Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 02, February-2017

A Mediating Layer for Heterogeneous I/O Devices in Virtual Environments

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

***Abstract*—In this paper, we propose Hyper I / O, a middleware that helps users to use a wide range of interactive devices freely in virtual reality applications. First, we derive an ontology structure that can accommodate various I / O devices and interaction techniques, and establish a 2-layer model between interaction and logical device. In this structure, N:N mapping between logical devices and physical devices is possible, and highly flexible I / O programming becomes possible. In addition, the specification of Hyper I / O devices as well as a protocol is defined. To do this, we derive the taxonomy of Hyper devices and developed the protocol using an XML based markup language.**

Keywords— Virtual Reality; Interaction; Multi-Modal; Middleware

I. INTRODUCTION

With the development of IT technologies, there are now many computing resources permeated throughout our daily lives. For example, low power processors and memory technologies have led to the deployment of high-performance processors and high-capacity memories in consumer and office appliances, which had previously been purely functional. In addition, recently, mobile devices such as smartphones and table PCs that cross the border between mobile devices and desktop PCs have higher performance and storage space than desktop PCs just a few years ago.

The permeating of computing resources is not limited to physical hardware devices alone. Device developers and telecommunication companies are working to provide new content and services suitable for each device or terminal in order to generate more demand for devices and additional revenue. In the case of a mobile device, this phenomenon can be confirmed in cases of services such as browsing the web, controlling a remote PC, and mobile banking in very small devices including smartphones and smart watches. Even in the case of home appliances, following the trend of convergence, a refrigerator equipped with the internet connection or an IPTV that provides functions such as interactive multimedia service, home banking, and home shopping.

In addition to rapid evolutions and changes in hardware devices, services, and contents, many changes have been made to user interfaces by utilizing various sensor technologies in recent years. In the case of TV, instead of the conventional remote control buttons, it is possible to adjust the desired channel and volume with only human gestures or voices. In the entertainment field, various multimodal interaction based interfaces are being introduced.

While the various devices, sensors, services, and contents are permeated around us, not all of them work together organically. Most device manufacturers and services/content providers use their own interfaces and I/O devices for their own benefit. So, although the amount of computing resources is very large, utilization is not high. In addition, since the conventional I/O method has a fundamental problem that it relies on the service provided by an operating system and the device driver provided by the device manufacturer, there is a limit to the operating system and application programs that have to use various external devices.

To use a device with the conventional I/O, application developers must use the device driver directly or use the system functions provided by an operating system (See Fig. 1). System functions actually the device driver provided by a manufacturer, so the application must rely on the device driver in whatever way. However, since the device driver operates only on the operating system supported by the manufacturer, it is difficult to use it in a mobile device or a home appliance operating on various operating systems.

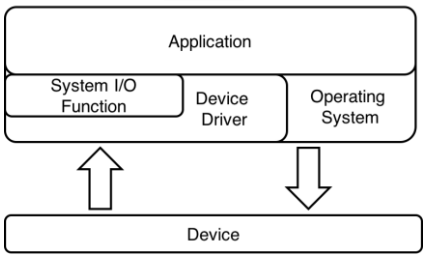

Fig. 1. Conventional method for I / O with devices

Even if there is no limitation of the operating system in the conventional I / O method, there is a problem that the device must be physically connected to the host hardware in which the device driver is installed. In recent years, USB is widely used as a physical connection method, but there are many other devices that use different methods such as serial RS232C, IEEE 1394, Bluetooth, wireless LAN, and infrared rays. Desktop PCs basically support a variety of physical connection methods. However, since mobile devices and home appliances often support only specific connection methods, we can not use a device without hardware support to convert the physical interface scheme

Many researchers have made great efforts to solve the above problems. In particular, we proposed an idea to integrate various devices with different platforms or characteristics, focusing on HCI (Human-Computer Interface)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 02, February-2017

[1]. However, most studies did not cover a wide range of devices and operating platforms, and tried to solve problems from a high-level perspective, such as users or applications. In this study, we defined a Hyper I/O technology at middleware level, which is a more fundamental approach to overcome the limitations of existing solutions, considering the speed of technology development that changes day by day.

II. RELATED WORKS

It is VRPN [2] 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 a remote host, a device on a different OS, etc.) of the device.

Our preliminary study, TUI SDK [1], 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 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 (See Fig. 2).

Fig. 2. TUI computer including various tangible user interfaces

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.

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.

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

BEACH (Basic Environment for Active Collaboration with Hypermedia) [6] project 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.

Plan B [7] is 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.

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.

TABLE II. REQUIREMENTS OF UBIQUITOUS COMPUTING ENVIRONMENTS

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.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 02, February-2017

Ubiquitous chip [9] is an I/O controller designed considering future ubiquitous. In this study, ubiquitous computing environment has three requirements as autonomy, flexibility, and organic cooperation. Ubiquitous chip is designed to meet these requirements (See Table II).

In [10], they propose an efficient user interface in the environment with various heterogeneous devices. 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.

An example of a "distributed user interface" technique similar to the one in the previous 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.

SUPPLE [12] introduces a technique for automatically generating user interfaces. 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 (See Table III).

TABLE III. 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

TransCom [13] is a pilot system using a different approach from the above studies. 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. DEFINITION OF HYPER I/O

The main purpose of this study is to define a Hyper I / O technology, a middleware for various heterogeneous devices, to solve the problems mentioned in Chapter 1 above. Hyper I / O is a middleware designed to freely use various heterogeneous devices regardless of operating systems and application types, as well as interface methods supported by host devices. In order to process I/O with a device in an application program, it is necessary to rely solely on the operating system and the device driver as shown in Fig. 1. However, when using Hyper I / O middleware (see Fig. 3), we can freely use the device regardless of the host operating system and the device driver.

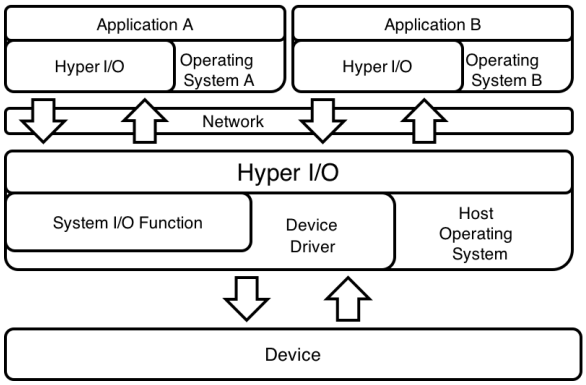

Fig. 3. Role of Hyper I / O middleware

In the past, the middleware approach described above has been widely used as an intermediary medium between servers and clients, but recently, a high-speed wireless LAN has been generalized in addition to a mobile platform having a small and powerful processor and memory, operating Hyper I/O middleware on each host platform does not incur large overhead. In addition, recently, not only mobile platforms such as smartphones and table PCs, but also home appliances such as TVs, set-top boxes for IPTV, and refrigerators are equipped with network functions such as LANs and wireless LANs.

IV. IMPLEMENTING HYPER I/O

In Fig. 2, Hyper I / O middleware is physically divided into two parts. One is the system on which an application is running and the other is the host on which a physical device is connected. From the point of view of implementation, these two separate layers can be named "Hyper IO Device Driver" and "Hyper I / O Mapper", respectively.

A. Hyper I/O Device Driver

Hyper I / O on the system where an application is running acts like a generic device driver. Application developers can use a variety of devices supported by Hyper I / O, which represent abstractly defined devices.

Prior to implementing a Hyper I / O Device Driver, we must first analyze and classify the interactions required by applications. At this point, rather than considering the limitations and characteristics of physical devices, the focus should be on the nature of the interaction that will be provided to users. Once the required interactions are classified, we can

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 02, February-2017

define an abstract device "Hyper Device" and implement a Hyper I/O Device Driver.

B. *Hyper I/O Mapper*

The Hyper I / O Mapper on a host where a device is connected through a physical interface is responsible for converting the I/O data from the physical device into the I/O data from the abstract device, Hyper Device. For application developers and users, the presence of this Hyper I/O Mapper is not critical, but it is a necessary part of running a physical device on the Hyper I/O middleware.

To implement the Hyper I/O Mapper, the characteristics of each physical device must be analyzed. The analyzed characteristics indicate how each physical device can be used. Most physical devices can be mapped to multiple Hyper Devices. In case of a specific Hyper Device, a plurality of physical devices can be combined to provide a desired I/O function (See Fig. 4).

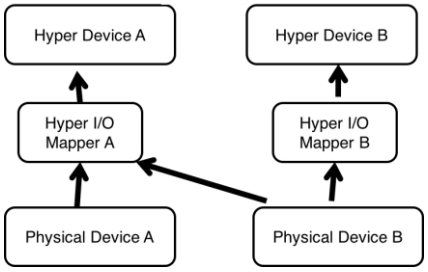

Fig. 4. Role of Hyper I / O Mapper

The Hyper I / O Mapper can be seen as a kind of server on the host where a device is physically connected. As with the Hyper Device Driver, we can use TCP/IP protocols, but if we use HTTP / HTTPS, the Hyper I / O Mapper will be part of a Web server.

V. STRUCTURAL DESIGN OF HYPER I/O

A. *Ontology structure of Hyper I / O*

The proposed Hyper I/O is designed to focus on two purposes as follows. The first is independent of operating systems and hardware platforms, and the second is a wide range of heterogeneous devices support. In order to meet these two purposes, we constructed two layers of "Interaction layer" and "Hyper Device layer" as shown in Fig. 5.

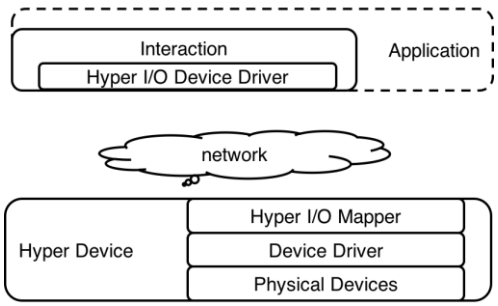

Fig. 5. Two layers of Hyper I / O

In Fig. 5, "Interaction", "Hyper Device," and "Network" do not specifically refer to the physical object, but to an abstract concept. For example, two layers may reside physically on the same host or may be on different physical devices connected to the network. Actually, even a physically identical device can act as an interaction layer or a Hyper Device layer.

B. *Hyper I/O Device*

We use a mobile device or an IT device to communicate with the application. The physical user interface provided to the user in these devices is ultimately used to interact with the application. Based on this, we have focused on the necessary interactions from the application's standpoint when devising Hyper I/O Devices in this study.

In order to provide maximum freedom and flexibility to application developers, a broad review of various interactions must be preceded. As a similar case study, we could find a systematic classification of the interaction techniques used in the virtual reality system. In [14], they systematically classified various interaction techniques and evaluated the performance of each technique. Although this study has been motivated to classify Hyper I / O devices, it is difficult to apply it to this study because the category of application program is limited to 3D virtual reality systems.

VI. EXAMPLES OF USING HYPER I / O MIDDLEWARE

This Chapter shows how physical devices are used as Hyper Devices using Hyper I / O middleware. Since it is beyond the scope of this study to develop a working prototype or middleware in advance, the contents of this Chapter are for illustrative purposes only, assuming that real middleware is implemented.

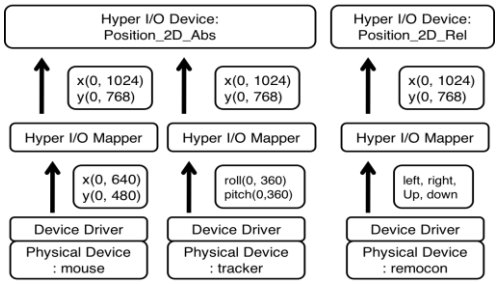

Fig. 6. An example of using middleware for Hyper Devices

Fig. 6 shows an example of 2 Hyper Devices ("Position_2D_Abs" and "Position_2D_Rel") interaction. Position_2D_Abs and Position_2D_Rel are for moving the pointer in a 2D-based GUI environment. The resolution of the screen (range of pointer movement) is 0 to 1024 on the x axis and 0 to 768 on the y axis. In this example, all three physical devices are used, demonstrating that the same interaction technique can be implemented with any device. The following is a description of each physical device.

A. *Mouse*

It is a typical physical device used to move pointers in a 2D-based GUI environment. However, in this example, we can see that the range of movement is different from each other, because the resolution of the device on which the actual GUI is displayed and the device using the mouse may be different. For example, a display device is an HDTV, and a device to which a mouse is connected is a PC.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 6 Issue 02, February-2017

B. Tracker

A tracker is a typical device used mainly in 3D virtual reality systems. Depending on the type, it is possible to measure the direction of 3 axes in the case of the low-cost type, and a tracker in the high-end class can measure not only in the direction of the 3 axes, but also in a 3D position. In this example, we are using a tracker capable of measuring the direction of 2 axes, one of which is the roll and the other is the pitch. Tracker-like devices include optical-based trackers using infrared cameras, camera-based computer vision commonly used in augmented reality, and gyroscope sensors.

```
<?xml version="1.0" ?>
- <hioml>
  - <hiodev name="abc">
    - <pos2D type="abs">
      - <x_range min="0" max="1024">
        - <mouse>
            <x_range min="0" max="640" />
          </mouse>
        - <tracker>
            <roll_range min="0" max="360" />
          </tracker>
        </x_range>
      - <y_range min="0" max="768">
        - <mouse>
            <y_range min="0" max="480" />
          </mouse>
        - <tracker>
            <pitch_range min="0" max="360" />
          </tracker>
        </y_range>
      </pos2D>
    - <pos2D type="rel">
      - <x_range min="0" max="1024">
        - <remocon>
            <dec_button id="left" unit="5" />
            <inc_button id="right" unit="5" />
          </remocon>
        </x_range>
      - <y_range min="0" max="768">
        - <remocon>
            <dec_button id="up" unit="5" />
            <inc_button id="down" unit="5" />
          </remocon>
        </y_range>
      </pos2D>
    </hiodev>
  </hioml>
```

Fig. 7. The XML that defines the Hyper Device in Figure 6

C. Remote Controller

It is a case where a pointer is moved by using a remote controller, as many cases are already used in HTPCs (Home Theater PC), HDTVs, and IPTVs. Recently, there is a remote controller including an analog stick, but in this example, the pointer is moved using a button. This example shows moving the pointer by pressing the arrow buttons on the remote controller. Looking at the XML document in Fig. 7, we can see that the moving unit is specified as "5" pixels.

The Hyper Device for the Position_2D_Abs and Position_2D_Rel interactions discussed so far can be defined as an XML-based markup language as shown in Fig. 7. This type of markup language has advantages not only in various operating systems and platforms but also in high-level network protocols and services such as HTTP.

VII. CONCLUSION

In this study, we defined the Hyper Device and created the application cases. As a result, it can be confirmed that there is no need to match the operating system platform for running an application program and the platform for driving a device. Application developers and users can use a variety of devices with an emphasis on the interaction itself, regardless of the operating system or platform type on which the device is running. In addition, defining well-defined abstract devices, such as Hyper Device and Hyper I / O Mapper, can overcome the physical limitations of devices and reduce the cost of developing application programs capable of various types of multimodal interactions.

In the future, it is necessary to examine the results of this study more closely and to confirm the clarity of the Hyper I / O technology and the possibility of implementation through prototyping. Then, if the standardized HCI framework, middleware ontologies, and protocols are expanded and developed, it is expected that it will create many economic added value in the upcoming IoT environment.

REFERENCES

[1] J. Seo, et al., "Implementation of an Authoring Tool for Tangible User Interface," Journal of Digital Contents, 8(7), pp. 9-16, 2008.

[2] 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.

[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.

[14] D. A. Bowman, D. B. Johnson, and L. F. Hodges, "Testbed Evaluation of Virtual Environment Interaction Techniques," Presence, Vol. 10, No. 1, pp. 75-95, 2001.