An Input Widget Framework for Multi-modal and Multi-device Environments

Nobuyuki Kobayashi, Eiji Tokunaga, Hiroaki Kimura,
Yasufumi Hirakawa, Masaaki Ayabe, Tatsuo Nakajima

Department of Computer Science.
Waseda University
{koba-n, eitoku, hiroaki, yasufumi, mazex, tatsuo}@dcl.info.waseda.ac.jp

Abstract

In future ubiquitous computing environments, our daily lives will be influenced by a lot of computer-supported services all over the place. To interact with those services intuitively, heterogeneous interaction techniques such as gesture recognition, auditory recognition and tangible user interfaces will appear. Besides, several kinds of services will support multiple input devices, not just one set of them. In such multi-modal environments, application programmers must take into account how to adapt heterogeneous input events to multi-modal services.

We propose an input widget framework that provides high-level abstraction for heterogeneous input devices, that we call meta-inputs, for distributed multi-modal applications. Our framework provides generic and standard interfaces between input devices and services. It enables developers to deploy input devices and services independently. Also, our framework supports context-aware runtime adaptation to switch input devices to handle services dynamically.

1 Introduction

Our daily lives will be dramatically changed by embedded devices which are highly networked in our environments. These devices will provide a lot of human-oriented services all over the place. These environments are called *pervasive computing* or *ubiquitous computing* [16]. In the vision of these environments, physical spaces and virtual resources are highly integrated with a variety of devices and sensors. Therefore, interaction techniques in these environments become more seamless and intuitive. In these environments, one of the most important issues is how to interact with a variety of computer embedded devices. Recently, a lot of researchers have developed intuitive interaction methods with heterogeneous input devices such as sensor embedded physical devices, recognition-based technologies, or their combination.

These interaction techniques are useful and intuitive, but it is difficult to adapt them to existing services because their input events are not standardized and generalized. In this paper, we describe an application framework that provides the standard and semantic interfaces that facilitate us to combine new input widgets and services. Existing middleware infrastructures such as BEACH [14] and Gaia [13] distinguish user interfaces and application logics in ubiquitous environments and they increase the reusability of software components, but they do not take into account defining standard and generic interfaces between them. The main goal behind our framework is to enable programmers to develop input widgets and service components independently with little or no knowledge of one another.

This paper is organized as follows. In the next section, we present related work. We describe design issues of input widgets in Section 3. In Section 4, we propose the architecture of an input widget framework, and the implementation is presented in Section 5. Section 6 shows the evaluation of our work, and Section 7 presents how to build application using our framework. In Section 8, we discuss about the experience of building our framework and its applications. Finally, we conclude the paper in Section 9.

2 Related Work

To develop our application framework, it is important to classify input devices and methods. Taxonomies of input devices have been proposed earlier by Buxton [15] and Card et al. [4]. They have classified input devices by the combinations of linear and rotary, position and force, absolute and relative and so on. But we have considered these schemes

Proceedings of the Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)
0-7695-2357-9/05 $20.00 © 2005 IEEE

are not enough for input devices in ubiquitous computing environments. However they focused attention on the low-level functionality of input devices, they didn't focus on the high-level abstraction of them. Also, they have not describe about recognition-based interactions as auditory recognition and gesture recognition.

iStuff [2] is a part of the work on the interactive workspace project in Stanford University. They categorized physical devices by the characteristics of dimensions, relative or absolute, resolution and so on. They designed the iStuff toolkit based on iROS Event Heap [1], that is the blackboard architecture in interactive workspaces, and the PatchPanel [3] that re-map events to applications dynamically. But their architecture have not ensured which components and services can connect each others. In addition, because they have not provided the standard interfaces between input devices and services, the application programmers must be aware of the event types between input devices and services.

Myers's Amulet [10] demonstrated that it is flexible and useful for programmers to separate input devices from application level code. His model encapsulated interactive behaviors into a few *Interactor* object types [9]. However he has focused on mouse, keyboard and window systems, he has not focused on heterogeneous physical or recognition-based devices. His toolkit has not addressed distributed environments.

The Gaia [13] project at the University of Illinois is developing a middleware for ubiquitous computing environments that is called "active spaces". They have extended the Model-View-Controller model to the *Model-Presentation-Adapter-Controller-Coordinator*(MPACC) [12] and they have separated inputs and outputs from application logics in ubiquitous computing environments. But their framework has not provided the abstraction of data flow between controllers and models. Because it has not offered semantic and standard interfaces between these components, it is difficult for programmers to develop applications without knowing details of each component.

ICON [5] is an editor designed to configure a set of input devices and connect them to actions into a graphical interactive application. Olwal and Feiner [11] describe an input processing as a dataflow diagram. Their research shows that modulating input mechanisms increases the reusability and flexibility of input modalities. But they do not discuss about the data type between input modules and the connectivity of them enough.

3 Design Space of Input Widgets

There are a variety of interaction techniques as multiple modalities and multiple devices in ubiquitous computing environments. Although these interaction techniques seems to be intuitive and effective, but a new type of complexities is added in ubiquitous computing:

- When using new interaction techniques to services, it is difficult to adapt them without adding or modifying programs on the services side.
- It is difficult that existing interaction techniques are bound to new services without adding or modifying programs on the controller's side.

To solve these issues, we must define the effective and expressive interfaces between input techniques and services. In order to define them, we have classified input techniques into several categories based on design spaces. In this section, we have examined a various aspects of input techniques.

3.1 Input Widgets

In this paper, to clarify the domain of input modalities, we defined *input widgets* as input methods and devices that provide explicit interaction to services. For examples, mouse, keyboards, speech recognition and sound recognition, gesture or posture captured by cameras are all considered as input widgets. Input widgets contain several input expressions. For examples, a standard mouse has two buttons and two-axis motion sensor and they have each role to services. We defined these available operations of input widgets as *input methods*. It is said that input widgets consist of a variety of input methods.

3.2 Input Capabilities

To classify the characteristics of input widgets, we have focused on input capabilities, that is the effectiveness and expressiveness of them. In this paper, we have classified them into five categories: modality, expression, roles, bounded or Infinite, and relative or absolute. We describe the details of these attributes in this section.

3.2.1 Modality

This attribute shows a mode of interaction styles. When we access ubiquitous services, we must represent commands by actions to occur input events. When we represent such an action, we need to express them by five senses such as touch, taste, hearing, eyesight, and smell. But taste and smell are not used as interaction techniques generally. Therefore we have considered that modalities are based on a basic mode and three interaction modes: tangible, auditory and visual.

Basic *Basic* is standard and traditional interaction forms using a set of standard mouse and keyboard. This modality needs fixed and stable space such as a table and chair. This modality is aimed at a single user with a single display, but it is not aimed at collaboration work with multiple-device environments.

Tangible There have been a lot of user interface researches trying to integrate real world and information systems using perception of real objects. Tangible user interfaces (TUIs) [8] are an effective approach providing intuitive physical user interfaces for information access and management. Phidgets [6] are a set of building blocks of the physical devices to make it easy for application programmers to develop physical controllers, sensors, and physical presentations. We consider components based on these ideas as *tangible widgets*.

Auditory *Auditory* is a voice or sound interaction with microphones such as speech interfaces of sound interfaces. This modality also can be used in a variety of situation such as walking, lying down on a bed, or being busy with both hands. But it is not useful when users are in a noisy place or in a place to be quiet.

Visual *Visual* is an vision-based recognition technique with cameras such as hand-gesture or eye-tracking and so on. This modality includes Optical Character Recognition(OCR) that can distinguish characters on papers. We can use this modality if we can use devices that capture images of objects.

3.2.2 Expression

Expression is an available operation or a sensed region of input widgets. For example, a standard mouse device has two buttons and one sensor that senses two-axis motions. That means its expression is two buttons and two axes. This attribute implies the number of methods of input widgets.

3.2.3 Roles

Input widgets have semantics and functions to control or modify objects in their environments. We have considered that we could categorize interaction roles of input widgets into a few terms such as *Trigger* (sending messages of something happened), *Pointing* (selecting objects or drawing images with pointers), *Move-Grow*(changing parameters of target objects), *Text Input* (input and editing texts).

3.2.4 Bounded or Infinite

Bounded or Infinite is whether a state of input widgets is bounded or not. The former is *BoundedValue* and the latter is *Infinite*. If the state is binary (that is a part of BoundedValue), this parameter is *Binary*.

3.2.5 Relative or Absolute

This attribute shows whether input modalities handle a relative value or an absolute value. For examples, a stylus provides absolute positional information and a mouse provides relative values of motion.

3.3 Taxonomy of input modalities

We have examined and classified interaction techniques depending on the design space (Figure 1). We have focused on lightweight, small and easily deployed devices that are effective and useful in ubiquitous computing environments.

Mouse A mouse is a handheld pointing device for computers, involving two buttons and one tracking device that detect two-axis motions. The role of this device is two triggers and two-axis move-grow actions mainly to control a pointer on GUIs. The mouse's 2D motion is typically translated into the motion of a cursor on the display.

Keyboards A standard keyboard has over 100 keys for text inputs. They are consisted of normal keys, function keys, modifier keys, direction keys and so on. The main function of this device is handling text input and sending text message to applications.

Tangible Widgets *Phidgets* [6] are a set of building blocks for low cost sensing and control devices with USB interfaces. A phidget slider can handle a one-axis absolute value. A phidget joystick sensor treats two-axis values with momentary switch. And a knob formed device by griffin technology, treats one button and a one-axis value. These devices are used to change parameters of services.

Auditory Recognition Audio recognition has two aspects of interaction. The one is the speech input, and the other is the sound input. The former can handle the text and enable verbal interaction with applications. The latter can extract the non-verbal properties of sound or voices such as pitch, volume, timing and so on, and it controls parameters of services.

Vision Recognition Vision-based devices such as cameras can recognize gestures and postures. Our implementation of hand-gesture recognition toolkit can determine the position and the orientation of fingers. These modalities send messages or relative changes of their states.

Input Widgets

Input Capabilities	Mouse	Keyboard	Phidget Slider	Phidget Joystick	Knob Device	Mic (Speech)	Mic (Sound)	Camera (HandGesture)
Modality	Basic	Basic	Tungible	Tungible	Tungible	Auditory	Auditory	Visual
Expression	Two Buttons, Two-axis	More than 100 keys	One-axis	Two-axis	Button, One-axis	Speech	Volume, Pitch, Timing	Gesture, Posture
Role	Trigger (two buttons), Move-Grow (two-axis)	TextInput	Move-Grow	Move-Grow	Trigger (button), Move-Grow (one-axis)	TextInput	Move-Grow (volume, pitch), Trigger (timing),	Trigger (Posture), Move-Grow (Gesture)
Bounded or Infinite	Binary (two buttons), Infinite (two-axis)	Infinite	Bounded Value	Bounded Value	Infinite (one-axis)	Infinite	Infinite (volume, pitch), Binary (timing)	Binary(Posture), Infinite(Gesture)
Relative or Absolute	Relative (two-asix)	Absolute	Absolute	Absolute	Relative (one-axis)	Absolute	Relative (volume, pitch)	Relative (Gesture)

Figure 1. Device Capabilities of Input Modalities

4 Architecture

In ubiquitous computing, various interaction devices are deployed independently and they work cooperatively. For examples, instead of mouse and keyboards, we may use a game controller for controlling mouse cursor and assign its button to click action. In addition we may want to input texts by speech recognition interface with a headset. Frequently, these widgets are not on the same host and we must take into account multiple devices and heterogeneous platforms. If input widgets and services are not built on a common infrastructure, it is difficult to increase interoperability of them. And to switch input widgets to operate services according to contexts, we should design the dynamic reconfiguration APIs in their infrastructure.

In this section, we introduce the design of an input widget framework for distributed interaction environments. This framework provides the high-level abstraction for application programmers to help them increase reusability and flexibility to handle input widgets. Also it provides standard and generic interfaces between input widgets and services. Therefore, application programmers are encouraged to develop input widgets and services independently with no or a little knowledge of each others. And it offers a dynamic reconfiguration mechanism between input widgets and services at runtime.

4.1 Meta-inputs

To ensure connectivity of input widgets and services, we have defined *MetaInputs* that are device-independent abstract proxies of software controllers. In turn, they offer standard and generic interfaces between input widgets and services. They take responsibility to generate typed events by method call from input devices, and send them to service components. We have considered the MetaInputs need to fill the following requirements to increase reusability of input widgets.

- Meta-inputs must employ a small and fixed set of generic interfaces that are separated from the particular devices or services.
- Meta-inputs should clarify their roles, their functions, and the type of event data.

They are organized by the characteristics of the roles and values of various input widgets based on the design space. Examining the categories of input widgets, we have defined the four types of meta-inputs such as *Trigger*, *Delta*, *BoundedValue* and *TextEntry*.

Trigger *Trigger* module is used to cause something to happen immediately when an event occurs. For examples, when a mouse button is pressed or when a particular gesture is recognized, this meta-input module is invoked. This module has a state of binary, and it has a *trigger* function.

Delta *Delta* module can express a relative change of values, such as when a mouse cursor is moved or an audio volume is turned up/down. This module express a change of a state of input widgets as an integer value, and it has *change*, *increase* and *decrease* functions.

Proceedings of the Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)
0-7695-2357-9/05 $20.00 © 2005 IEEE

Bounded *Bounded* module is used to adjust absolute bounded value to service such as a phidget slider and a pen stylus. A Bounded module has ranged a bounded value from 0.0 to 100.0, and it has an *adjust* function to modifier an absolute bounded value.

TextEntry *TextEntry* module supports sending text information to services. For examples, when we input the text with the keyboards or we speech with the microphone, this module is used. This module has the text input functions such as *putKey* and *putText*.

4.2 Architecture Overview

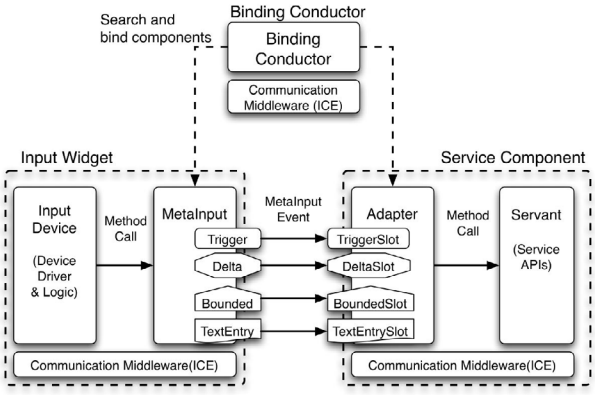

Figure 2. Interaction Components of the application framework

Our application framework is based on the distributed MVC model as same as Gaia [13] and BEACH [14]. Our framework consists of three parts, *Input Widgets*, *Service Components* and *Binding Conductor*. The overview diagram is shown in Figure 2.

4.2.1 Input Widgets

An input widget is an interaction component that provides input events to other components. It has a number of meta-inputs, that is the generic and standard interface module to attach other component's slots. Tasks of an input widget are to generate data from input devices, and to process these data, and to invoke methods of meta-inputs adequately.

Meta-inputs generate typed events to service components by method calls. Meta-inputs have a communication channel and provide location transparency and data transparency to programmers. An input widget has one or more meta-input modules, and each modules are able to send meta-input events to service components.

4.2.2 Service Component

Service Components consist of three parts, that is *Slot*, *Adapter* and *Servant*.

Slot can bind MetaInputs and receive meta-input events if their type are the same. They have a communication channel and they can receive the event from meta-inputs as long as their types are the same. *Adapter* has an event queue that aggregates received meta-input events. A callback function can be registered to the event queue and are invoked if the particular events are stored.*Servant* implements logic of applications and exports an interface. Servant has meta-input adapters, and adapters invoke callback methods of servant by observation of meta-inputs.

4.2.3 Binding Conductor

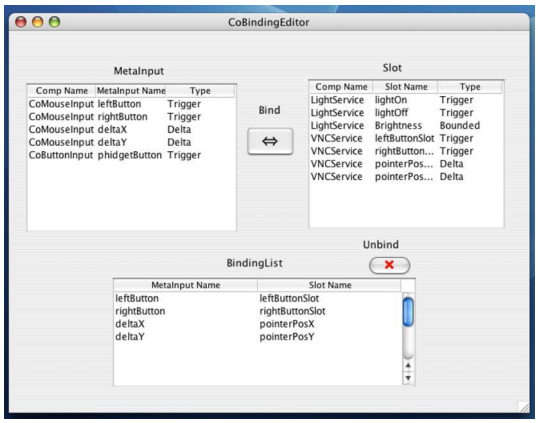

Figure 3. The GUI of Binding Conductor

To develop applications, our framework can couple input widgets with services. *Binding Conductor* components is responsible for composition between input widgets and service components(Figure 3). This component can register or unregister meta-inputs from meta-input slots dynamically. Therefore it enables users to switch input widgets at runtime according to users' contexts.

5 Implementation

Our framework is designed for distributed multi-modal environments, which are in heterogeneous platforms and languages. For this purpose, the framework is implemented as C++ and Java classes that can be extended by developers easily.

Our input widget framework consists of Internet Communication Engine (Ice) [7] IDL definitions of the component's interface and C++ and Java classes for the implementation. Ice is th e distributed object middleware simular to

CORBA. This means application developers can develop input widgets and services with both C++ and Java. And it will be ported to C#, Visual Basic and Python easily because of the capability of Ice IDL.

The framework is worked on a various platforms such as Windows, Linux and Mac OSX and so on. And it is easy to use other open source libraries and modules for processing input widgets and services.

These features provide obvious benefits to develop heterogeneous input widgets in ubiquitous computing environments because there are heterogeneous platforms and languages in these environments.

6 Evaluation

To support the development of multi-device and multi-modal environments in ubiquitous computing environments, it is necessary to provide enough processing performance. In order to evaluate the framework, we have focused on two aspects: the throughput of event processing and the performance of binding meta-input modules on this framework.

All the test have been performed in connected two machines that are the same conditions, which has a 100Base-T Ethernet networks, 802.11b wireless LAN, Pentium M 1.30GHz with 768MB of RAM, and Windows XP. All the times presented are the average result of ten experiments.

6.1 Throughput of Event Processing

We deployed the input widgets that has one Trigger meta-input and the services that count Trigger events on each hosts. After binding them, the input widget transmits the Trigger events a thousand times. When finished sending events, the service notified it back to the input widgets. The input widget measure the time from the beginning of sending events to receiving the notification from the service. We evaluated it in 802.11b and 100Base-T. And we calculate the number of event processing per one second. The results are shown in Figure 4.

In our experiments, we found that we can process the 135 events per one second in 802.11b, and the 294 events per one second in 100Base-T Ethernet networks on our framework. These numbers are enough for common distributed environments.

6.2 Binding MetaInputs and Slots

The same as the above, we deployed the Input Widgets that has one Trigger meta-input and the services that has the Trigger Slot on each hosts. And the BindingConductor repeats to bind them 1000 times. We evaluated it in 802.11b and 100Base-T. And we calculate that the time of binding a meta-input and a slot. The result is shown in Figure 5.

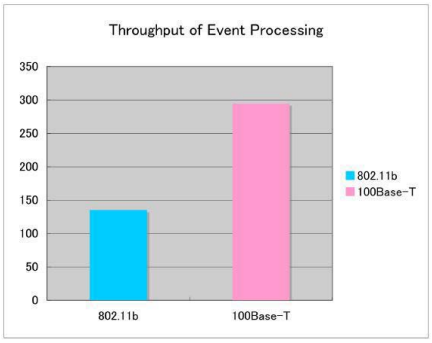

Figure 4. Throughput of Event Processing

In our experiments, we found that the time is 4.5msec in 802.11b and 2.3 msec in 100Base-T. These result shows that it is fast enough to process the binding modules on this framework.

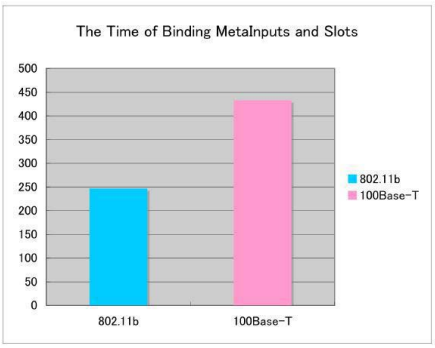

Figure 5. The Time of Binding MetaInputs and Slots

7 Building Applications

We have developed three multi-modal applications on this framework. The first application is the light control service using X10. The second application is the audio controlling services that is controlled by application scripts. The third application is the remote desktop application that shows how we reconfigure input widgets.

7.1 Home Appliance Service

In this section, we present the Home Appliance Service (Figure 6), which is an application based on our applica-

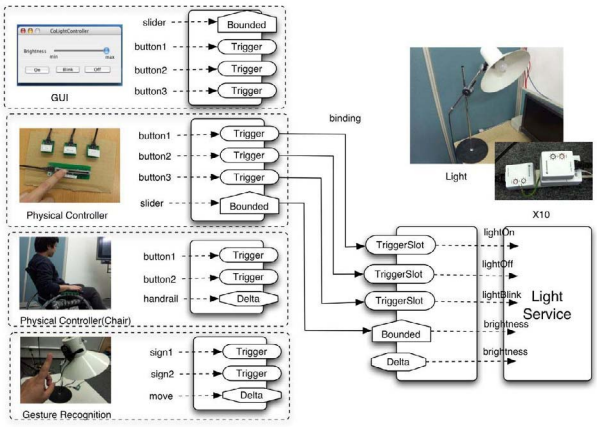

Figure 6. Home Appliance Service

tion framework that provides the functionality for controlling the room lights with several input widgets.

We designed four types of input widgets such as the GUI controller as a Java-Swing application, a set of physical controllers (phidgets), the chair which the physical sensors are attached to, and gesture recognition. However these input widgets are deployed on different hosts, they offer the equivalent functionalities. BindingConductor manages these input widgets and switches the suitable modalities according to the user's current situations.

7.2 Media Control Service

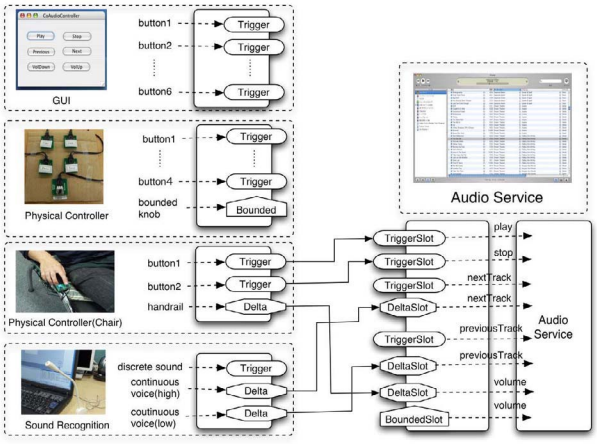

Figure 7. Media Control Service

In the second, we developed the Media Control Service (Figure 7), which is an application that provides functionalities for selecting and playing music files and changing its volume. The input widgets are almost reusable from the Home Appliance Service.

7.3 Remote Desktop Service

In the third, we developed the Remote Desktop Service (shown in Figure 8). When the user sits in front of the single general displays, the mouse and keyboards are often used. But if the user is in front of a large public display, it is difficult to use these personal devices. We developed the physical joystick and the controller with acceleration sensor, and it can control the mouse cursor without a basic mouse or a trackpad on the display.

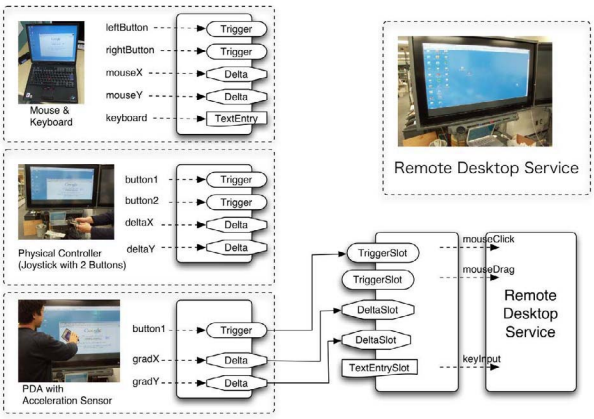

Figure 8. Remote Desktop Service

8 Discussion

In our framework, we can switch an input widget for another one at runtime when those input widgets have the same meta-input type. For example, a keyboard and a speech input modality have the TextEntry module as their meta-input module, so we can switch the speech interface from the keyboard dynamically to input texts according to situations. Moreover, we can substitute the gesture for the knob device to control the volume of the audio service because they have the same meta-input type as *Delta*. In that case, our framework facilitates the development and deployment of multi-modal and multi-device applications as below.

- Our framework enables us to bind new input widgets to service components without adding or modifying codes on the services side. Existing input widgets are easy to be bound to new services without adding or modifying programs on the controller side.
- Input modalities are dynamically reconfigurable when the meta-input types of the input widgets are same.

Proceedings of the Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)
0-7695-2357-9/05 $20.00 © 2005 **IEEE**

8.1 Interpretation of MetaInput Types

Even if meta-inputs are correctly used, it might become an improper combination for a user. For example, when we control the volume of the audio service with the physical knob, whether the right rotation is positive or negative depends on the developer of the input widget. Even if it can be the right connection between the input widget and the service, it may not be useful for users.

8.2 Capability Issues

According to physical characteristics, input widgets differ in qualities of their input events, that is error rate, accuracy, resolution and stable or unstable. If services demand on accurate controls, it is difficult to control it with high error-rate input widgets. For example, when controlling a mouse cursor, it is difficult to fix the position of it with the acceleration sensor or gesture recognition.

8.3 Monitor and Feedback

If there is no means to tell a user about binding status of input widgets and services, it is difficult to interact with services in these environments. It is necessary to design the monitor and feedback of the environment.

9 Conclusion

We described the challenge of a middleware infrastructure that supports heterogeneous input widgets in ubiquitous computing environments. We pointed out that providing semantic standard interfaces between input widgets and services increases independency, exchangeability and reusability of software components in multi-modal and multi-device interaction environments. In the future, we'll have more practical applications and evaluations on our framework.

References

[1] Johanson B. and Fox A. The Event Heap: A Coordination Infrastructure for Interactive WorkSpaces. *Proceedings of the 4th IEEE Workshop on Mobile Computer Systems and Applications*, 2002.

[2] Rafael Ballagas, et al. iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments. *In Proceedings of the ACM CHI 2003 Conference on Human Factors in Computing Systems*, pp. 537–544, 2003.

[3] Rafael Ballagas, et al. Patch Panel: Enabling Control-Flow Interoperability in Ubicomp Environments. *Proceedings of PerCom 2004. IEEE Computer Society*, pp. 241–252, 2004.

[4] Stuart K. Card, Jock D. Mackinlay, and G. Robertson. The Design Space of Input Devices. *CHI*, pp. 117–124, 1990.

[5] Pierre Dragicevic and Jean-Daniel Fekete. Input Device Selection and Interaction Configuration with ICON. *Proceeding of IHM-HCI 2001*, 2001.

[6] Chester Fitchett and Saul Greenberg. The Phidget Architecture: Rapid Development of Physical User Interfaces. *Workshop Application Models and Programming Tools for Ubiquitous Computing*, 2001.

[7] M. Henning, et al. Distributed Programming with Ice. *ZeroC*, 2003.

[8] H. Ishii and B. Ullmer. Tangible Bits: Towards Seamless Interfaces Between People, Bits, and Atoms. *Proceedings of CHI'97*, pp. pp.234–241, 1997.

[9] Brad A. Myers. A New Model for Handling Input. *ACM Transactions*, pp. 289–320, 1990.

[10] Brad A. Myers, et al. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering*, pp. pp.347–365, 1997.

[11] Alex Olwal and Steven Feiner. Unit: Modular Development of Distributed Interaction Techniques for Highly Interactive User Interfaces. *Proceedings of International Conference on Computer Graphics and Interactive Techniques*, 2004.

[12] Manuel Roman and Roy H. Campbell. A Middleware-Based Application Framework for Active Space Applications. *Middleware*, pp. 433–454, 2003.

[13] Manuel Roman, et al. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing Magazine*, 2002.

[14] Peter Tandler. The BEACH application model and software framework for synchronous collaboration in ubiquitous computing environments. *Journal of Systems and Software*, January 2004.

[15] Buxton W. Lexical and Pragramatic Consideration of Input Structures. *Computer Graphics*, pp. 31–37, 1983.

[16] M. Weiser. The Computer for the 21th Century. *Scientific American*, pp. 94–104, September 1991.