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

Nobuyuki Kobayashi
Student ID: 3603U060-8

Waseda University
Department of Computer Science
School of Science and Engineering
Supervisor: Prof. Tatsuo Nakajima

February 2, 2005

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 gestures, auditory recognition and tangible user interfaces will be 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-event framework that provides high-level abstraction for heterogeneous input devices, that we call *MetaInputs*, for distributed multi-modal applications. Our framework provides semantic and standard interfaces between input devices and services. It enables developers to deploy input devices and services independently. And also, our framework supports context-aware runtime adaptation that can switch input devices to handle services dynamically.

Contents

Chapter 1

Introduction

Multi-modal interaction is an inportant issue for ubiquitous computing environments. Until now, mouse and keyboard have emerged as usual input devices for desktop computers. But a desktop environments is aimed at one user, one set of hardware. In post-desktop , environments need to support heterogeneous interaction modalities for multiple devices, multiple users and multiple applications. The DCL research department proposes an application framework to tackle these difficulties in order to facilitate developing these environments.
In Section 1.1, we introduce the characteristics and issues of ubiquitous computing environments. Section1.2 describes the term of multi-modal interaction and the relation between multi-modal interaction and ubiquitous computing environments. In Section 1.3, we defined input widgets to explain modalities that generate input events in ubiquitous computing environments. The topic of Section 1.4 is the reason that we need application framworks in ubiquitous computing environments. To develop application frameworks, it is important to define the domain of framework. It is written in Section 1.5. The goal of this paper is described in Section 1.6 and the structure of this paper is introduced in Section 1.7.

1.1 Ubiquitous Computing

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. The 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. And interaction techniques in these environments become more seamless and intuitive. In these environments, one of the most important problems is how to interact with a variety of computer embedded devices. Recently, a lot

of researchers have developed intuitive interaction methods with ubiquitous services and heterogeneous interaction modalities such as sensor embedded physical devices, recognition-based technologies, or their combination.
And one of the important characteristics of ubiquitous computing environments is *context-awareness*. Various sensors that are attached to daily objects or users capture contextual information of environments. For examples, location sensors attached to objects or persons identify and trace the location of them. Camera devices may detect eye direction and analyze an attention of users. Such dynamic environmental information is used by ubiquitous software to give the users the right services at the right situation. The behavior of applications in ubiquitous computing environments should be changed according to the current situation.

1.2 Multimodal Interaction for Ubiquitous Environments

Nigay and Coutaz [11] mentioned that a multimodal system supports communication with the user through different modalities such as voice, gesture and typing. Literally,“multi” refers to “more than one” and the term “modal” may cover the notion of “modality” as well as that of “mode”.

- Modality refers to the type of communication channel used to convey or acquire information. It also covers the way an idea is expressed or perceived, or the manner an action is performed.
- Mode refers to a state that determines the way information is interpreted to extract or convey meaning.

One of the purposes of ubiquitous computing is the development of intuitive and flexible human-computer interaction environments. In existing desktop environments, we can use a set of mouse and keyboard for interaction with computers. But mouse is not useful for large-sized displays or multiple displays. And mouse and keyboard is not useful without a desk or a table such as a situation of walking, standing in front of large displays and so on. In ubiquitous computing environments, we need to consider various situations for using computers and services.

1.3 Input Widgets and Input Forms in Multi-modal Environments

In ubiquitous computing environments we have to focus on handling input events with heterogeneous modalities. In this paper, to clarify the target of input devices

and modalities, we defined *input widgets* as input forms 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. Although uncontrollable variables such as temperatures, heartbeat, weather and so on are not input widgets(often they are categorized as contexts), if application programmer designs them as explicit to services, they will be assumed as input modalities.
Input widgets contain several input expressions. For examples, standard mouse has two buttons and two-axis motion sensor. And these expression has each semantics in input widgets. We defined the available operations of input widgets as *input forms*. It is said that input widgets consist of a variety of input forms.
In multi-modal environments, heterogeneous input widgets and forms are researched for the realization of intutive human-computer interaction based on five senses such as touch, taste, visual, audito and smell. Because it is diffucult to interact virtual objects by taste and smell, researchers often have focused on the vision-based interaction and audio-based interaction, and physical interaction. We have classified broadly these input widgets to figure out the characteristics of them.

1.3.1 Basic Widgets

In current desktop systems, we always use mouse and keyboards to interact with GUI screens. We consider them as basic widgets. In current window systems, the workstation is supported to have one single mouse and keyboard. They may be the best practice for single user and one set of hardware in desktop systems. But in ubiquitous computing environments, there are more various complexities such as multi-user, multi-devices and so on. We should consider multiple event handling systems for ubiqiutous computing environments.

1.3.2 Recognition-based Widgets

The techniques of the speech-recognition and gesture-recognition are put to practical use. We can capture the audio data with the microphone device, and the motion data with the camera devices. We assume that it is effective to use these techniques as *recognition-based widgets* in ubiquitous computing environments.

1.3.3 Tungible Widgets

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) [10] is an effective approach providing intuitive physical user interfaces for information access and management. Phidgets [8] 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 *tungible widgets*.

1.4 An Application Framework in Ubicomp Environments

In ubiquitous computing environments, we need a framework to support us to develop applications. An Application framework offers benefits to application developer in three aspects. At first, it provides common functions in these applications as class libraries. Application programmers need not to repeat the similar codes in applications. Seconds, it provides high level abstractions that help programmers to develop applications according to the pre-established forms and the easy-to-understand APIs. It gives application programmers to the common semantics while building applications. Third, it hides complexities such as network communication, dynamic configuration and context-awareness. We believe that these aspects are very important to reduce the development cost of ubiquitous applications.

1.5 Domain of Our Framework

In ubiqutous computing environments, there will be a variety of services that helps us in dalily lives by heterogeneous networked sensors and devices. But to design the application framework practically, we have focused on current available services around us. Our application framework has been designed for three types of services: a remote desktop service, a RPC-based service including controlling information appliances and a script-based services that controls application on existing operating systems.

1.6 Goal of This Paper

In this paper, we describe the application framework that provides the standard and semantic interfaces, and ensure connectivity between these heterogeneous input modalities and services. Existing middleware infrastructures such as BEACH [14] and Gaia [13] distinguish user interfaces and application logics in ubiquitous environments so they increase reusability of software components, but they does not take into account defining standard and semantic interfaces between them. The main goal behind our framework is to enable programmers to develop input modalities and services components independently with little or no knowledge of one another.

1.7 Structure

This master's thesis discussed the input event framework that is part of a middleware that aims for fast and easy development of multi-modal and multi-device applications in ubiquitous computing environments. This chapter shows the need for such a software infrastructure, and the goal of this paper. This framework described in the introduction imposes certain requirements on the interaction framework. Additionally, this framework makes us easy to develop multi-modal applications in ubiquitous computing environments. Chapter 2 describes the related work of this framework. In Chapter 3, we discuss about the classification of input modalities by their characteristics to design the standard and semantic interface between input modalities and services. In Chapter 4, We describe the architecture design of this interaction framework and discuss the decisions and tradeoffs involved. In Chapter 5. We show the concrete implementations of this framework and interaction components on it. How this framework is used to build interaction components is the topic of Chapter 6. Chapter 7 describes the evaluation of this framework in qualitative and quantitative aspects. The discussion of this framework is covered in Chapter 8 and the paper is concluded in Chapter 8.

Chapter 2

Related Work

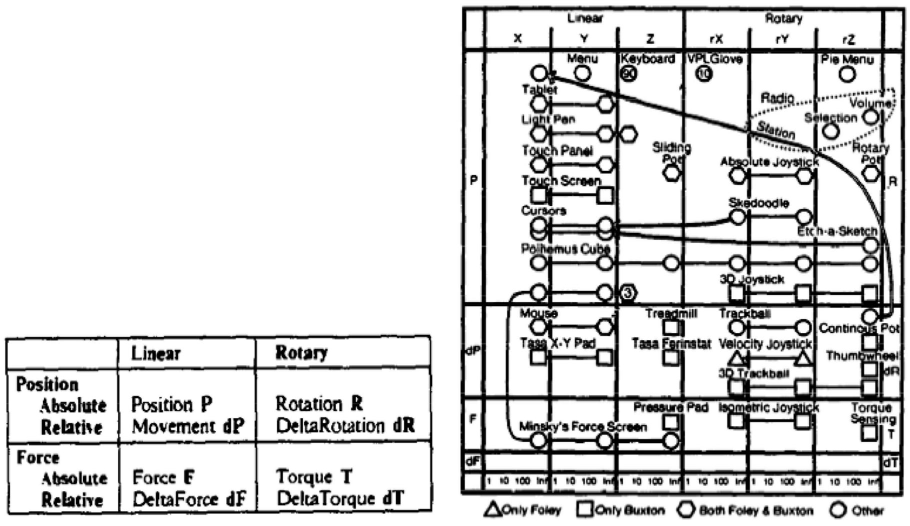
To develop our application framework, it is important to classify input modalities, that is input forms and devices that provide explicit interaction to services. And we need to inquire into the application frameworks in ubiquitous computing environments and examine how they fulfill our requirements of multi-modal and multi-device environments.
In this chapter, we examine the related works and reveal the differences between our framework and these works. These researches have focused on taxonomy of input modalities, distributed application frameworks, and toolkit for interaction environments.

2.1 Buxton et al. and Card et al.

The taxonomies of input modalities have been proposed earlier by Buxton [15] et al. and Card et al. [5]. They have classified input devices by their design space to clearify the characteristics of input devices. Baecker and Buxton consider all input modalities are combinations of physical properties such as linear and rotary, absolute and relative, position and force. They have such a observation that:

> Basically, an input device is transducer from the physical properties of the world into logical values of an application.

Based on their research, Card et al. [6] mensioned that there are two key ideas in modeling the language of input device interacton: a primitive movement vocabulary and a set of composition operators. They represents the input modalities as a six-tuple of primitive movement vocabulary: manipulation operator, input domain, current state, resolution of function, output domain and works. Manipulation operators represent the idea of Buxton's physical properties. Input domain means the range of values of input manipulation. Current state is a state of input



	Linear	Rotary
Position		
Absolute	Position P	Rotation R
Relative	Movement dP	DeltaRotation dR
Force		
Absolute	Force F	Torque T
Relative	DeltaForce dF	DeltaTorque dT

Figure 2.1: The design spaces of input devices by Card et al.

modalities and resolution of function shows how they can be controlled minutely. Output domain shows the range of output values and works are additional device properties.Their design space for input modalities is basically the set of possible combinations of the operators with primitive vocabulary. Their classification of input modalities are seen in Figure 2.1.
To be sure, they clearfy the detail of input modalities and classified them. But they focused attention on the low-level functionality of interaction modalities and they have not considered he semantics and roles adequately in interaction environments. Also, they have not describe about recognition-based interactions such as an auditory recognition and a gesture recognition. Moreover, their design spaces of input modalities did not contain varying modalities and social properties.

2.2 iStuff

iStuff [3] is a part of the researches on the interactive workspace project in Stanford University. It is designed as the physical user interface toolkit for ubiquitous computing environments. They developed the interaction toolkit in the following requirements:

- Flexible, lightweight devices.
- Platform independence and cross-platform capabilities
- Wireless protocol independence

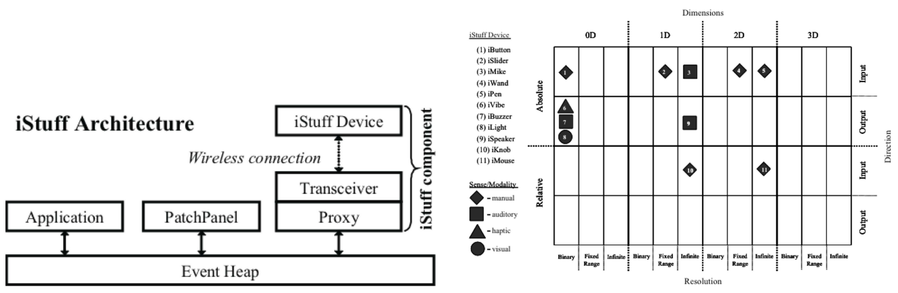


Figure 2.2: iStuff: Architecture and their design space

- Ease of integration with existing applications
- Support for multiple simultaneous users.

They create the toolkit based on iROS Event Heap, that is the blackboard architecture in interactive workspaces, and the PatchPanel [4] that re-map events to applications dynamically. These architecture is shown in Figure 2.2(left).
They developed the iStuff component that consist of wireless physical devices paired with a machine connected to the Event Heap [2] that has a transceiver and related software. To classify them into several categories,they propose a five-part design spaces: direction, modality, dimensions, resolution, dimensions and relative or absolute in Figure 2.2(right). Direction indecates whether a device is used to provide input, output, or both. Modality describes the modalities used to operate devices. Resolution shows the range of the values of input modalities such as binary, bounded value, and an infinite range. Dimension shows the value dimensions of input modailties such as 0, 1, 2, and 3D. And the relative or absolute describes the value of input modalities means relative or absolute.
However, their architecture have not ensured which components and services can connect each others. In addtion, because they have not provided the standard interfaces between input modalities and services, the application programmers have to be aware of the event types between components and services. And because of the event heap mechanism, it is difficult to process the event streams in their toolkit.

2.3 Gaia

The Gaia project at the University of Illinois is developing a middleware for ubiquitous computing environments that is called “active spaces”. They have extended

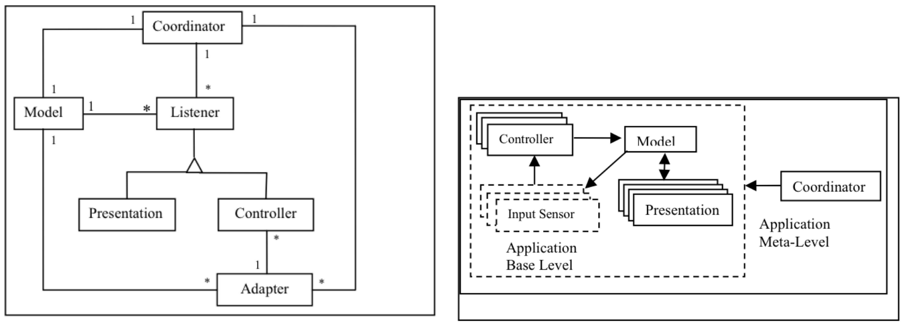


Figure 2.3: Gaia: Model-Presentation-Adapter-Controller-Coordinator

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 ubicomp environments(Figure 2.3). The term *Model* implements a application logic and document. And it offers an interface to operate the state of applications. *Presentation* component transforms the application's state into a representation such as a graphical or audible, a temperature or lighting variation ro any external representation that affects the user environment and can be perceived by any of human senses. A *Controller* is a component (hardware and software) capable of altering application's state through the Model's interface. An *Adapter* acts as a mediator between the controller and the model. And a *Coordinator* collects these components and construct active space applications.
Although we can build several distributed applications on this framework, but they have not mentioned about the connectivity of these components. They have provided the abstraction of distributed application components in ubiquitous computing. But their framework have not provided the abstraction of data flow between controllers and models. Because it has not offered high-level semantic interfaces between these components, it is difficult for application programmers to develop them without knowing each other. Because it is difficult to ensure the connectivity of these components, it is not sure that we could switch input modalities dynamically.

2.4 ICON

ICON [7] is the editor designed to configure a set of input devices and connect them to actions into a graphical interactive application(Figure 2.4). Using ICON, users can connect additional input devices such as tablets, voice recognition software, assistive devices and so on.

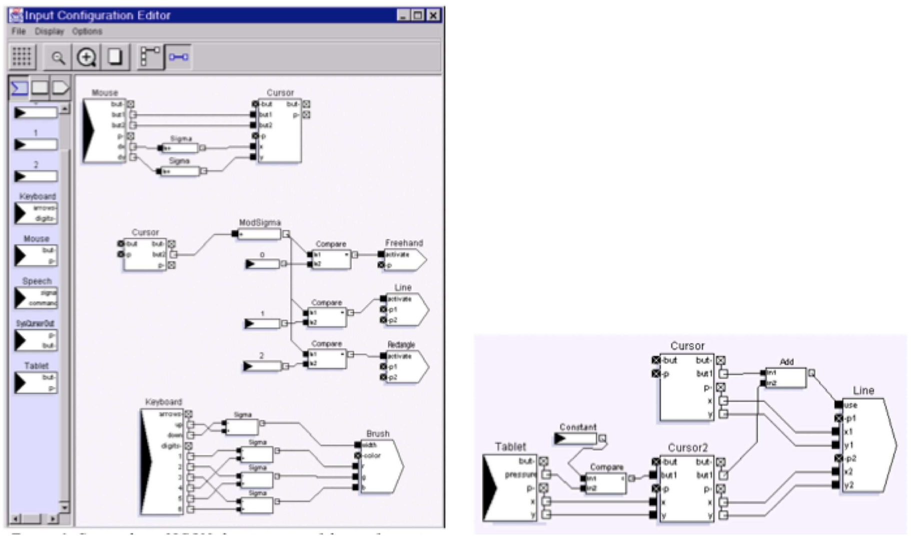


Figure 2.4: ScreenShot of ICON and its Model

It has three types of modules such as: output modules, processing modules and input modules. Output modules consist of typed output channels to transfer their data to other modules. When the editor is launched, input devices are detected and capabilities of each devices are interpreted as a set of typed channels. Processing modules have both input and output slots. And they are used for controlling switches, calcurating, operating boolean, comparisons and memorization of previous values. And input modules have input channels to receive data to process application specific tasks.
Although their framework supports multimodal interaction and dynamic configuration, they have not discussed the type of input and output channel adequately. Their channel type shows just the primitive of language and they have not considered the range of values and semancs of actions.

Chapter 3

Design Space of Input Widgets

There are a variety of input widgets in ubiquitous computing environments. Not only mouse and keyboard for basic desktop computings, but more intutive interaction techniques are to expected to be used to interact with services in these environments. In audio recognition techniques, speech input are used for text input. Gesture recognition techniques enable to send non-verval messages to services. And physical widgets such as sensor embeded daily objects help us to integrate with real world and virtual world.
Although these interaction techniques seems to be intuitive and effective, but in ubiquitous computing a new type of complexitiy is added:

- When using new input widgets to services, it is difficult to adapt them without adding or modifying programs on the services side.
- It is difficult that existing input widgets be bound to new services without adding or modifying programs on the controllers side.

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

3.1 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 defined five categories: modality, expression, roles, bounded or Infinite, and relative or absolute. We describe the details of these attributes in this section.

3.1.1 Modality

This attribute shows a mode of interaction styles. When we access ubiquitous services, we must represent a command by actions to occur input events. When we represent such an action, we need to express it by five senses such as touch, taste, hearing, eyesight, and smell. But taste and smell are not used as interaction techiniques generally. Therefore we have considered that modalities are based on three interaction mode, and one basic style.
Modality directly limits the situations of using input modalities. For examples, it is difficult to use basic modality such as mouse and keyboard when both hands are busy. Also, it is difficult to use audio interfaces when environments are noisy or have policies to be quiet.

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

Tangible *Tangible* is an interaction technique with sensored and computed physical objects such as joystick, knob and a device with various sensors. This modality includes mobile keypad on handheld devices. These object can be designed into various shapes and can be used in more various situations than the basic modality. But if there are a lot of tangible modality objects on a desk, it is difficult to manage these objects and it may reduce usability. It is important to design well and deploy these object at the right place.

Auditory *Auditory* is a voice or sound interaction with microphones such as speech input of sound input. This modality also can be used in a variety of situations. For examples, we can use speech command interface even if we are 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 a place to be quiet.

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

3.1.2 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 axis. This attribute implies the number of manipulated variables.

3.1.3 Roles

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

3.1.4 Bounded or Infinite

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

3.1.5 Relative or Absolute

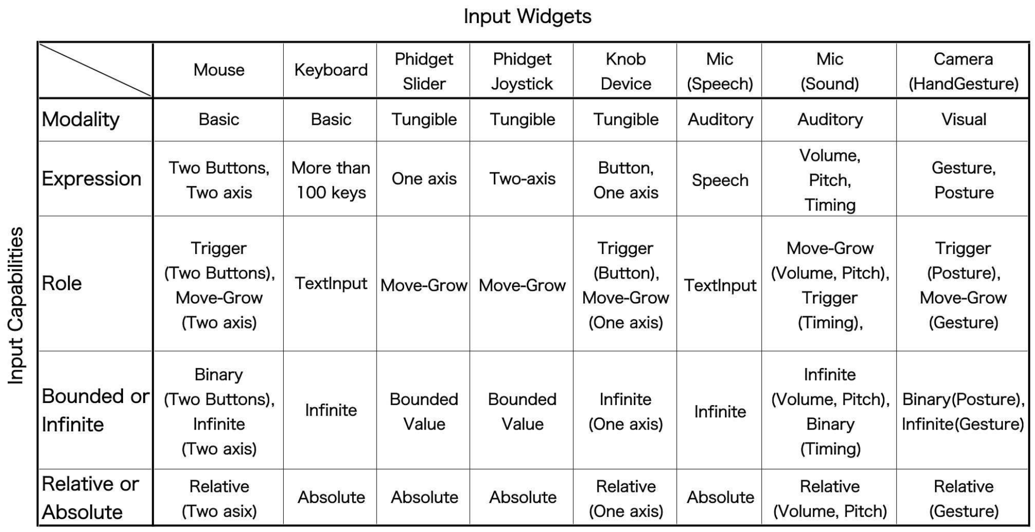
This attribute shows whether input modalities handle a relative value or an absolute value. For examples, a stylus provides absolute positional information and a mouse provides relative values of motion.

3.2 Taxonomy of input modalities

We have examined and classified interaction techniques depending on the design space. We have focused on lightweight, small and easily deployed devices that is effective and useful in ubiquitous computing environments(Figure 3.1).

3.2.1 Mouse

A mouse is a handheld pointing device for computers, involving a small object fitted with one or more buttons and shaped to sit naturally under the hand. The underside of the mouse houses a device that detects the mouse's motion relative



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 3.1: Device Capabilities of Input Modalities

to the flat surface on which it sits. The mouse's 2D motion is typically translated into the motion of a cursor on the display.
A standard mouse has two buttons and one tracking device that detects 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 value of the axis are infinite and relative. Five or more buttons have sometimes been built in to mice, especially ones designed for the Windows operating system. A few have included an extensive array of buttons. Depending on the user's preferences, these buttons might allow forward and backward web navigation, or scrolling through a browser's history. However, these newer functions are not supported by all software. The additional buttons are often used in computer games, where quick and easy access to a wide variety of functions (for example, weapon-switching in first-person shooters) can be very beneficial.

3.2.2 Keyboards

A standard keyboard has over 100 keys for input text. They are consisted of text 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. The value of the text input is infinite and absolute.

3.2.3 Tangible Widgets

Phidgets [8] 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.

3.2.4 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.

3.2.5 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 states.

Chapter 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 functions are not on the same host and we must take into account multiple devices and heterogeneous platforms. And if we want to switch input modalities to operate services according to contexts, we have to implements the dynamic reconfiguration APIs in these modules. But if input modalities and services are not built on a common infrastructure, it is difficult to implement them interoperably.
In this chapter, we introduce the design of application framework that we call *concerto* for distributed interaction environments. This framework provides the high-level abstraction for application programmers to help them easy to learn and increase reusability of their codes. Also it provides standard and semantic interfaces between input modalities and services. Accordingly application programmers are encouraged to develop input modalities and services independently with no or a little knowledge of each others. And it offers a dynamic reconfiguration mechanism between input modalities and services at runtime.

4.1 MetaInputs

To ensure connectivity of input modalities and services, we have defined the *MetaInputs* that are the device-independent abstract software controller proxies. In turn, they offer standard and semantic interfaces between input modalities and services. They take responsibility to generate the typed events by the 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 modalities.

- 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.

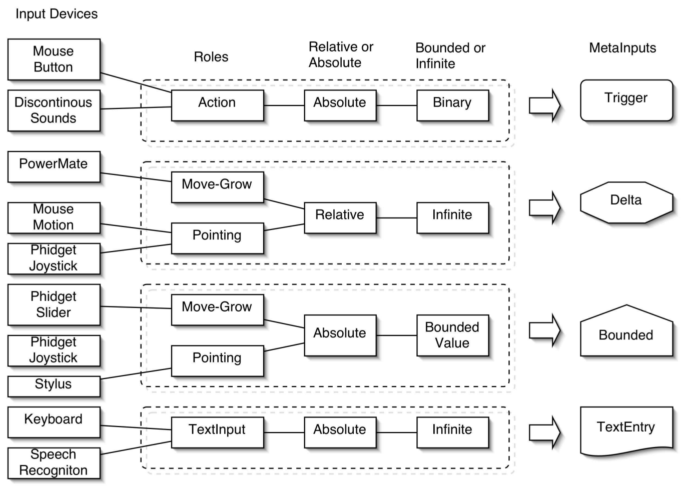


Figure 4.1: Extraction of meta-inputs from design space

They are organized by the characteristics of the roles and values of various input modalities based on the design space. Examining the categories of input modalities, we have defined the four types of MetaInputs such as *Trigger*, *Delta*, *BoundedValue* and *TextInput*.

Trigger The *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 MetaInput module is invoked. This module has the state of binary, and it has *trigger* functions.

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

BoundedValue The *BoundedValue* module is used to adjust absolute bounded value to service such as a phidget slider and a pen stylus. The BoundedValue module has treat the bounded value from 0.0 to 1.0, and it has the *adjust* function to modifier the absolute bounded value.

TextInput The *TextInput* module supports sending the 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 *inputChar* and *inputString* function and so on.

4.2 Architecture Overview

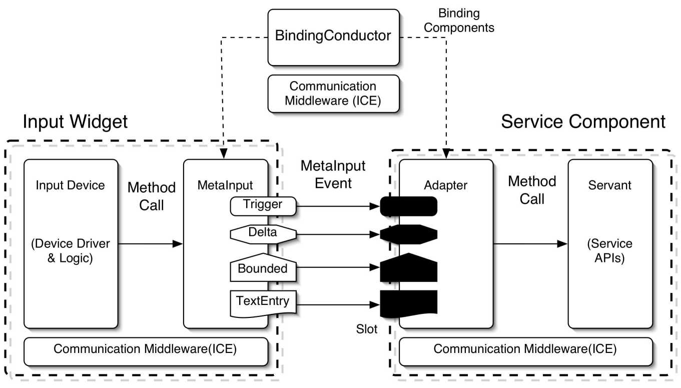


Figure 4.2: Interaction Components of the application framework

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 Modalites, Service Components* and *Binding Conductor*(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 semantic and standard interface module to connect other component's slots. The tasks of an input widget are to produce data from input devices, and to process these data, and to invoke methods of meta-inputs adequately. They indicates that we can design and implement input widget independently as a stand-alone conponent.

Meta-input modules generate typed events to service components by method calls. Meta-input modules are implemented as Ice objects [9] and provide location transparency and data transparency to programmers. An input modality has one or more meta-input modules, and each modules are able to send events to service components.

4.2.2 Service Component

Service components consist of three parts, that is *Slot*, *Adapter* and *Servant*.
Slot can bind meta-inputs and receive meta-input events if their type are the same. They are implemented as Ice objects and they can receive the event from meta-input modules as long as their types are the same.
Adapter has an event queue that aggregates received 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 the adapters invoke the callback methods of the servant by the observation of meta-input's events. Currently, we have assumed that our framework targeted existing services such as remote desktop services, script-based services (e.g., Applescript) and web services (e.g., controlling home appliances).

4.2.3 Binding Conductor

To develop applications, our framework can couple input modalities with services. *Binding Conductor* components is responsible for composition among input modalities, filters and services. And this component can register and unregister meta-inputs from slots dynamically. Therefore it enables users to switch input modalities at run-time according to users' contexts.

Chapter 5

Implementation

In this chapter, we describe the input event framework described in the previous chapter. It is designed for easy usage and fast development of distributed multi-modal applications. For this purpose, the framework are implemented as C++ and Java class that can be extended by developers.

5.1 Language and Platform Transparency

The input event framework consist of Internet Communitation Engine(Ice) IDL definitions of the component's interface and C++ and Java classes for the implementation. This means application developpers can develop input modalities and services with both C++ and Java. And it will be ported to C#, Visual Basic and Python easily because of Ice IDL.
The framework is worked on a various platforms such as Window, Linux and Mac OSX and so on. And it is easy to use other open source libraries and modules for processing input modalities and services.
These features provide obvious benefits to develop heterogeneous input modalities in ubuquitous computing environments because there are heterogeneous platforms and languages in these environments.

5.2 Interaction Components

The C++ or Java class CoComoponent implements an abstraction of an interaction components. The class is composed of one or more communication modules such as meta-inputs and their slots, and it has a application logics that manages these communication modules.The interaction component is classified into three types, such as CoInputWidget, CoComposer and CoService.

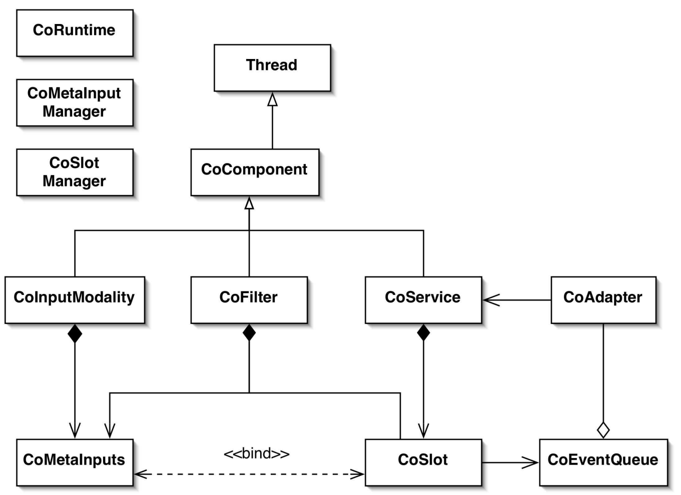


Figure 5.1: UML class diagram for interaction components

A CoInputWidget has one or more meta-inputs and the mechanism that invokes these meta-inputs. From a different view, a CoInputWidget explain their input forms as meta-inputs. For examples, the standard mouse is implemented as the input widget that has two trigger meta-inputs and two delta meta-inputs.
A CoComposer has one or more meta-inputs and one or more slots. The main role of this component is to composite or transform input events from input widgets. This component receives meta-input events from input widgets or other composer components.
A CoService has one or more slots and contains service logics. A CoService has adapters that interprets meta-input events and invoke high-level semantic methods such as pointerMove or pointerDrag. It facilitates service developers to adapt existing services to this framework.

5.3 Communication Modules

This framework has communication modules such as CoMetaInput and CoSlot. These modules are defined as Ice Interfaces and implemented as C++ or Java classes. We have designed a meta-input can be bind with multiple slots and a slot can be bind with multiple meta-inputs if the type of these modules are the same. There are four types of meta-inputs and slots such as CoTrigger, CoDelta, CoBounded and CoTextInput.
A CoTrigger is the subclass of CoMetaInput and it has the trigger method. And it can send the CoTriggerEvent that contains the state of CoTrigger that is true or

false. A CoTrigger only can be bind with CoTriggerSlot that is the subclass of CoSlots.
A CoDelta is the subclass of CoMetaInputs and it has the change method. This module can send a variation of input forms. This module support the ratio of values to slots.
A CoBounded has the adjust method. And it controls the absolute bounded value. Default value is from 0.0 to 1.0
A CoTextEntry has the setText method. And it can send the text message to CoTextEntrySlots. It can treat the modifier keys and function keys.
These modules encapsulates the communication logic to other modules. It can be said that these modules enable location transparency to other components.

5.4 Ice Interfaces

Our framework is built on the Internet Communication Engine(Ice) that is the modern object-oriented distributed middleware supporting various languages such as C++, Java, Python, C#, and various operating systems such as Windows, Linux, Mac OSX.
The binding interfaces in this framework are written in Ice IDL. The IDL interface definitions are implemented by C++ and Java classes.

5.5 Object Interaction

In this section the interaction of object on this framework will be described.
In CoInputWidget, the main class creates meta-inputs with CoMetaInputManager, that is the Mangaer class of meta-inputs. In CoServiceWIdgets, the main class creates slots with CoSlotManager. And CoBindingConductor binds their modules. if binding is completed, the meta-input modules can send their events to slots.

5.6 Building Input Widgets

We built several tangible modalities using a physical slider, a physical button and a physical joystick device from phidget toolkits [8].
We leveraged the Julius [1], that is a high-performance large vocabulary continuous speech recognition decoder software for speech recognition. And we built our original non-verbal sound interaction toolkit that distinguishes the pitch, volume, and whether the sound is continuous or discrete, that was adapted to Delta Meta-Input.

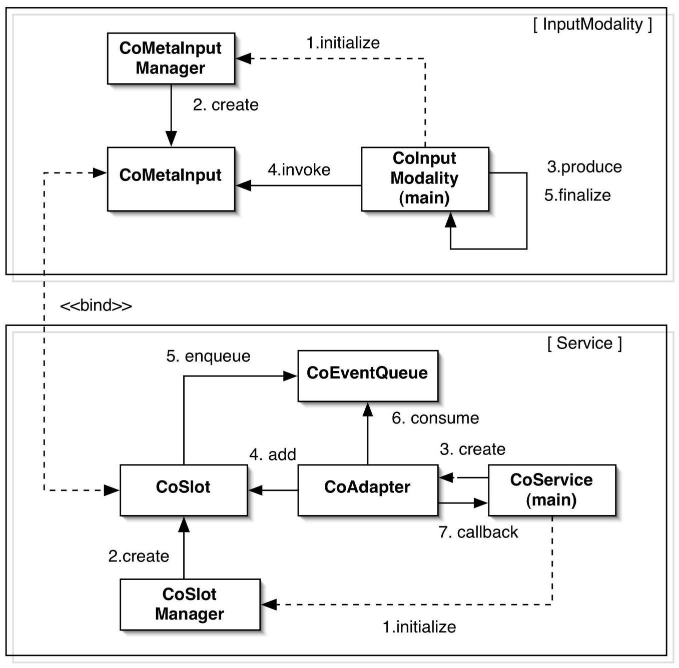


Figure 5.2: UML collaboration diagram for object interaction

Also, we built a vision-based hand-gesture module that extracts finger positions and orientations . This module can recognizell the gestures and postures of both hands.

5.7 Building Services

We have implemented three services and several input modalities. The first one is a remote desktop service using RFB protocols. The second one is an application script-based service that is a jukebox controller service by using AppleScript on MacOSX. The last one is RPC-based service. We have implemented a light switching service using X10 remotes.

5.8 Applications

We have developed three multi-modal applications on this framework. We have focused on three types of sevices, such as remote desktop services, script-based services and light control services. The first application is the remote desktop application that shows how we reconfigure input widgets. The second application is the audio controlling services that is controlled by application scripts. This example shows how we reconnect services to the same input widgets. And third application is the light control service using X10. This example shows the how input widgets collaborate each other.

5.8.1 Remote Desktop Service

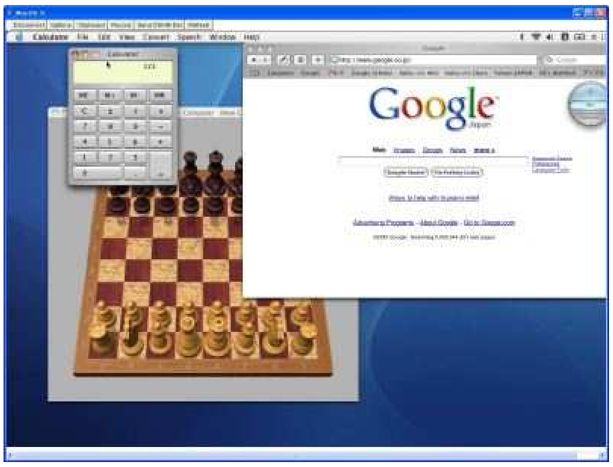
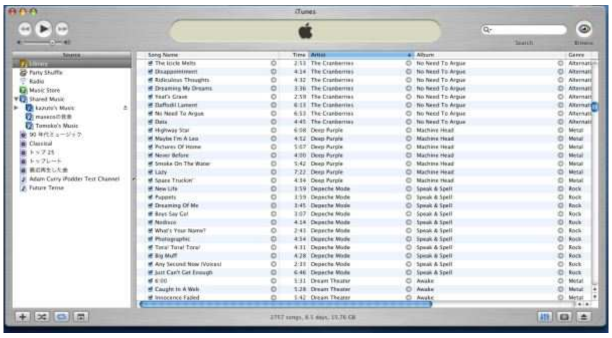


Figure 5.3: Remote desktop service

The first application is the remote desktop application. A user is working on the one single display and he moves around the room. If he want to use the large display without mouse and keyboard, he add the joystick to the system and he can control mouse pointer by it. And he want to use the gesture input to control photo application, he set camera and bind with them. If he swing his hand towards the camera, he can change the photo that shown on the large display.
VNC is the remote desktop system that transfer the bitmap image and we change the tightVNC viewer and develop gesture modules and phidget modules.

5.8.2 Script-based Service

The second application is the audio control application. A user sit the sensor attached chair. When he start application, he can controls the audio services by its chair. And when the movie starts, he can control the movie application with the same controller.
These scenario is designed by script-based application. And the mechanism of switching input widgets towards services. He need not to attend remote controllers or other devices.



Song Name	Time	Artist	Album	Genre
The Icicle Melts	2:53	The Cranberries	No Need To Argue	Alternati
Disappointment	4:14	The Cranberries	No Need To Argue	Alternati
Ridiculous Thoughts	4:32	The Cranberries	No Need To Argue	Alternati
Dreaming My Dreams	3:36	The Cranberries	No Need To Argue	Alternati
Yeat's Grave	2:59	The Cranberries	No Need To Argue	Alternati
Daffodil Lament	6:13	The Cranberries	No Need To Argue	Alternati
No Need To Argue	6:53	The Cranberries	No Need To Argue	Alternati
Data	4:45	The Cranberries	No Need To Argue	Alternati
Highway Star	6:08	Deep Purple	Machine Head	Metal
Maybe I'm A Leo	4:52	Deep Purple	Machine Head	Metal
Pictures Of Home	5:07	Deep Purple	Machine Head	Metal
Never Before	4:00	Deep Purple	Machine Head	Metal
Smoke On The Water	5:42	Deep Purple	Machine Head	Metal
Lazy	7:22	Deep Purple	Machine Head	Metal
Space Truckin'	4:34	Deep Purple	Machine Head	Metal
New Life	3:59	Depeche Mode	Speak & Spell	Rock
Puppets	3:59	Depeche Mode	Speak & Spell	Rock
Dreaming Of Me	3:45	Depeche Mode	Speak & Spell	Rock
Boys Say Go!	3:07	Depeche Mode	Speak & Spell	Rock
Nodisco	4:14	Depeche Mode	Speak & Spell	Rock
What's Your Name?	2:43	Depeche Mode	Speak & Spell	Rock
Photographic	4:54	Depeche Mode	Speak & Spell	Rock
Tora! Tora! Tora!	4:31	Depeche Mode	Speak & Spell	Rock
Big Muff	4:28	Depeche Mode	Speak & Spell	Rock
Any Second Now (Voices)	2:33	Depeche Mode	Speak & Spell	Rock
Just Can't Get Enough	6:46	Depeche Mode	Speak & Spell	Rock
6:00	5:31	Dream Theater	Awake	Metal
Caught In A Web	5:28	Dream Theater	Awake	Metal
Innocence Faded	5:42	Dream Theater	Awake	Metal

2757 songs, 8.5 days, 15.76 GB

Figure 5.4: Audio service example

5.8.3 Light Control Service

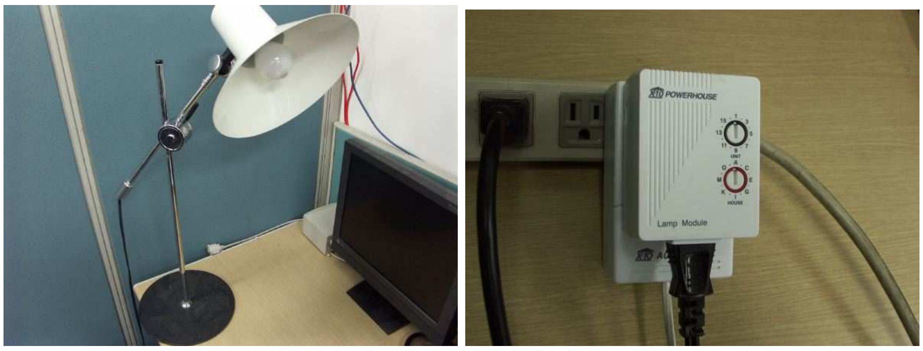


Figure 5.5: Light service example

The third application is the light controlling services. A user point his finger to light and say “On”. The light that are pointed becomes “on”. It is the composition example of gesture widget and speech widget. And he says “Ah-“, he can control the light brightness by its pitch continuously.
The light is controlled by X10 device that can controlles the voltage of devices.

Chapter 6

Discussion

6.1 Case studies

In our framework, we can switch an input modality for another one at runtime when those input modalities have the same meta-input type. For example, a keyboard and a speech input modality have the TextInput 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 multimodal and multi-device applications as below.

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

6.2 High-level events

We may need the high-level event mechanism. When we develop the audio application, we defined the function , next, back, start and stop. These function are reusable when controlling multimedia applications. It is important to consider the domain and reusable events.

6.3 Semantics between controller and services

We found the typed approach is not enough to share the semantics between service developper and input widget developper. For examples, if the delta value of knob device and x-axis of mouse is not the same. So it is not useful just mapped knob's delta to mouse pointer. It needs to be configured by user. And the input widget develpper and service developper may need to publish the configurable APIs.

6.4 Synchronization of input widgets

We give the timestanps to input widgets and lifetime to meta-input evnets. If we archive collaboration, we have to check the timing of events. We just implement simple synchronization algorithm. If we need the complex collaborations, we need the new algorithm or designed detail.

Chapter 7

Conclusion

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

Bibliography

[1] A.Lee, T.Kawahara, and K.Shikano. Julius: an open source real-time large vocabulary recognition engine. *in Proc. EUROSPEECH*, pp. 1691–1694, 2001.

[2] 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.

[3] 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.

[4] Rafael Ballagas, Andy Szybalski, and Armando Fox. Patch panel: Enabling control-flow interoperability in ubicomp environments. *Proceedings of PerCom 2004. IEEE Computer Society*, pp. 241–252, 2004.

[5] Stuart K. Card, Jock D. Mackinlay, and G. Robertson. The design space of input devices. *CHI*, pp. 117–124, 1990.

[6] Stuart K. Card, Jock D. Mackinlay, and George G. Robertson. The design space of input devices. *Proceeding of CHI 1990*, pp. 117–124, 1990.

[7] Pierre Dragicevic and Jean-Daniel Fekete. Input device selection and interaction configuration with icon. *Proceeding of IHM-HCI 2001*, 2001.

[8] Chester Fitchett and Saul Greenberg. The phidget architecture: Rapid development of physical user interfaces. *Workshop Application Models and Programming Tools for Ubiquitous Computing*, 2001.

[9] M. Henning, et al. Distributed programming with ice. *ZeroC*, 2003.

[10] H. Ishii and B. Ullmer. Tangible bits: Towards seamless interfaces between people, bits, and atoms. *Proceedings of CHI'97*, pp. pp.234–241, 1997.

[11] Laurence Nigay and Joelle Coutaz. A design space for multimodal systems: Concurrent processing and data fusion. *Proceedings of INTERCHI*, 1993.

[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.