

A framework to manage multimodal fusion of events for advanced interactions within Virtual Environments

Damien TOURAINE, Patrick BOURDOT, Yacine BELLIK, Laurence BOLOT

LIMSI-CNRS, University Paris XI Bât. 508, BP 133, F-91403 Orsay cedex (France).

Abstract

This paper describes the EVI3d framework, a distributed architecture developed to enhance interactions within Virtual Environments (VE). This framework manages many multi-sensorial devices such as trackers, data gloves, and speech or gesture recognition systems as well as haptic devices. The structure of this architecture allows a complete dispatching of device services and their clients on as many machines as required. With the dated events provided by its time synchronization system, it becomes possible to design a specific module to manage multimodal fusion processes. To this end, we describe how the EVI3d framework manages not only low-level events but also abstract modalities. Moreover, the data flow service of the EVI3d framework solves the problem of sharing the virtual scene between modality modules.

1. Introduction

Currently available Virtual Environments (VE) are mainly driven by conventional devices such as trackers or data gloves for input and by visual and auditory feedbacks for output. Our goal is to take advantage of every kind of advanced interactions the Human-Machine Communication domain is able to provide.

To this end, we are developing the EVI3d framework, a distributed architecture to manage advanced interactions within Virtual Environments. Actually, some devices require sophisticated recognition processes while others suppose real-time feedbacks and so on. Our purpose is to export such heavy tasks on dedicated computers. Consequently, it is necessary for VE applications to be embedded in a distributed architecture over a network that allows the cooperation of several computers possibly with different Operating Systems (OS).

The base component of this architecture is the *VEserver* that already manages several standard VE devices such as 6 DOF trackers as well as more exotic devices such as speech recognition systems. The purpose of this paper is to describe how EVI3d manages multimodal fusion of events. This kind of process can be considered as a client of the *VEserver* according to the fact that it requires events provided by the available VE devices. More fundamentally, it is a generic module of the EVI3d architecture that combines

multi-sensorial input events of any VE applications, according to knowledge of their functional possibilities.

The outline of this paper is the following. First, we present a short overview of the most advanced systems in the VR field. Then, we describe the constraints of the devices we use in order to reach realistic interaction within Virtual Environments. After an overview of the *VEserver* and its place in the EVI3d architecture, we finally describe the solution used to manage fusion of multimodal events.

2. Overview of VE systems

2.1. Software architectures for VE applications

A thorough description of the main requirements and approaches for networked VE systems is presented in²³. For the management of VE applications, there are mainly two kinds of architectures: the data-oriented distributed architectures and the event-oriented systems.

A first category of VE distributed architectures uses share memory approaches to distribute very large-scale databases. It is the case of IP multicast-oriented systems such as DIS¹⁹, NPSNET²⁶ or HLA¹⁶. A second class of solutions is the supplier/consumer approach, that the previous DIS-oriented systems provide, as well as DIVE⁴ or AVIARY²⁴. However, this approach can induce higher latency, while some of these systems do not allow preemptions between event processes.

The main advantage of this method is that every nodes of the application as a full knowledge of the whole database. Thus, we don't have to care about differences that could introduce artefacts between two nodes.

Another kind of software architecture for Virtual Environments is event-oriented systems. To this end, many libraries exist to provide device drivers services to VE applications. The most popular library is Trackd from VRCO. It is included in many VE software such as CAVELib, Division dVise and WorldToolKit. However, Trackd is not an open source software. Therefore, it is not easy to introduce new kind of devices such as voice or gesture recognition processes. Then, some laboratories developed and proposed their own open source device manager. The most common are the VRjuggler system¹⁰ and the Diverse Tool Kit (DTK), this last one being a part of the DIVERSE library¹¹. Conversely to the previous method where each node knows the whole database, each node know only the interacters it requires. However, this method requires much less bandwidth and is equivalent to the previous one in the case of only one node.

None of the previous systems include interaction through haptic devices. The only system designed to manage haptics within Virtual Environments is VRPN (Virtual Reality Peripheral Network)⁹. Moreover, this system also provide a driver to connect voice recognition and timed events. However, the system is restricted to one server with several clients. Thus, there is no solution to have several servers (and timing is provided by the unique server).

Regarding data fusion, the OpenTracker software²⁰ fully implements a system to fuse several informations from different trackers through XML language. The main drawback of OpenTracker is that it is a big standalone software that manages every thing in a centralized fashion. Our purpose is to build a modular architecture so that our implementation allows an application to disconnect the Multimodal fusion system for specific applications as well as avoid the use of the immersive kernel.

Many applications use flying menus with 3D widgets to interact within Virtual Environments. This kind of interactive paradigms obliges the user to frequently shift his attention from the 3D working space to some user interface sub-spaces: virtual objects as well as augmented ones are mixed with 3D widgets that induce a very confusing working scene. According to previous works on multimodal interfaces for CAD applications¹, our point of view is to combine "intelligent" devices within the virtual world to avoid those widgets: vocal commands, gesture recognition, tracking post-processing to control navigation and so on. To this end, we developped the VEsServer², an event oriented solution which manages standard workstation windowing events of (keyboard, mouse, focus, motion, expose, ...), as well as non-standard events such as those that are delivered by

VE devices or advanced recognition processes (voice, gesture...).

2.2. Speech recognition systems

Speech is the modality most used in the communication between human beings⁶. It has many advantages, e.g. It authorizes a significant flow of information (180 to 200 words per minute), larger than keyboard input. It makes it possible to refer and reach non-visible objects: contrary to pointing gestures, it is not necessary, with speech, to see an object to be able to point it out. Speech also makes it possible to refer to it in a descriptive way by indicating attributes or particular properties of the object to identify it. For instance, a graphical object can be identified by its attributes such as color, form, position, etc. Using graphical interfaces, it is sometimes necessary to descend a whole tree structure of menus before reaching a particular command. Thanks to speech, it becomes possible to reach it directly by naming it. Moreover, the replacement of keyboard shortcuts by words facilitates the memorization of the commands, because words have their own semantic content. To speak occupies neither the hands, nor the eyes. Consequently, it is possible, thanks to speech, to communicate and act (on real or virtual objects of the environment) at the same time.

To speak with a machine is not as simple as to speak with a human being. At the present time, there are a certain number of technological constraints. For example, to obtain very good recognition results, the size of the vocabulary should be limited to a few hundreds of words. It is necessary, moreover, to give a detailed attention to the choice of this vocabulary. Too short words (figures, spelled letters), or too acoustically close words ("led", "let") can be confused. A training phase remains necessary for speaker-dependent systems. The conditions under which the training was made (type of microphone, distance between the mouth and the microphone, ambient noise⁷, etc.) influence enormously on the performances of the system. It is thus significant to preserve the same conditions during the training and using phases. Contrary to the keyboard, the mouse, or the touch screen, which present no risk of error (that does not mean that the user cannot do mistakes), the speech recognition systems are not deterministic and constitute a new source of errors (in addition to the user). Indeed, the rate of recognition never reaches 100%, even in ideal conditions. The use of syntax or of an adapted dialog model however can significantly increase the rate of recognition.

2.3. Multimodal fusion process

The principle of combining several modalities and in particular speech with gestures, considerably increases the power of expressions^{12, 13}. A typical example of the contribution of speech to improve interaction concerns the selection task. A common problem with this task is the granularity of selection. For instance, in 3D design applications^{1, 5}, there are

different selection levels: point, curve, surface, volume, etc. Thanks to speech, it becomes possible to perform different types of selection with the same pointing operation. The desired granularity level is simply indicated with a word while performing the pointing operation.

However, combining speech with gestures in the same system requires certain characteristics of speech as well as gesture recognizers and the ability to provide some specific information which is important at the application level^{15, 18, 22, 17}. In order to correctly interpret the statements produced by the user, it is necessary to handle the events produced by the devices in a sequence that corresponds to the real chronological sequence. However, the difference between the response times of the different devices can be very important. This implies that, in general, the system receives an information stream in a temporal order which is not correct and therefore can lead to a bad interpretation of the statements (Even for human beings, it can be hard to understand the meaning of a sentence when the words are mixed up). To solve this problem, it is first necessary to know the production instants of each word (beginning and end times), so it will be possible to retrieve the right chronological order of events. Then, an event must be handled only after ensuring that no other event is currently being produced by another device, because it is possible that the other event can have an earlier start time. So, we will ensure that the next event produced will have a later time production and we can be sure that events from all devices will be handled in the real chronological order. Consequently, precise timing has a great importance in multimodal interfaces since it can convey information and have effects on the interpretation process of statements. The sequence alone does not allow multimodal statements to be correctly interpreted. It is necessary to have precise information about the temporal distribution of events, in particular the beginning and end times of each word or each gesture. In this way, it will become possible to detect temporal proximity between events, and then to decide if a fusion must occur or not.

3. EVI3d framework and VEsServer

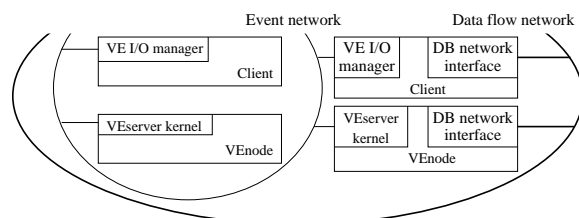


Figure 1: The general EVI3d framework.

The EVI3d framework (figure 1) is a general distributed

architecture which combines the advantages of an event approach with a data flow one. In a previous IEEE paper²⁵, we presented the data flow approach of the EVI3d architecture. This part of EVI3d allows sharing large databases between a scene generator and the computers requiring the full scene such as the graphics rendering computer. Among others, the haptic feedback computers require the full scene to be able to generate the right feedback according to the current scene.

However, in this section we describe two main components of the VEsServer to satisfy the requirements of multimodal fusion of events.

3.1. Distributed structure

As of today, many programs that manage special events coming from speech or gesture recognitions systems are platform dependent. Moreover, a voice recognition system fully uses the CPU (of a PII 450MHz) during 400 milliseconds to recognize a single vocal command. Consequently, it is necessary to dispatch the processes managing this kind of drivers on several machines over a network. The VEsServer manages the connections between those processes.

To this end, the VEsServer has a distributed structure according to several instantiations of a same class: the *VEnodes*. A *VEnode* is the software module composed of one *VEsServer kernel* and managing one or several devices of a single machine. To provide complete and correct information to the VEsServer clients, each *VEnode* has to know at any moment all the services that the other *VEnodes* are supposed to deliver. It is the goal of the VEsServer kernel of each *VEnode* to distribute this service information over the other *VEnodes*. On the other hand, each client of the VEsServer includes a special component that manages the connection of the client to its own *VEnode*. This component, called *VE I/O manager*, also allows the client to send messages to any device or client managed or connected to the VEsServer.

For each client, the VEsServer is a single software object represented by the *VEnode* the client is connected to. This is possible with the VEsServer kernel and the *VE I/O manager* that mask the connections and all the needs of communication between the VEsServer and its clients.

3.2. Driver manager

The VEsServer was designed to perform all the tasks inherent to a device manager (i.e.: it must be able to load, launch, kill and change the configuration of any driver). Due to the problem of asynchronous events, all the drivers have to run as different concurrent tasks. To avoid the problem of memory sharing, threads were preferred to processes in our implementation. Thus, each driver has its own detached thread within a same process, while the main thread of this process is dedicated to the VEsServer kernel.

We were also looking for the ability to start drivers while

the server is running. This is done by using dynamic shared objects via the `dlopen()` C system function. A nice side-effect is that the VEs server allows us to develop and test new drivers without having to recompile or to restart it. Thus a driver of a device is a shared object, dynamically loaded by the VEs server and designed to be launched as a separate thread.

3.3. Timing events

The section 2.3 shows the necessity of timing the events uniquely in order to process them adequately.

However, the distributed structure of the VEs server supposes to provide solutions to have a homogeneous timing of events. Furthermore, internal timers of each computer are not synchronized together and have specific clock deviations. Therefore, the VEs server includes a *time synchronization system*. This system provides a clock whose accuracy is about 1 ms over a 10 mb/s Ethernet network. This precision is generally enough, because our concern is the “interaction time”: under 1ms, users cannot distinguish between two events.

The aim of the time synchronization system is to get a universal time from one master. According to the localization of the time synchronization system within the VEs server kernel, each event is straight dated by the VEs node of its driver, while each message of a client is timed during its travel through the first VEs node.

The chosen paradigm is the master/slave one. Our timing system is quite similar to the Network Time Protocol (NTP) system¹⁴. However, conversely to NTP that provide an absolute time all around the real world, we only require a relative time between our servers. Moreover, we want to map the hierarchical structure of the different nodes on the distributed timing system to merge the re-election of masters in the case of failure.

3.4. Data flow services

Some applications, such as scientific simulations are working on large and/or dynamical data. Those data are generally processed by a supercomputer, while a dedicated machine is used for graphics visualizations. A standard network cannot provide enough bandwidth to transfer the millions of objects required by these applications. For instance, it is typically the case for Fluid Mechanics applications such as the one we develop at the LIMSI-CNRS laboratory (figure 2), where iso-surfaces are standalone objects composed of millions of triangles (figure 3).

In the spirit of the MPEG-2 data file format, the EVI3d framework architecture provides two kinds of data formats: those fully computed from scratch, the other ones produced from the first one by a delta of modifications. Consequently, our distributed architecture manages two channels:

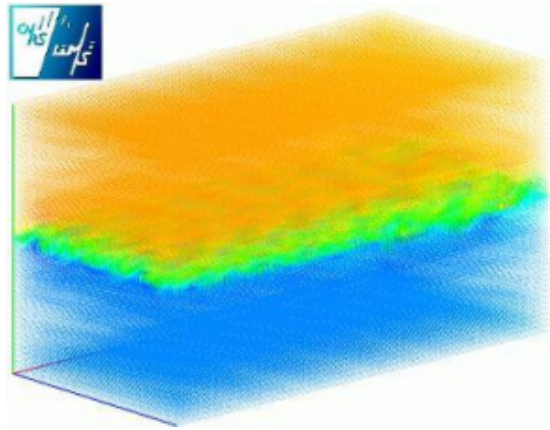


Figure 2: One of the 50 frames of the 3D database of a compressible turbulent mixing visualized within our Virtual Environment.

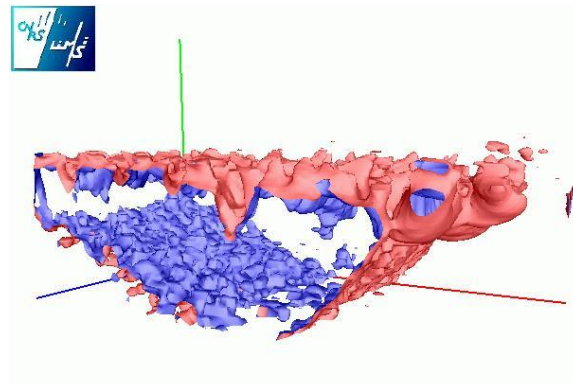


Figure 3: During the simulation of a compressible turbulent mixing, example of one frame of a dynamic iso-surface.

- **The event network** (for instance, a 10 mb/s Ethernet). This channel is completely controlled by the VEs server kernel and the VE I/O manager components. The data traveling through it are structured as events that only update the current state of a database.
- **A data flow network** (typically a gigabit Ethernet connection). This channel must be seen as a stream. The data concerned are heavy ones, such as large 3D databases, movies or sound signals. Those data can be considered as initial data or key frames needed to regenerate completely the scene. This channel is generally independent from the VEs server but requires it, for instance in order to initialize the protocol.

The general EVI3d framework for a VE application is shown figure 1. Each of the four boxes of this figure is designed to reside on separate machines. If only some modules of the EVI3d architecture are connected to the data flow net-

work, all modules are linked together via the VEs server kernel and the VE I/O manager components. In other words, the VEs server is the backbone of the EVI3d architecture.

4. Multimodal fusion of events

This section explains how some advanced interactions are designed within the EVI3d framework. First, we describe the management of modalities from low-level events. According to these considerations, it becomes possible to define the functional structure of a multimodal fusion process within our architecture.

4.1. Pre-processing of modalities

As we will see in the following sub-section, the multimodal fusion process is independent from the application. Consequently, the events provided to it must not only be correctly dated, but also have to carry high-level information. As explained in section 3.3, each event is already timed when it travels through the VEs server kernel of the device driver. To be semantically augmented, these events are preprocessed by *interpreters* depending on each modality.

The function of the interpreters is to translate the low level events issued by the devices (2D co-ordinates for the touch screen, character strings for the voice recognition system...) into higher level information. When events have been interpreted and dated, the resulting higher level information is sent to the multimodal fusion system. An interpreter is thus associated to each input modality. It uses its own *language model* and specific *domain information* to achieve the transformation of events:

- **Language models are static.** They contain a set of invariant (fixed) semantic information necessary for interpretation operation. For example, for the data glove, it will be a model of gestural language composed of the semantics of each gesture of the language. For the voice recognition system, it will be a spoken language model containing the meaning of each word or sentence.
- **Domain information is dynamic.** It represents the environment description of the application. This environment evolves in time following the evolution of the task. It contains a description of each “object” of the application. The dynamicity of the domain information requires its update every time the database is modified. To allow such update, the interpreters must be connected to the data flow services.

However, other modules can be clients of the low-level driver management. It is typically the case of “lexical” feedbacks such as the visualization of hand avatars. But more fundamentally, some modalities require a recognition step between the driver management and the interpreter module. Thus, a generic modality preprocessing is composed of the driver itself, the recognition process and the interpreter.

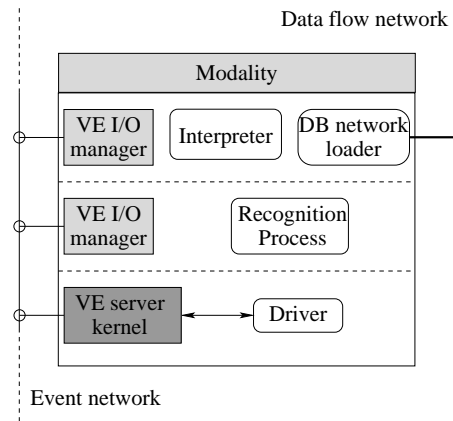


Figure 4: Preprocessing of events for a generic modality before the multimodal fusion.

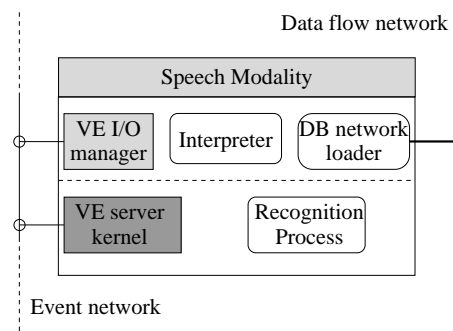


Figure 5: Preprocessing of vocal modality.

The figure 4 shows the whole preprocessing of events before sending them to the multimodal fusion system. An important point on this figure is the dashed lines that illustrate the ability of each component of this process to reside on different machines.

Please note that some of these components can be merged or splitted according to specific modality cases:

- **Speech recognition.** The low-level information provided by the driver is a heavy flow. Moreover, for some systems, we cannot manage the voice signal directly. In this case, the driver layer and the recognition process are combined to provide only the recognized words. The figure 5 shows the instantiation of the preprocessing for this modality.
- **Gesture recognition.** From another point of view, some modalities such as gesture recognition can require more than one device. The figure 6 shows that for gesture recognition there are two informations for a single hand: the data glove (hand configuration) and its 6 DOF tracking (hand global position). Furthermore, to make full recognition of the two hands, we need four devices.

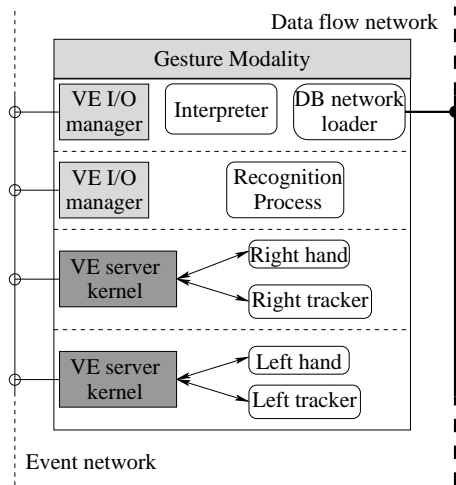


Figure 6: Preprocessing of gesture modality.

4.2. Multimodal fusion system

The multimodal fusion system is composed of two main parts: a *waiting queue* and a *fusion manager*.

4.2.1. The waiting queue

The waiting queue receives information from the modality interpreters. This information is classified in three categories corresponding to the three basic operations of interaction, following the user interaction model: $\langle \text{commands}, \text{arguments}, \text{values} \rangle$. It is sorted by the starting time of production of each information.

4.2.2. The fusion manager

The fusion manager is the heart of the multimodal process. Its activity consists of analyzing information contained in the waiting queue with the aim to build the original user command and to execute it. First, the command is identified in the waiting queue, then the *application model* (figure 7) gives the description of this command. After analyzing other informations in the waiting queue, the manager assigns values to arguments. When all arguments are valued, the manager starts the command execution.

The fusion manager handles the pieces of information one by one when they arrive in the waiting queue. To perform that process the fusion manager uses a set of rules defined from the analysis of the different possible configurations that can appear in the waiting queue. The result of the processing depends on the type of the current information, the state of precedent information and *matching conditions*.

The main problem to be treated by the manager is to assign right values to the corresponding arguments. The matching conditions are driven by three constraints: the *logic*

complementarity, the *type compatibility* and the *time proximity*.

- **Logic complementarity.** To match 2 informations there must be a logic complementarity between them. For instance, it is not possible to match 2 values or one value and one command. However, it is possible to match one value and one argument.
- **Type compatibility.** To match an argument and a value, the type of the value must be compatible with the argument type.
- **Temporal proximity.** This condition means that events corresponding to an input value and to an argument reference must be produced closely enough in time. In that case, we can affirm that the input value must be matched with the referenced argument.

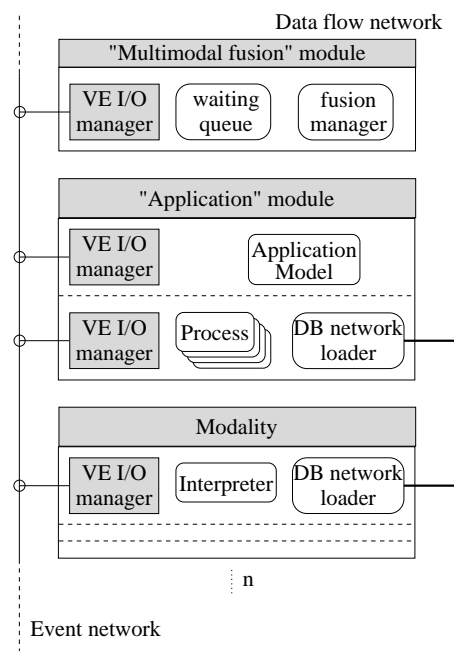


Figure 7: The multimodal system in the EVI3d architecture.

The figure 7 shows the integration of the whole multimodal fusion process in the EVI3d architecture. Each box may be seen as distinct process. Moreover, these different boxes can reside on different machines.

5. Current instantiation

This section will describe an example application using the EVI3D framework. Moreover, we will describe the benchmarks we have performed to assess the performances of our architecture.

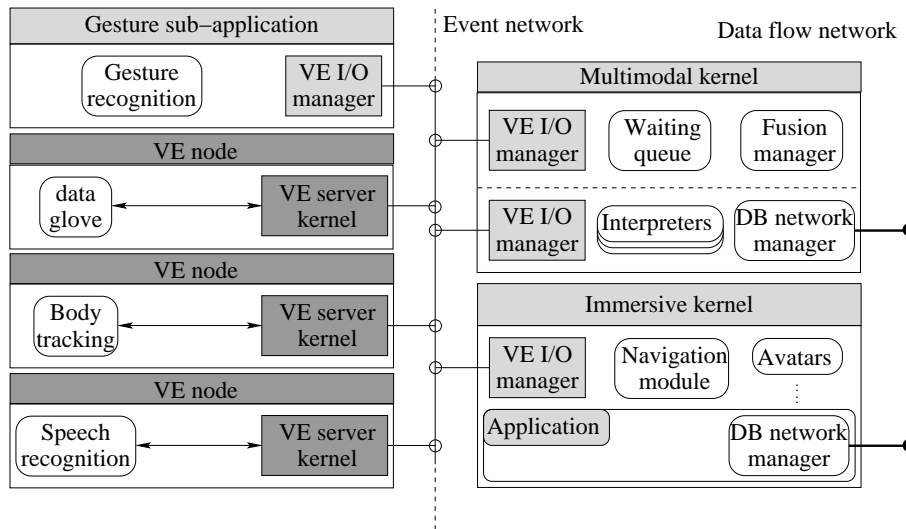


Figure 8: Current instantiation of the EVI3d framework.

5.1. EVI3d showcase: Nautilus

The figure 8 shows the current instantiation of the EVI3d framework on our demonstration application: Nautilus. This application is a 3D scene interactor that includes every interactive paradigm we have developed.

This application uses different devices: the body tracking includes a head tracking that is used by our free hand navigation module³. The data glove signal is send through the VEs server to a gesture recognition system. This system is based on a Rubine approach²¹.

The entire signals are sent to the immersive kernel for visual feedbacks (rendering of avatars of the interactors). Moreover they are also sent to the multimodal kernel.

As seen in section 4.1, the interpreters require the whole scene. To avoid the use of one data flow connection per modality, we merge all the interpreters in a single application that also includes the waiting queue and the fusion manager. This application concentrate the whole database as well as the scene management required by the detection collision to find the intersection between a tracker and the objects.

For example, the tracker of a laser beam can generate several different informations: the pointed direction of the beam and one information per object that the laser intersects. The system can fuse the high level event provided by this interpreter with a speech event that asks to “move” the pointed object to the position described by another position pointed by the same tracker immediately after. From another point of view, this task can be done by the designation gesture that our gesture recognition system “knows”: the user points a direction, the system recognizes the designation gesture and sends a message that also contains the orientation to the mul-

timodal kernel. The interpreter process this information like the previous one: one direction or several designations of objects.

5.2. Time synchronization precision

A first benchmark used three computers (see figure 9): a client under Linux was directly connected to a server under Linux (both distributed on different computers). This server was also connected to a master VEnode under IRIX.

The protocol of this benchmark is to use the keyboard driver for the slave and the master servers. We just do a keyboard interaction on the client computer and compare the time between the two events produced by the drivers respectively managed by the master and the slave servers.

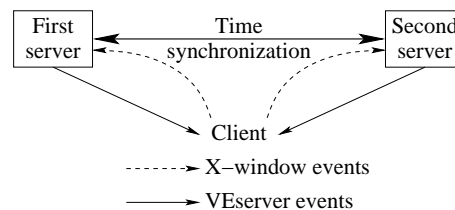


Figure 9: Benchmark for time latency measurement.

The main result (done on 100 iterations) is a latency of $(0.9 \pm 0.1)ms$, that matches our requirements for multimodal fusion of events.

However, the keyboard driver is based on the X connection between an X-window client (the keyboard driver) and the X-window server (the VEs server client computer). Thus,

due to latency of the X-window protocol and the different path from the VEserver client to each VEnode (switch, hub...) this protocol has to be improved.

We plan to implement another benchmark with a special RS-232 device that split specific information on two different ports simultaneously. Then, we will have the same signal on two computers and will be able to make better tests on time latency.

5.3. Latency between a server and a client

This second benchmark (figure 10) used three computers as well: three VEnodes and one VEserver client. However, in this protocol a single path connects each VEnode. The furthest VEnode resides on the same computer as the VEclient, and the messages travel through two remote servers before reaching the client. On the other hand, the VEnode that is on the same computer as the VEserver client as a special driver that simulates events with different frequencies. In this protocol, events are not timed through the time synchronization system, but through the `gettimeofday` function that provides the local time. Then, the events go through the two other VEnodes before arriving to the VEserver client. This client also gets the local time through the same function and compares the two dates.

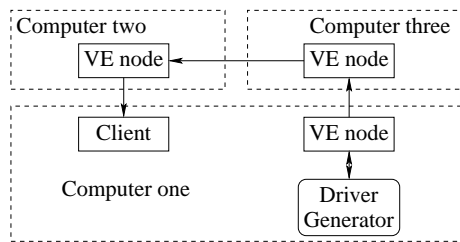


Figure 10: Benchmark for latency measurement through the server.

However, we have discovered a dysfunction in IRIX that can increase time latency of about 100 ms with high frequency data transfer. To avoid this problem, the VEserver includes a patch that may slow down the transfers. Using this patch, the main result of this benchmark is an average latency of $(3 \pm 0.2)ms$ for a range of 1Hz to 10kHz for the three connections between the main VEnode and the VEserver client.

Similar benchmarking remains to be done on other OSes (Windows NT).

6. Improvements with haptic devices

Conversely to visual feedbacks, which only require a low frame rate of 25 frames per second (mainly due to the retinal

persistence), haptic feedbacks typically require 1000 samples per second. Each time step includes several computations, including that of the distance between the virtual position of the device and each object of the scene.

VRPN is one of the first networked device managers that allow such interaction. Its haptic management is based on the use of libraries like GHOST (software manager for detection collision provided with Sensable technology's phantom device), Virtual Hand Toolkit (provided with the CyberGrasp, from VTi) and H-COLLIDE⁸. In VRPN, the interactions between the client and the drivers use the same connection for all the devices of a VE application. Then, the entire scene is managed using this connection. This system fully works in the case of a static scene that only needs to be moved. However, in the case of dynamic data, a single connection will overload such a standalone network.

The double channel used by our distributed architecture proposes a generic solution for the software management of haptic devices. Actually, within the EVI3d framework the driver of an haptic device is connected to the two channels. In the case of small updates, such as moves of a static scene, the information is provided through the VEserver by the event network. Conversely, when the database is changing drastically during a process, the driver is subscribed to the multicast group of the simulation calculator that will provide this dynamic data. The data flow network via the DB network loader routes those stream of data.

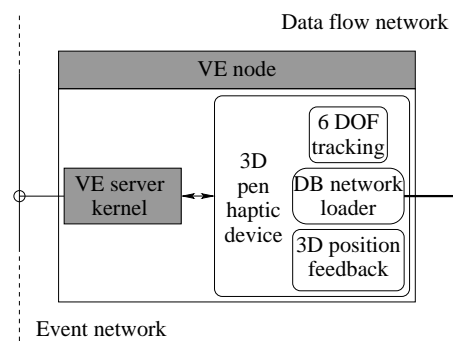


Figure 11: Integration of the Phantom device.

Figure 11 shows that the VEserver allows to directly include the DB network loader within the driver of a VE node, like in the case of our Phantom's driver. Moreover, the EVI3d framework includes the case of devices where the haptic part is physically dissociated from the input part. For instance, with the CyberGrasp, the VEserver allows the communication between the data glove drivers (the hand input part of the bottom VE node of figure 12) and the force feedback driver of the exoskeleton. Thus, input can be located on a computer to manage gesture recognition and its modality interpreter (cf. figure 6), while the feedbacks are managed by

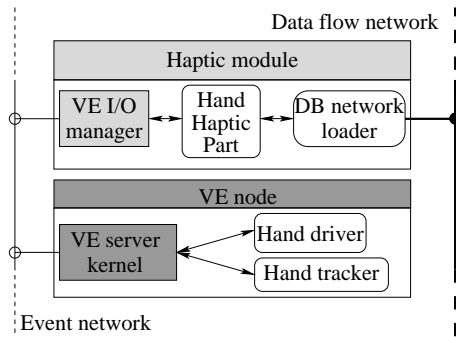


Figure 12: Integration of the CyberGrasp device.

another machine to deliver the necessary robotic command rate.

7. Conclusion

This paper has described the management of some advanced interactions by the way of the EVI3d framework. With this architecture, we implemented a complete multimodal fusion system. The interaction between the VEs server and this multimodal module is done by the introduction of modality interpreters distributed over several machines.

To validate the VEs server and its framework architecture, we developed several applications. To manage the functional commands of these applications, one VE node of the VEs server drives a speech recognition system under Windows. Other VE nodes, running under IRIX, deal with tracking systems (for head and pointing movements).

As seen section 6, we are interested in the use of output feedbacks. However, the haptic devices can be combined with auditory feedbacks according to the available devices. Thus, one of our future works will be to design a specific client, which can manage multimodal fusion processes. For instance, the behaviors of virtual objects are able to provide message events that have to be dispatched over several feedback modalities (visual, auditory, haptic...). Moreover, advanced processes (such as: sound effects, cinematic behaviors, scene generator...) can be dedicated to specific EVI3d modules.

This work is sponsored by the PERF-RV platform, a large Virtual Reality project of the French RNTL program that promotes research in software technologies.

Acknowledgements We wish to thank J.M. Vezien for his reviewing and comments on this paper.

References

1. P. Bourdot, M. Krus, and R. Gherbi. Cooperation between reactive 3D objects and a multimodal X window

kernel for CAD. In *Lecture Notes in Artificial Intelligence (1374) on Multimodal Human-Computer Communication : Systems, Techniques and Experiments*, pages pp. 188–212. Springer, January 1998.

2. P. Bourdot and D. Touraine. A hierarchical server to manage multimodal and collaborative interactions within Virtual Environments. In *Proc. of the 19th IASTED International Conference on Applied Informatics*, Innsbruck (Austria), February 2001.
3. P. Bourdot and D. Touraine. Polyvalent display framework to control virtual navigations by 6dof tracking. In *Proc. of IEEE International Virtual Reality Conference 2002*, Mars 2002.
4. C. Carlsson and O. Hagsand. DIVE - a platform for multi-user virtual environments. *Computer & graphics*, 17(6):663 – 669, 1993.
5. P. R. Cohen, D. McGee, S. Oviatt, L. Wu, J. Clow, R. King, S. Julier, and L. Rosenblum. Multimodal interaction for 2d and 3d environments. In *IEEE Computer Graphics and Applications*, volume 19(4), pages 10–13. IEEE Press, 1999.
6. P. R. Cohen and S. L. Oviatt. The role of voice input for human-machine communication. In *Proceedings of the National Academy of Sciences*, volume 92(22), pages 9921–9927. Washington, D. C. National Academy of Sciences Press, 1995.
7. Y. Gong. Speech recognition in noisy environments: A survey. In *Speech Communication*, volume 6, pages 261–291, 1995.
8. A. Gregory, M. Lin, S. Gottschalk, and R. Taylor. Hcollide: A framework for fast and accurate collision detection for haptic interaction. In *IEEE Virtual Reality*, 1999.
9. R. Taylor II, T. Hudson, A. Seeger, H. Weber, J. Juliano, and A. Helsen. Vrpn: A device-independent, network-transparent vr peripheral system. In *Proc. of Virtual Reality Software & Technology 2001 (VRST'01)*, Banff, Alberta (Canada), November 2001.
10. C. Just, C. Cruz-Neira, and A. Baker. Performance measurement capabilities of vr juggler : real-time monitoring of immersive applications. In *The 4th international immersive projection technology workshop*, 2000.
11. J. Kelso, L. Arsenault, C. Logie, F. das Neves, and R. Kriz. Diverse graphics interface for performer programmer's guide. Technical report, University Visualization and Animation Group of the Advanced Communication and Information Technology Center, Virginia Tech, July 2000. <http://www.diverse.vt.edu/dtk/>.
12. J. A. Larson, S. L. Oviatt, and D. Ferro. Designing

- the user interface for pen and speech applications. In *Conference on Human Factors in Computing Systems (CHI'99)*, 1999.
13. D. R. McGee, P. R. Cohen, and L. Wu. Something from nothing: Augmenting a paper-based work practice with multimodal interaction. In *Proceedings of the Designing Augmented Reality Environments Conference*, pages 71–80. ACM Press, 2000.
 14. D.L Mills. *Network time protocol (version 3): specification, implementation and analysis*. University of Delaware, March 1992. Request For Comment (RFC) : 1305.
 15. L. Nigay and J. Coutaz. A design space for multimodal systems : concurrent processing and data fusion. In *Proc. of INTERCHI'93*, pages 172–178. ACM Press, april 1993.
 16. US Depatement of defense. High Level Architecture (HLA) interface specification. submission to the IEEE committy, April 1998.
 17. S. Oviatt, P. Cohen, LZ. Wu, J. Vergo, L. Duncan, B. Suhm, J. Bers, Holzman, T. Winograd, J. Landay, J. Larson, and D. Ferro. Designing the user interface for multimodal speech and pen-based gesture applications: State-of-the-art systems and future research directions. In *Human-computer interaction*, volume 15 (4), pages 263–322, 2000.
 18. S. L. Oviatt, A. DeAngeli, and K. Kuhn. Integration and synchronization of input modes during multimodal human-computer interaction. In *Proceedings of Conference on Human Factors in Computing Systems (CHI'97)*, pages 415–422. New York: ACM Press, 1997.
 19. C. Pinon-Bouwens and LB. Mc Donald. Distributed interactive simulation: interoperability of dissimilar training devices for team training. In *Twenty-Third Annual Summer Computer Simulation Conference*, editor, *the 1991 Summer Computer Simulation Conference*, volume xxiii+1194, pages 934 – 939, CSC, San Diego, CA, USA, 1991.
 20. G. Reitmayr and D. Schmalstieg. An open software architecture for virtual reality interaction. In *Proc. of Virtual Reality Software & Technology 2001 (VRST'01)*, Banff, Alberta (Canada), November 2001.
 21. D. Rubine. *The automatic recognition of gestures*. PhD thesis, Carnegie Mellon University, 1991.
 22. A. Shaikh, S. Juth, A. Medl, I. Marsic, C. Kulikowski, and J. Flanagan. An architecture for multimodal information fusion. In *Proceedings of the Workshop on Perceptual User Interfaces (PUI'97)*, pages 91–93, 1997.
 23. S. Singhal and M. Zyda. *Networked Virtual Environments*. ACM Press, July 1999. ISBN=0-201-32557-8.
 24. D. N. Snowdon and A. J. West. The AVIARY VR-system. a prototype implementation. June 1994.
 25. D. Touraine and P. Bourdot. VEservers : a manager for input and haptic multi-sensorial devices. In *Proc. of 10th IEEE international workshop on Robot and Human Communication*, pages 2–7, september 2001.
 26. M. J. Zyda, D. R. Pratt, W. D. Osborne, and J. G. Monahan. NPSNET: Real-time collision detection and response. *The Journal of Visualization and Computer Animation*, 4(1):13–24, January–March 1993.