System Software for Audio and Visual Networked Home Appliances on Commodity Operating Systems

Tatsuo Nakajima

Department of Information and Computer Science
Waseda University
3-4-1 Okubo Shinjuku Tokyo 169-8555 Japan
tatsuo@mn.waseda.ac.jp

Abstract. This paper reports our ongoing project to build system software for audio and visual networked home appliances. In our system, we have implemented two middleware components for making it easy to build future networked home appliances. The first component is distributed home computing middleware that provides high level abstraction to control respective home appliances. The second component is a user interface middleware that enables us to control home appliances from a variety of interaction devices.
Most of our system have been implemented in Java, but several timing critical programs have been implemented in the C language, which runs on Linux. The combination of Linux and Java will be ubiquitous in future embedded systems. They enable us to port home computing programs developed on PC to target systems without modifying them, and Java's language supports enable us to build complex middleware very easily. Also, our user interface middleware enables us to adopt traditional user interface toolkits to develop home computing applications, but it allows us to use a variety of interaction devices to navigate graphical user interface provided by the applications.

1 Introduction

In the near future, home networks will connect a lot of networked home appliances such as digital TV, digital VCR, and DV cameras[15,20]. As the number of such appliances is increased, it becomes difficult to control the appliances individually. Also, it will not be realistic that each appliance has a different control device since we may not find an expected remote control device among a large number of the devices. Thus, it is desirable to use the nearest device such as PDAs and cellular phones to control the home appliances. On the other hand, these appliances will be connected to the Internet, and the services provided on the Internet will be integrated with the functions of home appliances to provide advanced services such as home shopping. The functionalities provided by the appliances will be very complicated, therefore high level abstraction is required to make it easy to build future advanced home applications that are constructed

R. Guerraoui (Ed.): Middleware 2001, LNCS 2218, pp. 273–294, 2001.

by the composition of a variety of services executed on the Internet or on the other home and personal information appliances.

We believe that the most important factor to develop future home appliances is the cost of products and the time to develop them. We need to consider a couple of issues to solve the problem. The first issue is to provide high level abstraction to make the development of home applications easy. Distributed middleware providing high level abstraction to control audio and visual home appliances for home appliances is a promising approach to solve the problem. However, it is not easy to develop such complicated middleware. Therefore, it is important to develop the middleware in a very portable way to reduce the cost to develop the middleware. Also, it is desirable to develop such complicated middleware by a sophisticated programming language that supports to build a large scaled software.

This paper reports our ongoing project to build system software for audio and visual networked home appliances. In our system, we have implemented two middleware components for making it easy to build future networked home appliances. The first component is distributed home computing middleware that provides high level abstraction to control respective home appliances. The second component is a user interface middleware that enables us to control home appliances from a variety of interaction devices.

Most of our system have been implemented in Java, but several timing critical programs have been implemented in the C language, which runs on Linux. The combination of Linux and Java will be ubiquitous in future embedded systems. They enable us to port home computing programs developed on PC to target systems without modifying them, and Java's language supports enable us to build complex middleware very easily. Also, our user interface middleware enables us to adopt traditional user interface toolkits such as Java AWT/Swing, GTK+, Qt to develop home computing applications, but it allows us to use a variety of interaction devices to navigate graphical user interface provided by the applications. Therefore, it will make the cost to develop home appliances dramatically cheap.

The remainder of this paper is structured as follows. In Section 2, we present the background of our work. In Section 3, we describe distributed middleware for audio and visual networked home appliances. Section 4 presents a user interface system to control home appliances. In Section 5, we show how our system works, and present the current status in Section 6. In Section 7, we describe several experiences with building our current prototype systems. Finally, in section 8, we conclude the paper.

2 Background and Motivation

Recently, embedded systems such as networked home appliances, internet appliances and personal area networks become very popular, and a lot of companies are developing very complex distributed embedded systems. In the future, the complexity will be increased exponentially, therefore middleware components providing high level abstraction will be very important.

2.1 Middleware for Home Computing Environments

In future computing environments, a variety of objects contain microprocessors and will be connected via various networks such as bluetooth, wireless LAN, and IEEE 1394. In [20], Mark Weiser describes some topics about "*Ubiquitous computing environments*", which promotes invisible embedded devices in every object near us. In his vision, embedded systems play a very important role to realize the goals, and one of the most popular environments to realize the goal is the *home computing environment*.

For building future advanced distributed home computing environments, high level standard interface is very important to reduce the cost of products and the time to develop them. This enables us to build a lot of reusable components that are used for building a variety of systems[12]. In the future, a lot of new standard middleware components will appear according to new requirements such as on-line shopping, smart spaces, and wearable computing. Therefore, it is important to use an infrastructure to build a variety of prototypes very quickly. The infrastructure needs to provide rich programming environments, a variety of device drivers, and various middleware components. We believe that an operating system that can be used universally, and distributed middleware for home appliances are key components to solve the above problems.

2.2 The Benefits of Our Approach

The aim of our project is to show the importance of middleware components to build advanced home computing systems. Our current project focuses on two issues. The first issue is to show the effectiveness of high level abstraction provided by our AV middleware component to build home applications. In future home computing environments, we will need to build large and complex middleware components. Therefore, our project is interested in how to build such complex software, and how to make the software portable. We believe that high level abstraction enables us to build advanced home computing applications quickly. The second issue is how to control home appliances in a very flexible way. The high level abstraction provided by our AV middleware component enables us to build flexible user interface, but it requires another middleware component to control user interface from various interaction devices.

There are several examples to show the effectiveness of our middleware components. We present four examples in this section. In the first example, home computing applications can be implemented on standard PCs, and the applications can be ported to target appliances without modifying them. The scenario can be realized by the adoption of COTS software components for building our middleware components, and shows that our approach makes the development cost very low. The second example is an application that changes graphical user interface according to a user's preference. The third example is an application that composes several home appliances, and the composition can be changed according to a user's preference. These two examples can be realized by the high level abstraction provided by our middleware components. In the fourth example, a home appliance can be controlled by various interaction devices. The

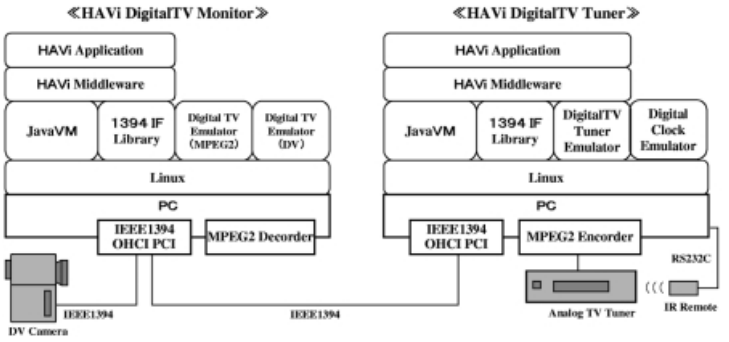

Fig. 1. Home Appliance System Architecture

devices are changed according to the situation of a user. For example, a PDA can be used to show graphical user interface that can be controlled on the PDA. If a user does not have a PDA, the nearest display can be used to display the graphical user interface that is controlled by a user's gesture. These flexible interaction scenarios can be realized by our middleware components although the graphical user interface is generated by using standard GUI toolkits.

3 Distributed Middleware for Networked Home Appliances

In this section, we describe distributed middleware for audio and visual networked home appliances, which we are building.

As shown in Figure 1, in our home computing system, we has implemented HAVi[7,8] that is a distributed middleware component for audio and visual home appliances. HAVi provides standard high level API to control various A/V appliances. The reason to choose HAVi as a core component in our distributed middleware for audio and visual home appliances is that HAVi is documented very well, and some real products will be available in the near future. Therefore, it is possible to use the products in our future experiments. The HAVi component is written in Java, and the Java virtual machine for executing HAVi runs on the Linux operating system. Also, continuous media streams such as DV or MPEG2 are processed in applications written in the C language running on Linux directly.

3.1 Software Designs for Complex Software

Our system provides a couple of high level abstraction for building new services in home computing because the abstraction makes it easy to implement a variety of home applications that need to compose several functionalities provided by different appliances. Also, the abstraction enables us to integrate home appliances with various services on the Internet.

However, it is not easy to implement the high level abstraction. Also, the abstraction should be used on many types of appliances. Thus, the implementation of the abstraction should be ported on various applications in an easy way. One approach to solve the problem is to adopt a software platform that is widely available. The solution allows us to build complex software on various appliances in an easy way if the platform can be used on a wide range of appliances.

However, there is no software platform that is suitable for a various types of software because we need to take into account several requirements to design complex software, and it is necessary to consider the tradeoff among the requirements since it is impossible to satisfy all requirements to develop complex software. Therefore, it is desirable to adopt several software platforms simultaneously according to the characteristics of respective software components. The approach allows us to select a suitable software platform for implementing each program.

In our approach, we configure several software platforms in a layered structure. We call the structuring *multi-layered software platforms*. Software in our system is divided into several components. One component requires high productivity, but it does not require severe timing constraints. On the other hand, it is more important to satisfy timing constraints for another component than to increase productivity. In our current implementation, the former component is implemented on a Java virtual machine, and the latter component is built on Linux directly. Also, the Java virtual machine runs on Linux in a hierarchical fashion. If our system needs to be ported on other operating systems, the component that runs on Linux directly should be ported, but a component implemented in Java needs not to be modified.

3.2 Structure of Our Home Computing System

Our system consists of five components. The first component is the Linux kernel which provides basic functionalities such as thread control, memory management, network systems, and file systems, and device drivers. The kernel also contains IEEE 1394 device drivers and MPEG2 decoder drivers which are required to implement HAVi. The second component is a continuous media management component running on Linux, and written in the C language. The component sends or receives media streams to/from IEEE 1394 networks and MPEG2 encoder/decoder devices, and it sends media streams to a speaker and a window system to simulate various audio and visual home appliances. The component also encodes, decodes or analyzes media elements in a timely manner.

The third component is a Java virtual machine to execute a middleware component such as HAVi. Java provides automatic memory management, thread management, and a lot of class libraries to build an sophisticated middleware components and home computing applications in an easy way. Also, Java Swing and AWT enable us to build sophisticated graphical user interface to write applications for home appliances. Therefore, a lot of standard middleware components for home computing assume to use Java due to the rich class libraries. The fourth component is the HAVi middleware component described in the next section. In our system, the component is written in Java. The last component is home com-

puting application programs that invoke HAVi API to control home appliances. The component adds extra functionalities to actual home appliances. One of the most important roles of the application component is to provide graphical user interface to control home appliances.

3.3 Home Appliances on Linux

This section describes an overview of our home computing system. First, we give a brief description about HAVi which is a distributed middleware for networked home appliances. Next, we show how our system emulates a variety of networked home appliances.

Middleware for Networked Home Appliances: A variety of audio/visual home appliances such as TV and VCR are currently connected by analog cables, and these appliances are usually controlled individually. However, there are several proposals for solving the problem recently. Future audio/visual home appliances will be connected via the IEEE 1394 interface, and the network will be able to deliver multiple audio and video streams that may have different coding schemes. HAVi(Home Audio/Video Interoperability) Architecture proposed by Sony, Matsushita, Philips, Thomson, Hitachi, Toshiba, Sharp, and Grundig is a standard distributed middleware for audio and visual home appliances, and provides an ability to control these appliances in an integrated fashion by offering standard programming interface. Therefore, an application can control a variety of appliances that are compliant to HAVi without taking into account the differences among appliances even if they are developed by different vendors.

The HAVi architecture also provides a mechanism to dynamically load a Java bytecode from a target device. The Java bytecode knows how to control a target appliance since the code implements a protocol to communicate between a client device and a target appliance. Therefore, the client device needs not to know how to control the target appliance, and the target appliance can use the most suitable protocol for controlling it. Also, the bytecode may be retrieved from a server on the Internet, therefore it is easy to update software to control target appliances.

The architecture is very flexible, and has the following three advantages.

- The standardized programming interface that provides high level abstraction hides the details of the appliances. Therefore, it is possible to build applications that are vendor independent. Also, the application can adopt future technologies without modifying target appliances. The feature enables us to build a new appliance by composing existing appliances.
- The software to control target appliances can be replaced by retrieving from the Internet. The feature removes the limitations of traditional home appliances since the functionalities may be updated by modifying software.
- The Java bytecode loaded from a target appliance enables us to use the most appropriate protocol. The feature allows us to choose the state of the art technologies without modifying existing applications when a company releases the appliance.

Especially, the first and the third advantages are suitable for developing a prototype very rapidly since the advantages allow us to adopt future technologies very easily without modifying existing appliances and home application programs.

The HAVi component consists of nine modules defined in the HAVi specification. In HAVi, the module is called *software element. DCM(Device Control Module)* and *FCM(Function Control Module)* are the most important software elements in HAVi. Each appliance has a DCM for controlling the appliance. The DCM is usually downloaded to a HAVi device, and it invokes methods defined in DCM when the appliance needs to be controlled. A DCM abstracts an appliance, and contains several FCMs. Each functionality in an appliance is abstracted as a FCM. Currently, tuner, VCR, clock, camera, AV Disk, amplifier, display, modem, and Web Proxy are defined as FCM. Since these FCMs have a standard API to control them, an application can control appliances without knowing the detailed protocol to access the appliances.

Home Appliance Emulation on Linux: The continuous media management component processes continuous media streams such as MPEG2 and DV streams. To process media elements in a timely manner, the components are implemented in the C language and are running on the Linux kernel directly.

The continuous media management component is controlled by a Java bytecode provided by the appliance implementing the component. As described in the previous section, the bytecode is downloaded to a HAVi device such as set-top boxes, and installed as a DCM software element. The bytecode implements a protocol to transmit a control message to a target appliance, and the message is processed by the continuous media management component to control media streams.

The continuous media management component contains several threads. Let us assume a continuous media management component that processes an audio stream and a video stream. The audio stream is delivered to a speaker, and the video stream is displayed on a window system.

The component contains five threads. The first thread initializes the component, sends a Java bytecode representing a DCM to a HAVi device, and waits for receiving commands from the HAVi device. When an application on the HAVi device invokes a method of the downloaded DCM, a control message is delivered to the component. Although HAVi does not define the protocol between a HAVi device and an appliance, the 1394 AV/C command may make it easy to implement DCMs.

The second thread receives packets from a IEEE 1394 network, and collect them as media elements. The media elements are classified into video frames and audio samples, and they are passed to the third and fourth threads respectively. In our system, threads are communicated through time-driven buffers to carry media elements. The time-driven buffer stores media elements in the FIFO order, but obsolete media in the buffer elements are dropped automatically. The third thread retrieves video frames from a time-driven buffer, and sends them to a window system. Also, the fourth thread retrieves audio samples from a time-driven buffer, and sends them to an audio device. The last thread monitors

the performance of a system, and controls a QOS value of each media stream according to the priority of media streams.

In our system, each media element has a header containing information to identify the media format, and it also contains timestamps to process the media element correctly. Currently, we adopt a RTP header as a header in our system to be assigned to each media element.

To process the timestamp assigned to each media element correctly, each machine's clock should be synchronized. However, there is a clock skew among different clocks, thus we require a mechanism to synchronize them. Also, to control an overload condition, each media stream's QOS should be changed. The detailed description to process continuous media in our system can be found in [11].

3.4 Applications Components

The purpose of an application in our system is to allow us to control home appliances with a variety of devices such as PDAs and cellular phones. In the future, we will have a variety of home appliances, but each appliance traditionally offers a different remote controller to control it. However, the approach has the following problems.

- To control an appliance, we usually need a specific remote controller.
- Everyone needs to use the same user interface to control an appliance.
- When a user controls an appliance, a system does not understand the situation of the user.
- When there are several appliances, a user needs to control them independently.

For solving the problem, an application should have three mechanisms. The first mechanism is to support various types of interaction devices to control an appliance. For example, we like to use an interaction device carried by us to control the appliance. The problem can be solved by the user interface middleware described in the next section. The second mechanism is to take into account our situations. For example, if a system knows the location of a user, the system can control an appliance in a more appropriate way. Also, if a system knows the preference of a user, there may be a better way to control the appliance. The third mechanism is to compose multiple functions in different appliances as one appliance. The mechanism allows us to define a new appliance very easily. The second and third mechanisms should be provided as appropriate high level abstraction to build application components easily. In the future, we like to provide the more high level abstraction to provide the second and third mechanisms on HAVi API.

3.5 A Sample Scenario

In this section, we present a scenario to build an emulated digital TV using our system. The scenario has actually been implemented using our prototype and

give us a lot of experiences with building distributed home computing environments.

In a program emulating a digital TV appliance, a Linux process receives a MPEG2 stream from an IEEE 1394 device driver, and displays the MPEG2 stream on the screen. The sender program that simulates a digital TV tuner receives the MPEG2 stream from a MPEG encoder, and transmits it to the IEEE 1394 network. Currently, the MPEG2 encoder is connected to an analog TV to simulate a digital TV tuner, and the analog TV is controlled by a remote controller connected to a computer with a serial line. A command received from the serial line is transmitted to the analog TV through infrared.

The receiver process contains three threads. The first thread initializes and controls the program. When the thread is started, it sends a DCM containing a Java bytecode to a HAVi device. In our system, the HAVi device is also a Linux based PC system. When the HAVi device receives the bytecode, a HAVi application on the HAVi device shows graphical user interface to control the emulated digital TV appliance. If a user enters a command through the graphical user interface, the HAVi application invokes a method of the DCM representing a tuner. Then, the HAVi device executes the downloaded bytecode to send a command to the emulated digital TV appliance, and the first thread will receive the command.

After the second thread in the emulated digital TV appliance receives packets containing MPEG2 video stream from an IEEE 1394 network, it constructs a video frame, and inserts the frame in a time driven buffer. The third thread retrieves the frame from the buffer, then the video frame is delivered to a MPEG2 decoder according to the timestamp recorded in the video frame. Finally, the decoder delivers the frame to a NTSC line connected to the analog TV display.

4 User Interface System for Networked Home Appliances

Our user interface system is based on the stateless thin-client system such as Citrix Metaframe[3], Microsoft Terminal Server[10], the AT&T VNC system[18], and Sun Microsystems Sun Ray[16].

In the future, a lot of new interaction devices will appear, but it is not realistic to rewrite all existing application programs for the new interaction devices. *Universal interaction* realized by our user interface middleware enables us to use the new interaction devices without modifying the applications programs that adopt traditional GUI toolkits. The benefits of universal interaction enable us to behave uniformly either in home, in offices, or in public spaces when controlling a variety of appliances.

4.1 Design Issues

There are several ways to realize the goals described in the previous section. The most important issue is how to support a variety of interaction devices. Traditional user interface systems assume a few types of interaction devices. For example, mouses, track points, and touch screens are used as input devices.

There are three alternative approaches to build a flexible user interface system as shown below.

- Builds a new user interface system from scratch.
- Builds a new device independent layer that invokes respective traditional user interface systems for a variety of interaction devices.
- Captures input and output events of traditional user interface systems, and transforms the events according to interaction devices.

We have chosen the third approach since our home computing system implementing HAVi requires to use Java AWT, and we like to use various home computing applications developed on HAVi in the near future without modifying them. Recent interests in the use of embedded Linux to build home appliances make our approach more practical since Linux usually adopts the X window system as its basic user interface system, and our system enables a variety of applications on embedded Linux to be controlled by various interaction devices. The approach makes the development of a variety of embedded applications very easy since the applications can use traditional and familiar user interface toolkits.

4.2 Universal Interaction

In our approach, we call the protocol that can be universally used for the communication between input/output interaction devices and appliances *universal interaction*. *Universal interaction* enables us to control a variety of home appliances in a uniform way. This means that our behavior is not restricted by where we are or which appliance we like to control. Therefore, our approach provides very natural interaction with home appliances.

The output events produced by appliances are converted to *universal output interaction events*, and the events are translated for respective output interaction devices. Also, input events generated in input interaction devices are converted to *universal input interaction events*, and the events are processed by applications executed in appliances.

A universal interaction proxy that is called the UniInt proxy as described in the next section plays a role to convert between the universal interaction protocol and input/output events of respective interaction devices in a generic way. The proxy allows us to use any input/output interaction devices to control appliances if the events of the devices are converted to the universal interaction protocol. This approach offers the following three very attractive characteristics.

The first characteristic is that input interaction devices and output interaction devices are chosen independently according to a user's situation and preference. For example, users can select their PDAs for their input/output interaction. Also, the users may choose their cellular phones as their input interaction devices, and television displays as their output interaction devices. For example, users may control appliances by their gesture by navigating augmented real world generated by wearable devices.

The second characteristic is that our approach enables us to choose suitable input/output interaction devices according to a user's preference. Also, these

interaction devices are dynamically changed according to the user's current situation. For example, a user who controls an appliance by his/her cellular phone as an input interaction device will change the interaction device to a voice input system because his both hands are busy for other work currently.

The third characteristic is that any applications executed in appliances can use the any user interface systems if the user interface systems speak the universal interaction protocol. In our approach, we currently adopt keyboard/mouse events as universal input events and bitmap images as universal output events. The approach enables us to use traditional graphical user interface toolkits such as Java AWT, GTK+, and Qt for interfacing with any interaction devices. In fact, a lot of standards for consumer electronics like to recently adopt Java AWT for their GUI standards. Thus, our approach will allow us to control various future consumer electronics from various interaction devices without modifying their application programs. The characteristic is very desirable because it is very difficult to change existing GUI standards.

4.3 System Architecture

Our system uses the stateless thin-client system to transfer bitmap images to draw graphical user interface, and to process mouse/keyboard events for inputs. The usual thin-client system consists of a viewer and a server. The server is executed on a machine where an application is running. The application implements graphical user interface by using a traditional user interface system such as the X window system. The bitmap images generated by the user interface system are transmitted to a viewer that are usually executed on another machine. On the other hand, mouse and keyboard events captured by the viewer are forwarded to the server. The protocol between the viewer and the server are specified as a standard protocol for showing a desktop on various client systems[1]. In the paper, we call the protocol the RFB protocol. The system is usually used to move a user's desktop according to the location of a user[15], or shows multiple desktops on the same display, for instance, both MS-Windows and the X Window system.

In our system, we replace the viewer to a UniInt(Universal Interaction) proxy that forwards bitmap images received from a UniInt server to an output device. In our system, a server of any thin-client systems can be used as the UniInt server. Also, UniInt proxy forwards input events received from an input interaction device to the UniInt server.

Our system consists of the following four components as shown in Figure 2. In the following paragraphs, we explain these components in details.

- Home Appliance Application
- UniInt Server
- UniInt Proxy
- Input/Output Devices

Home appliance applications generate graphical user interface for currently available home appliances to control them. For example, if TV is currently available, the application generates user interface for the TV. On the other hand, the

[1] The AT&T VNC system calls the protocol the RFB(Remote Frame Buffer) protocol.

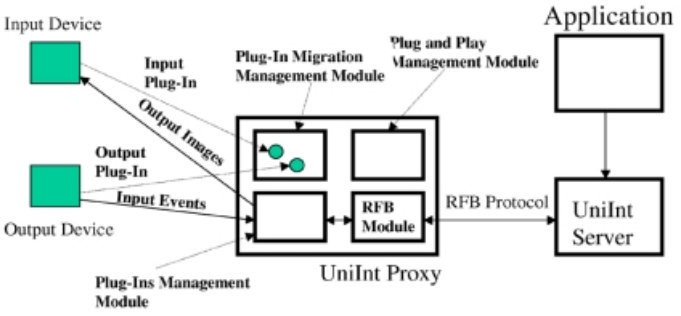

Fig. 2. System Architecture

application generates the composed GUI for TV and VCR if both TV and VCR are currently available.

The UniInt server transmits bitmap images generated by a window system using the RFB protocol to a UniInt proxy. Also, it forwards mouse and keyboard events received from a UniInt proxy to the window system. In our current implementation, we need not to modify existing servers of thin-client systems, and any applications running on window systems supporting a UniInt server can be controlled in our system without modifying them.

The UniInt proxy is the most important component in our system. The UniInt proxy converts bitmap images received from a UniInt server according to the characteristics of output devices. Also, the UniInt proxy converts events received from input devices to mouse or keyboard events that are compliant to the RFB protocol. The UniInt proxy chooses a currently appropriate input and output interaction devices for controlling appliances. Then, the selected input device transmits an input plug-in module, and the selected output device transmits an output plug-in module to the UniInt proxy. The input plug-in module contains a code to translate events received from the input device to mouse or keyboard events. The output plug-in module contains a code to convert bitmap images received from a UniInt server to images that can be displayed on the screen of the target output device.

The last component is input and output interaction devices. An input device supports the interaction with a user. The role of an input device is to deliver commands issued by a user to control home appliances. An output device has a display device to show graphical user interface to control appliances.

In our approach, the UniInt proxy plays a role to deal with the heterogeneity of interaction devices. Also, it can switch interaction devices according to a user's situation or preference. This makes it possible to personalize the interaction between a user and appliances.

4.4 UniInt Proxy

The current version of UniInt proxy is written in Java, and the implementation contains four modules as shown in Figure 2. The first module is the RFB protocol

module that executes the RFB protocol to communicate with a UniInt server. The replacement of the module enables us to use our system with different thin-client systems. The module can use the same module implemented in a viewer of a thin-client system. The second module is the plug-in management module that receives input and output plug-in modules from interaction devices, and dynamically links the modules in the UniInt proxy. The third module is the plug-and-play management module that detects currently available input and output interaction devices. The last module is the plug-in migration management module that manages the migration of input and output plug-in modules between interaction devices and a UniInt proxy.

Management for Available Interaction Devices: The plug and play management module detects the currently available input and output devices near a UniInt proxy. In our system, a unique ID is assigned for each type of input/output devices. The UniInt proxy broadcasts beacon messages periodically. In the current prototype, interaction devices are connected via IEEE802.11b, Ethernet or infrared networks. When each interaction device receives a beacon message, it replies an acknowledgement message. The acknowledgement message contains the unique ID to identify the device type. If the UniInt proxy receives several acknowledgment messages from multiple interaction devices, it chooses one device according to the preference determined by the UniInt proxy. Also, when a newly detected device replies an acknowledgement message, the device may be chosen as a currently used interaction device, if the device is more preferable than the currently used device.

When a UniInt proxy chooses a new device after the detection of the device, it sends an acknowledgement message before using the device. Then, the UniInt proxy sends a terminate message to the device that is used until now. Finally, the UniInt proxy waits for receiving a plug-in module from the new device. The preference to select input and output devices is registered in a UniInt proxy for each user. If the system cannot detect who likes to use an appliance, a default preference is chosen. Also, each plug-in module supports an event to switch the currently used input and output devices. For example, a user can send a command to change a currently used output device to a UniInt proxy. The UniInt proxy switches the current output device to the next one until the user selects his favorite output device.

Migration Management of Plug-In Modules: When receiving an acknowledgement message from the UniInt proxy, the input/output devices send plug-in modules to the UniInt proxy. In our system, we are using the MobileSpaces mobile agent system[14] to transmit plug-in modules between input/output devices and a UniInt proxy. After the plug-in modules are downloaded in the Java virtual machine executed in a UniInt proxy, the plug-in migration module sends a migration complete message to the input/output devices. The MobileSpaces system supports hierarchical agents, and the agents can be communicated when they reside at the same hierarchical level. Therefore, we also implement a UniInt proxy as a mobile agent, but the agent does not be migrated to other hosts.

However, the feature may be used to move a UniInt proxy to a near computer from input and output devices.

Lifecycle Management of Plug-In Modules: The plug-in management module contains an input and an output plug-in module. The input plug-in module receives events from an input device that is currently selected. The event is converted to a mouse or a keyboard event, and the event is transferred by the RFB protocol to a UniInt server. For example, if a user touches a button drawing a right arrow, the event is transmitted to the input plug-in module delivered from a PDA device. The event is translated to a mouse movement event to the right, and the event is finally forwarded to a window system.

Also, after bitmap images are received by the RFB protocol module in a UniInt proxy from a UniInt server, an output plug-in module processes the images before transmitting to an output device. For example, a color image received from a UniInt server is converted to a black and white image. Also, the size of the image is reduced to show on a PDA's screen.

4.5 Context-Awareness in Home Computing Applications

The role of home computing applications is to allow us to control home appliances easily. The interaction between a user and home appliances will become more complex since the functionalities of appliances will be richer and richer. Also, the number of appliances will be increased in the future. Therefore, future home applications should support context aware interaction with a variety of appliances.

There are two types of context awareness that should be taken into account. The first one is to personalize the interaction. The interaction is also customized according to a user's situation. The second type is to deal with currently available multiple appliances as one appliance.

It is usually difficult to personalize the interaction with an appliance since a system does not know who controls the appliance. In our system, we assume that each user has his own control device such as a PDA and a cellular phone. If these devices transmit the identification of a user, the system knows who likes to control the appliance. However, in our system, there is no direct way to deliver such information to a home computing application from interaction devices since we assume that the application adopts traditional user interface systems that do not support the identification of a user. Therefore, in our current implementation, each user has a different UniInt server that executes personalized applications for each user. The application provides customized user interface according to each user's preference. The UniInt proxy chooses an appropriate UniInt server according to the identification of a user acquired from an input device.

Our system needs to know which appliances are currently available according to the current situation. In the current system, we assume that an application knows which appliances can be available. For example, if the application supports three home appliances, the application needs to provide seven graphical user interfaces with the combination of the three appliances. The user interfaces are selected according to the currently available appliances. In our system, we

assume that each home appliance is connected via IEEE 1394 networks. Since IEEE 1394 networks support a mechanism to tell which appliances are currently connected and whose power switches are turned on, it is easy that the application easily knows the currently available appliances, and selects the most suitable user interface.

4.6 Input and Output Interaction Devices

In our system, a variety of input and output devices are available, and the input devices and output devices may be separated or combined. For example, a graphical user interface can be displayed on the screen of a PDA or a large display device on TV. The user interface displayed on TV can be navigated by a PDA device. Therefore, a user can choose a variety of interaction styles according to his preference. Also, these devices can be changed according to the current situation. For example, when the currently used interaction devices are unavailable, another interaction device may be selected to control appliances.

We assume that each device transmits a plug-in module to a UniInt proxy as described in Section 4.4. However, some input devices such as a microphone may not be programmable, and it is difficult to support the communication to a UniInt proxy. In this case, we connect the device to a personal computer, and the computer communicates with a UniInt proxy to deliver a plug-in module. However, it is difficult to know when a program on the personal computer returns an acknowledgement message when a beacon message is received from a UniInt proxy since the computer does not understand whether a microphone is currently available or not. Therefore, in the current prototype, we assume that the device is always connected and available.

5 How Does Our System Work?

In the section, we describe how our system works using our middleware components. When an application recognizes that currently available appliances are a television and a video recorder, it shows a graphical user interface for controlling them. Since the current user does not interested in the TV program reservation, the application draws a user interface containing power control, TV channel selection, and VCR function to its UniInt server. As described in Section 4.5, a UniInt proxy selects a UniInt server executing an application drawing a customized user interface for the user who likes to control these appliances, and the selected UniInt server transmits bitmap images containing the interface to the UniInt proxy.

Let us assume that the UniInt proxy detects that the user has a PDA device. The PDA device delivers input and output plug-in modules to the UniInt proxy. The UniInt proxy transcodes bitmap images transmitted from the UniInt server using the output plug-in module before transmitting the images to the PDA device. In this case, 8 bits color images whose image size is 640x480 are reduced to monochrome images whose size is 180x120. Also, input events on the touch screen of the PDA device are converted to mouse and keyboard events

by the input plug-in module. However, we assume that the user likes to see the graphical user interface on a bigger display now. The user transmits a command to the UniInt proxy by tapping on the screen. When the UniInt proxy detects it, the bitmap images containing the graphical user interface are forwarded to the display system, and the images are converted by the output plug-in module provided by the bigger display before transmitting them. Also, the user interface will be displayed again on the screen of the PDA device by tapping the PDA's screen.

6 Current Status

Currently, we have four PCs and one DV camera connected by an IEEE 1394 network. The first PC executes MS-Window98[2], and has an MPEG2 encoder that is connected to a analog TV tuner via a NTSC cable. The PC also executes a program that controls a remote controller to access the TV tuner. The program translates a command from the PC to an infrared command that enables us to control the commercial analog TV tuner. The DCM for our digital TV emulator transmits commands to the program, then the commands are finally forwarded to the analog TV.

The second PC executes Linux, and has an MPEG2 decoder. The PC executes an MPEG2 receiver process. The MPEG2 decoder has a NTSC interface which is connected to an analog TV display. The continuous media management component on the PC has the DCM for a MPEG display.

The third PC contains the DCM for a DV camera. The DCM contains a code to control a DV camera using the AV/C command. A program to detect a DV camera running on the PC transmits the DCM to a HAVi device when a DV camera is connected to the IEEE 1394 network. The PC also emulates a DV monitor that receives DV frames from an IEEE 1394 broadcast synchronous channel, and the continuous media management component for the DV monitor also has the DCM for the DV monitor. The fourth PC emulates a HAVi device. When the PC receives DCMs from the DV monitor or the digital TV emulation program, an application running on the PC shows graphical user interface. By navigating the interface, the methods provided by the DCMs are invoked by the application, and commands to control continuous media management components emulating home appliances will be delivered.

We found that the approach that building middleware components on Linux and Java is very attractive to develop embedded software because the software can be executed on embedded devices without modifying it. This decreases the development cost of embedded devices dramatically. Also, even if we cannot adopt Linux on target embedded devices, the porting of the software is not difficult. For example, we have ported our implementation of HAVi on an embedded device that has a Pico Java II chip. In the experiment, we are able to port HAVi without modifying it, therefore the porting of the software was dramatically easy.

[2] Currently, we are using MS-Window98 to emulate a digital TV tuner because we could not find a MPEG2 encoder that can be used on Linux.

Our user interface system has adopted the AT&T VNC system as a thin-client system. Our application shows a graphical user interface according to currently available appliances as described in the previous section. Also, the cursor on a screen that displays a graphical user interface can be moved from a PalmPilot as shown in Figure 3. However, if the device is turned off, the cursor is controlled by the keyboard and the mouse of a personal computer. It is also possible to show a graphical user interface on the PDA device according to a user's preference. Also, the current system has integrated cellular phones to control home appliances. NTT Docomo's i-mode phones have Web browsers, and this makes it possible to move a cursor by clicking special links displayed on the cellular phones.

In our home computing system, we have adopted Linux/RT[9] that extends a standard Linux by adding a resource reservation capability. The Linux also provides an IEEE 1394 device driver and an MPEG2 decoder. Also, IBM JDK1.1.8 for the Java virtual machine is used to execute the HAVi middleware component.

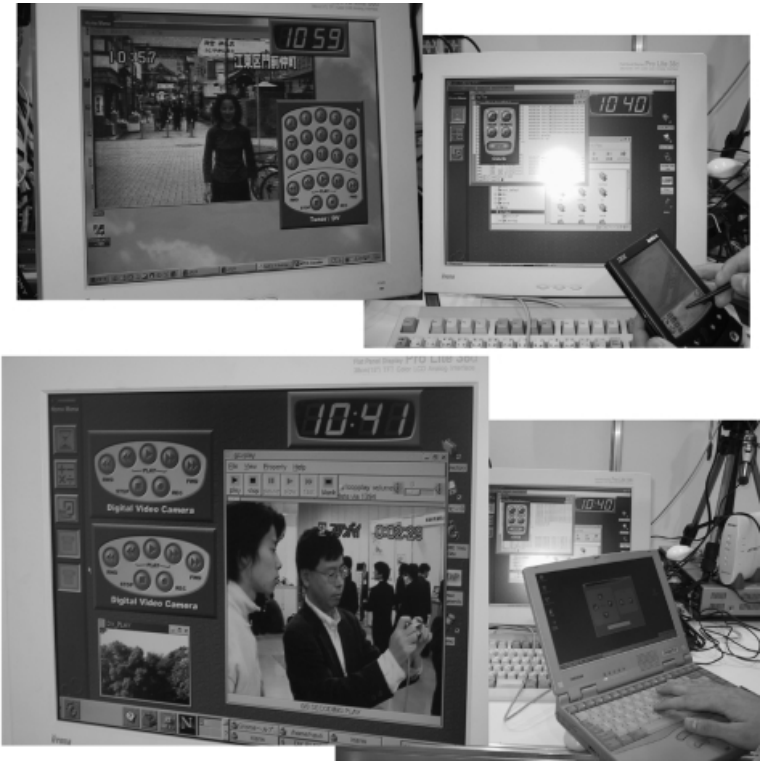

Fig. 3. Controlling HAVi with Various Interaction Devices

7 Experiences

This section describes several experiences with building our prototype systems.

Home Appliance Applications: One of the most important issues to build future home appliances is their usability. Especially, the usage of appliances should be customized according to each user's preferences and situations. For example, current home appliances such as TV and VCR provide rich functionalities such as TV program reservation. Since the remote controller has a lot of buttons, usual users confuse how to use these appliances. On the other hand, our system easily changes user interface by replacing applications that invoke HAVi API. This means that it is possible to change user interface according to the preference of a user. Also, there is a possibility to compose multiple functions in several existing appliances, and it enables us to define a new appliance in an ad-hoc way. Therefore, our system is very suitable to do a variety of experiments for developing future advanced home appliances.

However, in HAVi, Registry returns a list of software elements, when an application retrieves an necessary function such as a tuner function and a camera function. The current HAVi does not provide enough information to select the functions. Therefore, a variety of attributes should be stored in Registry to select the most suitable function to develop advanced context-aware home applications.

Also, the customization of user interface requires to represent a variety of aspects of our daily life such as location information of users and various objects or a user's emotion. To make an application to be adapted according to context information portable, the representation of such information should be standardized. However, it is not easy to represent the context information and the preference of a user. To represent such information, we need to model the entire world. This makes the specification of middleware to become too big to use for home appliances, thus we need more efforts in this area.

Distribution Transparency: The issue discussed in this section is how to manage distribution. Home appliances connected by networks store a variety of information that should be consistent whenever some failures occur or the configuration of networks is changed. Also, to build stable distributed systems, it is important to define rigorous semantics for all operations in systems.

The following two issues are completely ignored by the current researches on home computing. The first issue is very important because home computing environments assume that the configuration changed at any moment. There is a lot of work to maintain consistency in distributed systems such as transactions[5] and process groups[2], and these concepts are useful to build distributed home appliances. For example, a digital TV appliance may invoke a transaction for on-line shopping. The database to monitor the behavior of users should keep the consistency even if an application is turned off while modifying the database. We believe that transactions are also a useful concept in home computing environments, and it is important to develop an appropriate transaction model for home computing environments. Also, fault tolerance is an important topic in

home computing environments since it is not desirable to fail to record a broadcast program due to the crash of an appliance. In this case, an application should be implemented by using a process group concept to store critical information.

The second issue is important when multiple functions in different appliances are composed. These functions provide standard interface to access them. For example, each tuner function provides the same interface even if the vendors are different. However, since the semantics of the interface is not define rigorously, it is not ensured to behave in the same way. Because the distribution of these functions is hidden from an application in a transparent way, it is not easy to build stable applications when a failure occurs[6,19]. We should take into account the difficulties of distributed systems when designing distributed middleware components.

High Level Abstraction: We have implemented several home computing applications on our implementation of HAVi. One of these applications is a remote control application that is integrated with a Web server. It enables the application to convert commands from the embedded Web server to commands to access HAVi API. This means that it is easy to build an application to control home appliances from a variety of control devices embedding Web browsers. We believe that the key point to realize fantastic home computing environments is to provide high level API that enables us to build a variety of home applications.

The high level API provided by HAVi allows us to build advanced home applications that are personalized for each person. For example, it is easy to build a program customizing graphical user interface according to the preference of a user. The program identifies a user, and changes the layout of of graphical user interface. However, it is not easy to identify the current user who likes to use an appliance. Also, it is not easy to represent context information to customize the behavior of applications. For example, it is not simple to generate graphical user interface according to the currently available appliances.

The high level API enables us to compose several appliances, but it is not realistic to create an application for respective compositions. Therefore, it is important to build an application to compose appliances from a composition specification that is a high level language specific for describing the composition of appliances. The specification should be declarative, and it should be possible to compose several specifications to reuse existing specifications. Our experience shows that the current high level API does not have enough power to represent the composition. Especially, the naming of functions in appliances and a variety of services is a key issue to develop a useful composition specification.

COTS Platforms: We have implemented HAVi using Java, and continuous media management components on Linux. Our platform increases the portability of our middleware by adopting COTS components. A variety of open software make us to develop prototype applications very quickly. For example, free DV decoder software enables us to create our DV monitor in a very short time, and a free Web server enables us to create a remote control application easily. We believe that the adoption of open platform allows us to build a variety of prototype

applications very cheaply. Also, Linux allows us to use the same operating system both for development machines and target devices. Especially, recent Linux kernel provides a variety of functions that are suitable for embedded systems such as ROM file systems and flash file systems. Therefore, the development cost of home appliances will be dramatically decreased by the adoption of Linux.

Object-orientation, automatic memory management, sophisticated synchronization support and rich class libraries enable us to develop complex middleware such as HAVi very quickly. Especially, the development of traditional embedded applications is very difficult by synchronization misuse and memory leak. However, Java's synchronization mechanism is well structured, therefore concurrent programs become robust quickly. Also, Java's automatic memory management prevents us from memory leaks that are especially serious in multi-threaded programming. The fact is very important for developing future home appliances since the development time should become shorter although the complexity will be increased exponentially.

Limitation of Our User Interface System: In our system, a bitmap image that contains a graphical user interface is transferred from a UniInt server to a UniInt proxy. Since the image does not contain semantic information about its content, the UniInt proxy does not understand the content. For example, it is difficult to extract the layout of each GUI component from the image. Therefore, it is not easy to change the layout according to the characteristics of output devices or a user's preference. Also, our system can deal with only mouse and keyboard events. Thus, the navigation of a graphical user interface can be done by emulating the movement of a cursor or pressing a keyboard and mouse buttons. If the limitation makes the usability of a system bad, other approaches should be chosen. However, navigating a graphical user interface from a PDA and a cellular phone provides very flexible interaction with home appliances. Our experiences show that home appliances usually allow us to use a large display and show graphical user interfaces on the display to control the appliances. Thus, we believe that our system has enough power to make future middleware components for home appliances flexible.

Better Control for Home Appliances: Our system can control any applications executed on standard window systems such as the X window system. In our home computing system, traditional applications coexist with home computing applications, and these applications can also be controlled in an integrated way by our system. For example, we can navigate an MP3 player or a Netscape browser running with home computing applications via our system. However, the overlapping window layout is painful to be navigated by our user interface system. We consider that the tiled window strategy is more suitable for controlling home appliances. Also, our experience shows that we can control both home appliances and traditional applications such as presentation software and web browsers by using our system in a comfortable way if our system supports the movement of a mouse cursor at a variety of speeds.

Porting Issues in Embedded Systems: In embedded systems, underlying platforms provide a variety of advanced features. The features are different on respective platforms, and it is impossible to standardize these features since the standardization may hide some details of respective platforms. Also, it is important to support Quality of Service(QOS) parameters to accommodate a variety of resource constraints. However, a general QOS specification is not suitable for embedded systems since the specification will be too complex and big. Therefore, it is desirable to provide a specialized QOS specification for each platform. The QOS specification should take into account the predictability of underlying software such as scheduling behavior and worst case execution time.

However, the approach makes the portability of software bad since it is necessary to rewrite programs when they are ported on other platforms. We need to develop a new methodology to develop a portable program that can be adapted to respective platforms. We have a plan to enhance our middleware components with our new methodology[12]. The methodology is based on famous separation of concerns principle. In the methodology, we are working on transparently incorporating advanced features provided as non standardized API and a domain specific QOS specification on existing programs. Also, the methodology includes a method to predict the behavior of programs, which is important to predict the behavior of a program when it is enhanced to access advanced underlying features.

8 Conclusion

This paper has described system software for audio and visual networked home appliances. Our system consists of two components. The first component is distributed middleware for audio and visual networked home appliances. The component provides high level abstraction to control home appliances. The second component is a user interface system to allow us to use a variety of interaction devices to control graphical user interface provided by home applications.

Our system increases the portability of system software for home appliances dramatically since we have adopted commodity software such as Java and Linux. Also, our user interface system allows home computing applications to adopt standard user interface middleware, but it enables us to use a variety of interaction devices instead of a mouse and a keyboard. In the future, we like to prove that our approach to adopt commodity software as much as possible is useful to build other complex future embedded systems such as personal appliances like cellular phones and PDAs.

Ubiquitous computing environments that are located in any spaces will be integrated by connecting them via the Internet in the near future[13]. The environments are extremely heterogeneous, but requires applications to be extremely portable. Also, in the environments, we need to connect different distributed middleware components such as HAVi, Jini and UPnP. We consider that network systems that are customized according to the characteristics of respective applications are very important in the near future[1].

References

1. H.Aizu, I.Satoh, D.Ueno, T.Nakajima, "A Virtual Overlay Network for Integrating Home Appliances", In the Proceedings of the 2nd International Symposium on Applications and the Internet, 2002.
2. K.Birman, "The Process Group Approach to Reliable Distributed Computing", Communications of the ACM, Vol.36, No.12, 1993.
3. Citrix Systems, "Citrix Metaframe 1.8 Background", Citrix White Paper, 1998.
4. T. Hodes, and R. H. Katz, "A Document-based Framework for Internet Application Control", In Proceedings of the Second USENIX Symposium on Internet Technologies and Systems, 1999.
5. J. Gray and A.Reuter, "Transaction Processing: Concepts and Techniques", Morgan Kaufmann, 1993.
6. R. Guerraoui, "What Object Oriented Distributed Programming Does not Have to be, and What it may be", Informatik, 2 1999.
7. HAVi Consortium, "HAVi Specification: Specification of the Home Audio/Video Interoperability(HAVi) Architecture, http://www.havi.org/
8. R.Lea, S.Gibbs, A.Dara-Abrams, E. Eytchson, "Networking Home Entertainment Devices with HAVi", IEEE Computer, Vol.33, No.9, 2000.
9. Timesys, "Linux/RT", http://www.timesys.com.
10. Microsoft Corporation, "Microsoft WIndows NT Server 4.0: Technical Server Edition, An Architecture Overview", Technical White Paper 1998.
11. T. Nakajima, et. al., "A Framework for Building Audio and Visual Home Appliances on Commodity Software, In the Proceedings of IASTED International Conference on Internet and Multimedia Systems and Applications(IMSA2001), 2001.
12. T. Nakajima, "Towards Universal Software Substrate for Distributed Embedded Systems", In Proceedings of the International Workshop on Object-Oriented Real-Time Dependable Systems, 2000.
13. T. Nakajima, et. al., "Technical Challenges for Building Internet-Scale Ubiquitous Computing", Technical Report, Waseda University, 2001.
14. I. Sato, "MobileSpaces: A Framework for Building Adaptive Distributed Applications using a Hierarchical Mobile Agent System", In Proceedings of IEEE International Conference on Distributed Computing Systems, 2000.
15. Andy Harter, Andy Hopper, Pete Steggles, Andy Ward, Paul Webster, "The Anatomy of a Context-Aware Application", In Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, 1999.
16. Sun Microsystems, "Sun Ray 1 Enterprise Appliance", http://www.sun.com/products/sunray1/.
17. User Interface Markup Language, http://www.uiml.org/
18. T.Richardson, et al., "Virtual Network Computing", IEEE Internet Computing, Vol.2, No.1, 1998.
19. J.Waldo, G.Wyant, A.Wollrath and S.Kendall, "A Note on Distributed Computing", Technical Report, Sun Microsystems Labolatories, Inc., 1994.
20. Mark Weiser, "The Computer for the 21st Century", Scientific American, Vol. 265, No.3, 1991.