Making Existing Interactive Applications Context-Aware

Tatsuo Nakajima, Atsushi Hasegawa, Tomoyoshi Akutagawa, Akihiro Ibe, and Kouji Yamamoto

Department of and Computer Science, Waseda University
3-4-1 Okubo Shinjuku Tokyo 169-8555 JAPAN
tatsuo@dcl.info.waseda.ac.jp

Abstract. We propose a new approach to build context-aware interactive applications. Our approach enables us to use existing GUI-based interactive applications although a variety of interaction devices can be used to control them. This means that we can adopt traditional GUI toolkits to build context-aware applications. The paper describes design and implementation of our middleware that enables existing applications to be context-aware, and presents some examples to show the effectiveness of our approach.

1 Introduction

Context-awareness[4,11] will reduce complexities to access our environments since the environments recognize our situation to fill many parameters that are chosen by us in traditional approaches. The parameters are retrieved by various sensors embedded in our environments. However, implementing context-aware applications is hard for usual programmers. Also, we already have many existing GUI-based interactive applications. In the future, these applications will like to be modified to support context-awareness, but the modification is not easy.

In this paper, we propose a new approach to build context-aware applications. Our middleware enables the GUI-based interactive applications to be context-aware without modifying them. Also, programmers can adopt traditional GUI toolkits such as GTK+ and Java Swing to build context-aware applications. Our current work focuses on home computing environments where home appliances such as TV and VCR are connected by a home network. Most of current home appliances do not take into account context-awareness. Also, standard middleware for home computing has adopted traditional GUI toolkits such as Java AWT. Therefore, our middleware is attractive to build context-aware home computing applications in an easy way.

2 Ubiquitous Computing Applications

In ubiquitous computing environments, one of the most important problems is how to interact with a variety of objects embedding computers. The interaction

H. Kosch, L. Böszörményi, H. Hellwagner (Eds.): Euro-Par 2003, LNCS 2790, pp. 1082–1090, 2003.

between us and computers embedded in various objects has been studied. However, it is necessary for us to consider how to manage these interaction devices. Let us consider that a user uses a mobile device to control home appliances or to retrieve interesting information. In this case, applications running on the device need to take into account the mobility of devices. Also, we like to use various display devices to show control panels of home appliances. For example, when a user sits on a sofa, he will use his PDA to control the appliances. The control panel is displayed on the PDA, and he navigates the panel to control appliances. However, if he can find a large display near him, he likes to use it to control the appliances. The control panel should be displayed on the large display, and the panel can be navigated from his PDA. Moreover, if the display is a touch panel, he will navigate the panel by touching it. We believe that it is important to use a variety of interaction devices according to his situation, and one interaction device is not enough to cover various situations. In this case, a user interface middleware to offer context-awareness is necessary to change various interaction devices according to the situations.

There are many researches to develop context-aware applications[6,11]. Also, there is a generic framework to develop context-aware applications[4]. However, we believe that there are two important issues that should be considered to build context-aware applications. The first issue is how to retrieve context information. We need various sensors and common interface to provide higher level abstraction to abstract various sensor information. For example, context toolkit[4] and sentient information framework[5] provide high level abstraction to hide details of context information retrieved from the sensors.

The second issue is a mechanism to adapt software structure according to retrieved context information[9]. Traditionally, most of context-aware applications change their software structures in an ad-hoc way, and it is not easy to take into account new context information. We believe that it is important to adopt a framework to build adaptive applications in a systematic way.

Although there are many researches to attack the problems described above, it is not easy to develop context-aware applications. Because implementing adaptive applications is still ad-hoc, and programmers need to learn advanced programming techniques such as design patterns and aspect-oriented programming to build context-aware applications that can take into account new situations. Also, we need to reimplement existing applications if we like to modify the applications to support context-awareness. For example, if a context-aware application needs to display video streams according to the location of a user. The application should implement the redirection of the video streams to transmit them to computers that is desirable to display them[1].

We believe that we need a simpler approach to build context-aware applications. Especially, it is important to build context-aware applications from existing applications to build ubiquitous computing environments in a practical way. Our middleware based on a thin-client architecture enables us to build context-aware applications in a simple way. Existing applications are completely separated from our middleware components. Our middleware enables us to inte-

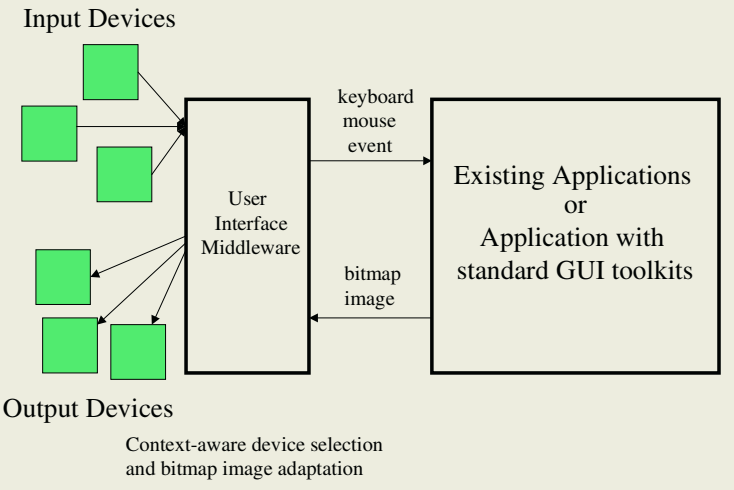

Fig. 1. Basic Architecture

grate new interaction devices and context-awareness without modifying existing applications.

3 Design and Implementation

3.1 Basic Architecture

Fig. 1 shows an overview of our middleware infrastructure. In the architecture, an application generates bitmap images containing information such as control panels, photo images and video images. The approach is simple because popular operating systems provide a mechanism to retrieve bitmap images generated by applications. Also, these applications can receive keyboard and mouse events to be controlled. The user interface middleware receives bitmap images from applications and transmits keyboard and mouse events. The role of the middleware is to select appropriate interaction devices by using context information. Also, input/output events are converted to keyboard/mouse events according to the characteristics of interaction devices.

Our system uses the 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 such as Citrix Metaframe[3], Microsoft Terminal Server[7], Sun Microsystems Sun Ray[13], and AT&T VNC(Virtual Network Computing) system[10] 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 is usually executed on another machine. On the

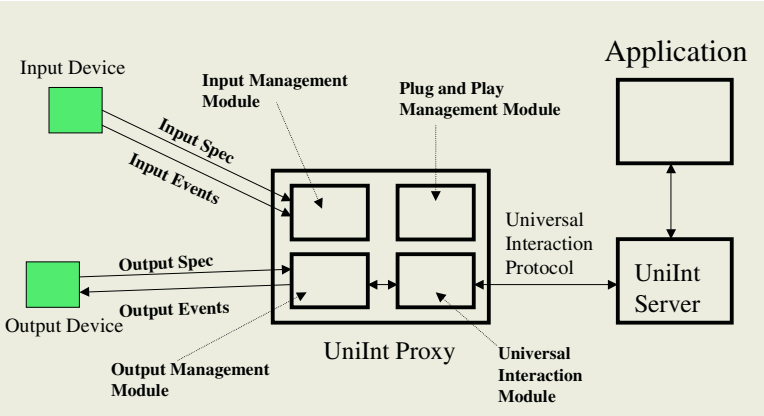

Fig. 2. System Architecture

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 universal interaction protocol. The system is usually used to move a user's desktop according to the location of a user[6], 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 of a thin-client system to the UniInt(Universal Interaction) proxy that forwards bitmap images received from a UniInt server to an output device. Also, UniInt proxy forwards input events received from an input interaction device to the UniInt server. In our approach, a server of any thin-client systems can be used as the UniInt server.

Our system consists of the following four components as shown in Fig. 2.

- Home Computing Application
- UniInt Server
- UniInt Proxy
- Input/Output Interaction Devices

Home computing applications[12] 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 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 universal interaction 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, it converts events received from input devices to mouse or keyboard events that are compliant to the universal interaction protocol. The UniInt proxy chooses a currently appropriate input and output interaction devices for controlling appliances. To convert interaction events according to the characteristics of interaction devices, the selected input device transmits an input specification, and the selected output device transmits an output specification to the UniInt proxy. These specifications contain information that allow a UniInt proxy to convert input and output events.

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.

3.2 Implementation of UniInt Proxy

The current version of UniInt proxy is written in Java, and the implementation contains four modules as shown in Fig. 2. The first module is the universal interaction protocol module that executes the universal interaction 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 and play management module. The module collects information about currently available interaction devices, and builds a database containing information about respective interaction devices. The third module is the input management module. The module selects a suitable input interaction device by using the database contained in the plug and play management module. The last module is an output management module. The module also selects a suitable output interaction device. Also, the module converts bitmap images received from the universal interaction module according to the output specification of the currently selected output interaction device.

Management of Available Interaction Devices: The plug and play management module detects currently available input and output devices according to context information. The module implements the Universal Plug and Play Protocol to detect currently available interaction devices. In our system, we assume that all interaction devices can be connected to the Internet. An interaction device transmits advertisement messages using the simple service discovery protocol(SSDP). When a UniInt proxy detects the messages, it knows the IP address of the interaction device. Then, the UniInt proxy transmits an HTTP GET request to the interaction device. We assume that each interaction device contains a small Web server, and returns an XML document.

The XML document contains information about the interaction devices. If the interaction device is an input device, the file contains various attributes about the device, which are used for the selection of the most suitable device. For an output device, the file contains information about the display size and the attributes for the device. The plug and play management module maintains a database containing all information about currently detected interaction devices.

Adaptation of Input and Output Events: The role of the input management module and the output management module is to determine the policies for selecting interaction devices. As described in the previous section, all information about currently available interaction devices are stored in a database of the plug and play management module. The database provides a query interface to retrieve information about interaction devices. Each entry in the database contains a pair of an IP address and a list of attributes for each interaction device, then the entry whose attributes are matched to a user's preference provided in a query is returned.

The current implementation of the input management module receives all input events from any currently available input devices. When a new input device is detected, a new thread for receiving input events from the newly detected device is created. All input events are delivered to the universal interaction module, and they are processed by applications eventually.

The output management module converts bitmap images received from the universal interaction module according to the display size of an output device. The size is stored in the database of the plug and play management module. When an output device is selected, the display size is retrieved from the database. The bitmap image is converted according to the retrieved information, then it is transmitted to the selected output device.

3.3 Current Status

Our system have adopted the AT&T VNC system[10] as a thin-client system, and the VNC server can be used as the UniInt server without modifying it. The current prototype in our HAVi-based home computing system[12], where HAVi is a standard specification for digital audio and video, emulates two home appliances. The first one is a DV viewer and the second one is a digital TV emulator. 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 Compaq iPAQ. However, if the device is turned off, the cursor is controlled by other devices such as a game console. 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, Linux 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 contains several photos to demonstrate our system. Currently, our home computing applications are executed on HAVi, and a control panel is written by using Java AWT. In the demonstration, if both a DV camera and a digital TV tuner are simultaneously available, the control panels for them are combined as one control panel as shown in the photo(Top-Left). The control panel can be navigated by both a cellular phone(Top-Right) and a game

console(Bottom-Left). Also, the control panel can be displayed and navigated on a PDA(Bottom-Right).

Our middleware proposed in this paper enables us to use various interaction devices and to interact with the appliances in a context-aware way. By integrating home appliances with our middleware, a user is allowed to choose the most suitable interaction device according to his situation.

Fig. 3. Controlling Our HAVi based Home Computing System

4 Scenarios of Our Approach

In this section, we present two scenarios to show the effectiveness of our approach. The first scenario enables us to interact with existing applications running on MS-Windows or Linux in a location-aware fashion. The system has been implemented, and shows the effectiveness of our approach. The second scenario is a location-aware video phone.

4.1 Location-Aware Interaction

The system enables a user to control a home appliance in a location-aware way. In the future, our home will have many display devices to show control panels

for controlling home appliances. A user usually likes to use the nearest display device to him to control a variety of home appliances. For example, if a user sits on a sofa, the control panel of a home application is displayed on his PDA. On the other hand, he is in front of a display device. The control panel is shown on the display device. In this case, if the display is a touch panel, he can access the control panel by touching the display. However, if the display is a normal display, he can navigate the control panel from his PDA or a game console.

4.2 Ubiquitous Video Phones

The second example is a ubiquitous video phone that enables us to use a video phone in various ways. In this example, a user speaks with his friend by using a telephone like a broadband phone developed by AT&T Labolatories, Cambridge[2]. The phone has a receiver like traditional phones, but it also has a small display. When the phone is used as a video phone, the small display renders video streams transmitted from other phones. The display is also able to show various information such as photos, pictures, and HTML files that are shared by speakers. Our user interface system makes the phone more attractive, and we believe that the extension is a useful application in ubiquitous computing environments.

When a user needs to start to make a dinner, he will go to his kitchen, but he likes to keep to talk with his friend. However, a phone is put in a living room, and the traditional phone receiver is not appropriate to continue the conversation with his friend in the kitchen because his both hands may be used for cooking. In this case, we use a microphone and a speaker in the kitchen so that he can use both hands for making the dinner while talking with his friend. In the future, various home appliances such as a refrigerator and a microwave provide displays. Also, a kitchen table may have a display to show a recipe. These displays can be used by the video phone to show a video stream. In a similar way, a video phone can use various interaction devices for interacting with a user. The approach enables us to use a telephone in a more seamless way.

Our system allows us to use a standard VoIP application running on Linux. The application provides a graphical user interface on the X window system. However, our system allows a user to be able to choose various interaction styles according to his situation. Also, if his situation is also changed, the current interaction style is changed according to his preference. Currently, we are incorporating an audio stream in our system, and it is easy to realize the example described in the section by adopting our approach.

5 Conclusion and Future Work

In the paper, we have proposed a new approach to build ubiquitous computing applications. Our middleware enables us to convert existing applications to context-aware applications without modifying existing applications. Therefore, a programmer needs not to learn new APIs to write context-aware applications.

Currently, we are extending our middleware to support video streams and audio streams. Also, we are implementing a location-aware video phone described in this paper. Our system can transmit video streams to any output devices if the bandwidth is enough to process the video streams. We like to incorporate a dynamic QOS control[8] scheme in our system. This means that the quality of video streams is changed according to the processing power of a machine to execute UniInt proxy programs.

To support audio streams, we are working on an audio snoop program in a Linux kernel. The program runs in the kernel, and captures events about audio streams. The program communicates to UniInt proxy, and the UniInt proxy transmits audio streams to appropriate speakers, and receive audio streams from appropriate microphones. The audio streams can be converted according to the capabilities of microphones and speakers similar to other interaction devices.

References

1. J. Bacon, J Bates, and D. Halls, "Location-Oriented Multimedia", IEEE Personal Communications, Vol.4, No.5, 1997.
2. AT&T Laboratories, Cambridge, "Broadband Phone Project", http://www.uk.research.att.com/bphone/apps.html.
3. Citrix Systems, "Citrix Metaframe 1.8 Background", Citrix White Paper, 1998.
4. A.K. Dey, D. Salber, and G.D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", Human-Computer Interactions, Vol.16, No.2–4, 2001.
5. Diego Lopez de Ipina, "Building Components for a Distributed Sentient Framework with Python and CORBA", In Proceedings of the 8th International Python Conference, 2000.
6. 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.
7. Microsoft Corporation, "Microsoft WIndows NT Server 4.0: Technical Server Edition, An Architecture Overview", Technical White Paper 1998.
8. T. Nakajima "A Dynamic QOS Control based on Optimistic Processor Reservation", In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, 1996.
9. T. Nakajima "Adaptive Continuous Media Applications in Mobile Computing Environment", In Proceedings of the IEEE International Conference on Multimedia Computing and Systems, 1997.
10. T.Richardson, et al., "Virtual Network Computing", IEEE Internet Computing, Vol.2, No.1, 1998.
11. B.N. Schilit, N.Adames, and R. Want, "Context-Aware Computing Applications", In Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications, 1994.
12. K.Soejima, M.Matsuda, T.Iino, T.Hayashi, and T.Nakajima, "Building Audio and Visual Home Applications on Commodity Software", IEEE Transactions on Consumer Electronics, Vol.47, No.3, 2001.
13. Sun Microsystems, "Sun Ray 1 Enterprise Appliance", http://www.sun.com/products/sunray1/.