International Journal of Engineering and Applied Sciences (IJEAS)
ISSN: 2394-3661, Volume-4, Issue-2, February 2017

A Survey of Middlewares for Multi-Modal Interactions

Jinseok Seo

***Abstract*— The user interfaces in computing environments are becoming experience - oriented, diversified, and personalized from the WIMP (Widows, Icons, Mouse, and Pointer) interface, and a wide variety of interaction devices are continuously being developed to support these needs. Therefore, in order to combine these new devices with existing application software, a method for connecting operating systems and the existing software is needed. Researchers in various fields such as virtual reality, home appliances, and ubiquitous computing have studied middleware to solve this problem. This paper is a survey of these middlewares.**

***Index Terms*— Multi-Modal, Interaction, Middleware.**

I. INTRODUCTION

Already, many IT devices such as smartphones, table PCs, notebook PCs, game consoles, and IP TVs are being used throughout our lives and are becoming indispensable nowadays. In addition, hardware-related technologies such as smaller and less power-consuming processors and memories have evolved in recent years, despite the stronger performance, and much more IT devices are expected to be used in our daily lives in the near future.

Many researchers around the world are making efforts to efficiently use and manage such overflowing devices, and especially, they are paying a great deal of attention in the fields related to HCI (Human-Computer Interface). However, most HCI researches are limited to the development of recognition algorithms (e.g. voice, gestures) or hardware devices for specific modalities, while the development of system software to support various kinds of modalities and input devices and standards have not been studied yet.

In the case of virtual reality, there has been an effort to support programmers to develop interactions using multiple devices and to develop them through the same interface. However, technologies that support a broad range of public services or devices have not yet been developed. Microsoft continues to make efforts to support various devices and platforms through platforms such as Plug and Play and .NET, but it is dependent on OS such as MS Windows.

In addition, with the recent successful commercialization of smartphones, console games, and haptic devices and the arrival of the ubiquitous environment where computers must be contacted anywhere, the current WIMP (Windows, Icons, Mouse, and Pointer) interface evolves into a next-generation experience-oriented and personalized interface, and it is necessary to develop system software to support this evolution. However, in order to incorporate such an evolved interface into existing application software, it causes a lot of overhead due to the lack of proper mediating layer between OS and application software. Therefore, it is important to establish appropriate standards for easy integration of these services and interactions and to develop the middleware required for them.

Because of this background, this study investigated middlewares supporting multimodal interactions and various types of devices. As a result, we have found the efforts and results in various fields such as IoT (Internet of Things), home appliances, ubiquitous computing as well as virtual reality. In this paper, we analyze 13 representative cases among the results.

Jinseok Seo, Division of Digital Contents Technology, Dong-eui University, Busan, Korea

II. MIDDLEWARES FOR MULTI-MODAL INTERACTIONS

A. VRPN

It is VRPN [1] that provided the most motive to this study. VRPN is a library for originally developed for virtual environment applications. It is composed of classes for using various VR input devices (tracker, 3D mouse, glove, etc.) distributed on a network. Complex virtual environment applications often operate in a distributed environment consisting of various operating systems. VRPN provides the same interface regardless of the physical location (the same process, another process on the local host, a device on another host, a device on a different OS, etc.) of the device.

Generally, the most difficult thing for a virtual reality application programmer is to develop a control program for devices operating in various operating systems. However, when VRPN is used, various heterogeneous devices can be used in an application program simply by adding a very simple source code.

In order for an application to be able to use various devices, of which the control methods and the supporting operating systems are different from each other, in a consistent way, VRPN has classified each device according to its interface type. Types include Tracker, Button, Analog, and Dial. When a device provides a variety of interactive interfaces, the client program maintains a separate network connection for each interface and sends and receives data streams even though it is the same device.

If multiple interfaces are provided in a single physical device, an abstract logical device is defined at an intermediate stage and operated as though there are multiple physical devices. And even one interface provides a very flexible structure for many different uses. As a result, from the standpoint of an application program, VRPN can be used by connecting a logical device that provides the necessary interface regardless of the type of physical device to be used,

the underlying operating system, or the driver software for the device. This project shows that the same device, or even the same interface, should be aware of various usage scenarios depending on their use.

B. *Tangible User Interface (TUI) SDK*

Our preliminary study, TUI SDK [2], defined a logical device in the middle layer and applied it to games. In this study, we developed SDK and authoring tool for a TUI (Tangible User Interface) computer (see Fig. 1) composed of various input devices (camera, multi-touch panel, RFID, accelerometer, gyroscope sensor, button). We also implemented a software module and included it in our authoring tool so that contents using the conventional keyboard and mouse can be run on this computer.

In this SDK, to use the camera-based vision sensor or accelerometer in the existing contents using only the keyboard and mouse, mapping relationships are defined in a scripting language called TML (TUI Markup Language), which is based on XML, so that data or events generated by each physical sensor can be automatically transformed into a form required by the contents.

Physical devices are not only different in the events and data generated by their characteristics, but also have different control methods. For example, in the case of an accelerometer, it should be controlled by a polling method in which the data value of the sensor must be checked continuously with a constant update period. In the case of a touch sensor, a callback function is called in response to a specific event.

In this study, we developed a TUI SDK that allows users to receive events or data from devices in a consistent fashion, regardless of the characteristics of the devices. All you have to do is request the desired data type or event. To this end, the TUI SDK defined TPIS, a collection of class libraries that wrap physical devices, and defined the TLIS library that represents logical devices that can be used in a consistent way

Fig. 1: TUI Computer including various tangible user interfaces

The main role of the TUI SDK is to help define the mapping relationship between the logical device class and the physical device class, which can be dynamically redefined when the content is running (at runtime). Once dynamic redefinition is enabled, content developers do not have to terminate the running content and can instantly see the results of their modifications without having to recompile and re-run the programming code. Finally, the TUI SDK defines an XML-based scripting language that allows content developers without expertise in computer programming to utilize various multi-modal devices for their contents.

C. *3DML*

An example of an XML-based can also be found in 3DML [3]. 3DML helps you implement the interaction in 3D content using an XML-based markup language without using a programming language such as C ++. Basically, it uses data flow architecture to connect input devices, interaction methods, and feedback objects.

3DML models interactions by defining and combining blocks, where each block acts as a filter. Each block has a port (e.g., position, orientation, etc.) for receiving input and a port (e.g., selectObject) for sending output values. Blocks are used to process data from input ports and select objects to be delivered. In 3DML, various basic interaction techniques required for 3D content are defined as filters. In addition, these basic interaction filters can combine to form complex interactions.

D. *GlovePIE*

GlovePIE (Glove Programmable Input Emulator) [4] is a program that originally started as a utility for games on the PC platform. Input values from various devices such as joysticks, game pads, mice, keyboards, MIDI input devices, trackers, and VR Gloves can be used in various applications including not only computer games but also general applications such as MP3 players.

It is similar to the above TUI SDK that provided a way to use input values from various in existing applications, but GlovePIE defines an object-oriented scripting language and provides very flexible and powerful control capabilities. Although it takes a lot of time to get used to non-experts who are unfamiliar with computer programming, using a variety of arithmetic functions or equations makes it possible to express complex interactions as much as possible in a programming language such as C or C ++.

E. *UbicompBrowser*

UbicompBrowser [5] interacts with various devices in a way that extends the WWW (World Wide Web). This project has two characteristics as follows. The first suggests a way to send input data from the web to various output devices around us, including mobile devices. The second is to use Uniform Resource Identifiers (URIs) to provide consistent access and control to resources, such as TVs and light switches, as well as resources on the web.

In this study, they propose “Extended URL” (See Table I) to access various resources and devices. It can be divided into “Protocol / Media-type” and “Media Content.” For example, “tv://local/ARD” means to set the channel to “ARD” on the TV in the same location as the mobile device you are using, and “x10://local/light?low” means that it weakens the intensity of the light. This consistent approach and control scheme not only has the advantage of being able to manage

various devices and resources very efficiently, but also has the advantage of utilizing existing HTTP protocol-based resources and systems as it is when constructing future IoT environments.

Table I: Extended URLs of UbicompBrowser

URL	Protocol / Media Type	Media Content
tv://local/ARD	television	German TV channel ARD
radio://local/SWR3	radio	German radio station SWR3
x10://local/light?low	house automation control sequence	dim the light low

To realize UbicompBrowser, two important problems have to be solved. The first is a "Detection and management" problem for detecting and managing devices and resources in various places in the surrounding environment. The second is a "Matching resources" problem for finding and delivering the corresponding resources from a specific device's request.

The most important thing for "detection and management" is to constantly monitor the environment and update the directory for resources and devices to the latest. This should take different policies depending on the characteristic of each device. The policy should also be different depending on whether the device has a built-in automatic registering function or not. Another option is to register each device with a tag (e.g., RFID).

The "matching resource" problem is to find and deliver the specific resource requested by the user. UbicompBrowser uses rule-based decision-making techniques. The rules are set according to 4 conditions as the following Table II.

Table II: Decision making rules in UbicompBrowser

Rule	Description
Rule 1	The environment described in "device directory"
Rule 2	Application limitations or additional rules
Rule 3	Characteristics of media
Rule 4	User preferences and context information

F. Beach

BEACH (Basic Environment for Active Collaboration with Hypermedia) [6] proposes a software architecture for using various devices with different characteristics in ubiquitous computing environment. This is similar to the "UbicompBrowser" above, but it offers a more general and fundamental solution. More importantly, the implications of this study are very well summarized in terms of what to consider to design a middleware similar to the purpose of this study.

For example, various input / output devices distributed on the network (such as a large display device, a tabletop type PC, a device providing various multi-modal experiences and interfaces, etc.) serve as a BEACH client and are connected to the BEACH server. The most important feature of the system using BEACH is multi-user support, which is possible because it provides the ability to efficiently process concurrent events from multiple BEACH clients intertwined with the BEACH server. To do this, the event structure is layered into low-level / high-level events, and a separate dispatching strategy is supported for each event level.

G. Plan B

This study proposes "Plan B" [7], an operating system for IoT environments to be introduced in the near future. Most researchers are approaching from the perspective of middleware for pervasive environments such as IoT or ubiquitous computing, but this study is approaching from the point of view of file system which is a part of operating system.

In Plan B, all resources (including various input / output devices) are treated as virtual files in a kind of distributed network environment. This file has a mechanism similar to that of NFS (Network File System), which is widely used in unix-based operating systems. Resources representing a single I / O device are exposed on the network with their names and attributes, and from a user's point of view, the physical location of each resource (regardless of local host or remote host), the user can use it freely according to his/her own needs. This system has been in operation for many years at the Sistemas Institute of Rey Juan Carlos University in Spain.

The structure of Plan B is much simpler than expected. Thanks to this simple structure, it can be applied to various fields irrespective of characteristics and kinds of devices, and the overhead of system operation is also very little. Hosts that serve resources need only expose their externally accessible devices as network files.

Serviceable resources in Plan B include not only input / output devices such as a keyboard and a mouse, but also widgets that are components of a graphical user interface (GUI). This is similar to the server-client architecture of the X-Window system. Various UI widgets that are served from a remote host are freely available on the local host. For example, it is possible to use a UI widget with a physical device, such as a sound device, from a remote host.

H. Smart Baton

The Smart Baton [8] system is a remote-control system for home appliances and office appliances. The basic configuration of this system uses a PDA combined with a laser pointer for remote control, and each home appliance and office machine includes a laser receiver and a network function.

The feature of this system is that it allows users to select various devices easily and clearly, provides a user interface suitable for each device through PDA, supports not only multiple users, but also security functions. First of all, it is unique that it used the idea of a laser pointer to easily and clearly select the desired device. The laser pointer communicates with the laser receiver of each device to identify the device. Once the identification is made, the subsequent communication is performed through the network in consideration of security, and through the user interface provided on the LCD screen of the PDA, a very abundant application and control freedom is given unlike the other remote control systems.

I. Ubiquitous Chip

In the ubiquitous computing environment of the future, the environment of our daily lives can be regarded as a network of computers connected to each other. It is clear that various

computers that are very different in characteristics or purpose from each other, which are different from the PCs or mobile devices we use today, are connected to each other in a complex way.

This study introduces a ubiquitous chip [9], which is a kind of device for I / O control developed with this environment in mind. In this study, ubiquitous computing environment has three requirements as follows (See Table III). The ubiquitous chip is designed to meet these requirements.

Table III: Requirements of ubiquitous computing environment

Req.	Description
Autonomy	It must be processed and operated by itself without human intervention or manipulation.
Flexibility	It should be flexible enough to be applied to various needs.
Organic Cooperation	Multiple computers must be able to organically combine to perform complex tasks.

One chip may be used independently, and multiple chips may be combined organically to perform complex functions. Each chip can define its own rules according to its purpose, so it can perform its own tasks under the rule base without human intervention (autonomy), can freely define new rules (flexibility). In addition, many chips can have a structure that operate organically (organic cooperation), so all of the above three requirements are met.

The rules in this system have an Event-Condition-Action (ECA) structure. When a certain event satisfies a predefined condition, it takes its predefined action. To enable this rule-based control, each chip has a built-in microprocessor (PIC16F873) and a lithium-ion battery, with connectors for connecting various external devices or additional chips.

J. Distributed User Interface Elements

In recent years, the price of display devices using LCDs and LEDs has fallen, and flat-type display devices have become very popular output devices. In addition, small and lightweight mobile devices such as Tablet PCs developed from small laptops are being used in many parts of our lives. Moreover, in a future, these flat-type display devices and mobile devices will all be connected to the network and used together organically.

In [10], they propose an efficient user interface in the environment mentioned above. There are two typical features in this study. The first is that the user interface is transparently distributed to the various display devices connected through a network. This means that the components of GUI can be freely distributed regardless of the type of the platform or the operating system of the display device. The second feature is that a user interface for a specific application can be collaborated by sharing with a large number of users. Each user can freely collaborate with other users using the user interface of the his/her own display device.

For example, the user interface provided through the original web browser can be distributed to three different devices (e.g., a smartphone, a tablet PC, and a notebook PC) depending on the service purpose. First, the smartphone can be used exclusively for the zooming user interface. Second, the tablet PC can be used for scrolling only. Finally, the notebook PC can display a user interface for browsing information.

K. NOTOS

An example of a "distributed user interface" technique similar to the one in the previous section used for actual medical use can be found in NOTOS [11]. NOTOS suggests a structure that allows the user interface to be distributed to dynamically available devices. In order to determine a user interface adaptable to a specific device, the following three steps are performed.

1) ***Stage 1****: The "Peer-Device Discovery" module determines the available devices and the interaction components of each device.*
2) ***Stage 2****: The "Rule Engine" determines how to distribute user interface components by considering the characteristics and limitations of each device.*
3) ***Stage 3****: The "DUI Engine" transmits the user interface component to the device through the network.*

When the user interface is distributed in such a manner, in order to receive input from the user rather than merely displaying information, the event generated from each device must be efficiently transmitted to the main process of the application program. In the NOTOS, the "DUI Engine" plays this role. In a process that continuously inspects a specific event, it let the DUI engine know the information about an event desired by the user in advance and requests to notify when the event occurs. Then, the DUI engine immediately delivers the event generated in each dispersed user interface to the promised "Event-Listener." For example, by distributing a graphical user interface for a movie player running on a PC to a mobile device, a user can remotely control the movie.

L. SUPPLE

In the techniques to distribute graphical user interfaces to devices connected to a network, as in [10] or [11] above, one of the important problems is to transform each graphical user interface component to suit the characteristics (screen resolution, size, physical user interface, etc.) of the device. In [10] and [11], components have been implemented in advance in consideration of the characteristics of each device, but it is expected to be difficult to apply them to various practical applications. The reason is that, as is always the case in the near future, the more kinds of devices become available, so it will take a lot of time and cost to manually create all the components for every device.

SUPPLE [12] introduces a technique for automatically generating user interfaces to solve the above problem. In this study, they looked at the problem of creating the most suitable user interface as a "Decision-theoretic optimization problem," and aimed at minimizing the efforts of the user to manipulate the user interface components to be rendered on the actual device. In addition to the effort to use each component, they defined the following three variables as characteristics of each component.

Table IV: Variables for calculating the properties of each UI component

Variable	Description
Functional specification of user interface	The type of data that will actually be exchanged between the user and the application
Device model	User interface components available on each device
User model	Activity performed by the user on each device

M. TransCom

In the environment where low-cost mobile devices are widely used as in the present, the biggest problem is that it is difficult to organically interwork with each other due to too many different types of devices and operating systems. If the same operating system is used, it is not a problem. However, when the operating system is changed, the network protocol to be used is changed. Therefore, the reusability and utilization of data dependent thereon are also reduced. For example, documents or data created on an iPhone using iOS will have no meaning on mobile devices using Microsoft's Windows Mobile phones. A bigger problem is that every device has its own applications and data representations, which leads to a waste of resources (data storage, processors, etc.).

TransCom [13] is a pilot system to solve the above problem. This system uses a kind of "thin client" technique, which is designed so that client devices can use various remote operating systems, applications, and data regardless of operating system or network protocol type. Once the operating system on the remote server is received over the network and booted, you can use the desired application and data on the remote server as well. Of course, compared to client devices using proprietary operating systems, the overhead of network transmission is large. However, in a near future, network technology that supports transmission speeds of several tens to hundreds of times will be realized, so it is not expected to be a big problem. In addition, as noted in [13], testing using the pilot system took about 48 seconds to boot the operating system, and the Microsoft Word 2003 took only 1.26 seconds to start.

III. DISCUSSION

As you saw in Chapter 2, we can see that many efforts are being made to efficiently use and manage various kinds of devices. Among them, there was an effort to efficiently manage many devices connected to the network, and there was a case where a software development kit (SDK) was developed to provide a convenient development environment for developers. Also, there have been many researches focusing on interfaces with users, such as research on generation and conversion of GUI components for various devices.

However, in this age of rapid change of IT technologies and paradigms, such approaches are reaching their limits. Most researches are limited to specific categories of devices, and a middleware. Therefore, I think it is necessary to study I / O structure that supports various modality based on the wider types of devices, but does not depend on a specific operating system or a middleware.

REFERENCES

[1] Taylor II, Russell M., et al. "VRPN: a device-independent, network-transparent VR peripheral system," Proceedings of the ACM symposium on Virtual reality software and technology. ACM, 2001.
[2] J. Seo, et al., "Implementation of an Authoring Tool for Tangible User Interface," Journal of Digital Contents, 8(7), pp. 9-16, 2008.
[3] P. Figueroa, M. Green, and H. J. Hoover, "3DML: A Language for 3D Interaction Techniques Specification," Eurographics, 2001.
[4] C. Kenner, "GlovePIE," URL: http://glovepie. org/glovepie. php [last accessed 2013-02-04] (2007).
[5] M. Beigl, A. Schmidt, M. Lauff, and H. W. Gellersen, "the UbicompBrowser," Proceedings of the 4th ERCIM Workshop on User Interface for All, 1998.
[6] P. Tandler, "Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices," Proceedings of UbiComp 201: Ubiquitous Computing, LNCS 2001, pp. 96-115, 2001.
[7] F. J. Ballesteros, G. Guardiola, K. Leal, E. Soriano, "Plan B: An Operating System for Ubiquitous Computing Environments," IEEE Pervasive Computing, 2006.
[8] A. Saito, M. Minami, Y. Kawahara, H. Morikawa, and T. Aoyama, "SmartBaton Systems: a universal remote control system in ubiquitous computing environment," International Conference on Consumer Electronics, pp. 308-309, 2003.
[9] T. Terada, M. Tsukamoto, K. Hayakawa, T. Yoshihisa, Y. Kishino, A. Kashitani, and S. Nishio, "Ubiquitous Chip: A Rule-Based I/O Control Device for Ubiquitous Computing," LNCS 3001, pp. 238-253, 2004.
[10] K. Luyten and K. Coninx, "Distributed User Interface Elements to support Smart Interaction Space," Proceedings of the Seventh IEEE International Symposium on Multimedia, 2005.
[11] M. Bang, A. Larsson, E. Berglund, and H. Eriksson, "Distributed user interfaces for clinical ubiquitous computing applications," International Journal of Medical Informatics, 545-551, 2005.
[12] K. Gajos, and D. S. Weld, "SUPPLE: Automatically Generating User Interface,"In Proceedings of IUI'04, 2004.
[13] Y. Zhang, and Y. Zhou, "Transparent Computing: A New Paradigm for Pervasive Computing," LNCS 4159, pp. 1-11, 2006.

Jinseok Seo received the M.S. and Ph.D. degrees in Computer Science and Engineering from Postech, Korea, in 2000 and 2005, respectively. Since 2005, he joined the division of digital contents technology, Dong-eui University, Busan, Korea. His main research interests are artificial intelligence for computer games, game engines, virtual reality, and augmented reality.