Mobile Services for Universal Interaction." [24] In "An Agent-based Bidirectional Intelligent Remote Controller," [19] the security is handled by the underlying mobile agent platform.

Since my project is an application built on top of the M2MI layer, if the M2MI becomes secure it would also promote the above application layer [33]. My project aims to show the controls over various devices; the application-level security is not a part of the demonstration. The various security issues remain for further work.

3 A Generic Framework for the Personal Omni-remote Controller using M2MI

This section depicts *a generic framework for the personal omni-remote controller using M2MI*. I am proposing an omni-remote controller system that could be an auxiliary application in wireless devices, such as a PDA, cellular phone, wireless PC/Laptop, and so on. Every appliance (servers) supporting the omni-remote controller system can be controlled by the omni-remote controllers (clients).

3.1 Three Use Cases

This section describes what a user wants to do with the omni-remote controller and how people use the appliances through the omni-remote controller. For representing a broad range of applications, three use cases—the thermostat control, the TVs with parental control, and the washing machine control—are chosen.

3.1.1 Thermostat

The thermostat is one of the most common appliances in buildings and houses. In order to control or monitor a thermostat, a user usually has to go the location of the control panel. However, it will be much more convenient if we can use a wireless device, such as a PDA, to control and monitor the thermostat without being physically present in front of its control panel.

Through the use of a wireless remote controller, a user expects the system to support the function of status reporting and to have a friendly UI. For example, a PDA remote controller of a thermostat should be able to display the current room temperature and the current set point of the thermostat. Also, when a user wants to adjust the room temperature, he or she should be able to click on the temperature-up or -down buttons on the PDA's touch screen. Basically, the wireless remote controller should provide all the control functions that a built-in control panel usually has.

3.1.2 Parental TV

Parental TV gives parents the power of controlling which channels their children are allowed to watch. When parents cannot watch TV with their children, they often feel unsafe to let children switching channels freely because of some inappropriate contents. If there is a type of TV system that can prevent children from watching protected channels or the system can notify parents when children want to watch protected channels, that would be a great help.

When an omni-remote controller system is engaged for parental TV control, a user expects more functionality other than what a common TV remote controller often provides. For instance, if a child tries to go to R-rated channels, the parental TV system will automatically deny it. When, he/she tries to choose to PG or PG-13 rated channel,

parents will be notified through their PDA and then decide if their child can watch the channel or not by clicking on the YES or NO button on their PDA. If the answer is yes, the parental TV system will switch to the requested channel and parents go to watch it together. If the answer is no, the parental TV system cannot switch the channel and the child's request is denied.

3.1.3 Washing Machine

Dorm residents often find themselves in a laundry room without an available washing machine, especially during weekends. It will be more convenient if dorm students can check the availability of washing machines before they go there. Also, if no washer is currently available, students should be able to register their load(s) for the next available washing machine.

When engaging an omni-remote controller system, students expect to know the availability of the washing machines in a laundry room. For example, while watching TV, Alice sees her piled-up clothes and decides to get it all cleaned up. Before going to do her laundry, she wants to make sure there is a free washing machine in the dorm's laundry room. Thus, she grabs her PDA and sees if there is an available washer. When there is one available, she can reserve it and go to do her laundry. If none of them are available right away, she should be able to register her load(s) to the group of washers in the laundry room and wait for the next available washing machine. The omni-remote controller system will notify Alice her if it's her turn. Furthermore, it will be great if the omni-remote controller system allows people on the waiting list to swap their waiting positions. For example, when Alice puts her name on the waiting list, she sees her best friend Bob's is in the first line. Because Alice has an appointment with her classmates soon, she wants to swap the waiting position with him. She then sends out a request to Bob and Bob agrees. Now, Alice can do her laundry right after the next washer from the group becomes available.

3.2 High Level Architecture

This section introduces the high level architecture of my proposed personal omni-remote controller system. I will describe the architecture issues of an omni-remote controller system in Section 3.2.1. Consequently, Section 3.2.2 depicts a generic, personal omni-remote controller system. Finally, a scenario is given to illustrate the proposed system in Section 3.2.3.

Before proceeding, for each client and server device to join the omni-remote controller system, it must be so *smart* that it is able to 1) compute instructions, 2) save its states, and 3) communicate wirelessly with others using broadcast communication schemes. The broadcast scheme is important because one broadcast message from an entity can be received by other entities in the vicinity. Even though most of the contemporary devices are neither able to save the state nor to communicate wirelessly, the goal of having smart devices is not too far away with the rapid advance of new technology.

3.2.1 Architectural Issues

This section is organized around the architectural issues for an omni-remote controller system. These issues include the type of communication, the type of server, and the type of client. Each of them will be discussed in the followings.

The Type of Communication

There are two types of wireless communication commonly-used in an omni-remote controller system—point-to-point and broadcast. When engaging the point-to-point communication in an omni-remote controller system, one message is sent out to one receiver. For example, a control message is sent from a client to a server and a feedback message is sent back from a server to a client. However, when broadcast communication is used, one message can be received by all the nearby clients and servers. For example, a server's current-status message is broadcast and all the interested clients and servers can receive this message.

When communicating wirelessly, there are two kinds of communication mediums often used—IR and RF. Currently, when clients and servers want to talk to each other, a series of two-way, point-to-point communications are often engaged like the cases in the Pebbles project [3] and in the XWeb [18]. Although RF is used in these projects, they do not take advantage of a broadcast communication. In these projects, one message is sent out to one receiver, although a broadcast message can be received by all nearby entities. Thus, how to take advantage of the RF as a communication medium? For example, in the mobile agent project [21], instead of sending out the same state-messages to each mobile agent platform, the state-message can be broadcast once and every interested platform in the vicinity will receive it.

An omni-remote controller system is often used for controlling appliances in a local area. Thus, when choosing a communication medium, we need to consider if the medium can easily cover the operational range of a local area, such as a house. Furthermore, all the clients, such as PDAs, and servers, such as TVs, in this area should be able to communicate with each other via this medium although some obstacles may exist among them. Therefore, RF is an ideal choice because RF can pass through walls and obstacles but IR cannot.

The Type of Server

There are two types of servers—*specific server* and *generic server*. A *specific server*, such as SONY-SU27 TV, only provides the services to its corresponding clients, such as the remote controller of SONY-SU27 TV. When various types of clients try to discover services, a *specific server* only responds to its corresponding clients. For example, GE Light-100 server will only respond to GE Light-100 client and Toshiba VCR-200 server will only respond to Toshiba VCR-200 client.

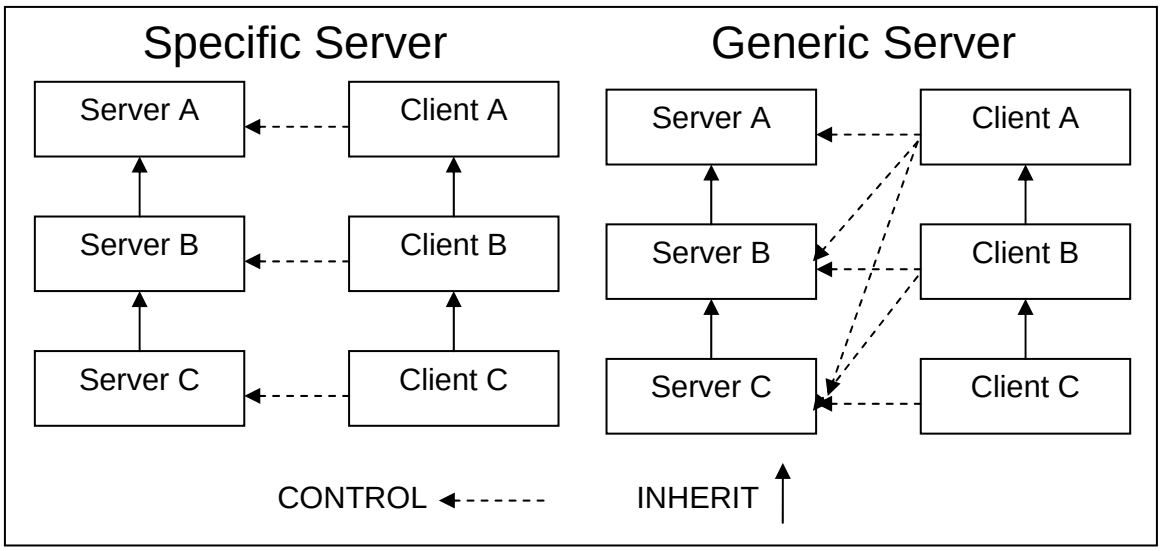


Figure 3-2-1: Specific server verse Generic server

A *generic server* can be fully controlled by its corresponding client and also can be "partially" controlled by the clients of the *super classes* of this server. As shown in Figure 3-2-1, Generic Server C can be fully controlled by Client C and partially controlled by Client A and B because Client A and B know the partial functions of Server C that are inherited from Server A and B. (See **Type Hierarchy** for details.) When various types of clients do the service discovery, a *generic server* will respond to its corresponding client as well as the clients of its super classes. For example, a dimmer-light server "extending" the functionality (interfaces) of a light server will respond to a dimmer-light client as well as a light client.

Even though the light client can discover the dimmer-light server, it does not "know" how to control the dimmer light server beyond the functions of a light server. For instance, if the light server only has TURN ON and TURN OFF functions, the light client can only turn on and turn off the dimmer light server although the dimmer light server can also be dimmed. (A dimmer light client fully controls a dimmer light server.)

In an omni-remote controller system, we might want to have both types of servers depending on the usage. If a server is designed to only be controlled by its corresponding client, a *specific server* is used. If a server is designed to support both fully and partially controls of its functionality, a *generic server* is necessary. For example, there is a type of ice maker having both the power on/off and temperature set-point function. A worker whose job is to turn on the ice makers in the morning and turn them off in the evening only needs a client that can operate the power on/off function of ice makers. In this case, a *generic server* for the ice makers is needed because the system needs to support partial control of the ice maker. However, in the case that these ice makers are only controlled by administrators, a *specific server* is used.

The Type of Client

There are two types of clients—*specific client* and *generic client*. A *specific client* is a devoted controller to a certain type of server. For example, a Sony-V100 TV remote

controller is designed for controlling Sony-V100 TVs, but not other models. In my proposed system, an advanced type of server is built up by combining several common types (components) through Object-oriented Type Hierarchy and each type has its corresponding client.

In the ice maker example, a worker, who is responsible for turning ice makers on and off, only needs a *specific client* that implements the TURN ON and TURN OFF function. No other control functions are needed. In the same example, when fully controlling these ice makers is required we will implement the ice maker's corresponding client as the *specific client*. Depending on the required (partial or full) controls on devices, a *specific client* can be implemented to either fully or partially control a server type. The similar idea is used in XWeb [18].

A *generic client* controls different types of servers through their control modules. Each control module can be treated as a *specific client*. The main job of a *generic client* is to find out all the nearby servers and bring up the corresponding control module of a certain server type when a user requests. Initially, a *generic client* finds out all the servers in the vicinity and notices the user the available service types such as TV, light, and so on. When the user selects one from these service types, the *generic client* brings up its corresponding control module (if available) from the *module repository*, that is a place to store all the control modules. The selected control module then provides a UI for controlling this type of servers. Thus, through the use of control modules, a *generic client* becomes a real omni-remote controller.

In the ICrafter project [28], the authors use an *Interface Manager* to achieve the same objective as the control module approach does. An *Interface Manager* uses the description of a client and the *service description*s of the target server(s) to generate a suitable UI for the client. The difference between the two approaches is that an *Interface Manager* engages an auto-UI generation while the control module is manually created. For the *Interface Manager* approach, a central server hosting the *Interface Manager* is necessary and the quality of the generated UI cannot be guaranteed.

3.2.2 Architecture

Built on top of the M2MI layer, the personal omni-remote controller system is a broadcast-based system that consists of a number of client and server devices. In Figure 3-2-2, a high level software structure of the system is shown. As seen in the figure, the omni-remote controller client application and the omni-remote controller server application operate on top of the M2MI layer, and M2MI is based on Many-to-Many Protocol (M2MP)—a broadcast protocol. This software structure applies to the M2M connections, in which clients and servers can talk to each other at the same time.

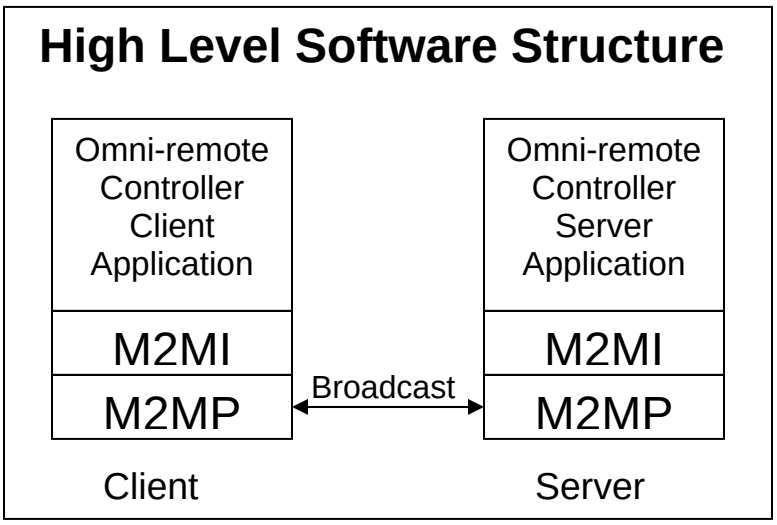


Figure 3-2-2: High level software structure of the personal omni-remote controller system

This system is built on top of M2MI layer that is based on M2MP [1]. M2MP is a message protocol that "broadcasts messages to all nearby devices using the wireless network's inherent broadcast nature instead of routing messages from device to device." This means the protocol layer broadcasts a message and all the entities in the vicinity can receive this message. In M2MI layer, when an object makes a method call, every object out there implementing the interface will call the method. To be specific, when an object makes a M2MI method call, the call is broadcast through M2MP layer, and all nearby entities implementing this method call will be invoked. Applications preferring the M2M approach are multi-user application and collaborative middleware systems, such as chat rooms, group games, neighboring file systems, combat fields, and omni-remote controller systems [2, 31].

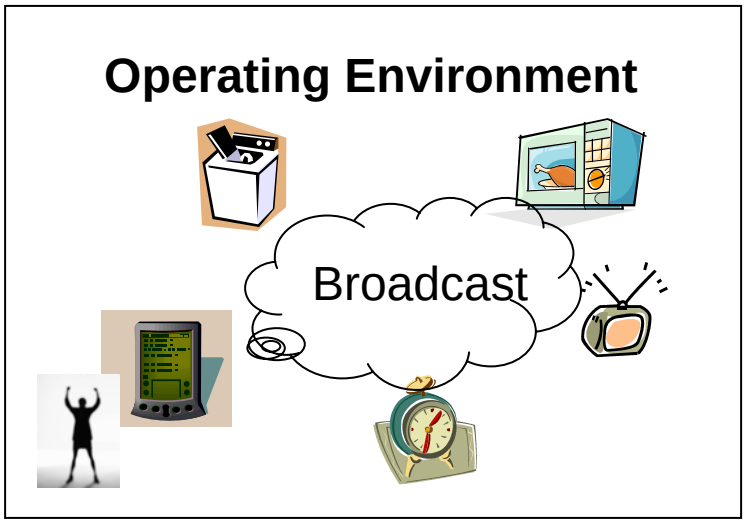


Figure 3-2-3: The operational environment

The reason for using RF as the communication medium of this system is that the distance of operation can easily reach 50 meters indoors and 250 meters outdoors. Figure 3-2-3 shows a user using a PDA to interact with the local “smart” devices.

Device Control Module:

To control appliances, a user installs a *generic client* application in his or her portable device, such as a PDA. Through the use of control modules, this client application becomes an omni-remote controller that can control various types of services. The main job of a *generic client* is to discover all the available service types in the vicinity and then to bring up its corresponding control module (if available). Of course, the corresponding control module needs to be installed beforehand. Each control module, implementing the client interface of a service type, is a remote controller for this server type. When a user wishes to use a certain type of service, a *generic client* will plug in the server’s corresponding control module, and the module will serve as a bridge between the user and the server.

In the XWeb [18] project, the researchers claimed “if every new service requires the installation of a new piece of software, the accessibility of that service is sharply diminished.” That is, if a user needs to install new software for controlling every new server, the service becomes less accessible. In contrast to their opinion, I believe the accessibility of a service will increase if using the control module approach, because, instead of automatically generating an unexpected UI, the control module approach provides an accurate control representation of an appliance. Furthermore, since there are various types of client platforms (for example, a PDA client with 3.8” or 5” screen and a notebook client with a 15” screen), a service type can have different control modules devoted to each type of client platform. That is, each control module is designed to suit the specification of a client platform, and thus the compatibility issue between a UI and a client platform becomes less significant. In the XWeb project, the authors mentioned that for each type of client, the XWeb system needs to tune the UI of a service according to the capability of a client. This means the server’s control UI has to be modified to suit a client device because a client platform can be in any form. However, the tuning approach may cause unexpected UI or control results. When using the control module approach, a devoted control module to each type of client platform will be brought up, and there will be no ambiguity compared to an auto-UI generation approach.

Some people might worry that a large number of control modules is needed for each service type. However, since there are limited types of client platforms in the market and device vendors would like to provide control modules for as many different platforms as possible, it should be achievable to adopt the module approach.

The construction of a control module varies depending on the specification of a client platform. When a user wants to access a new service type, he/she needs to add the corresponding control module to a *generic client*’s *module repository*. Once the *generic client* discovers the requested service type, it will bring up its corresponding control

module for the user. The process of adding corresponding control modules can be automated, such as downloading them from the website of device vendors.

Figure 3-2-4 shows the idea of the control module approach: that is, a devoted device control module is needed for each type of client platform.

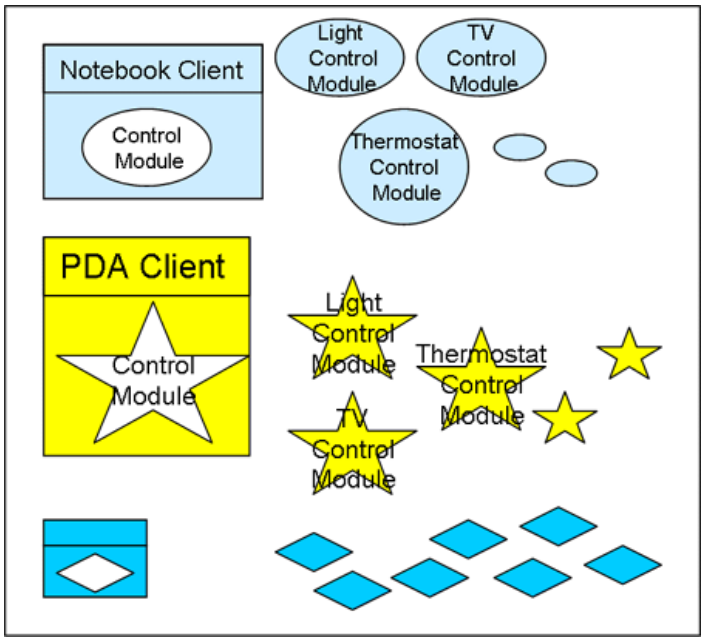


Figure 3-2-4: A device control module is needed for each type of client platform

As shown in the figure, a thermostat server should have a thermostat control module devoted to a notebook client, a thermostat control module devoted to a PDA client, and so on.

Pros of the module approach:

- One *generic client* controls many different servers through their corresponding control module. It becomes a real omni-remote controller.
- Manufacturers know their products well and therefore an accurate control utility is guaranteed. Since the control module and server application are both implemented by the same device vendor, the possibility of device malfunction is largely reduced. In the Pebbles project, the researchers try to automatically generate UI, which may cause device malfunction due to the faults that happen during the interpretation process.
- Manufacturers create UIs for their products. For a client with a graphic display such as a PDA, vendors can provide a GUI with various styles. That is more convenient and visually comfortable compared to the auto-generated, boring GUI. Furthermore, a configurable GUI can also be added to allow user configuration.
- Collaborative functions become possible. Since device manufacturers implement both controller and server, the semantic meaning of interactive functions can be added and fully implemented. If interpreting from a descriptive language (such as

the case in the Pebbles project), only a minimal semantic meaning of function is possible in the automated process. For example, a message received by the parents' application is handled differently from the message received by the child's application.
- When a new control module becomes available, a user can simply replace the existing module with the new one.

Cons of the module approach:
- Manufacturers have to provide different control modules for various client platforms.
- The device collaboration can be very complex.
- While releasing control modules, manufacturers need to take care of the backward compatibility issues for control and collaboration functions.

Device Collaboration:

In my proposed system, a client can interact not only with other servers but also with its peer clients, as does a server. Since each message is broadcast, the message will arrive at all nearby clients and servers (the unwanted message will be discarded). Depending on the receivers (client or server), the message is processed differently. For example, when a server broadcasts its status, clients receiving this message will use the status to update the buttons in the control panel while servers receiving the same message will use this information for working collaboratively with their peers.

Because server applications and client control modules are both implemented by the same manufacturer, client and server devices can be designed to work collaboratively or independently based on their usage. For example, the control of a regular TV device can be "one client controlling one server," whereas the control of a parental TV device can be "many clients working with one server." That is, if working independently, a channel-change-message from a client goes directly to the TV. And, if working cooperatively, a channel-change-message from a child will be handled by the Parental TV as well as his or her guardian(s).

The semantic meaning of the implemented interface of each entity is the most critical part in system collaboration, and detailed discussions about this will be given in Section 4, the **Demonstration** part of this paper. In the **Parental TV** scenario of Session 4, one-to-many collaboration is demonstrated when a child asks to watch a protected channel. The M2M collaboration is demonstrated when dorm residents want to swap their waiting position with others in the **Washing Machine** scenario.

3.2.3 Example

In this section, I give a scenario to illustrate the system architecture. More complex examples will be presented in Section 4.

System Characteristics

Server side omni-remote controller system:

A server application performs the following functions:

Internally

- Controls/interacts with the embedded system of devices
- Executes the device function/logic
- Does system maintenance such as logging

Externally

- Beacons for its presence (see **Service Discovery** section for details)
- Reports its current status (state reporting)
- Executes client commands
- Cooperates with other servers

Client side omni-remote controller system (both generic and specific clients):

A client application performs the following functions:

Internally

- Provides a user with proper UI (s)
- Interacts with a user
- Executes the client logic

Externally

- Performs Service Discovery
- Interacts with server(s)

Let us consider the scenario happening in Joe's house where there are three smart devices—two dimmer-lights (one is in the room and another is in the kitchen) and a TV in the room. The dimmer-light can be turned on/off and its brightness is adjustable. The TV has functions of turning on/off the power, switching channel, changing volume, and mute. A user with a *generic client* application is able to control these three appliances with their respective control modules. Initially, all the appliances are functioning and beaconing for their presence periodically. Joe turns on his PDA and runs the client application of the omni-remote controller system. Then, his PDA receives the beacons from all the appliances and displays the available service types—DIMMER-LIGHT and TV—on the PDA's screen. Joe clicks on the DIMMER-LIGHT button and its preinstalled, corresponding control module is brought up. The control module, which is the *specific client* of the Dimmer-Light servers, does a service discovery and finds two servers—the Dimmer-light-room and Dimmer-light-kitchen. When Joe selects the Dimmer-light-room, the light's UI shows up. He then clicks on the TURN ON button and the dimmer light in the room is on.

After adjusting the light to a comfortable level, Joe chooses the TV service on his PDA and the TV control module is brought up. This module then does the service discovery to find the TV-room server and brings up the TV's UI. While this TV is selected, the control module requests the TV's status periodically in order to update the

UI. Because the TV is initially set “Off,” only the “On” button on the UI is clickable. Joe clicks the “On” button and the TV turns on. At this time, the arrived status message of the TV server triggers the module’s logic function, which grays out the “On” button, and then reflects the server’s status on the UI and enables the other buttons based on the device logic. While Joe is watching TV, a phone call comes in. He mutes TV and picks up the phone. After finishing talking on the phone, Joe forgets if he muted the TV or not. Then, Joe checks the control panel and finds that the “MUTE” button is gray-out and the UN-MUTE button is enabled. He then clicks on the UN-MUTE button and continues watching TV.

3.3 Type Hierarchy

A type hierarchy defines the interface for a system. In the following sections, I will first introduce the type hierarchy used by other researchers and then explain the type hierarchy used in my proposed personal omni-remote controller system.

3.3.1 Introduction

In order to develop a universally adaptable omni-remote controller system, a standard interface is needed for defining the common structure. In the Pebbles project [4], a hierarchical grouping of the servers’ functions—called the *group tree*—is used to define the appliances’ functions. This project uses XML as a descriptive language to depict the functions of each appliance. When the client receives the XML description of a server, it generates its corresponding UI from the XML. In the Universal Interaction [24] project, server interfaces are defined by the IDL (*Interface Definition Language*) that consists of the common elements of the appliances. The clients use the description of a server, defined by the IDL, to create the corresponding UI. The same technique is used in the Universal Interaction Appliance [25] project that uses the *reference platform* that is similar to what I just described.

In the personal omni-remote controller system, I employ the object-oriented (OO) type hierarchy to define the common elements of appliances. To join in the personal omni-remote controller system, the client and server need to either implement the predefined interface of its type or extend the predefined interface to form a new type of interface and implement it. No matter which way is chosen, a client-server pair has to apply itself to the appropriate interface. For example, a TV and its remote controller (a client-server pair) can either implement the predefined TV server interface and the TV client interface or extend the existing TV server interface and the TV client interface to form a new type of interface (descendant), such as a Sony-TV server interface and a Sony-TV client interface. Since the descendant’s interface inherits all the characteristics of the ancestral client, the ancestral client is able to call the descendant interface. That means a client of a service type can control the servers of this service type plus its descendant servers on the “KNOWN” interfaces. Thus, if a TV (even a different brand) is the descendant of a generic set of TV controlling functions, a *generic client* can control this TV on basic functions.

By adopting the OO type hierarchy, the same interface can be implemented with a different semantic meaning. For example, a "forward channel" call (*TV.forwardChannel()*) would switch TV-A to the next channel while the same call would switch TV-B to the next *available* channel. This call makes a difference when the next channel on the channel list is temporarily disabled. When receiving the command, TV-A will still switch to the channel without a signal; however, TV-B will skip this channel and jump to the next available one. Thus, the semantic meaning of a function can be implemented differently even though the interface for the appliances is the same.

3.3.2 Basic Type

For the purpose of describing the architecture of a basic type, we will assume there is one client and one server.

Basically, every appliance can be turned on and off. Thus, it is natural to build the hierarchical tree from the TURN ON and TURN OFF methods. Instead of having the *turnOn()* and *turnOFF()* methods separate, I simplify them into a single method with a boolean parameter indicating on or off. A M2MI call such as S*erver.setOn(true)* turns on a device while a call such as *Server.setOn(false)* turns it off.

In addition to the *setOn* method, there are two critical paired functions—*service discovery* and *state reporting*—in the infrastructure of my proposed system. The service discovery function consists of the *getDeviceInfo* and *reportDeviceInfo* methods. These methods are reserved primarily for the service discovery process. To use them, a client calls a server with its client handle, such as *Server.getDeviceInfo(ClientHandle),* and the server responds by calling the client with the server's bootstrap information, such as *ClientHandle.reportDeviceInfo(ServerBootstrap)* (See Section 3.4 **Service Discovery** for details).

The state reporting function includes the *getStatus* and *reportStatus* methods. These methods are used to update a client's UI and execute a client's device logic, such as enabling or disabling buttons on the GUI. When a client calls a server with its handle to request the server's current status (*Server.getStatus(ClientHandle)*), the server calls back with its status (*ClientHandle.reportStatus(ServerStatus)*). This procedure serves as a feedback process for the personal omni-remote controller system. In the project "An Agent-based Bidirectional Intelligent Remote Controller," [19] the bidirectional communication between the controller and the appliance is used to establish the feedback function. This matches the functionality of the state reporting pair. In both systems, we have the client recognizing the state of the server.

Figure 3-3-1 shows a block diagram for the basic server interface and the basic client interface.

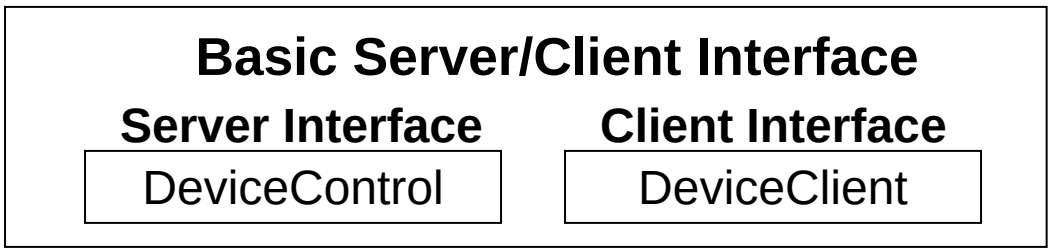


Figure 3-3-1: The basic client and server interfaces

The server and client interfaces are shown as follows.

```
public interface DeviceControl
{   public void getDeviceInfo(DeviceClient theClient);

    public void getStatus(DeviceClient theClient);

    public void setOn(boolean theValue);
}
public interface DeviceClient
{   public void reportDeviceInfo(DeviceControlInfo theInfo);

    public void reportStatus(DeviceControlStatus theStatus);
}
```

Besides the service discovery pair and the state reporting pair, a basic server has a *setOn* method for a client to call. A typical application using the basic server/client interface is the light device/controller. Assuming that a light is a smart device and it implements the basic server interface, after the initial service discovery phase a user with a light controller (a client device) implementing the basic client interface can turn on or off the light by calling the *LightServer.setOn(true/false)* method. Furthermore, the client controller updates the server status whenever the *LightClient.reportStatus(theStatus)* is invoked by the light server.

When a server reports its bootstrap information to a client, it calls the client's *reportDeviceInfo* method with its information defined by the DeviceControlInfo interface. The DeviceControlInfo "extends" ServerInfo interface that defines the server's bootstrap information in the personal omni-remote controller system. Figure 3-3-2 shows a block diagram for DeviceControlInfo interface extending ServerInfo interface.

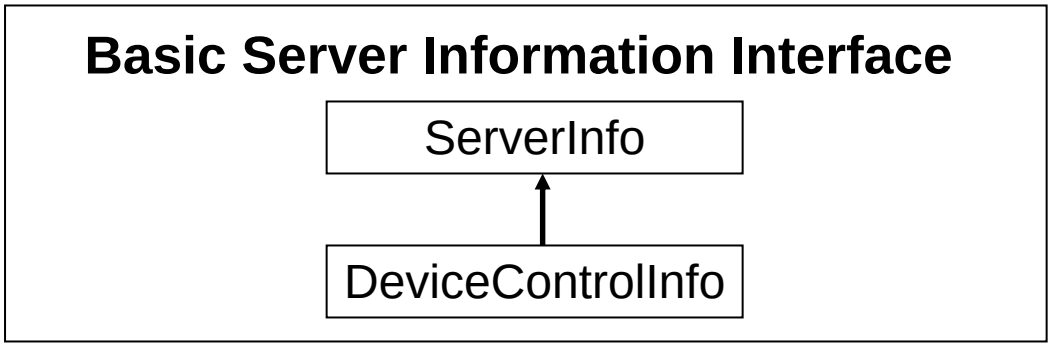


Figure 3-3-2: The basic server information interface hierarchy

The ServerInfo interface and its descendant DeviceControlInfo interface are shown as follows.

```
public interface ServerInfo
{   public String getDeviceID();
    public String getDeviceName();
    public String getDeviceModel();
    public String getDeviceVersion();
    public String getDescription();
    public Date getServerInfoCreatedDate();
}
public interface DeviceControlInfo extends ServerInfo
{   public DeviceControl getServerHandle();
}
```

The methods of the ServerInfo interface are described as follows.

- **getDeviceID()**: A server's identification that should be universally unique
- **getDeviceName()**: The name of a server
- **getDeviceModel()**: The service type of a server. It is used for a *generic client* application to match a server with its corresponding control module (if any) in the *module repository* of this *generic client*
- **getDeviceVersion()**: Used for tracking the version of a server when matching up the server's control module
- **getDescription()**: A brief description of a server, such as a server's location and so on
- **getServerInfoCreatedDate()**: Indicates the date of creating a beaconing message and used for knowing the availability of a server

The DeviceControlInfo interface consists of the ServerInfo interface plus its unihandle [1, 2].

- **getServerHandle()**: Used to invoke M2MI method call

When a server reports its state, it calls a client's *reportStatus* method with this server's status defined by the DeviceControlStatus interface. The DeviceControlStatus is shown as follows.

```
public interface DeviceControlStatus
{   public boolean isOn();
}
```

When a client receives a server's status, it uses the **isOn** method to know if the server is on or off. The **isOn** method returns "true" if the server is on and "false" otherwise.

3.3.3 Object-oriented Type Hierarchy

An object-oriented type hierarchy has the following three advantages: 1) reusing interfaces, 2) partially controlling servers, and 3) redefining methods. For example, since every device has the basic type's TURN ON and TURN OFF methods, any new service type can extend the basic interface. By doing do, the basic interface is reused. Figure 3-3-3 is a block diagram for the interface hierarchy of a dimmer light server and client.

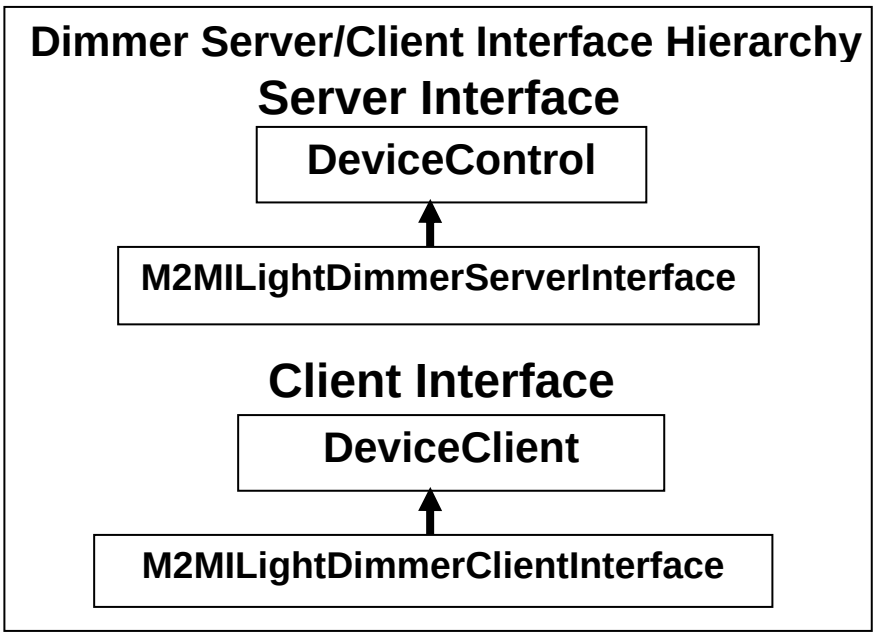


Figure 3-3-3: A dimmer client and server interface hierarchy

The dimmer light server interface and the dimmer light client interface are shown as follows:

```
public interface M2MILightDimmerServerInterface extends DeviceControl
{   public void getDeviceInfo(M2MILightDimmerClientInterface theClient);

    public void getStatus(M2MILightDimmerClientInterface theClient);

    public void setDimmer(float theValue);

    public void setUp();

    public void setDown();
}
public interface M2MILightDimmerClientInterface extends DeviceClient
{   public void reportDeviceInfo(M2MILightDimmerServerInfoInterface theInfo);

    public void reportStatus(M2MILightDimmerServerStatusInterface theStatus);
}
```

Except for the *setOn* method inherited from the basic device, the server interface also has the *setDimmer* (with a float parameter), *setUp*, and *setDown* methods. The dimmer client extends the basic client to develop a new type of client corresponding to the dimmer server. As we can see, the interface can be reused in this OO type hierarchy.

When a dimmer server reports its bootstrap information to a dimmer client, it calls the client's *reportDeviceInfo* method with its information defined by the M2MILightDimmerServerInfoInterface. The M2MILightDimmerServerInfoInterface extends the DeviceControlInfo interface. Figure 3-3-4 shows a block diagram of the M2MILightDimmerServerInfoInterface extending the DeviceControlInfo interface.

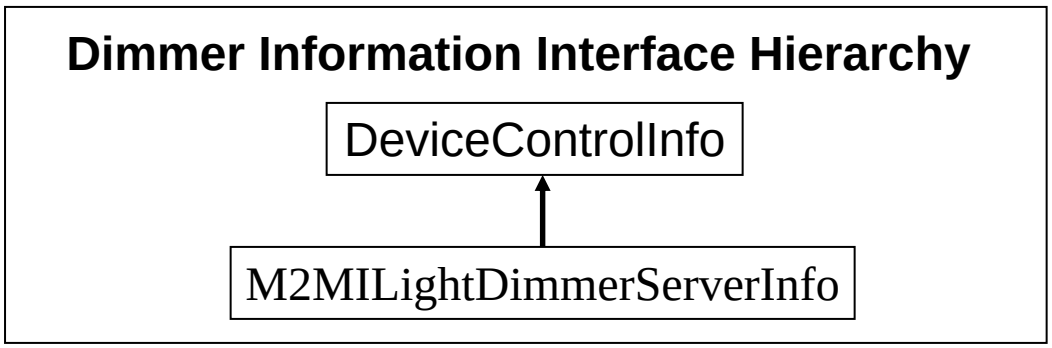


Figure 3-3-4: A block diagram for the dimmer server information interface hierarchy

The DeviceControlInfo is the same as I described before and its descendant M2MILightDimmerServerInfoInterface is shown as follows:

```
public interface M2MILightDimmerServerInfoInterface extends DeviceControlInfo
{   public M2MILightDimmerServerInterface getServerHandle();
}
```

The dimmer server information consists of the server's basic information plus its unihandle.

When a dimmer server reports its state, it calls a dimmer client's *reportStatus* method with this server's status. This server's status is defined by the M2MILightDimmerServerStatusInterface that is extended from the DeviceControlStatus interface (show in Figure 3-3-5).

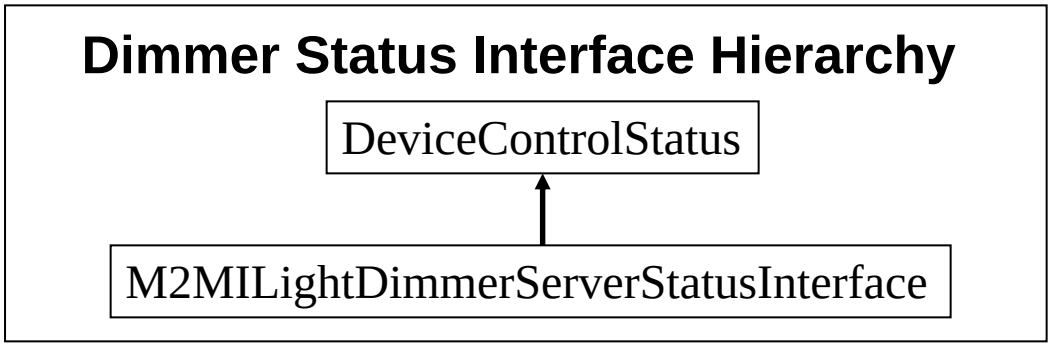


Figure 3-3-5: A block diagram for the dimmer server status interface hierarchy

The M2MILightDimmerServerStatusInterface is shown as below:

```
public interface M2MILightDimmerServerStatusInterface extends DeviceControlStatus
{   public float getDimmer();
}
```

A dimmer client uses the **getDimmer** method to know the dim level of the server. Because the dimmer's status is defined by the M2MILightDimmerServerStatusInterface, a client can use the **isOn** method to know if this server is on or off as well.

After describing the first advantage of using OO type hierarchies (reusable interfaces), let us see the second one. That is, a client of service type A is able to partially control types extended from type A. To put it in other words, a client of a super class can use the "known" interface to control the servers that are partially built up by the interfaces of the

super class. For instance, since the dimmer interface extends the basic device interface, a basic client will know how to call the *setOn* function of the dimmer device. This is to say that the basic client can turn on or off a dimmer device. However, a dimmer client is not able to control a basic device because the dimmer client expects its target server to have all the *setOn*, *setDimmer*, *setUp*, and *setDown* interface.

Furthermore, the OO type hierarchy provides backward compatibility. When new functions are added to a server, there will be a new client-server pair for this service type. Then, the new updates of the server and client applications are published and we want the server and client to be updated. However, if a client device, for some reason, is not able to upgrade the newest client application, it still can control the newly updated server. Because the client device knows the server's original interfaces, there will be no problem to control the original functions of the newly upgraded server. A personal omni-remote controller system with such a property is more favorable.

The OO type hierarchy of my implemented servers, clients, and server's status are shown respectively in Figure 3-3-6, Figure 3-3-7, and Figure 3-3-8. The details of the Thermostat, the Parental TV, and the Washing Machine system will be described individually in the **Demonstration** section.

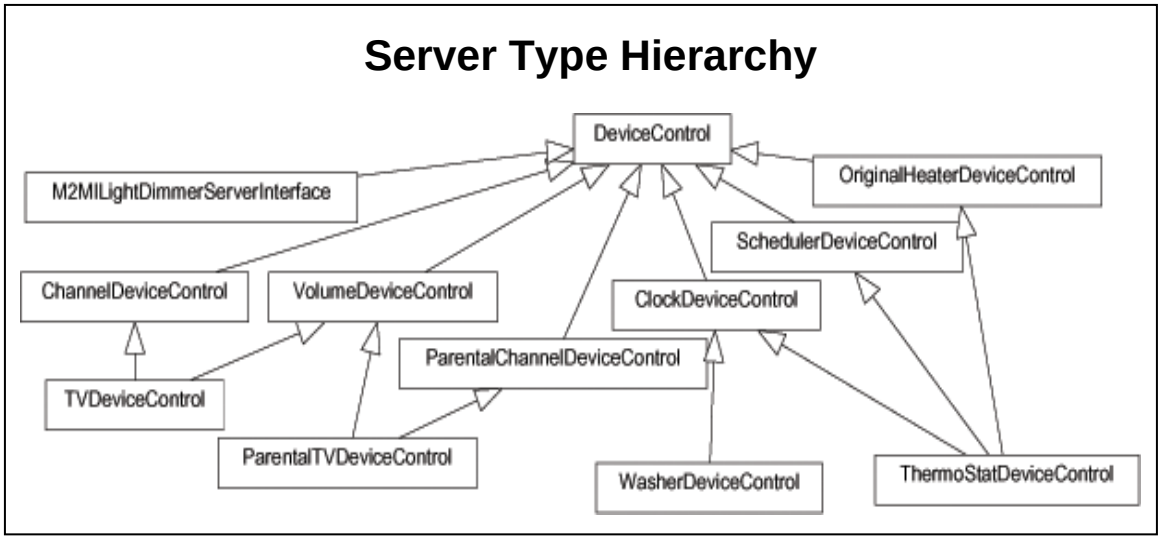


Figure 3-3-6: The server type hierarchy

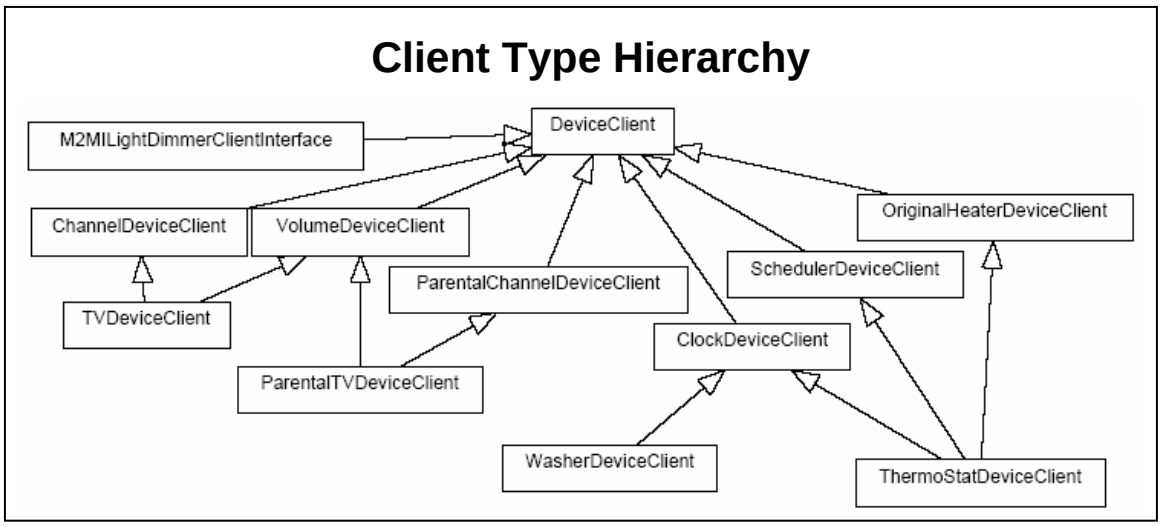


Figure 3-3-7: The client type hierarchy

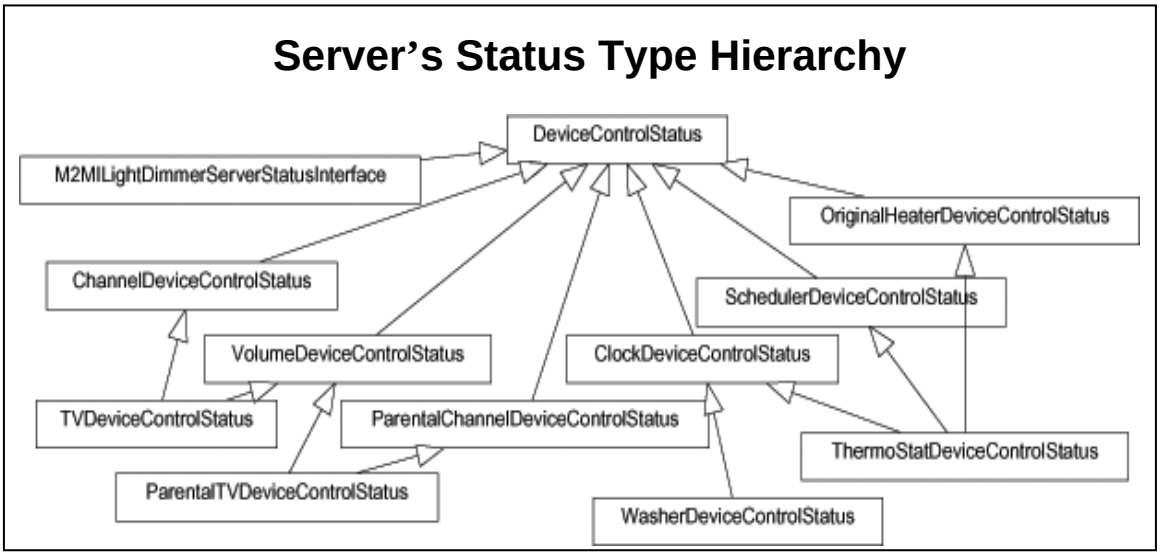


Figure 3-3-8: The server status type hierarchy

3.3.3.1 Thermostat Interface

The interface of the thermostat server inherits from the *Scheduler interface*, the *Clock interface*, and the *Original Heater interface*. These three interfaces are extended from the *basic server interface*. Each interface unit itself is a common element in the OO type hierarchy of the personal omni-remote controller system. Figure 3-3-9 shows a block diagram for the interface hierarchy of a thermostat server. Note, the *getDeviceInfo* and *getStatus* interface in each block is paired respectively with the client's *reportDeviceInfo* and *reportStatus* interface to perform service discovery and state reporting.

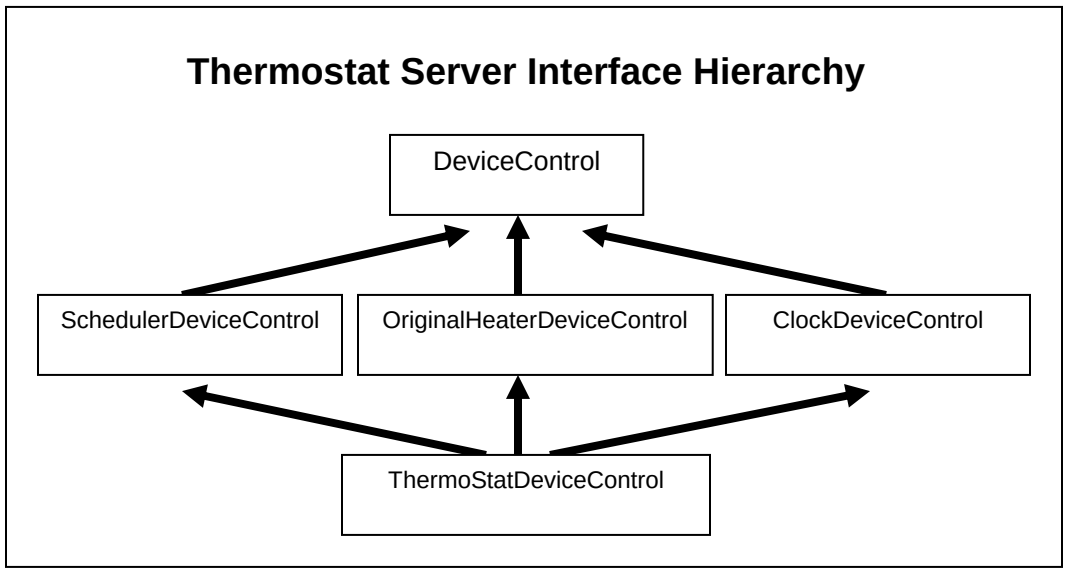


Figure 3-3-9: A block diagram for the interface hierarchy of a thermostat server

The DeviceControl (Basic Server) Interface: The interface shown below is used to turn on or off the thermostat system.

```
public interface DeviceControl
{  public void getDeviceInfo(DeviceClient theClient);
   public void getStatus(DeviceClient theClient);
   public void setOn(boolean theValue);
}
```

- *setOn(boolean)*: TRUE to turn on and FALSE to turn off the thermostat.

The Scheduler Interface: The interface shown below is used to change the thermostat's set point schedule.

```
public interface SchedulerDeviceControl extends DeviceControl
{  public void getDeviceInfo(SchedulerDeviceClient theClient);
   public void getStatus(SchedulerDeviceClient theClient);
   public void addSchedule(Schedulable theSchedule);
   public void removeSchedule(Schedulable theSchedule);
   public void clearAllSchedules();
}
```

- *addSchedule(SchedulableObject)*: Add a schedule to the system
- *removeSchedule(SchedulableObject)*: Remove a schedule from the system
- *clearAllSchedules()*: Reset the schedules

The Original Heater Interface: The interface shown below is used to control a basic heater/AC system. A user can set his or her desired temperature, set the power to either "AC" or "Heater," and set the fan to either "Auto" or always "On."

```
public interface OriginalHeaterDeviceControl extends DeviceControl
{   public void getDeviceInfo(OriginalHeaterDeviceClient theClient);
```

```
    public void getStatus(OriginalHeaterDeviceClient theClient);
    public void setPoint(int thePoint);
    public void setACOn();
    public void setHeaterOn();
    public void setFanAuto();
    public void setFanOn();
}
```

- *setPoint(int)*: Set a desired temperature
- *setACOn()*: Set the AC "On"
- *setHeaterOn()*: Set the heater "On"
- *setFanAuto()*: Set the fan to "Auto"
- *setFanOn()*: Set the fan to "On"

The Clock Interface: The interface shown below is used to set the system time.

```
public interface ClockDeviceControl extends DeviceControl
{   public void getDeviceInfo(ClockDeviceClient theClient);
    public void getStatus(ClockDeviceClient theClient);
    public void setClock(Date theDate);
}
```

- *setClock(Date)*: Set the system time

The Thermostat Interface: The interface shown below is for the thermostat type.

```
public interface ThermoStatDeviceControl extends OriginalHeaterDeviceControl,
SchedulerDeviceControl, ClockDeviceControl
{   public void getDeviceInfo(ThermoStatDeviceClient theClient);
    public void getStatus(ThermoStatDeviceClient theClient);
}
```

A thermostat client controls a thermostat server using the server's interface. In the thermostat application, it is sufficient for a client to have just the service discovery and state reporting interfaces. Figure 3-3-10 shows a block diagram for the interface hierarchy of a thermostat client.

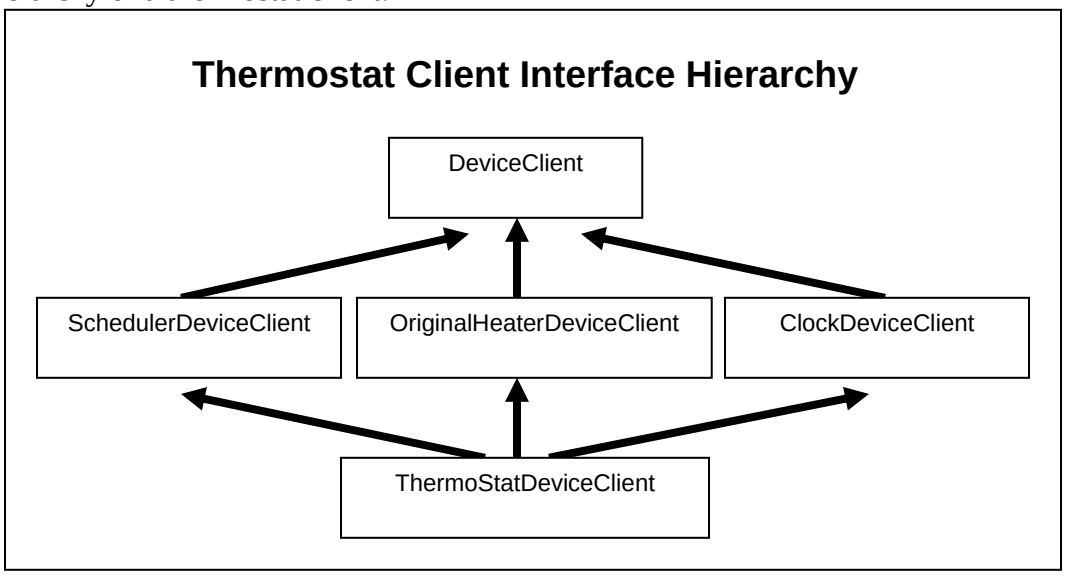


Figure 3-3-10: A block diagram for the interface hierarchy of a thermostat client

The interfaces shown below are implemented by their corresponding client.

The DeviceCient (Basic Client) Interface:

```
public interface DeviceClient
{   public void reportDeviceInfo(DeviceControlInfo theInfo);

    public void reportStatus(DeviceControlStatus theStatus);
}
```

The Scheduler Interface:

```
public interface SchedulerDeviceClient extends DeviceClient
{   public void reportDeviceInfo(SchedulerDeviceControlInfo theInfo);

    public void reportStatus(SchedulerDeviceControlStatus theStatus);
}
```

The Original Heater Interface:

```
public interface OriginalHeaterDeviceClient extends DeviceClient
{   public void reportDeviceInfo(OriginalHeaterDeviceControlInfo theInfo);

    public void reportStatus(OriginalHeaterDeviceControlStatus theStatus);
}
```

The Clock Interface:

```
public interface ClockDeviceClient extends DeviceClient
{   public void reportDeviceInfo(ClockDeviceControlInfo theInfo);

    public void reportStatus(ClockDeviceControlStatus theStatus);
}
```

The Thermostat Interface:

```
public interface ThermoStatDeviceClient extends OriginalHeaterDeviceClient,
SchedulerDeviceClient, ClockDeviceClient
{     public void reportDeviceInfo(ThermoStatDeviceControlInfo theInfo);

      public void reportStatus(ThermoStatDeviceControlStatus theStatus);
}
```

The server's information interfaces (the SchedulerDeviceControlInfo interface, the OriginalHeaterDeviceControlInfo interface, the ClockDeviceControlInfo interface, and the ThermoStatDeviceControlInfo interface) are all extended from the DeviceControlInfo interface (as shown in Figure 3-3-11).

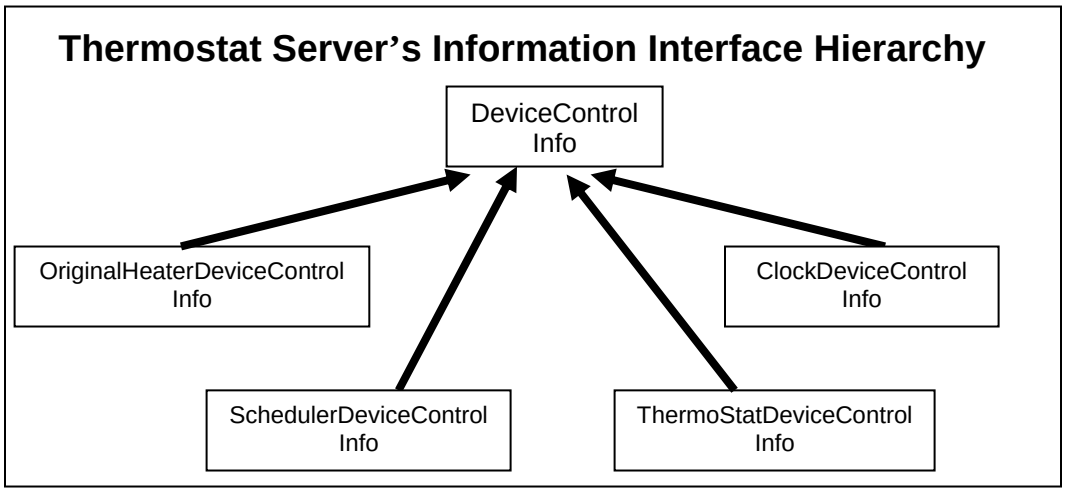


Figure 3-3-11: A block diagram for the interface hierarchy of a thermostat server's information

Each of these interfaces shown below has a **getServerHandle** method that returns a unihandle.

```
public interface OriginalHeaterDeviceControlInfo extends DeviceControlInfo
{   public OriginalHeaterDeviceControl getServerHandle();
}
public interface SchedulerDeviceControlInfo extends DeviceControlInfo
{   public SchedulerDeviceControl getServerHandle();
}
public interface ThermoStatDeviceControlInfo extends DeviceControlInfo
{   public ThermoStatDeviceControl getServerHandle();
}
public interface ClockDeviceControlInfo extends DeviceControlInfo
{   public ClockDeviceControl getServerHandle();
}
```

Figure 3-3-12 shows a block diagram for the interface hierarchy of a thermostat server's status.

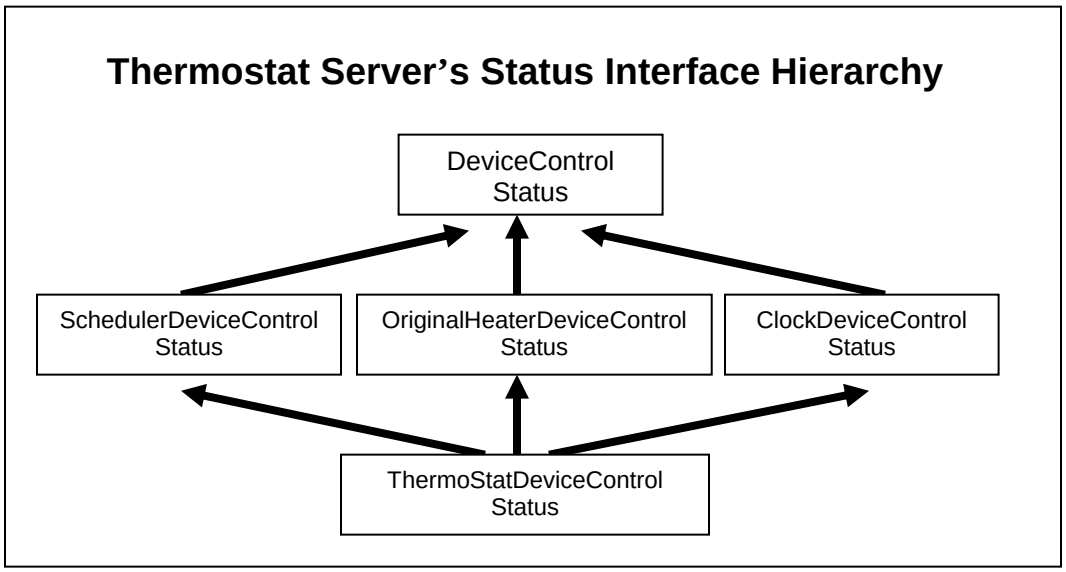


Figure 3-3-12: A block diagram for the interface hierarchy of a thermostat server's status

Followed by the figure, the status interfaces are given.

SchedulerDeviceControlStatus interface:

```
public interface SchedulerDeviceControlStatus extends DeviceControlStatus
{   public Schedulable[] getSchedules();
}
```

- *getSchedules()*: Returns the schedule(s)

```
public interface Schedulable
{   void setEnable(boolean theValue);

    boolean isEnabled();

    void setName(String theName);
```

```
    String getName();
    void setActivity(Object theActivity);
    Object getActivity();
    void setDate(Date theDate);
    Date getDate();
}
```

- *setEnable(true/false)*: Set this schedule to be enabled or disabled
- *isEnabled()*: Return true if this schedule is enabled and false otherwise
- *setName(theName)*: Set the name of this schedule
- *getName()*: get the name of this schedule
- *setActivity(theActivity)*: Set an certain activity to be performed
- *getActivity()*: get the scheduled activity
- *setDate(theDate)*: Set the date of performing the scheduled activity
- *getDate()*: Get the date of performing this scheduled activity

OriginalHeaterDeviceControlStatus interface:

```
public interface OriginalHeaterDeviceControlStatus extends DeviceControlStatus
{   public int getSetPoint();
    public int getCurrentTemperature();
    public boolean isACOn();
    public boolean isHeaterOn();
    public boolean isFanAuto();
    public boolean isFanOn();
}
```

- *getSetPoint()*: Get the set point
- *getCurrentTemperature()*: Get the current temperature
- *isACOn()*: Return true if the "AC" is "On" and false otherwise
- *isHeaterOn()*: Return true if the "heater" is "On" and false otherwise
- *isFanAuto()*:Return true if the fan is "Auto" and false otherwise
- *isFanOn()*: Return true if the fan is "On" and false otherwise

ClockDeviceControlStatus interface:

```
public interface ClockDeviceControlStatus extends DeviceControlStatus
{   public Date getClock();
}
```

- *getClock()*: Get the date of the clock

ThermoStatDeviceControlStatus interface:

```
public interface ThermoStatDeviceControlStatus extends OriginalHeaterDeviceControlStatus,
SchedulerDeviceControlStatus, ClockDeviceControlStatus
{
}
```

By using the OO type hierarchy, when a thermostat client receives a thermostat server's status, the client can use these built-up methods to know the server's status.

3.3.3.2 Parental TV Interface

The interface of the parental TV server inherits from the *Volume interface* and the *Parental Channel interface*. These two interfaces are extended from the *basic server interface*. Each interface itself is a common element in the OO type hierarchy of the personal omni-remote controller system. Figure 3-3-13 describes a block diagram for the interface hierarchy of a parental TV server. Note, the *getDeviceInfo* and the *getStatus* interface in each unit are respectively paired with the client's *reportDeviceInfo* and *reportStatus* interface to perform service discovery and state reporting.

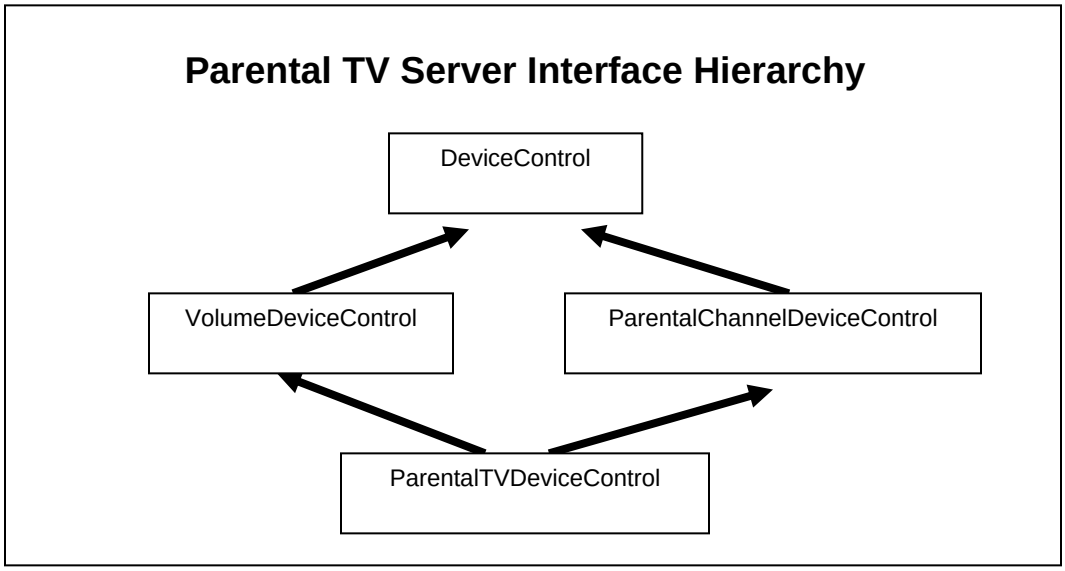


Figure 3-3-13: The block diagram of the interface hierarchy of parental TV server

The Basic Server Interface: This interface is used to turn a parental TV on or off.

- *setOn(boolean)*: TRUE to turn on and FALSE to turn off the TV.

The Volume Interface: The interface shown below is used to change a parental TV's volume.

```
public interface VolumeDeviceControl extends DeviceControl
{   public void getDeviceInfo(VolumeDeviceClient theClient);

    public void getStatus(VolumeDeviceClient theClient);

    public void setMute (boolean theValue);

    public void setVolume (float theVolume);

    public void setVolumeUp();

    public void setVolumeDown();
}
```

- *setMute(boolean)*: TRUE to mute the TV and FALSE to un-mute it.
- *setVolume(float)*: Set the TV's volume where 0.0 is the lowest and 1.0 is the highest.
- *setVolumeUp()*: Turn the TV's volume up.
- *setVolumeDown()*:Turn the TV's volume down.

The Parental Channel Interface: The interface shown below is used to switch the channel. When calling this interface, a user needs to provide the user's information and then the parental TV sees if the user is allowed to change to a requested channel or not.

```
public interface ParentalChannelDeviceControl extends DeviceControl
{   public void getDeviceInfo(ParentalChannelDeviceClient theClient);

    public void getStatus(ParentalChannelDeviceClient theClient);

    public void setChannel(Right theRight, int theChannel);

    public void forwardChannel(Right theRight);

    public void backwardChannel(Right theRight);
}
```

- *setChannel(theUserRight, theChannel)*: Set the channel number with the user's identity.
- *forwardChannel(theUserRight)*: Forward the channel with the user's identity.
- *backwardChannel(theUserRight)*: Backward the channel with the user's identity.

The Parental TV Interface: This interface is used for configuring the features of a parental TV.

```
public interface ParentalTVDeviceControl extends ParentalChannelDeviceControl,
VolumeDeviceControl
{   public void getDeviceInfo(ParentalTVDeviceClient theClient);

    public void getStatus(ParentalTVDeviceClient theClient);

    public void setBothGrants(Right theRight, boolean theValue);

    public void setDisableProtected(Right theRight, int theMinute);

    public void setChannel(Right theRight, int theChannel, ParentalTVDeviceControl theTV);
}
```

- *setBothGrants(theUserRight,boolean)*: Return TRUE if a parental TV's control policy is set to "Both grants" and return FALSE if the TV is set to "Single grant." Depending on which control policy is set, a child has to either get both of his or her parents' permission or get any one of parents' permission before changing to a protected channel.
- *setDisableProtected(theUserRight, theDuration)*: Parents can enable or disable the parental control function by calling this interface. Once this function is disabled, a child can watch any channel.
- *setChannel(theUserRight, theChannel, theParentalTV)*: Kid's control device uses this interface to request switching to a protected channel. This call will go to the parental TV and the parental control devices. Note: The third parameter is used to know which parental TV a child refers to because there might be more than one parental TVs in a house.

A parental TV client controls a parental TV server using the server's interface. In this case, it is sufficient for a client to have just the service discovery and state reporting interfaces. Figure 3-3-14 shows a block diagram for the interface hierarchy of a parental TV client.

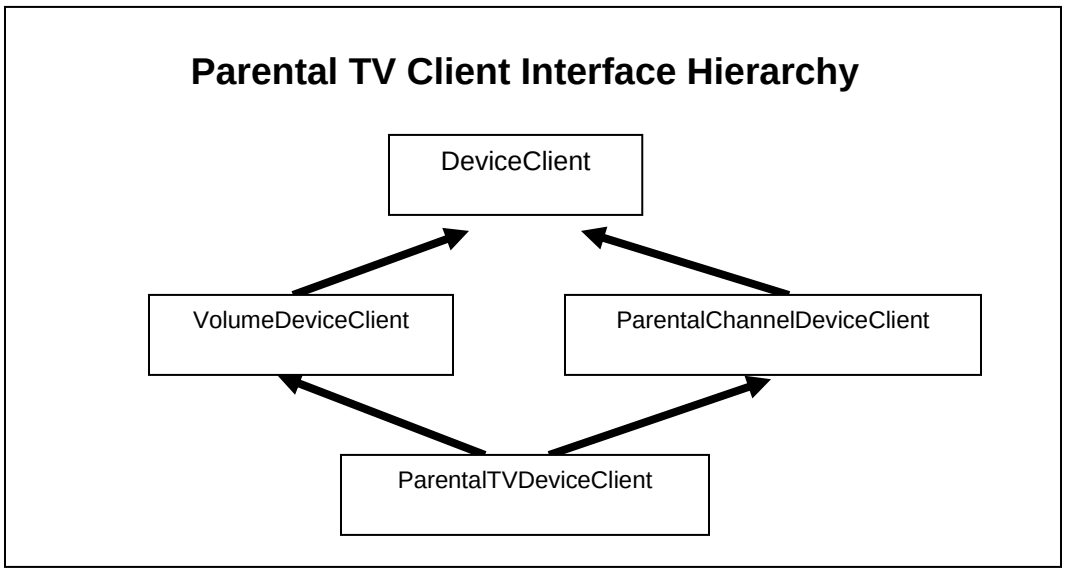


Figure 3-3-14: The block diagram of the interface hierarchy of Parental TV client

The interfaces shown below are implemented by the corresponding clients.

The Volume Interface:

```
public interface VolumeDeviceClient extends DeviceClient
{   public void reportDeviceInfo(VolumeDeviceControlInfo theInfo);

    public void reportStatus(VolumeDeviceControlStatus theStatus);
}
```

The Parental Channel Interface:

```
public interface ParentalChannelDeviceClient extends DeviceClient
{   public void reportDeviceInfo(ParentalChannelDeviceControlInfo theInfo);

    public void reportStatus(ParentalChannelDeviceControlStatus theStatus);
}
```

The Parental TV Interface:

```
public interface ParentalTVDeviceClient extends ParentalChannelDeviceClient,
VolumeDeviceClient
{   public void reportDeviceInfo(ParentalTVDeviceControlInfo theInfo);

    public void reportStatus(ParentalTVDeviceControlStatus theStatus);
}
```

The server's information interfaces (the VolumeDeviceControlInfo interface, the ParentalChannelDeviceControlInfo interface, and the ParentalTVDeviceControlInfo interface) are all extended from the DeviceControlInfo interface (as shown in Figure 3-3-15).

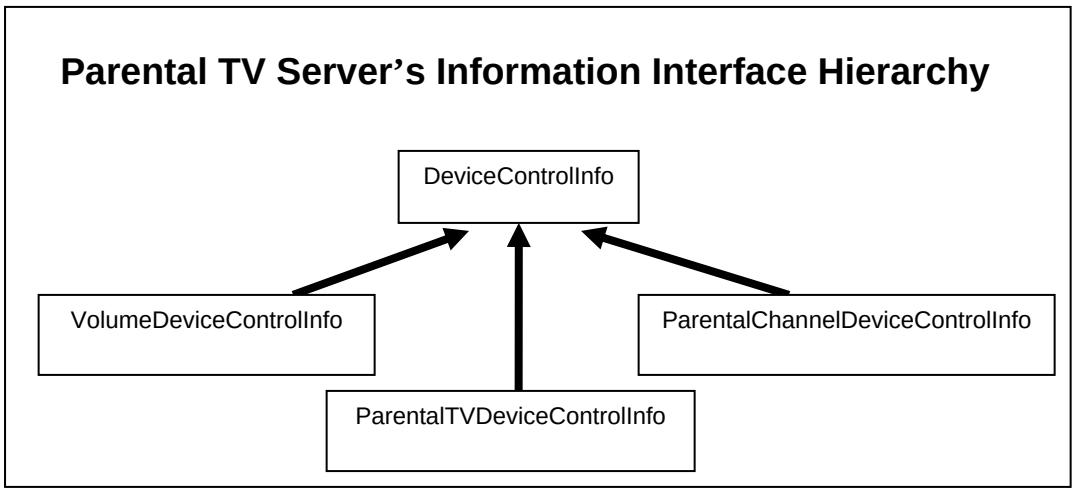


Figure 3-3-15: A block diagram for the interface hierarchy of a parental TV server's information

Each of these interfaces shown as follows has a **getServerHandle** method that returns a unihandle.

```
public interface VolumeDeviceControlInfo extends DeviceControlInfo
{   public VolumeDeviceControl getServerHandle();
}
public interface ParentalChannelDeviceControlInfo extends DeviceControlInfo
{    public ParentalChannelDeviceControl getServerHandle();
}
public interface ParentalTVDeviceControlInfo extends DeviceControlInfo
{    public ParentalTVDeviceControl getServerHandle();
}
```

Figure 3-3-16 shows a block diagram for the interface hierarchy of a parental TV server's status. Followed by this figure, the interfaces are described.

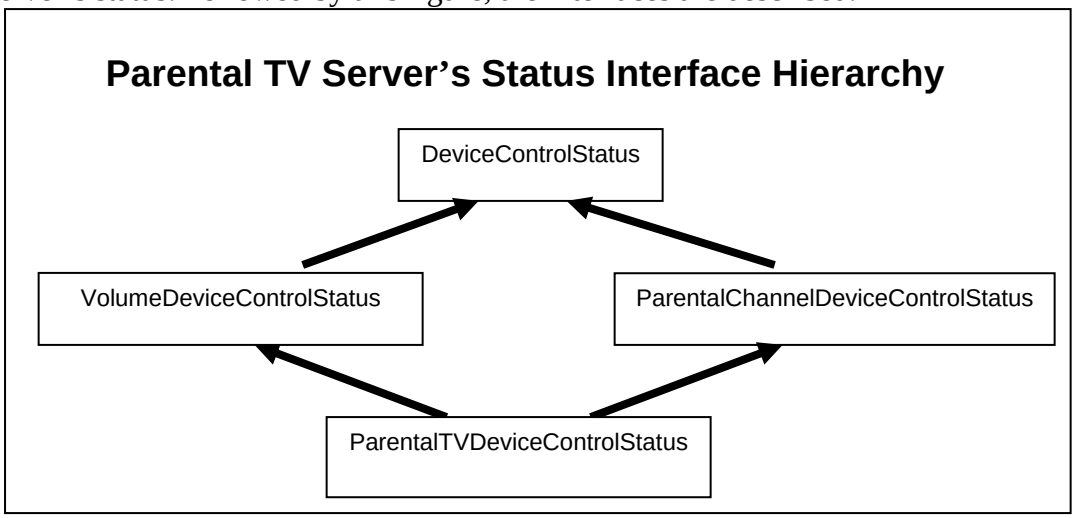


Figure 3-3-16: A block diagram for the interface hierarchy of a parental TV server's status

VolumeDeviceControlStatus interface:

```
public interface VolumeDeviceControlStatus extends DeviceControlStatus
{   public float getCurrentVolume();

    public boolean isMute();
}
```

- *getCurrentVolume()*: Get the current volume
- *isMute()*: Return TRUE if the system is muted or FALSE otherwise

ParentalChannelDeviceControlStatus interface:

```
public interface ParentalChannelDeviceControlStatus extends DeviceControlStatus
{    public int getCurrentChannel();

     public TreeSet getAvailableChannels();
}
```

- *getCurrentChannel()*: Get the current channel number
- *getAvailableChannels()*: Get the available channel number

ParentalTVDeviceControlStatus interface:

```
public interface ParentalTVDeviceControlStatus extends
ParentalChannelDeviceControlStatus, VolumeDeviceControlStatus
{   public boolean isBothParentsGrantsNeeded();

    public int getDisableMinutes();
}
```

- *isBothParentsGrantsNeeded()*: Return TRUE if both parents' permissions are required and FALSE otherwise
- *getDisableMinutes()*: Get the duration of disabling the TV protection functionality in minutes

By using the OO type hierarchy, when a parental TV client receives a parental TV server's status, the client can use these methods to check the server's status.

3.3.3.3 Washer Interface

The interface of the washing machine server inherits from the *Clock interface* that is extended from the *basic server interface*. Note notice that the **Thermostat Server** inherits from the *Clock interface* too. This shows one of the advantages of using an OO type hierarchy—that is, reusable common interface elements. Figure 3-3-17 shows a block diagram for the interface hierarchy of a washer server. As with the thermostat and parental TV application, the *getDeviceInfo* and the *getStatus* interface in each unit are paired with the corresponding client's *reportDeviceInfo* and *reportStatus* interface to perform service discovery and state reporting.

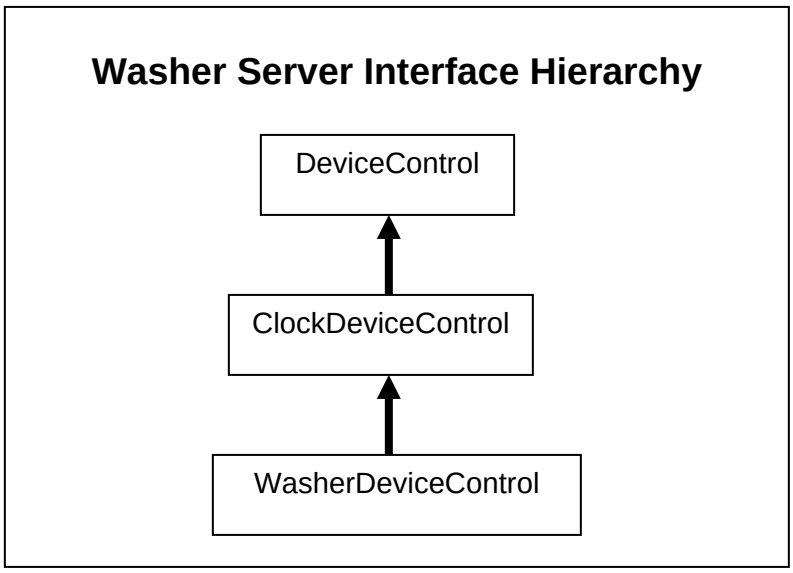


Figure 3-3-17: A block diagram for the interface hierarchy of a washing machine server

The Basic Server Interface: This interface is used to turn the system on or off.

- *setOn(boolean)*: TRUE to turn on and FALSE to turn off a washer.

The Clock Interface: This interface is used to set the system time

- *setClock(Date)*: Set the system time

The Washer Interface: The interface shown below is used to interact with a washer. In this server interface, the same control message is repeated about 3 times to cope with network malfunction and message loss. Thus, a date object is introduced for ignoring the repeated message. Also, the date (a long value in Java) is used for identifying a user's load in the system.

```
public interface WasherDeviceControl extends ClockDeviceControl
{   public void getDeviceInfo(WasherDeviceClient theClient);

    public void getStatus(WasherDeviceClient theClient);

    public void changeWaitingTimePreToWashing(int theMSec, Date theDate);

    public void changeWaitingTimePostToIdle(int theMSec, Date theDate);

    public void addLoad(M2MIWasherUsage theUsage);

    public void cancelLoad(Right theOne);

    public void startLoad(M2MIWasherUsage theUsage);
}
```

- *changeWaitingTimePreToWashing(duration, date)*: Set the waiting time for the system to start a reserved load. For example, the reserved washer will wait ten minutes (count from the reservation time) for Alice to start her load.
- *changeWaitingTimePostToIdle(duration, date)*: Set the waiting time for picking up a finished load.
- *addLoad(theUsage)*: Add a load to a certain group of washers with the load information.

- *cancelLoad(theRight)*: Remove all the loads from a user.
- *startLoad(theUsage)*: Start a reserved load.

A washer client controls a washer server using the server's interface. In this application servers and clients need to interact with each other and clients should be able to communicate with each other for a task of swapping slot. Therefore, interfaces for doing these tasks are necessary. Figure 3-3-18 shows a block diagram for the interface hierarchy for a washer client.

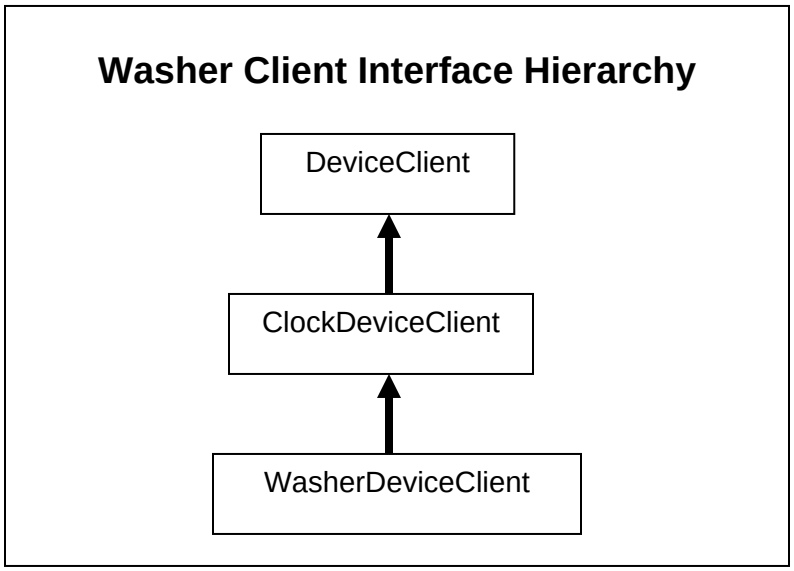


Figure 3-3-18: A block diagram for the interface hierarchy of a washing machine client

The *DeviceClient* and *ClockDeviceClient* interfaces have been discussed previously.

The Washer Client Interface: The interface shown below is used to interact with a washer client. In this interface, the same control message is repeated about 3 times to deal with network malfunction and message loss. Thus, a date object is introduced for ignoring the repeated message. Also, the date is used for identifying a user's load in the system.

```
public interface WasherDeviceClient extends ClockDeviceClient
{   public void reportDeviceInfo(WasherDeviceControlInfo theInfo);

    public void reportStatus(WasherDeviceControlStatus theStatus);

    public void reportAvailable(WasherDeviceControlInfo theWasherInfo, M2MIWasherUsage theUsage, Date theStartDate);

    public void reportLoadFinished(WasherDeviceControlInfo theWasherInfo, Date theFinishDate);

    public void swapWaitRequest(String theRequester, String theRequestee, Date theDate);
}
```

- *reportAvailable(theWasherInfo, theLoad, theStartDate)*: A reserved washer reports its availability to the requesting client and the reservation will be due at "the start date."
- *reportLoadFinished(theWasherInfo, theFinishDate)*: When a washer finishes a load, this interface will be used to inform the owner of the load who has to pick up the load before "the finish date."

- *swapWaitRequest(theRequester, theRequestee, Date)*: This interface is used for a user to request swapping a waiting position with another. The requestee uses the same interface to respond the request.

The server's information interface (the ClockDeviceControlInfo and WasherDeviceControlInfo interface) are all extended from DeviceControlInfo interface (as shown in Figure 3-3-19).

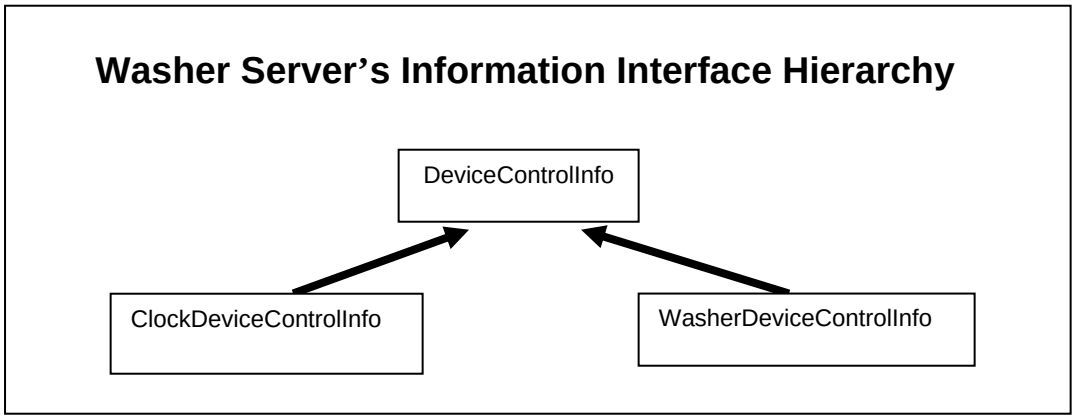


Figure 3-3-19: A block diagram for the interface hierarchy of a washer server's information

Each of these interfaces below has a **getServerHandle** method that returns a unihandle.

```
public interface ClockDeviceControlInfo extends DeviceControlInfo
{   public ClockDeviceControl getServerHandle();
}
public interface WasherDeviceControlInfo extends DeviceControlInfo
{   public WasherDeviceControl getServerHandle();
    public String getGroupName();
}
```

Besides the **getServerHandle** method, the WasherDeviceControlInfo interface contains the **getGroupName** method for identifying which group a washer belongs to.

Figure 3-3-20 shows a block diagram for the interface hierarchy of a washer server's status.

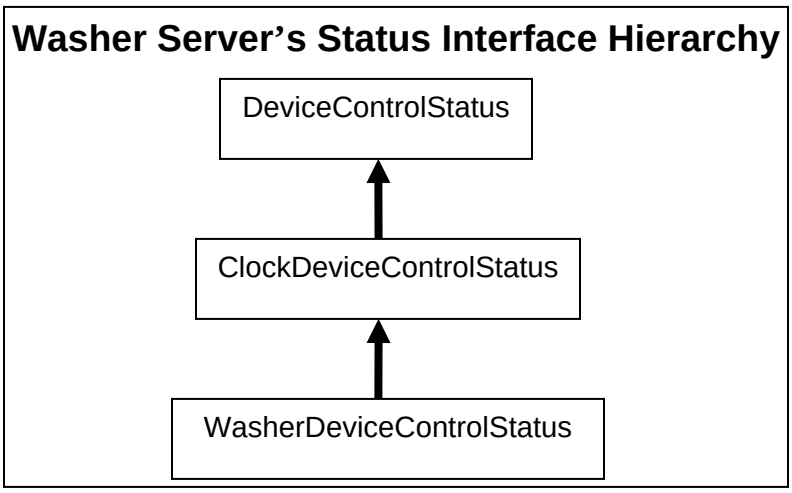


Figure 3-3-20: A block diagram for the interface hierarchy of a washer server's status

Followed by this figure, the interfaces are described.

ClockDeviceControlStatus: has been discussed before and will not repeat here.

WasherDeviceControlStatus interface:

```
public interface WasherDeviceControlStatus extends ClockDeviceControlStatus
{     public String getWasherID();
      public int getTimeToStart();
      public int getTimeToIdle();
      public TreeSet getWaitingList();
      public int getWasherState();
      public String getWasherStateString();
      public String getCurrentUserName();
}
```

- *getWasherID ()*: Get this washer's ID.
- *getTimeToStart ()*: Get the default waiting time to start a load (the time from "Pre-washing" state to "Begin washing" state). Note: Four states are defined in this system: "Pre-washing," "Begin washing," "Post-washing," and "Idle."
- *getTimeToIdle()*: Get the default waiting time to becoming idle (the time from "Post-washing" state to "Idle" state).
- *getWaitingList()*: Get the washer's waiting list.
- *getWasherState()*: Get the washer's current state.
- *getWasherStateString()*:Get the washer's current state in a string representation.
- *getCurrentUserName()*: Get the username who is currently using this washer (if any).

By using the OO type hierarchy, when a washer client receives a washer server's status, the client can use these methods to check the server's status.

3.4 Service Discovery

Because broadcasting in the M2MI [1, 2] does not guarantee the delivery of message and client, and server devices might be in or out of the "picture" at any moment, a service provider must continuously send out the "KeepAlive" message and bootstrap information to let other devices know its availability and the way of using it. In my proposed system, a periodical beacon works well for these two purposes. If a periodical beacon of a server is not received, this server will be treated as "out of service" because other devices do not know if it is "alive" or not.

Before being able to control a server, a client must have some information about the server, such as its name and type. To obtain this information, there are two ways—*client beacon* or *server beacon*. *Client beacon* means that a client device broadcasts a request message to servers and the servers respond if they are available. Based on the servers' replies along with their bootstrap information, the client knows how many service providers are available and how to use each of them. S*erver beacon* means that each service provider has a daemon that periodically broadcasts the server's bootstrap information. When a client device receives the beacon of a server, it recognizes the existence of this server in a certain area.

3.4.1 Design

My proposed personal omni-remote controller system employs *a dual mode service discovery scheme*—that is, the use of both *server beacon* and *client beacon*. When a user enters an area, his or her portable device discovers the available services by waiting for servers' beacons. If the user does not want to wait, he or she can request servers to beacon immediately. Thus, in *a dual mode service scheme*, the *server beacon* scheme is used by default and the *client beacon* scheme is used by request. While a client device is waiting for servers' beacons, a server's beaconing daemon periodically broadcasts a M2MI call of *reportDeviceInfo* interface with a parameter of the bootstrap information. All the nearby clients will discover this server when their *reportDeviceInfo* function is called. When requesting servers' beacons, a client broadcasts a M2MI call of the *getDeviceInfo* interface with its M2MI client handle. This call goes to all the nearby servers, and the servers use the client handle to call the client's *reportDeviceInfo* interface with their bootstrap information. Thus, a couple of seconds later this client gets all the available servers' beacons in the vicinity.

The advantages of using *a dual mode service discovery scheme* are:

- Since client devices are usually battery-powered, energy consumption is a critical problem. By using this scheme, a client will consume less energy because it does not need to beacon periodically.
- Because this system mainly uses a *server beacon* scheme for service discovery, it minimizes the network traffic. Because, for a *server beacon*, only one server's beacon message is necessary while for a *client beacon* there must be two messages—a request message from a client and a response message from a server.

- Because this system uses a *server beacon* scheme by default, it also minimizes the overhead of service discovery. A server beaconing message is for all the nearby clients in *server beacon* while in *client beacon* a server's response beaconing message is only for the requesting client. For example, when a TV server beacons periodically, this beaconing message will be sent to all the nearby clients. However, when a TV client requests the nearby TV servers to beacon, each TV servers' response message will be sent only to the requesting client.

4 Demonstration

For a device (client or server) to join the personal omni-remote controller system, it needs to meet the hardware and software requirements for running the application. To meet the hardware requirement, a device must be "smart" enough to compute instructions, save its states, and communicate with other devices wirelessly. To meet the software requirement, a device must have the Java 2 Platform, Standard Edition (J2SE) installed in order to run M2MI/M2MP, the underlying layer of the personal omni-remote controller system. To demonstrate the framework of the system, I have implemented the control simulations for three different devices with different capabilities. These devices are a thermostat, a TV with parental control (parental TV), and a washing machine.

In an ordinary operational environment for a personal omni-remote controller system, there could be many appliances (servers) and omni-remote controllers (clients). Servers announce their presence, broadcast their status, and interact with clients and peer servers. Also, clients are designed to discover all the servers in the vicinity and bring up a UI for interacting with any known servers, based on a user's request. Since each client needs to interact with various types of servers, the *generic client* design is well suited. I have created the control modules of a thermostat, parental TV, and washing machine for PDA clients and have implemented a simulated PDA *generic client* to manage these control modules.

In the following sections, I will describe the class implementation, the GUI, and the sequence diagram of device interaction for each of the three devices. While designing these demonstrations, the goal is to minimize communication and interfacing while achieving maximum functionality.

4.1 Service Discovery Implementation

In the personal omni-remote controller system, servers beacon their presence using a beacon daemon. To be specific, a server's beacon daemon broadcasts the server's bootstrap (beaconing message) at a specific interval, such as 60 seconds. Setting up the beaconing time interval is application-specific. A short beaconing time increases the network traffic while a long beaconing time diminishes the accessibility of a service. However, when a user enters a service area and cannot access a certain service right away, he or she can request a server beacon (*client beacon*) as long as he or she is sure about the existence of the servers. But, as we discussed before, a *client beacon* has some disadvantages too, such as increasing the traffic on a network.

To report a server's bootstrap to the *generic clients*, the service discovery pair of the basic device is used. That is, since each type of servers inherits from the basic device server, they know how to interact with the basic device client that is implemented by all the *generic clients*. Thus, the beaconing daemon of each server uses an omnihandle of the basic device client to report its bootstrap through an M2MI method call (*reportDeviceInfo*). The sequence diagram shown in Figure 4-1-1 depicts a typical sequence for service discovery.

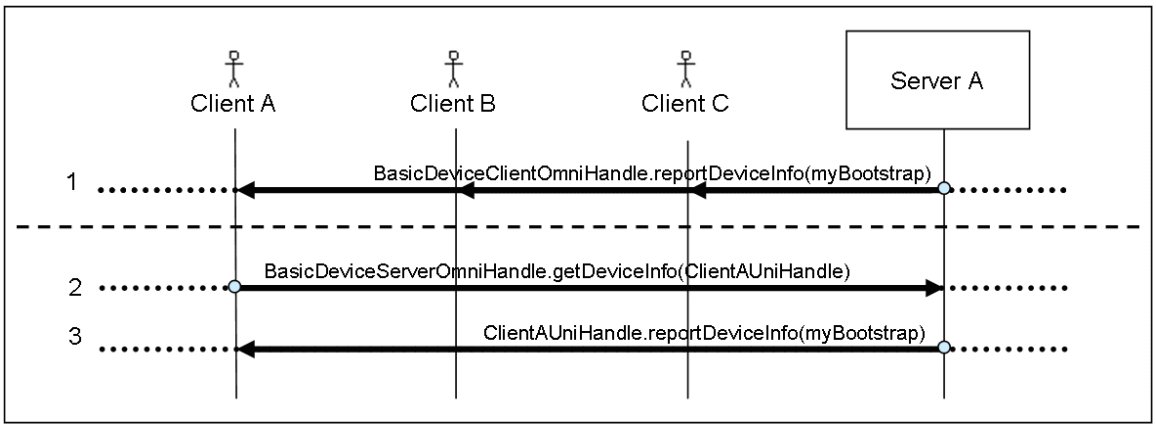


Figure 4-1-1: Service discovery sequence diagram

Server beacon:

Time 1: Server A reports its bootstrap to *generic clients* A, B, and C through the *reportDeviceInfo* interface of the basic device client's omnihandle. The three clients receive Server A's bootstrap.

Client beacon:

Time 2: Client A broadcasts a request for service discovery by using the *getDeviceInfo* interface of an omnihandle of the basic device server. Server A receives this request.

Time 3: Server A uses the client's unihandle to report its bootstrap.

At Time 2, if there are many servers in the area, all of them will receive Client A's request and report their bootstrap to Client A.

Figure 4-1-2 shows the components of the bootstraps used in this system

- **Server ID**: A server's identification. Should be universally unique.
- **Server's M2MI Handle** [1, 2]: Used to invoke a M2MI method call.
- **Server Name**: The name of a server.
- **Server Model**: The service type for a server. It is used in a *generic client* application to match a server with its corresponding control module (if any) in the *module repository* of the *generic client*.
- **Server Version**: Used for tracking the version of a server when matching up the server's control module.
- **Server Description**: A brief description of a server, such as a server's location.
- **Information Created Date**: Indicates the date of creation for a beaconing message. Used to determine the availability of a server.

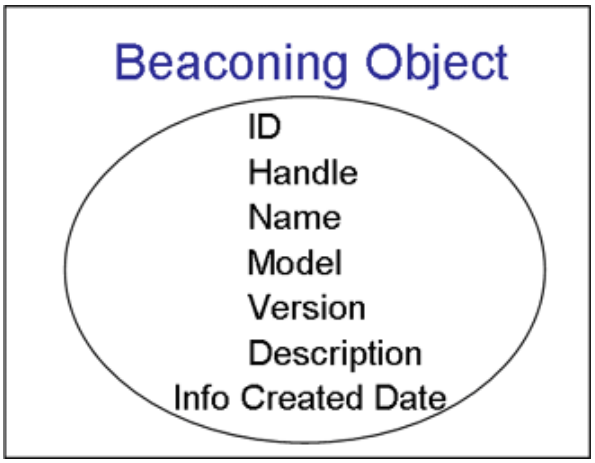


Figure 4-1-2: Server's beaconing object

When a server broadcasts its bootstrap, all the nearby clients will receive the message, determine the server type, and prompt a user with this information. In Figure 4-1-3, the available service types in an area are displayed on a notebook's *generic client* application. As we can see, the service types are organized in a tree structure.

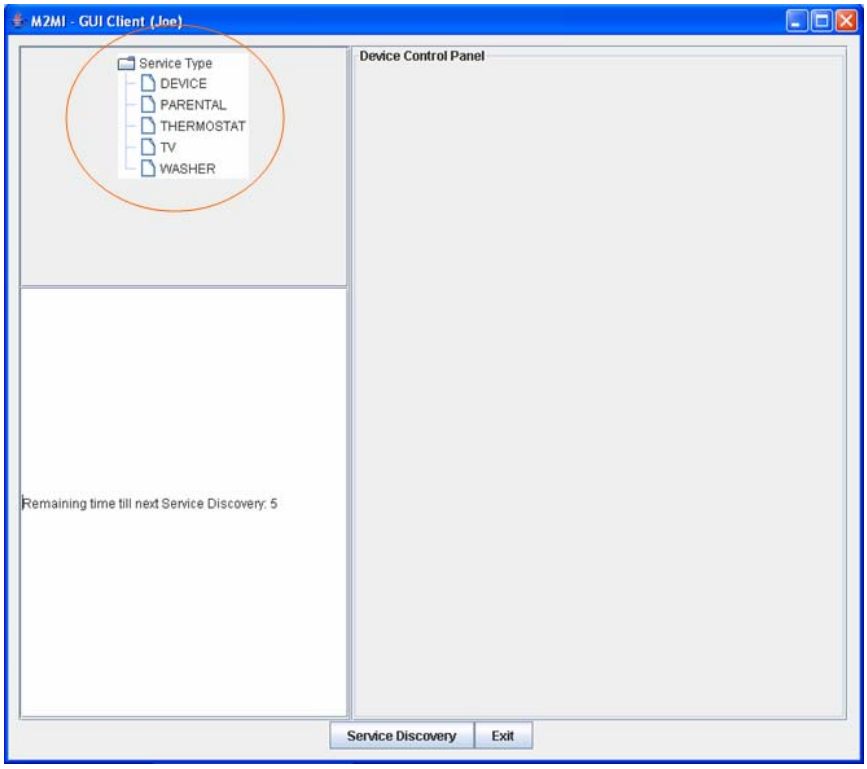


Figure 4-1-3: The discovered service types organized in a tree structure

In the figure, the top-left panel shows the types of the available services (model names). For example, if one or more TV servers are available in an area, the TV service type will be displayed in the service type tree (as shown in Figure 4-1-3).

When a user clicks on a desired service type, for example "TV," a preinstalled TV control module will be brought up to perform a service discovery to find all the available TV servers in the vicinity. All the available TV servers respond to the module's request and their names are listed on its control panel (as shown in Figure 4-1-4). To control a certain TV server, the user just clicks on the TV server's name and the GUI of this TV shows up.

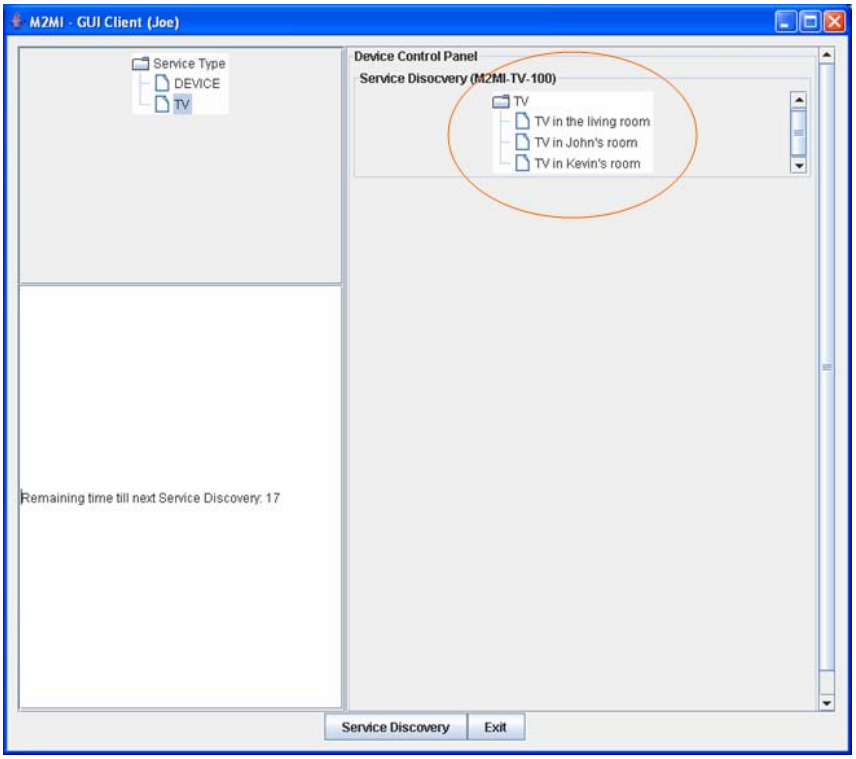


Figure 4-1-4: When clicking on the TV service type on the left-top corner, all the available TV appliances are displayed in the device control panel on the right-top corner.

To make the service discovery process more clear, let us consider a scenario in Joe's house. In order to turn on the TV in the living room, Joe first turns on his PDA (if it is not "On" yet) and runs the omni-remote controller client application. He sees that the TV service type is not shown on his PDA and then clicks on the service discovery button to do a service discovery. When the TV service type is found, the PDA refreshes its screen with the TV service type added in its service type tree. Joe clicks on the "TV" and all the available TV servers are shown on the PDA's control panel. He then clicks on "TV in the living room" and the GUI of the TV shows up. Now Joe can control the TV as he wishes to.

Just now, we talked about the process of a service discovery from a user's point-of-view. Now, let's see what is going on in the back-end of the system during a service discovery. Once the PDA is turned on, the *generic client* application is running and waiting for any receiving servers' beacon (*server beacon*). After a reasonable waiting time, the client application receives the beacons from all the available servers in the

house. If Joe does not want to wait and, instead, wants to access the TV in the living room right away, he just clicks on the service discovery button to perform a *client beacon* (Of course, he is very sure about the existence of the TV in his living room) and all the available servers will respond to the beaconing request. When all the available service types are found, a service type tree will be shown on his PDA's control panel. After Joe clicks on the TV service type, the client application goes to check the *module repository* and see if there is any control module that knows how to control the TV service type. If there is, the TV control module is picked out. This module then performs a service discovery to find all the individual TV servers and displays their names on the control panel. If the TV control module is not in the *module repository*, the message "No control module available" will be shown on the control panel. In this case, the TV control module will be installed.

4.2 Thermostat

Thermostats are among the most commonly-used appliances in daily life. In order to control or monitor a thermostat, a user usually has to go the location of its control panel. However, it will be much more convenient if we use a wireless device, such as a PDA, to control and monitor a thermostat without being physically present in front of its control panel. The following scenario describes this thermostat application.

After finishing an evening class in winter, Alice walks back to her dorm room and suddenly remembers that she had turned off the thermostat in her room that morning. Alice grabs her PDA from her pocket and turns it on. The omni-remote controller client application is initiated and a list of the available appliances in her room appears. Alice clicks on the thermostat and its control panel shows up on the screen. She clicks on the TURN ON button and the current room temperature "55F" shows up. It is too cold for her. So, Alice sets the room temperature to "75F." Hopefully, Alice's room will be heated before she gets home.

4.2.1 Class

Server Implementation:

- A server daemon beacons once every 30 seconds for the *generic clients*.
- The server beacons for its presence upon the request of the thermostat client application.
- The server broadcasts its current status upon the request of the thermostat clients.
- The server accepts commands through the M2MI calls.
- The server changes its temperature setting according to the schedule set by a user.
- The server updates its GUI whenever the server state is changed.

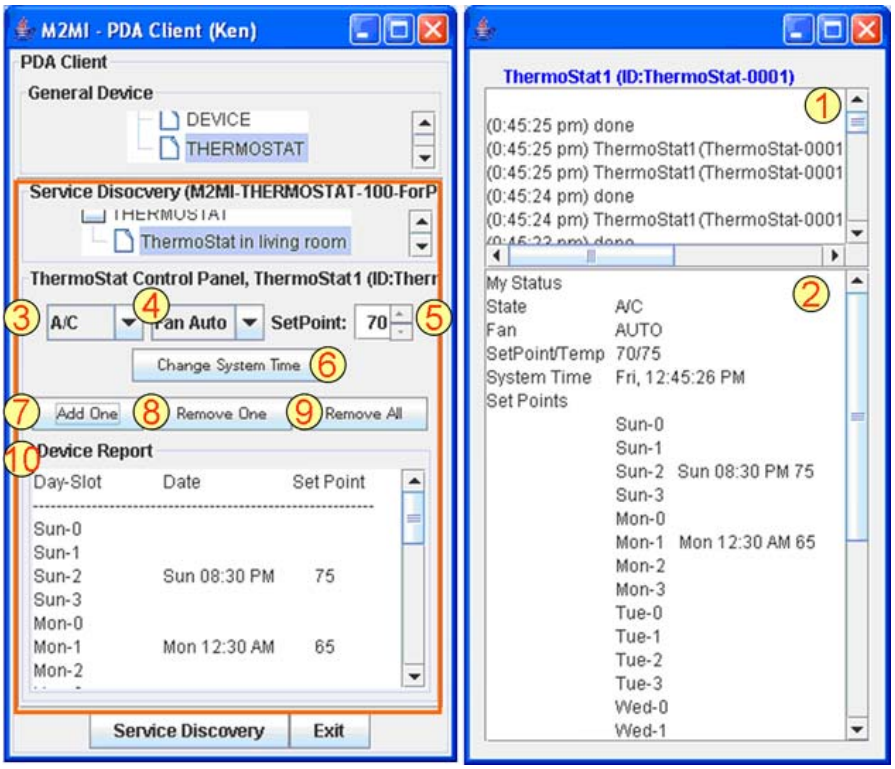
Client Implementation (the control module):

- The client displays the GUI to control the thermostat
- The client performs service discovery to find all available thermostat servers and displays their names on the control panel.

- The client requests the target server's status once a second to update the control panel.
- The client uses the target server's state to execute the GUI logic. For example, when the system is off, all the GUIs, except the TURN ON GUI, will be disabled.

4.2.2 Client and Server GUIs

The following two screenshots show the Java GUI implementation of the thermostat system. The orange box on the left hand side is the control panel of the thermostat system displayed on a user's control device (Client GUI). The window on the right hand side is a thermostat server application that shows its current status (Server GUI).



Server GUI (right-hand side):

- The text area 1: Display the system log of a thermostat server.
- The text area 2: Show the current system status. In my implementation, the thermostat has four set-point slots a day. "Sun-0" refers to the first set point slot on Sunday. "Sun 08:30 PM 75" means "On Sunday at 8:30 pm the set point was changed to 75."

Client GUI (shown on the left-hand side):

- Drop-down box 3: Used to turn on (either A/C on or heater on) or turn off the thermostat.
- Drop-down box 4: Used to set the fan to either "Fan Auto" or "Always On."
- Spinner 5: used to set a desired room temperature

- Button 6: Used to change the system time.
- Button 7: Used to add a set point schedule.
- Button 8: Used to remove a set point schedule.
- Button 9: Used to remove all schedules.
- Text area 10: Display a target server's set point schedule.

4.2.3 Sequence Diagram and Example

In the programming part of the thermostat application, I assume that only one client (a control module) can control a one thermostat server. However, in some cases, more than one client will control a thermostat at the same time. To allow this, we can run a second (or as many as we want) client application.

Figure 4-2-1 shows the sequence diagram of the initial interactions (including service discovery).

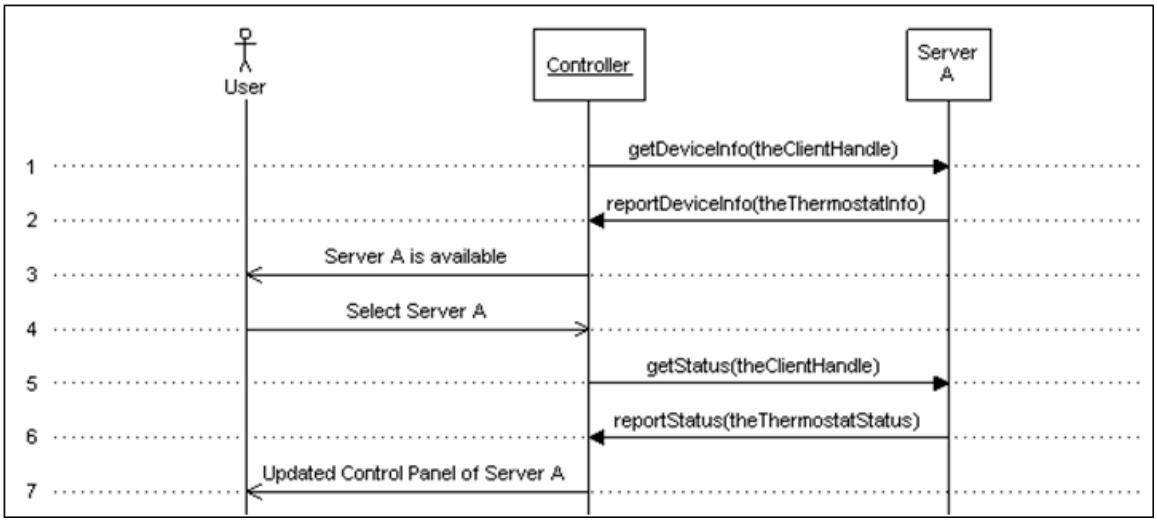


Figure 4-2-1: The sequence diagram of the initial service discovery.

Time 1: The Controller (the thermostat control module) broadcasts a message for discovering the local thermostat servers and Server A (a local thermostat server) receives this message.

Time 2: Server A calls the Controller's *reportDeviceInfo* interface with its bootstrap. The Controller receives this bootstrap and adds Server A to the list of the available thermostats.

Time 3: The Controller updates the list of the available thermostats and the User finds Server A on his or her control device.

Time 4: The User selects Server A.

Time 5: The Controller asks for the status of Server A and Server A receives the request.

Time 6: Server A reports its current status and the Controller receives it.

Time 7: The Controller shows the GUI of Server A with this server's status.

After the initial interaction phase, the Controller will start a *state reporting* daemon that requests Server A's status once a second in order to keep track of Server A's status. When a user controls this thermostat, the feedback of the command is from the server's state is reported as well. This design guarantees that every client has the newest server information and a valid GUI when multiple clients are used.

Figure 4-2-2 shows a scenario of two thermostat users.

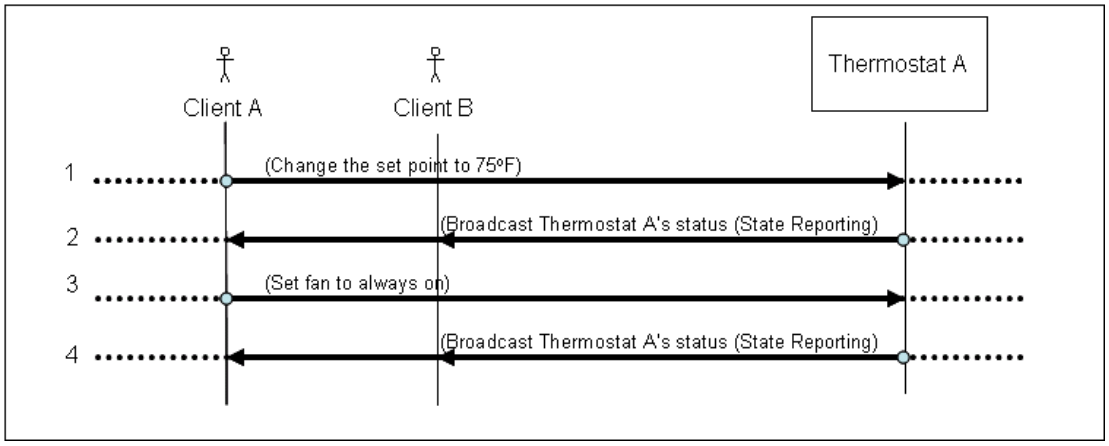


Figure 4-2-2: The sequence diagram of two thermostat users.

After the initial service display stage as described previously, Client A and Client B know the existence of Thermostat A. Assume Thermostat A's current set point is "70" and its fan setting is "Auto."

Time 1: Client A sends a command to change Thermostat A's set point to "75" degree. Thermostat A receives this message and changes its set point to 75.

Time 2: Thermostat A broadcasts its current state (state reporting), and Client A and Client B receives this message. Two clients know that Thermostat A's current set point is 75.

Time 3: Client A sends a command to change Thermostat A's fan setting to "On." Thermostat A receives this message and changes to "Fan On" setting.

Time 4: Thermostat A broadcasts its current state, and Client A and Client B receives this message. Two clients know that Thermostat A's current fan setting is "On" instead of "Auto."

No matter how many thermostat clients there are in the vicinity, the idea is basically the same. When a thermostat client changes the state of a thermostat server, all thermostat clients in the vicinity will know it at once through a server's state broadcast (state reporting).

4.3 Parental TV

In a family, we assume that parents decide what shows their children can watch on TV. That is, children have to get permission from both parents before switching to a protected channel. This seemingly simple activity actually involves three two-way

communications: between child and mother, between child and father, and between the child and the TV being watched. Especially when dad and mom are not in the same place, the child has to run around the house to get his or her parents' permission.

As you can see, the overhead of the two-way communications discussed above is obvious. However, if kid, mom, dad, and TV can talk together instead of communicating one on one, things may get easier. When a kid yells "Can I watch HBO," the voice passes over to Mom, Dad, and the TV. Mom then responds by yelling "YES" or "NO" back, and so does dad. Kid does not even need to do any operation on the TV. It gets the kid's request and the parents' responses and makes an operation based on the received messages.

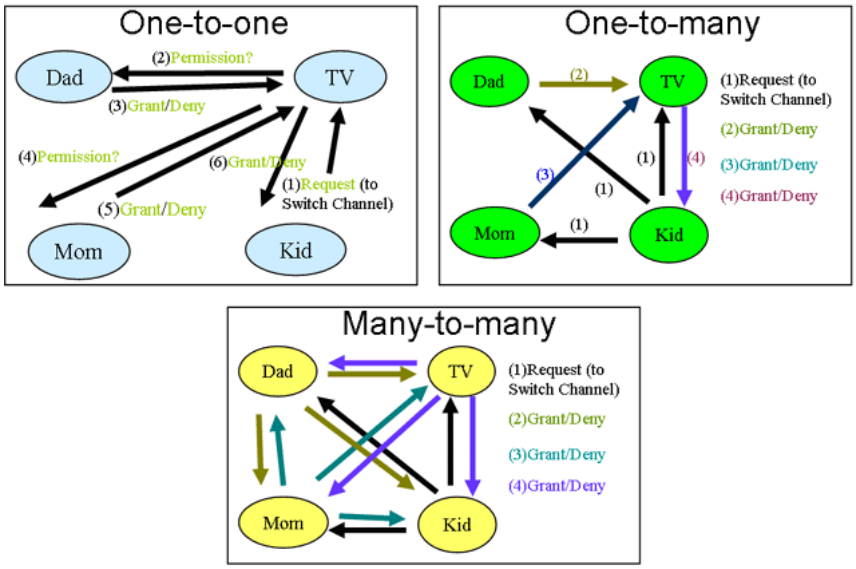


Figure 4-3-1: The one-to-one, one-to-many, and many-to-many communication process when a kid requests for switching to a protected channel.

Figure 4-3-1 gives us a clear understanding about three kinds of communications: one-to-one, one-to-many, and many-to-many. In the above scenario, by using a one-to-one communication,

(1) Kid asks to change the channel. This message is sent to TV.
(2) TV asks Dad to see if he grants Kid's request.
(3) Dad responds.
(4) TV asks Mom to see if she grants Kid's request.
(5) Mom responds.
(6) TV responds to Kid granting or withholding permission to change the channel.

As we see in this scenario, 6 steps are needed to complete the activity but only the TV knows the states (actions) of the other entities. That is, only the TV knows the responses from Dad and Mom.

By using one-to-many communication,

(1) Kid asks to change the channel. This message broadcasts to TV, Dad, and Mom.

(2) Dad responds.
(3) Mom responds.
(4) TV responds to Kid granting or withholding permission to change the channel.

As we see in this scenario, 4 steps are necessary but still only TV knows other unities' states (actions).

By using many-to-many communication, 4 steps are necessary and everyone knows the states (actions) of others.

(1) Kid asks to change the channel. This message broadcasts to TV, Dad, and Mom.
(2) Dad broadcasts its response.
(3) Mom broadcasts its response.
(4) TV broadcasts its response to the Kid's request.

We can see that four broadcasts is sufficient to finish this activity. Furthermore, since everyone knows the states (actions) of others, if one parent denies the request first, the other parent does not need to respond at all.

4.3.1 Class

Server Implementation:

- A server daemon beacons once every 30 seconds for the *generic clients*.
- A server daemon beacons once every 30 seconds for the TV clients.
- The server has a daemon for state reporting (once a second) for the TV clients.
- The server checks the identity of a client before executes a command.
- The server follows the flow diagram (shown in Figure 4-3-2) to deal with the kid's request for a protected channel.

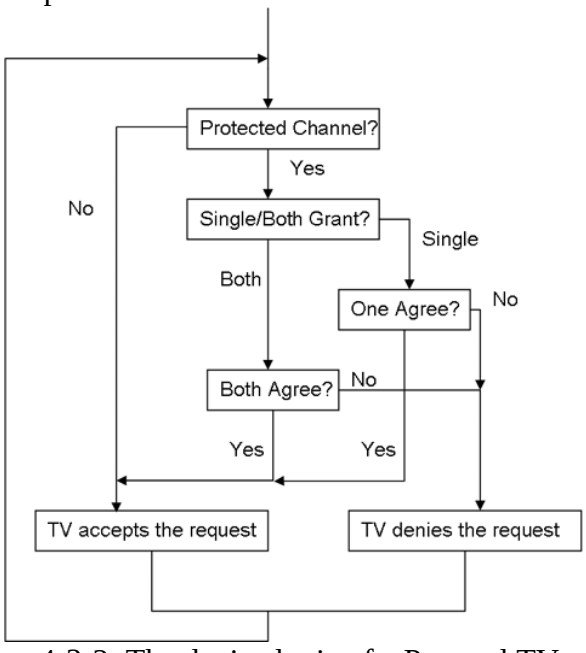


Figure 4-3-2: The device logic of a Parental TV server for kid's channel request.

Client Implementation (the control module):

- The client does service discovery to find out all the available TVs and displays their names on the control panel.
- The client provides control GUI for a Parental TV.
- The client updates a TV server's control panel whenever its newest state report arrives.
- The client uses a target server's state to execute the GUI logic. For example, when a TV server is turned off, all the buttons (except the TURN ON button) will be disabled.

4.3.2 Client and Server GUIs

The following three screenshots show the Java GUIs implementation of the Parental TV system. The parent client (in the middle) includes all the functions provided by a TV server while the child client (in the left) only has some controls appropriate for kids. A TV server's status displayed on both parents' and kid's control panel, will be updated with the change of its status. The orange box shows the plug-in control module of a TV server. The window on the right hand side displays a server's current status.

Server GUI (on the right-hand side):

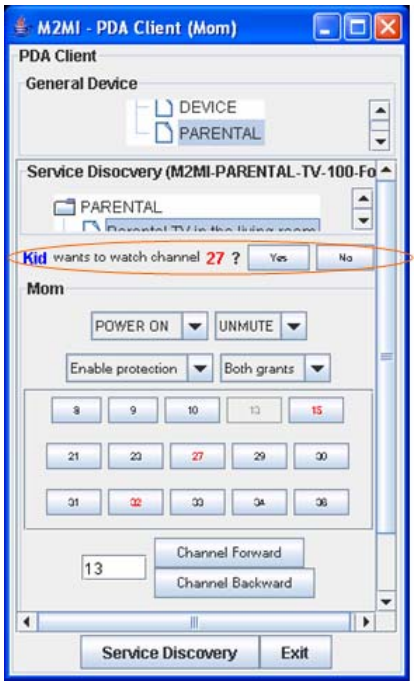
- Text area 1: Display the server log.
- Text area 2: Show the server status.

Client GUI:

- Drop down box 3: Used to turn on or off a server.
- Drop down box 4: Used to mute or un-mute a server.

- Drop down box 5: Used to enable or disable the parental control function.
- Drop down box 6: Used to set the parental control policy to either “Both grants” or “Single grant.”
- Button area 7: The available channel buttons. The button with a red number means a protected channel.
- Button area 8: Used to forward and backward channel. When a kid clicks the “Forward” or “Backward” button, the TV server will skip the protected channels and automatically jump to the next non-protected one.
- Button area 9: Used to turn up or down the volume of a server.

In general, when the parental control function is enabled, and if a kid only uses the forward and backward buttons, he/she will never be able to access a protected channel. When a kid clicks on the button of a protected channel, the request will be sent to the TV server and parents’ control devices. Then, parents give a “grant” or “deny” response and based on their response, the TV server makes an appropriate action. A screenshot of a kid’s message for requesting channel 27 is shown below.



4.3.3 Sequence Diagram and Example

In the programming part of the Parental TV system, the M2M communication and interaction involving two parent clients, one child client, and one Parental TV server is demonstrated. In order to receive the M2MI call of a switching channel message from a child, the TV control module must implement the Parental TV server interface, as well as the Parental TV client interface.

To simplify the sequence diagram of device interaction, service discovery and state reporting are not shown. However, we must keep in mind that the TV server’s status is

broadcast once a second. Figure 4-3-3 shows the sequence diagram of a kid's request to switch to a protected channel.

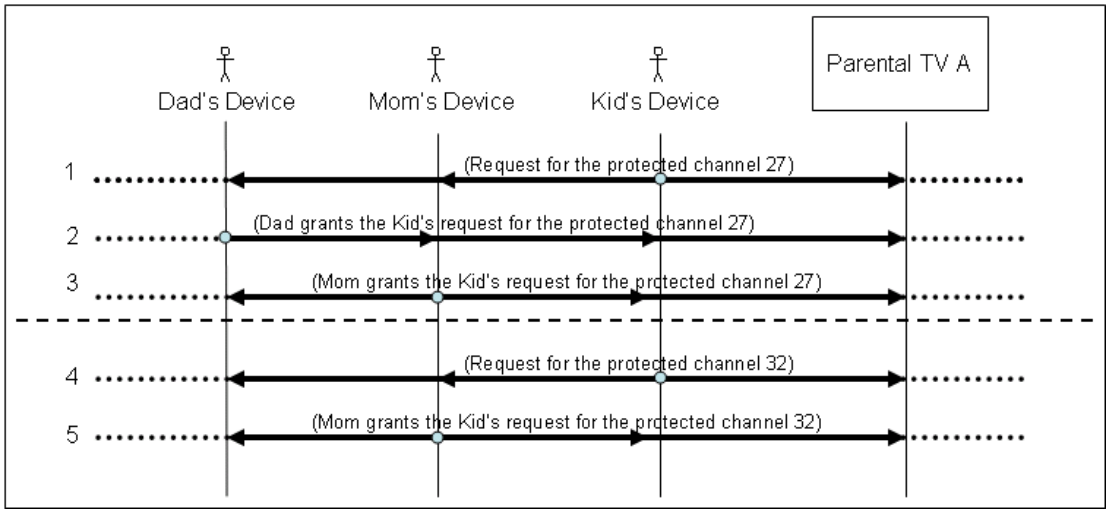


Figure 4-3-3: The sequence diagram of a kid's request for a protected channel.

When the parental control function is enabled and the parental control policy is set to "Both grants."

Time 1: Kid's device broadcasts a request for the protected channel 27. Mom and Dad's devices and Parental TV-A receive this request. Since Kid is not authorized to watch Channel 27 until he or she gets both parents' approval, TV-A will remember the request but not perform it. Meanwhile, the client application on Mom and Dad's devices will prompt them to respond to the child's request.

Time 2: Dad chooses to grant Kid's request and the message is broadcast to Mom, Kid, and TV-A. Since the parental policy is set to "Both grants," TV-A will remember Dad's response and wait for Mom's decision.

Time 3: Mom grants Kid's request too and the message is broadcast to Dad, Kid, and TV-A. TV-A notices that both Dad and Mom grant kid the authority of watching Channel 27 and then switches to Channel 27. Note: Kid gets the feedback by seeing the current channel changed to channel 27 on TV-A and/or on the control panel.

When the parental control function is enabled and the parental control policy is set to "Single grant":

Time 4: Kid's device broadcasts a request for the protected channel 32. Mom and Dad's devices and Parental TV-A receive this request. Since Kid is not authorized to watch Channel 32, TV-A will memorize the request and wait for the parents' decisions. Meanwhile, the client application on Mom and Dad's devices will prompt them to respond to the child's request.

Time 5: Mom chooses to grant the Kid's request and this message is broadcast to Dad, Kid, and TV-A. Since the parental policy is "Single grant," TV-A will change the current channel to Channel 32 right after receiving Mom's granting massage. However, Dad's prompting message will disappear because the

client application knows the control policy and receives Mom's granting message.

When other scenarios such as "Mom (Dad) denies but Dad (Mom) grants" happen, TV-A will still switch a required channel as long as the control policy is "Single grant." If the control policy is "Both grants" and when Mom (Dad) denies a request, Dad's (Mom's) prompting message will disappear. Since one of parents has already denied a request, it is not necessary for the other to give any response. In this case, the TV of course will not switch to a protected channel.

This design is for the purpose of producing the minimal traffic for each control operation. As we can see, the M2MI reduces a tremendous amount of network traffic compared to a one-to-one communication. Furthermore, since each device is able to know the state of others, a highly collaborative system is possible without adding extra traffic to the network.

4.4 Washing Machine

Dorm residents often find themselves in a laundry room without a washing machine available, especially during weekends. It will be more convenient if they check the availability of washing machines before going to do their laundry. Also, if no washer is currently available, students should be able to register their load(s) for the next available one.

The following scenario describes the control of washing machines in a dorm. While watching TV, Alice sees her piled-up clothes and decides to get them cleaned up. Before going to do her laundry, she wants to make sure there is a free washing machine in the dorm's laundry room. She grabs her PDA and sees if there is an available washer. If there is one available, she can reserve it and go to do her laundry right away. If none of them are available right now, she should be able to register her load(s) and wait for the next available one. The omni-remote controller system will tell her when it is her turn. Furthermore, the omni-remote controller system should allow Alice to swap her waiting position with others on the waiting list. For example, when Alice puts her name on the waiting list, she sees her best friend Bob's name is in the first line. Because Alice has an appointment with her classmates soon, she wants to swap the waiting position with him. She sends out a request to Bob and Bob agrees. Then, Alice can do her laundry as soon as the next washer becomes available.

4.4.1 Class

Server Implementation:

- A server daemon beacons once every 10 seconds for the *generic clients*.
- A server daemon beacons once every 10 seconds for the washer clients.
- The server has a daemon for state reporting (once a second) for the washer clients.
- The server accepts command through a M2MI call.

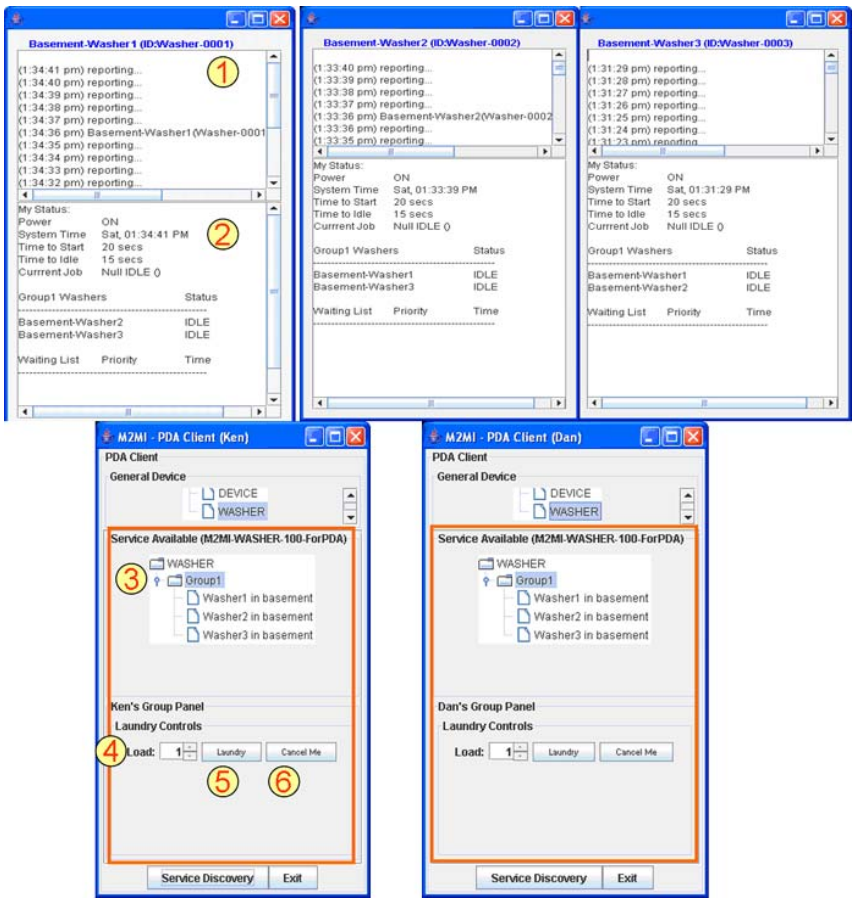
- Each server has a job queue to accept requests from clients. A group of washers work cooperatively by listening to the broadcasting message of their peers and managing their job queue accordingly. For example, when a user registers his or her load to a washer group, the user's control device broadcasts a request and all the washers in the group receive this message. When one of these washers takes the job, its peers will remove this job from their job queue.

Client Implementation (the control module):

- The client displays the control GUI of a washer group.
- The client discovers all the nearby washers and then groups them by their group ID.
- The client updates the control panel whenever a new state-reporting message from servers arrives.
- The client uses servers' states to execute the GUI logic.
- The client sends out "swapping waiting position request" to another client.

4.4.2 Client and Server GUIs

The following screenshots show the Java GUIs of the Washing Machine system. They are 3 washer servers (on the top) and 2 clients (at the bottom).



Server GUI:

- Text area 1: Display a server log.
- Text area 2: Display a server status. It also shows its peers and people on its waiting list (if any).

Client GUI:

- Service Tree 3: The available washers are sorted under its group name. It is possible to find more than one group in an area. The following screenshot shows that there are two washer groups in an area.

When a group is selected, a control panel for the group shows up.

- Spinner 4: Used to set the number of loads required by register.
- Button 5: Used to send out a request for registering the number of loads.
- Button 6: Used to remove all the jobs from a washer's waiting list.

When a certain washer server is selected, the control panel of the target washer shows up (as shown below).

- Button 7: Begins a washing cycle.
- Drop down box 8: Used to send out a request for swapping the waiting position with another user.
- Text area 9: Display all the people who are on the waiting list.

When a user clicks on the "laundry" button (Button 5), a message is broadcast to a group of washers. One of the washers in this group takes the job and sends an "available" message to the user. The message (circled in orange) is shown on the following screenshot.

After the user is ready to wash his or her clothes, he/she can click on the START button (which shows up when the reserved washer's name is clicked) to begin a washing cycle. After the load is done, the washer will send a "load completed" message to the user.

4.4.3 Sequence Diagram and Example

In the programming part of this washing machine system, I demonstrate an example of the M2M communication and system collaboration, which involves three washer servers and two washer clients. The M2M communication happens when a user wants to swap his or her waiting position with others. Through the M2M communication, a position-swapping task can be done with two broadcasting messages—a requester's requesting message and a requestee's response message. When a user broadcasts a request for swapping position, all washer servers, in order to receive the M2MI calls, need to pretend they are washer clients by implementing the washer client interface. Thus, both a requester's message and a requestee's response can be received by washer servers. By this, washer servers know the interaction between a requester and a requestee and then swap their waiting position (when applicable). The sequence diagram of swapping a waiting position is shown in Figure 4-4-2.

System collaboration is needed when a certain group of washers work together on managing the laundry loads. A sequence diagram (shown in Figure 4-4-1) describes the system collaboration process. To simplify the sequence diagram, service discovery and state reporting are not shown. However, it is important to remember that each server's status is broadcast once a second. This means all the other entities in a certain area are able to get informed almost immediately after the state of a server has been changed.

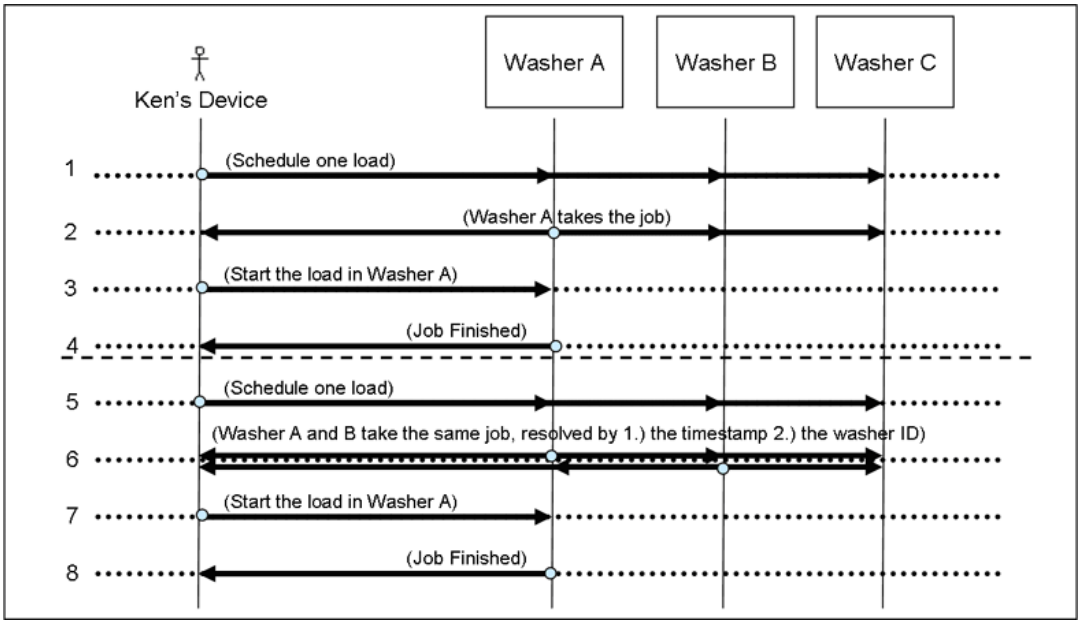


Figure 4-4-1: The sequence diagram of the washer control.

Let us assume a user will pick up his or her load right after it is done and Washer A, B, and C are in the same group.

Time 1: Ken's control device broadcasts a request for one load. This message goes to the respective queues of Washer A, B, and C. Each washer will wait for a random time interval (usually within 2 seconds) before it dequeues a job to avoid a race condition.

Time 2: Washer A dequeues the job and broadcasts its availability message to Ken's device, Washer B, and Washer C. Washer B and Washer C remove the job from their queues since Washer A took the job already. Ken's device prompts Ken that Washer A is reserved for him.

Time 3: Ken's load is ready and he clicks on the "Start" button. This command goes to Washer A and Washer A then begins a washing cycle.

Time 4: Washer A finishes the job and sends Ken a completion message.

The above example shows a typical sequence of interactions. A race condition described as below might happen sometimes.

Time 5: Ken's device broadcasts a request for one load. This message goes to the respective queues of Washer A, B, and C. Each washer will wait a random time before it dequeues a job.

Time 6: Washer A and Washer B have a very close random time value and they dequeue Ken's job almost at the same time. Both of them broadcast the "available" message. When Washer C receives the messages, it simply removes the job from its queue. When Washer A (Washer B) receives the message from Washer B (Washer A), it will first check the created date of the received message. In the example in the sequence diagram, Washer A broadcasts the "available" message slightly earlier than Washer B. Then, Washer A takes the job and Washer B will cancel the task and move on to the next job in its queue. If both Washer A and Washer B have the exactly same random time, which is possible, the "tiebreaker" is the name of the washers in alphabetical order. When Ken's device receives the first available message, it will prompt the user. And, when it receives the second available message in regards to the same job, the client device will use the time to compare first and then the name if needed. After figuring out which washer does the job, if necessary, it will change the prompting message.

Time 7: Ken goes to the laundry room and is ready to do his laundry. He clicks on the "Start" button. The command goes to Washer A and Washer A begins a washing cycle.

Time 8: Washer A finishes the job and sends Ken a completion message.

As we can see in the sequence diagram, some reasonably complex operations are done with just a few broadcasts in this washing machine system. But, when using a two-way communication, it requires more time and becomes more complex compared to what our system can do in the same scenario. Also, to "automatically" generate a system that performs highly collaborative tasks is too complex or impossible. That is, the semantic meaning of the functions between clients and servers are so complicated that they cannot be reached by the auto generation scheme. Thus, by using the M2M communication and

manually creating client and server applications, we can have a truly omni-remote controller system.

The sequence diagram in Figure 4-4-2 shows the position-swapping operation between two users.

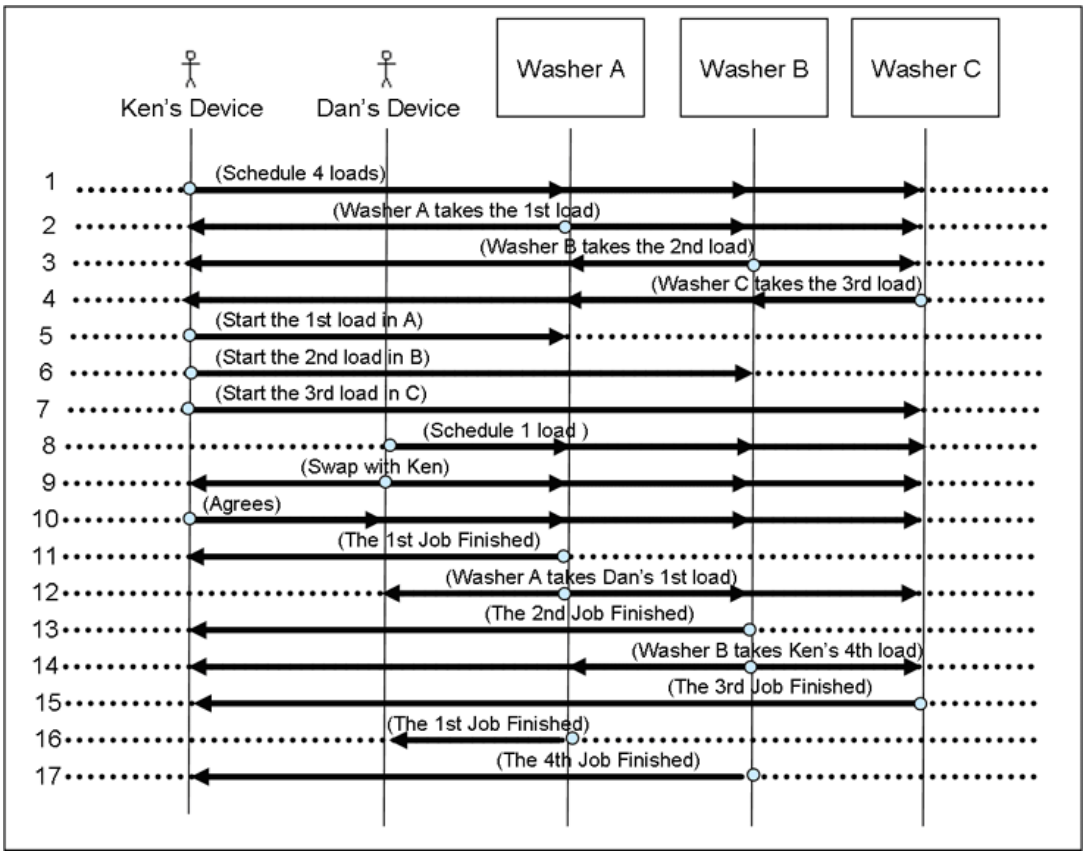


Figure 4-4-2: The sequence diagram of the swapping position request.

Let's assume a user will pick up his or her load right after it is done and Washer A, B, and C is in the same group.

Time 1: Ken's device broadcasts a request for 4 loads. This requesting message goes to the respective queues of Washer A, B, and C. Each washer will wait a random time interval before it dequeues the job.

Time 2: Washer A dequeues the 1st load job and broadcasts an "available" message. Washer B and C remove this job on their queues. Ken's device prompts Ken that Washer A is reserved for his 1st load.

Time 3: Washer B dequeues the 2nd load job and broadcasts an "available" message. Washer A and C remove this job. Ken's device prompts Ken that Washer B is reserved for his's 2nd load.

Time 4: Washer C dequeues the 3rd load job and broadcasts an "available" message. Washer A and B remove this job. Ken's device prompts Ken that Washer C is reserved for his 3rd load.

Time 5: Ken starts his 1st job.

Time 6: Ken starts his 2nd job.

Time 7: Ken starts his 3rd job.
Time 8: Dan's device broadcasts a request for 1 load. This message goes to the respective queues of Washer A, B, and C.
Time 9: Dan requests swapping waiting position with Ken using his control device. Not only Ken but also Washer A, B, and C receive Dan's requesting message by implementing the washer client interface. Washer A,B, and C know Dan wants to swap position with Ken.
Time 10: Ken agrees and the message is broadcast. Dan's device receives it and prompts Dan with Ken's decision. Washer A, B, and C receive Ken's response message too and then swap the load position of Dan and Ken.
Time 11: Washer A finishes Ken's 1st job and sends Ken a message to pick up.
Time 12: Washer A dequeues the job which is Dan's 1st load and broadcasts an available message to Dan. Washer B and C remove this job. Dan's device informs Dan that Washer A is reserved for Dan's 1st load.
Time 13: Washer B finishes Ken's 2nd job and sends Ken a completion message.
Time 14: Washer B dequeues the job which is Ken's 4th load and broadcasts the "available" message to Ken. Washer A and C remove this job. Ken's device prompts Ken that Washer B is reserved for his 4th load.
Time 15: Washer C finishes Ken's 3rd job and sends Ken a completion message.
Time 16: Washer A finishes Dan's 1st job and sends Dan a completion message.
Time 17: Washer B finishes Ken's 4th job and sends Ken a completion message.

As we can see in this demonstration, the M2MI reduces a tremendous amount of network traffic compared to the one-to-one communication. Since any entities in the washing machine omni-remote controller system are able to know the state of other entities, a system cooperative system can be designed.

4.5 System Summary

My proposed personal omni-remote controller system has the following significances:

- *Smart Devices*: The client and server devices must be "smart" enough to compute instructions, save their states, and communicate with other devices wirelessly. The "smart device" idea can be found in most of the omni-remote controller projects, such as [3, 24].

- *Radio Frequency:* As opposed to the projects that use IR as a communication medium, my system uses RF, which covers a wider area and is able to penetrate obstacles.

- *Communication Scheme*: The one-to-one, one-to-many, and M2M communications are all possible in the use of the M2MI. Depending on the application, a system designer can choose the best-suited communication scheme. In most of the omni-remote controller projects, the one-to-one (two-way) communication scheme is usually used.

- *Platform Independent (Java platform)*: My proposed personal omni-remote controller system is built by using the Java language which has cross-platform ability. With some extra work, we can convert the existing version of M2MI/M2MP to suit the Java 2 Platform, Micro Edition (J2ME), which is designed for resource-limited devices such as a PDA. Thus, a M2MI/M2MP version for J2ME will provide an environment for running the omni-remote controller client application in portable devices.

- *Decentralized System:* The personal omni-remote controller system, built on top of the M2MI/M2MP, is a decentralized system that does not require a central server. Since the M2M paradigm is a new communication model, none of the existing omni-remote controller projects use this design as far as I know.

- *Object-oriented Type Hierarchy:* An object-oriented type hierarchy has the advantages of reusing interfaces, partially controlling servers, and redefining methods.

- *Device Control Module:* Each control module controls a type of device. A *generic client* application can use the control module of a type to control its corresponding server. I found many omni-remote controller projects use XML to automatically generate a control interface for server. However, auto generation is error-prone. When using the control module approach, an accurate UI can be provided.

- *System Collaboration:* one of the reasons that client and server devices can work collaboratively is because both client modules and server applications are implemented by the same device vendor. None of the projects I have found provide the functionality of device collaboration.

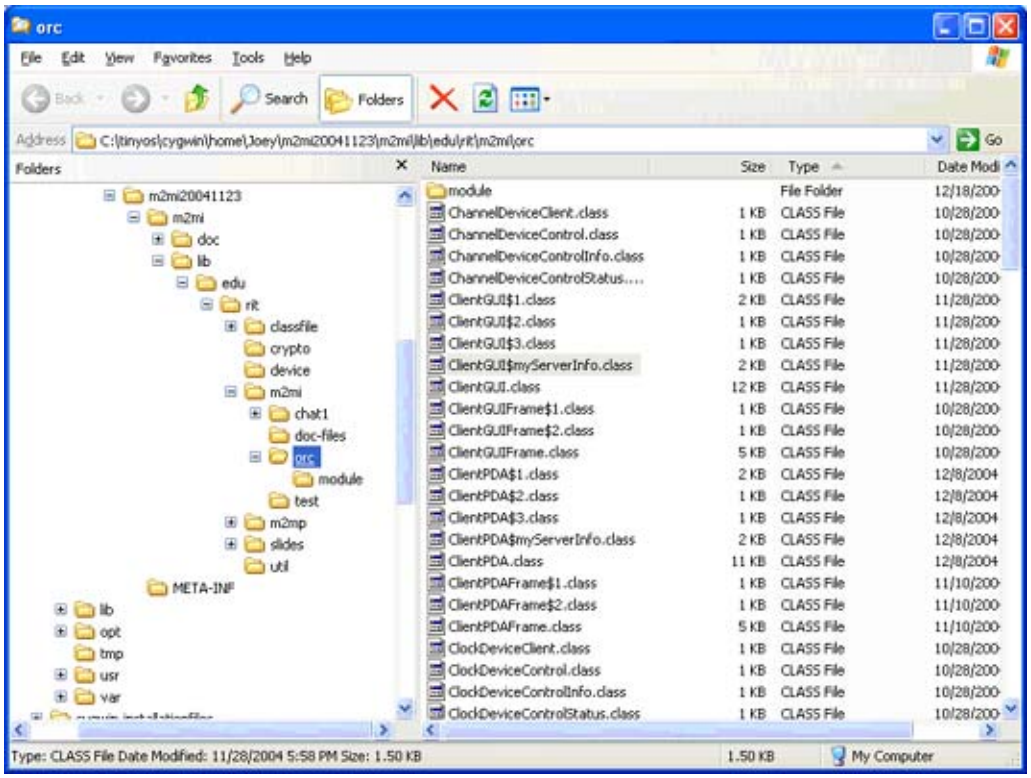
- *A dual-mode service discovery:* The service discovery scheme used in this system has the advantage of both *client* and *server beacons*. A client device can either passively wait for server's beacon or actively request all the available servers to provide their beaconing message.

5 User's Manual

This section gives the software requirements and the instructions for running the demonstration code of my programming projects. Ideally, each simulated client and server application runs in separate PCs/Notebooks. However, it is possible to have all the client and server applications running in one single computer. Before running the demo, a user's system(s) must meet the following requirements.

Software Requirements:

1. Java 2 Standard Edition (J2SE): Install J2SE (or higher) in your system
2. The Many-to-Many Invocation Library (M2MI): Install M2MI (Java library) in your system following the instructions at http://www.cs.rit.edu/~ark/m2mi.shtml and http://www.cs.rit.edu/~ark/m2mi/doc/index.html. These webpage should guide you through the installation and configuration.
3. Unpack the orc.jar file to the folder *..\m2mi[yyyymmdd]\m2mi\lib\edu\rit\m2mi*. The path might look like the following screen snapshoot.



4. Repeat step 1 through 3 for each of your systems if any.
5. You have successfully installed all the required software.

Note: The interfaces of the devices are placed under the *orc* folder and the implementations (server applications and clients' control modules) are put in the *orc/module*/ folder.

Start Application

To run a simple test

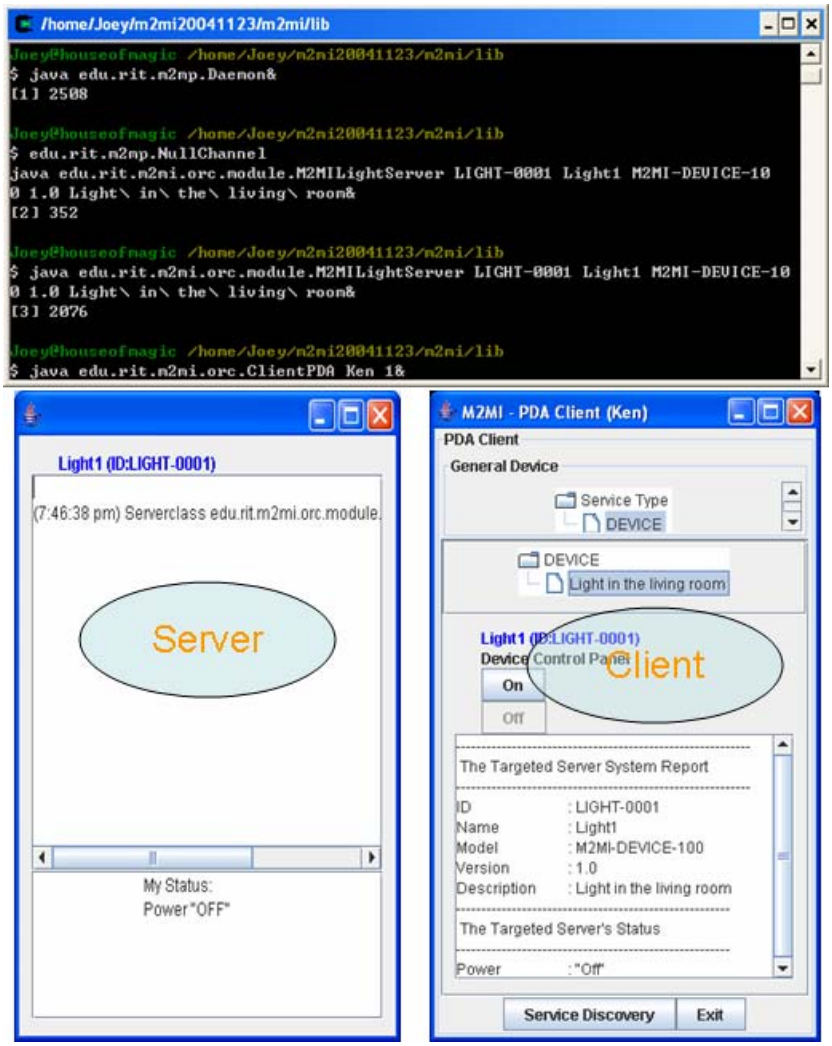
1. Start a terminal
2. Change the directory to *../m2mi[yyyymmdd]/lib*
3. Run the M2MP daemon:
   java edu.rit.m2mp.Daemon&

4. Start a light server:

java edu.rit.m2mi.orc.module.M2MILightServer LIGHT-0001 Light1 M2MI-DEVICE-100 1.0 Light\ in\ the\ living\ room&

   (You should see a Light server GUI as the following screenshot shows.)

5. Start a PDA *generic client*:
   java edu.rit.m2mi.orc.ClientPDA Ken 1&
   (You should see a PDA client GUI as the following screenshot shows.)



You should be able to use the client application accordingly.

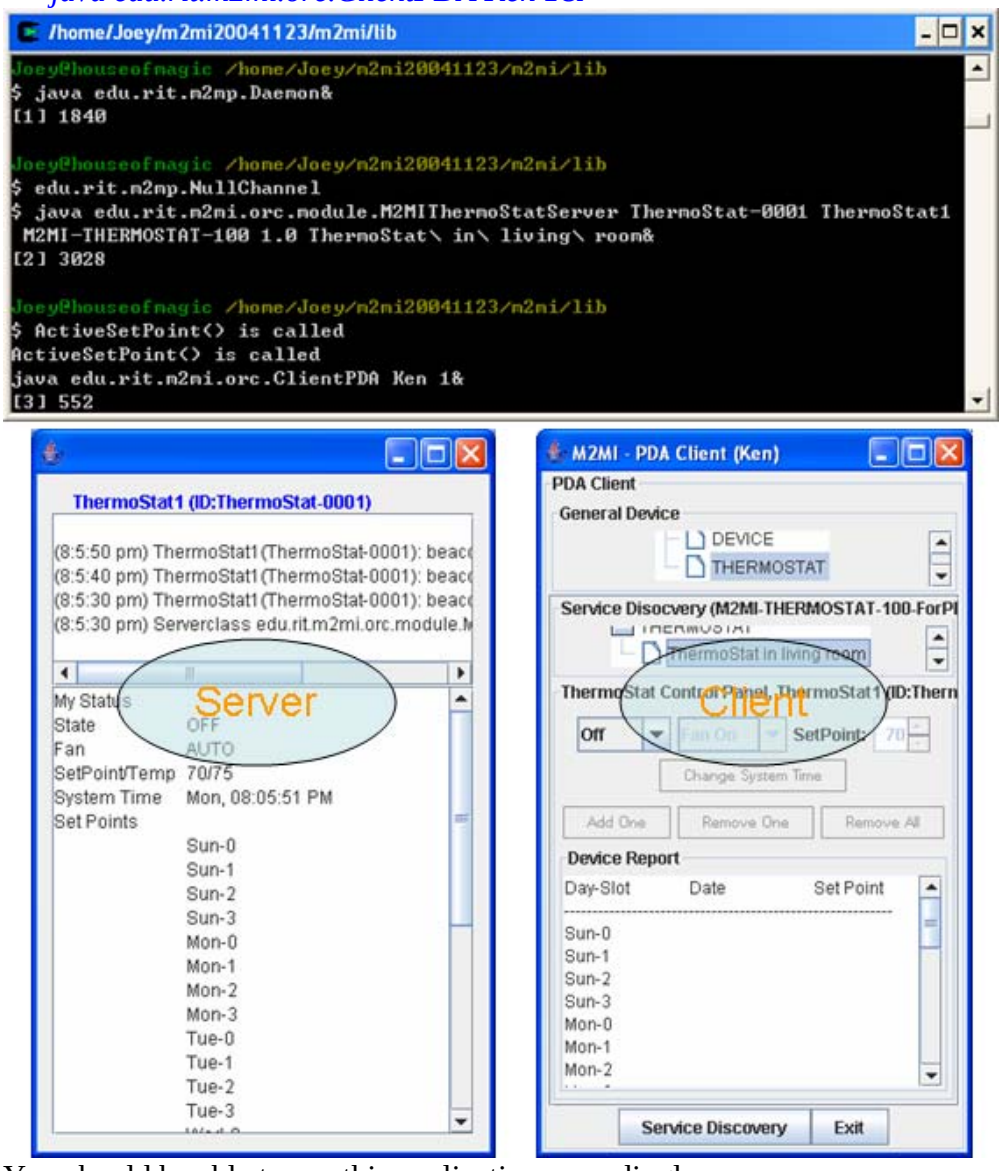
To run the Thermostat demonstration

1. Start a terminal
2. Change the directory to *../m2mi[yyyymmdd]/lib*
3. Run the M2MP daemon:
   java edu.rit.m2mp.Daemon&

4. Start a ThermoStat server:

java edu.rit.m2mi.orc.module.M2MIThermoStatServer ThermoStat-0001 ThermoStat1 M2MI-THERMOSTAT-100 1.0 ThermoStat\ in\ living\ room&

5. Start a PDA *generic client*:
   java edu.rit.m2mi.orc.ClientPDA Ken 1&



You should be able to use this application accordingly.

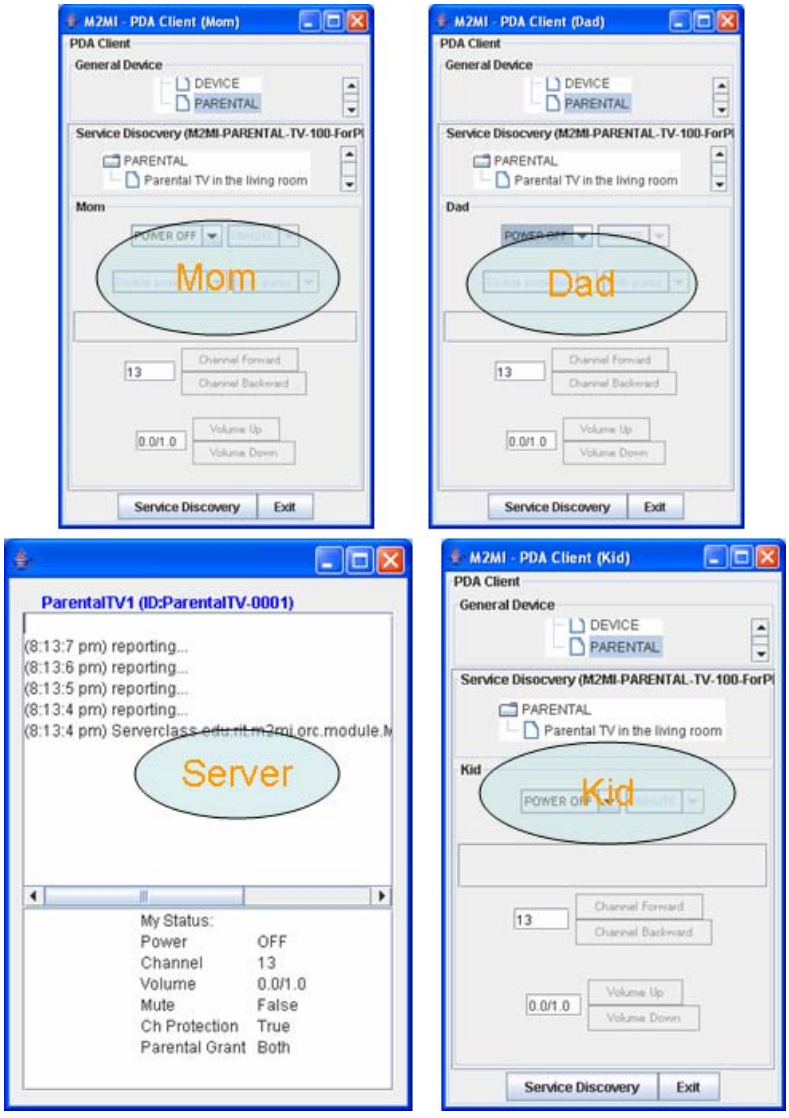
To run the Parental TV demonstration

1. Start a terminal
2. Change the directory to *../m2mi[yyyymmdd]/lib*
3. Run the M2MP daemon:
   java edu.rit.m2mp.Daemon&

4. Start a parental TV server:

java edu.rit.m2mi.orc.module.M2MIParentalTVServer ParentalTV-0001 ParentalTV1 M2MI-PARENTAL-TV-100 1.0 Parental\ TV\ in\ the\ living\ room&

5. Start PDA *generic clients*:
   java edu.rit.m2mi.orc.ClientPDA Kid 1&
   java edu.rit.m2mi.orc.ClientPDA Mom 2&
   java edu.rit.m2mi.orc.ClientPDA Dad 3&



You should be able to use this application accordingly.

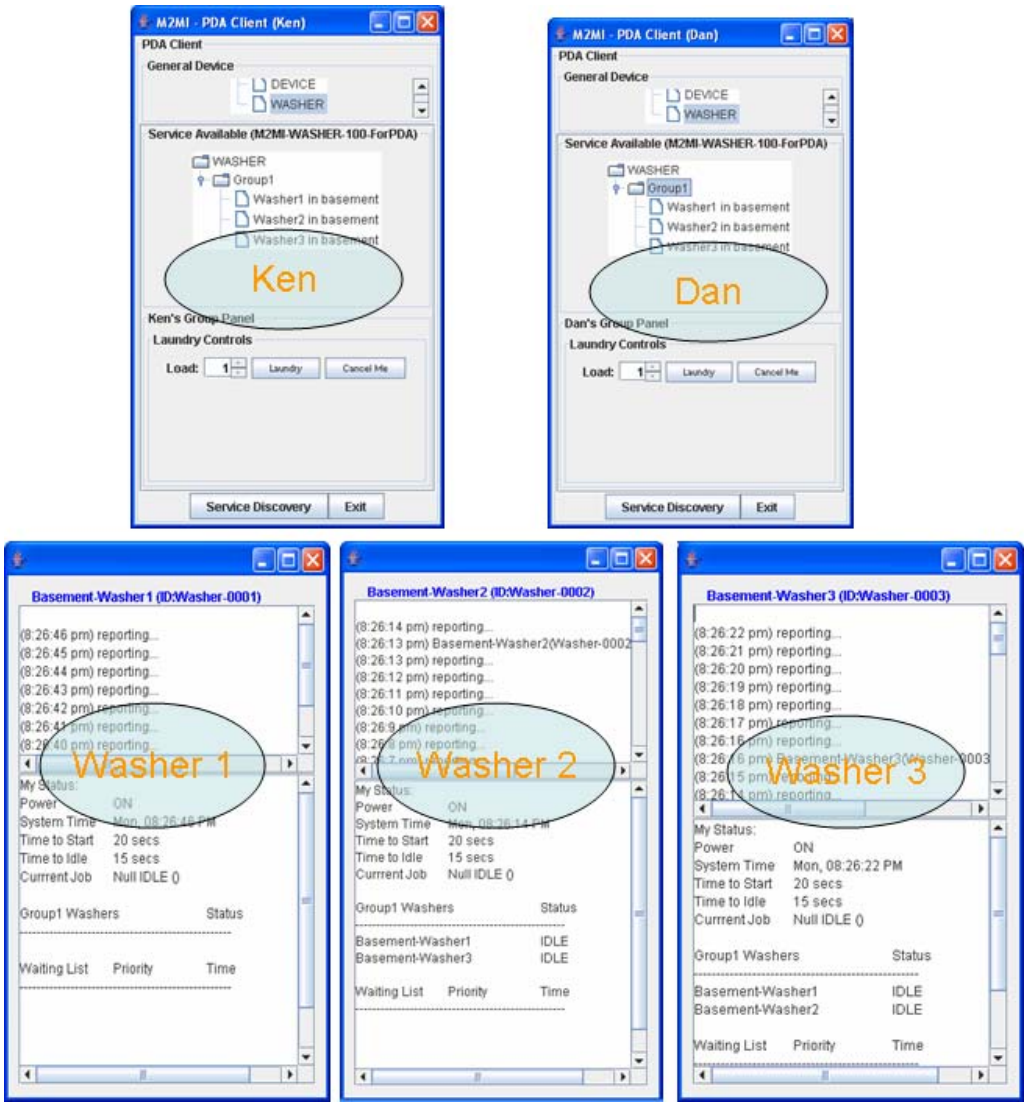
To run the Washing Machine demonstration

1. Start a terminal
2. Change the directory to *../m2mi[yyyymmdd]/lib*
3. Run the M2MP daemon:
   java edu.rit.m2mp.Daemon&

4. Start the washer servers:

java edu.rit.m2mi.orc.module.M2MIWasherServer Washer-0001 Basement-Washer1 M2MI-WASHER-100 1.0 Washer1\ in\ basement Group1&
java edu.rit.m2mi.orc.module.M2MIWasherServer Washer-0002 Basement-Washer2 M2MI-WASHER-100 1.0 Washer2\ in\ basement Group1&
java edu.rit.m2mi.orc.module.M2MIWasherServer Washer-0003 Basement-Washer3 M2MI-WASHER-100 1.0 Washer3\ in\ basement Group1&

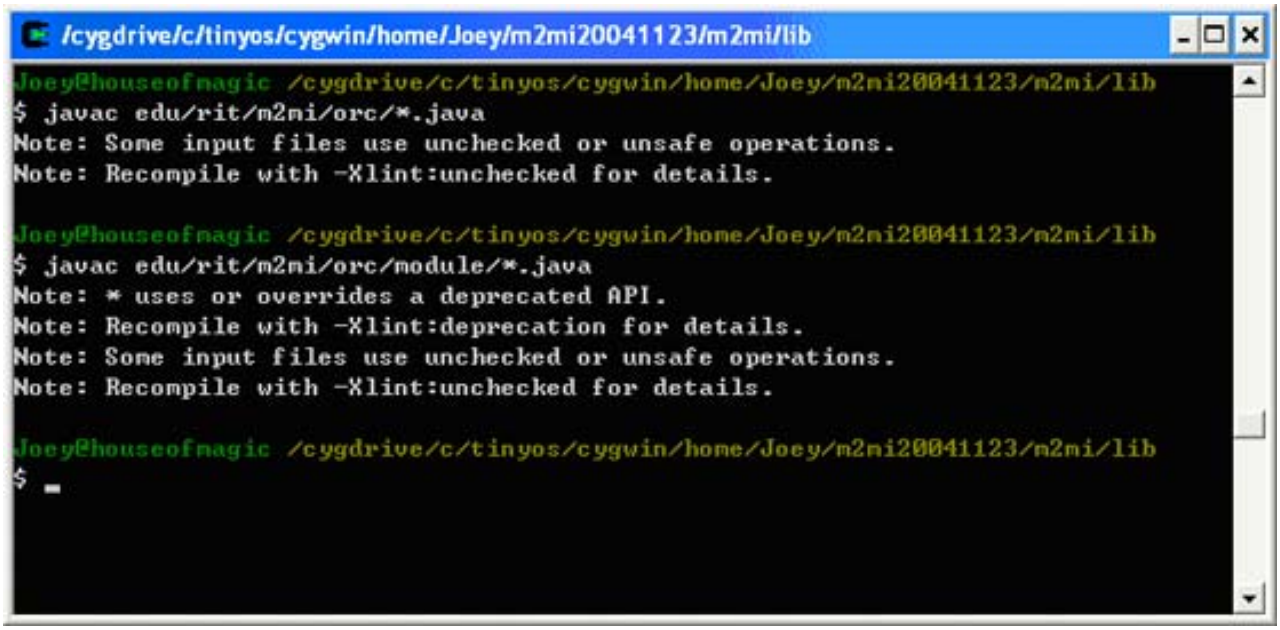
5. Start the PDA *generic clients*:
   java edu.rit.m2mi.orc.ClientPDA Ken 1&
   java edu.rit.m2mi.orc.ClientPDA Dan 1&



You should be able to use this application accordingly.

Compile the source code:
The files of *.java* and *.class* are included in the archive *orc.jar*. If you want to edit the source code, you can recompile them by entering *javac edu/rit/m2mi/orc/*.java* and *javac edu/rit/m2mi/orc/*.java* while in the *../m2mi[yyyymmdd]/lib* folder

Note: If a user wants to modify source code, *netBeans*[3] is recommended for editing and compiling the source files.



[3] See http://www.netbeans.org/

6 Future Work

Much work remains for the future. Topics include system collaboration in broadcast-based system, dynamic upgrade, and hot swapping of the control module, the M2MI security, the agent-based system for the broadcast-based system, and more interfaces to the hierarchy.

First, in the washing machine demonstration, I was able to implement the washers to work collaboratively. A further investigation on the system collaboration in the M2M environment is needed. Topics such as “the principles of system collaboration in the M2M environment” and “augmenting system collaboration to solve complex problems” will be interesting.

More research is needed for the issues of the dynamic upgrade and hot swapping. While a user is running the omni-remote controller system, if a new version of the control module is available (online), it is favorable to install the new control module dynamically and to perform a hot swap operation while the system is up running.

There are several security issues in the omni-remote controller system. As described in [33], the general security requirements of the M2MI-based system are *confidentiality*, *participant authentication,* and *service authentication*. 1) If a controller is not part of the omni-remote controller system, it should not be able to understand the content of the M2MI messages. 2) If a controller is not authorized by the server, it should not be able to perform the M2MI calls in that server. 3) If the server is not being authenticated, the omni-remote controllers should not trust that the server would perform the services it claimed. As described in [32], one way to solve the security problems is to have a *regional central security repository manager*. The manager would authenticate the user for the legibility of using the local servers and the services of the servers. When a new controller device enters the region, it will first get signed by the regional central security manager. Then, the controller discovers the local services. After that, it accesses the authorized services with the signed Key. Even though it is much easier to manage the security in one centralized place, the single-point-of-failure and the denial-of-service attack will also need to be taken into consideration. A decentralized key management in the *ad hoc* networks may be favorable [33]. One possible way of doing so is to have a shared hash table containing each member’s contribution of the group key and to use this group key for security purposes. The authors describe “each member of the group has an entry in the hash table [shared among all members of the group], which includes that member’s contribution to the group key.”

One future topic is the feasibility of the (mobile) *agent-based system* in the M2MI-based system. An agent takes one input from a user and, depending on the semantically meaning of the input, performs a series of operations. Furthermore, in the project of mobile agent [21], instead of working the job on a user’s client device, the user’s task is actually migrated from one mobile agent platform to another. However, does it make sense to continue the job if the task initiator leaves the network while the task is processing? Or, if we decide to stop the task when the task initiator leaves the network,

how would the system as a whole roll back to the initial state (if it is possible and necessary)?

Extending the interface hierarchy to handle more types of devices is needed to standardize the interfaces for various types of devices. Thus, we need to add more interface to the interface hierarchy.

Finally, and most importantly, the M2MI/M2MP is currently developed under J2SE; however, the J2SE cannot be installed in most of the mobile devices. Instead, J2ME is designed for these resource-limited, portable devices. Thus, a translation from the current implementation of the M2MI to J2ME is needed for the omni-remote controller system to run in these mobile devices.

7 Conclusion

This thesis proposed a real personal omni-remote controller system by the use of the M2MI. This system is designed for a user to control those "smart" appliances in a certain area with a portable device, such as a PDA. In order to control a new appliance, the control module of this appliance must be installed beforehand. After a user's control device performs service discovery, the control module will be brought up by a *generic client* application. Then, the user can access this appliance by using its control module. No training (or code learning) is necessary in this omni-remote controller system, and it is platform-independent through the use of Java language. Compared to the approach of an auto-UI generation, a manually created control module is able to provide a user with a more accurate UI. The M2MI-based, omni-remote controller system reduces network traffic, increases system efficiency, and promotes device collaboration. Client-Server, Client-Client, and Server-Server interaction are all possible in this system. Furthermore, by using *Object-oriented Type Hierarchy*, it has the advantages of 1) reusing interfaces, 2) partially controlling servers, and 3) redefining methods. Finally, a decentralized dual-mode service discovery scheme is proposed for discovering servers in an *ad hoc* environment. That is, a client can discover the servers by using primarily *server beacon* and then secondarily *client beacon.*

8 References

[1] Alan Kaminsky and Hans-Peter Bischof. Many-to-Many Invocation: A new object oriented paradigm for ad hoc collaborative systems. *17th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA 2002)*, Onward! track, Seattle, Washington, USA, November 2002.

[2] Hans-Peter Bischof and Alan Kaminsky. Many-to-Many Invocation: A new framework for building collaborative applications in ad hoc networks. *CSCW 2002 Workshop on Ad Hoc Communication and Collaboration in Ubiquitous Computing Environments*, New Orleans, Louisiana, USA, November 2002.

[3] The Pittsburgh Pebbles PDA Project. http://www.pebbles.hcii.cmu.edu/ retrieved January 12, 2005.

[4] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joe Hughes, Thomas K. Harris, Roni Rosenfeld, Mathilde Pignol. Generating Remote Control Interfaces for Complex Appliances. *CHI Letters: ACM Symposium on User Interface Software and Technology, UIST'02, 27-30* Oct. 2002, Paris, France. pp. 161-170.

[5] Pebbles Research Group. Using a Hand-Held as a Personal Universal Controller. http://www.pebbles.hcii.cmu.edu/puc/index.php retrieved January 12, 2005.

[6] Jeffrey Nichols and Brad A. Myers. Studying the Use of Handhelds to Control Smart Appliances. *International Workshop on Smart Appliances and Wearable Computing. IWSAWC 2003. In the Proceedings of the 23rd IEEE Conference on Distributed Computing Systems Workshops (ICDCS'03)*. May 19-22, 2003, Providence, Rhode Island. pp. 274-279.

[7] B. Myers. Using Handhelds and PCs Together. *Communication of the ACM*, Vol. 33, No. 11, November 2001. http://www-2.cs.cmu.edu/afs/cs/project/pebbles/www/papers/pebblescacm.pdf retrieved April 16, 2004.

[8] MAYA Design. PDG | Personal Universal Controller (PUC). http://www.maya.com/web/what/clients/what_client_pdg_puc.mtml retrieved April 17, 2004.

[9] Deborah Caswell and Philippe Debaty. Creating Web Representations for Places. *Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*. 2000, pp. 114 – 126. http://cooltown.hp.com/cooltown/ retrieved January 10, 2005.

[10] Donald Odell, AirMouse Remote Controls. An Optical Pointer for Infrared Remote Controllers. *Consumer Electronics, 1995., Proceedings of International Conference on*. June 1995.

[11] Jerry Fails, Dan Olsen. Light Widgets: Interacting in Every-day Spaces. http://icie.cs.byu.edu/Papers/LightWidgets.pdf retrieved December 13, 2004.

[12] Jerry Fails, Dan Olsem. MagicWand: The True Universal Remote Control. http://icie.cs.byu.edu/Papers/MagicWand.pdf retrieved December 13, 2004.

[13] Pacific Neo-Tek. OmniRemote Springboard Module. http://www.pacificneotek.com/omnisb.htm retrieved December 13, 2004.

[14] InVoca 12-in-1 Touch Screen Universal Remote. http://www.avagifts.com/s/Stereo_Equipment/InVoca_153_12_in_1_Touch_Screen_Universal_Remote_31580.htm retrieved December 13, 2004.

[15] Philips. ProntoPro. http://www.pronto.philips.com/index.cfm?id=636 retrieved December 13, 2004.

[16] LogiTech. Harmony-Internet Powered Universal Remote Controls. http://www.logitech.com/index.cfm/products/features/harmony/US/EN,CRID=2078,ad=g03 retrieved December 13, 2004.

[17] Ismail Coskun and Hamid Ardam, Ankara/Turkey. A Remote Controller for Home and Office Appliances by Telephone. *Consumer Electronics, IEEE Transactions on.* November 1998.

[18] Dan Olsen, Sean Jefferies, Travis Nielsen, William Moyes, and Paul Fredrickson. Cross-modal Interaction using XWeb. *Proceedings of UIST '00, San Diago, CA, USA.* November 2000, pp.191-200.

[19] Takuo Osaki, Tomohiro Haraikawa, Tadashi Sakamoto, Tomohiro Hase, and Atsushi Togashi. An Agent-based Bidirectional Intelligent Remote Controller. *Consumer Electronics, IEEE Transactions on 06/19/2001 -06/21/2001, Los Angeles, CA , USA.* Augest 2001.

[20] Takuo Osaki, Ryuji Nakayama, Tomohiro Haraikawa, and Atsushi Togashi. An Implementation of Intelligent Remote Controller Based on Mobile-Agent Model. *Consumer Electronics, 2002. ICCE. 2002 Digest of Technical Papers. International Conference on 2002*. 2002, pp. 202- 203.

[21] Soko Aoki, Jin Nakazawa, and Hideyuki Tokuda. Autonomous and Asynchronous Operation of Networked Appliances with Mobile Agent. *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on 2002*. 2002, pp. 743-748.

[22] Alexandre Sanguinetti, Hirohide Haga, Aya Funakoshi, Atsushi Yoshida, and Chiho Matsumoto. FReCon: a fluid remote controller for a Freely connected world

in a ubiquitous environment. *Personal and Ubiquitous Computing*. 2003, pp. 163 – 168.

[23] Jia-Ren Chien and Cheng-Chi Tai. The Information Home Appliance Control System – A Bluetooth Universal Type Remote Controller. *2004 **IEEE** International Conference on Networking, Sensing and Control, Grand Hotel, Taipei, Taiwan*. ICNSC2004.

[24] Todd Hodes, Randy Katz, Edouard Servan-Schreiber, and Lawrence Rowe. Composable Ad-hoc Mobile Services for Universal Interaction. *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking.* 1997, pp. 1-12.

[25] K. Eustice, T. Lehman, A. Morales, M. Munson, S. Edlund, and M. Guillen. A Universal Information Appliance. *IBM Systems Journal*, Vol. 38, No. 4, October 1999.

[26] Masahito Teuka, Yoshiyuki Honda, and Motonobu Kato. Development of Bi-Directional Remote Controller Protocol and Systems for Domestic Appliances. *IEEE Transactions on Consumer Electronics*, vol. 46, No. 3, August 2000.

[27] A. Fox, B. Johanson, P. Hanrahan, and T. Winograd. Integrating Information Appliances into an Interactive Workspace. *IEEE Computer Graphics and Applications*, Vol. 20, No. 3, May, 2000, pp. 54-65.

[28] Shankar Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd. ICrafter: A Service Framework for Ubiquitous Computing Environments. *Proceedings of the 3rd international conference on Ubiquitous Computing.* 2001, pp. 56-75.

[29] Yu-Chung Yang, Fan-Tien Cheng. Autonomous and Universal Remote Control Scheme. *IECON 02 [Industrial Electronics Society, IEEE 2002 28th Annual Conference of the]* November 2002, pp. 2266 - 2271.

[30] Tobin J. Lehman, Alex Cozzi, Yuhong Xiong, Jonathan Gottschalk, Venu Vasudevan, Sean Landis, Pace Davis, Bruce Khavar, and Paul Bowman. Hitting the Distributed Computing Sweet Spot with TSpaces. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, Vol. 35, Issue 4, March 2001, pp. 457 – 472.

[31] Brad Myers, Robert Malkin, Michael Bett, Alex Waibel, Ben Bostwick, Robert C. Miller, Jie Yang, Matthias Denecke, Edgar Seemann, Jie Zhu, Choon Hong Peck, Dave Kong, Jeffrey Nichols, Bill Scherlis. Flexi-modal and Multi-Machine User Interfaces. *IEEE Fourth International Conference on Multimodal Interfaces, ICMI'02*, Pittsburgh, PA. October 14-16, 2002. pp. 343-348.

[32] Niels Ferguson, Bruce Schneier. *Practical Cryptography*. Wiley, 2003.

[33] Hans-Peter Bischof, Alan Kaminsky, and Joseph Binder. A new framework for building secure collaborative systems in ad hoc network. *Second International Conference on AD-HOC Networks and Wireless (ADHOC-NOW '03)*, Montreal, Canada, October 2003.

[34] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T Spaces. IBM Systems Journal 37, No. 3, 454-474. 1998.