N96-14984

Virtual Environment Architecture for Rapid Application Development

Dr. Georges G. Grinstein, David A. Southard, J. P. Lee
The MITRE Corporation,
Bedford, MA 01730-1420

ABSTRACT

We describe the MITRE Virtual Environment Architecture (VEA), a product of nearly two years of investigations and prototypes of virtual environment technology. This paper discusses the requirements for rapid prototyping, and an architecture we are developing to support virtual environment construction. VEA supports rapid application development by providing a variety of pre-built modules that can be reconfigured for each application session. The modules supply interfaces for several types of interactive I/O devices, in addition to large-screen or head-mounted displays.

Key words: Virtual reality, virtual environments, software, architecture, prototyping.

INTRODUCTION

One of MITRE's duties as a Federally Funded Research and Development Center (FFRDC) is to objectively evaluate and compare current technologies, and to recommend courses of action for numerous government programs. As such, we have actively been involved in assessing workstation, graphics, and user-interface technology. Since these areas form the basis for virtual environment technology (VET), we are in a good position to further explore the intrinsics and the impact of this new technology [1-3]. We are currently developing MITRE's Virtual Environment Architecture (VEA), to be used as a foundation for several applications in battle management, mission planning, electric utility simulation, and medical support [4,5].

VEA supports event-driven simulation, using a real-time clock with variable time expansion, and a flexible mechanism for integrating application-specific modules, such as knowledge bases, planners, and simulation models. It is written in C++ on a Silicon Graphics RealityEngine system, and executes object modules in parallel when multiple processors are available. VEA can support a wide variety of user interface devices. Voice, gesture, aural, and visual interactions are supported for the creation of multimodal,
multisensory, and highly interactive environments. Visualizations can be presented on large screen or head-mounted stereoscopic displays.

Requirements for rapid prototyping

Rapid prototyping of environments, objects, behaviors, synthetic tools, and evaluation of new peripheral devices is an important part of the virtual environment design process. Virtual environments are typically very large entities, comprising diverse source code for graphics, expert systems, knowledge bases, and peripheral device drivers. Distributed systems for collaborative activities further complicate the integration of new technologies. As software developments and hardware offerings continue forward, a virtual environment application must be able to accommodate rapid change. Requirements for rapid prototyping are needed to face the larger domain of requirements placed on virtual environments, and are critical to the overall solutions of the problems VET places on developers.

In a comparative study of four virtual environment architectures (including VEA), Masterman [6] analyses the functional requirements for virtual environments, and the solutions offered by each of the representative systems. The functional requirements are: *interactive response* times to user inputs; *multiple interface devices*, and *multiple modes* of input and output; *distributed processing; easy integration* of application code; *extensibility* to new interfaces and applications.

System usability ultimately depends on minimizing the latency between user actions and their manifestation in the virtual environment. Interactive response must always be maintained in the environment. This is complicated by the requirement that multiple devices of a multimodal and multisensory nature are the rule in such applications. Distributed computing is inevitable, which means that integration of additional systems and devices occur across multiple, possibly remote platforms. Ease of integration allows maximum portability and must consider the addition of separate, autonomous application code written in heterogeneous languages. Finally, ease of extension is required for the evaluation of new concepts and products without a large programming effort.

These requirements are accommodated by several common design themes. Object-orientation results in a highly modular environment, and it addresses extensibility and ease of integration. Object behavior can also be easily encapsulated. Mechanisms for parallelism, in line with present and future hardware capabilities, address the interactive response and distributed computing requirements. Asynchronous, message-based communication results from the object-oriented and parallelism approaches, and it is necessary for distributed computing. The layering of device abstraction further assists the integration and extension issues, as the device interface must accommodate change because of the continual introduction of new devices for virtual environments.

Why a new architecture?

We wanted to take specific approaches to each of the design themes. On the basis of our experiences and observations, these approaches seemed best:

Object Orientation. VEA is implemented in C++. The kernel, devices, and abstract domain objects are all treated as autonomous objects. Most system protocols are implemented high in the class inheritance hierarchy, so that object instances derive most of their systems behavior from their base classes. In contrast, for example, VEOS (Virtual Environment Operating Shell) is written in primarily in XLisp, with application and interface modules in C [7]; Division's dVS uses an object-oriented C library interface [8]; and IBM's VUE (Veridical User Environment) uses a rule-based dialog manager [9].

Parallelism. VEA assumes a tightly coupled, shared memory, multiprocessing model of computation. In this respect VEA differs from most other architectures, which assume a looser configuration, distributed over a local area network. VEA's approach is consistent with current super graphics workstation architectures (as exhibited by Silicon Graphics, for example), as well as our perception of architectural trends. Network distribution will be supported in the future by adding network adapter software modules. Contrast VEOS with its loose, heterogeneous model; dVS with its specific selected hardware platforms; and VUE, similar to VEOS, with a loose and heterogeneous model.

Message-Based Communication. VEA defines two modes of communication: messages and events. Messages are synchronous (blocking); events are asynchronous (non-blocking). Both messages and events can be executed in parallel.

Layering of Device Abstraction. VEA defines two layers for device abstraction: *filters* and *managers*. Filters provide an interface between devices and generalized I/O events. Managers transform the I/O events into higher-level events.

Integration and extensibility

There are several aspects of VEA that provide for ease in integration and extensibility. New sessions with alternate devices, displays, and users can be reconfigured without recompiling by selecting a different configuration file or by altering any of the environment parameters. Object knowledge and behaviors are independent and separated from the interaction style. This provides the capability of having dramatically different environments. We have provided templates for generic device classes that support quick integration of new devices: most of the interface is provided as boilerplate. Finally we have developed VEA with the intent of eventually supporting multiple users. There is much work to be done in understanding the ramifications of users sharing a design space. For example, what happens when two users grab the same object and attempt each to modify it in different ways?

HARDWARE AND SOFTWARE PLATFORM

We decided to target super graphics workstations from Silicon Graphics, since we are familiar with their products, and because it meets the requirements for a high-performance graphics rendering across a wide product line. The RealityEngine graphics option provides hardware texturing and anti-aliasing support. In principle, at least, portability can be achieved by replacing the graphics module with alternative code, as all platform-specific graphics code resides in one object module.

I/O Devices

After having evaluated numerous devices and software packages, we realized that applications require a variety of I/O devices; however, software to support them is scarce. Table 1 summarizes the devices we are currently working with, which we have categorized into generic classes. These devices were evaluated and their limitations characterized through experimentation and empirical observations in our virtual environment laboratory. Software drivers have been written as necessary.

Table 1. I/O Device Categories

Class	*Examples*
Posture	VPL DataGlove, Exos Dexterous HandMaster
Locators	Polhemus, Ascension Flock of Birds, SpaceBall, Global 3D Controller
Pointers	mouse, Logitech 2D/6D mouse, Origin Instruments DynaSight tracker
Graphics	Large-screen dual-projector stereoscopic display, Virtual Reality Group head-mounted display, SGI RealityEngine
Tactile	Exos tactile display
Speech	Voice Navigator (input), Voice Impact (output)
Audio	IDI synthesizer
Text	keyboard
Button	buttons box, mouse, SpaceBall
Valuator	dials

For voice I/O we are using inexpensive commercial products such as Macintosh-based Voice Navigator for voice input. Even with its speaker-dependent limitation it provides for an interesting integration demonstration. We plan to use Voice Impact for output. It records and generates voice messages. We also plan to experiment with a MIDI sound synthesizer for non-speech audio output.

Software tools

Numerous software tools were acquired and evaluated. We selected the following tools: C++ for the object oriented programming, Software Systems' MultiGen for object modeling, IRIS Performer for visual simulation support, and NASA's C Language Integrated Production System (CLIPS) for the knowledge-bases, which will be used to model intelligent object behavior. IRIS Performer provides excellent rendering

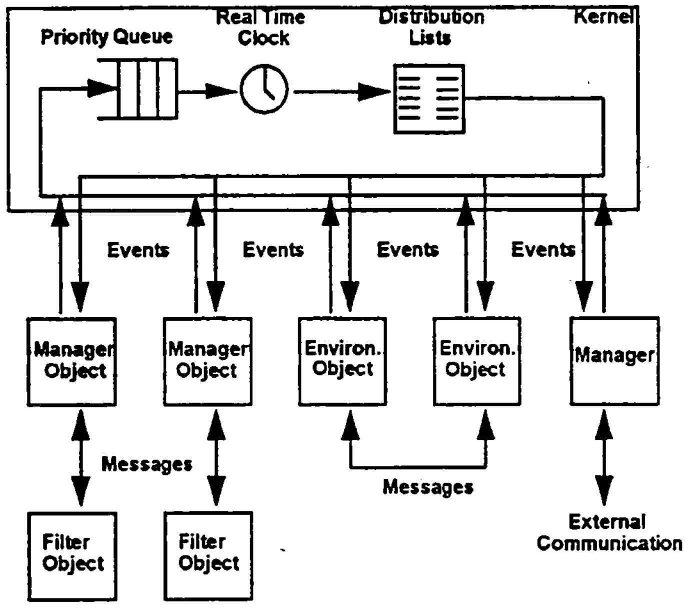

Figure 1. VEA Overview

performance on the whole line of workstations, takes advantage of multiprocessing capabilities on multi-CPU systems, and is the most cost-effective among the visual simulation toolkits on the market. CLIPS provides an object-oriented knowledge representation language, as well as a rule-based production system, both of which are easily accessible from C and C++.

THE VIRTUAL ENVIRONMENT ARCHITECTURE

VEA uses events for representing certain kinds of simulated events and communications between simulated objects. The VEA kernel uses UNIX system facilities to implement its event-handling mechanisms. Figure 1 provides an overview of the event mechanism in the kernel. Events and objects are initialized from a configuration file. Events are placed in a priority queue, ordered by time. Objects are placed in the object list. When an object is initialized, it subscribes to the types of events that it is capable of accepting. The subscriptions are placed in the distribution lists, which are organized by event type. Objects may subscribe, or drop a subscription, at any time during the simulation.

Events

Events have properties that distinguish them from the more general concept of messages. Object-oriented languages such as C++ support messages as *member functions*, or *methods*. VEA contains additional facilities that support events. Events are characterized by the following attributes:

The primary purpose of an event is to effect a global state change.

Events occur in time. Conceptually, all objects receive events simultaneously.

An event can be processed by any object that registers to receive it. Each object interprets an event as appropriate to its function. A sender does not know a priori who will receive its event.

Events may be initiated by the users of the system, by simulated objects, or by external influences.

Events are not elements in a communication dialog or protocol.

Event types include: button, gesture, graphic, locator, pointer, position, posture, sound, speech, text, valuator. The intention is to keep the number of events small, and to keep their meanings as flexible as possible. There may be some overlap between event types. For example, a posture event could be represented by a kind of button event. For convenience, however, a separate posture event allows a manager object to monitor sequences of postures for possible gestures. This design realizes the layering of device abstraction.

Clock

VEA contains a real-time clock, which is implemented with the UNIX interval timer. A time expansion factor is provided, which allows the simulation to run faster or slower than real-time. Most objects will not need to access the clock itself. The objects usually work with the time stamp reported in the events that they receive.

Multiprocessing

When the simulation clock reaches the time indicated by the event at the top of the priority queue, the event is dispatched for distribution to the objects. All objects on the distribution list for that event type receive a pointer to the event. Each object can then process the event, according to its own interpretation. Each object is run in its own process. In principle, all objects may proceed simultaneously. In practice, parallelism is limited by the number of processors available, as well as by other resources shared by all processes. A separate "lightweight" process is created for each event. These processes remain in existence only as long as necessary to process each event. Access to common data structures is synchronized using UNIX system semaphores, for which we created a C++ class interface.

As each object computes, it updates its internal state to reflect the consequences of the event. This may involve a message dialog with the sender of the event. The result of the processing might be new events added to the priority queue, or it might simply update an object's internal state. When an object completes its processing, the object becomes free to process another event, if another one is waiting. The kernel deletes an event when all recipients have released it.

Software Bus

In our prototypes, there is a need to use previously developed, stable tools. Such modules include planners, schedulers and simulation models. We called these tools *external modules* and our intent is to provide a mechanism, termed a *software bus,* by which such modules can easily be integrated within VEA. The first example of a external module we integrated is a CLIPS knowledge base. It is anticipated that the soft[ware bus will support a distributed interface for a wide variety of simulation models.

CURRENT EFFORT

Our current application is aimed at battle management. This application is intended to provide situation awareness to commanders, who must make critical decisions quickly. In addition, the system should facilitate the decision-making process directly. That is, the system should provide tools that enable the decision-maker to probe the current situation, obtain needed information, and assess the implications of various options. Someday, such an environment might host intelligent agents, which could suggest alternative courses of action, and once a decision is made, actually begin to implement those decisions in the real world.

A central aspect of this application is a detailed terrain scene. The terrain model will represent not only the terrain relief, but also cultural and natural features, such as buildings, roads, crop lands, rivers, and forests. Some of the stationary features will be militarily significant: supply depots, defense installations, etc. The scene will be populated with numerous moving vehicles, such as trucks, tanks, and aircraft. The vehicles could represent objects derived from real-time tactical data links, or they might be simulation models. The decision-maker will have the ability to move through the scene, to place himself at any viewing position on the ground, or in the air. The viewing position could be attached to any moving vehicle, so that the viewpoint tracks the vehicle as it moves.

The user will be able to designate objects, and to query them for identification, history, and any intelligence information that might be available. In a simulated battle scenario, the information will be dynamic, as new reports come in about each object. For simulated scenarios, it is important that the simulated objects move and behave as they would in reality. For example, trucks stay on roads, tanks avoid lakes, aircraft avoid the ground, and enemy forces engage or avoid each other, depending on their rules of engagement, and the current situation. The user may wish to directly reconfigure various assets, then have mobility and cost models advise whether the new arrangement is feasible, how long it would take to reach the new configuration, and what the costs would be.

Another aspect of this application is the *what if* scenario. In this case, it will be desirable to set-up simulations with several different initial conditions, then to view a simulation at faster-than real-time speeds. Conversely, if too many things are happening at once, the user may wish to slow down the scenario, so that all the object interactions can be observed.

The simulation and database systems needed to support such a scenario do not currently exist as integrated systems, but parts of these capabilities do exist throughout the military. Our job is to begin to pull the pieces together into a useful system.

The user of this application is quite different than envisioned for most virtual environments. A command officer will not be willing to suit-up in cumbersome, restrictive apparatus. There will be no opportunity to train users how to use the equipment. The devices used must be natural and easy to use, and the user interface must be intuitive.

Potential Enhancements

We have a number of areas in mind for expansion. Our immediate plans are to expand from a single-platform, multiprocessing system, to a networked, multi-platform, collaborative system for multiple users. Virtual environments represent a technology by which remote teams can work together on science and engineering problems. Following that, we envision adding intelligent agents. These agents could monitor the environment, and notify the user when certain situations arise, or carry out tasks on behalf of the user.

We would like to include support classes for physically based modeling. This would include basic Newtonian dynamics and collision models. This would allow us to create many interesting virtual environments, in which objects behave as one would expect. Many object behaviors are completely mechanical, so there is no need for the knowledge base to intervene in these cases.

Another potential area includes advanced artificial intelligence information systems. Many applications exist for advanced user interfaces for schedulers, and reactive and adversarial planners. MITRE has developed context dependent natural language parsers and our intention is to eventually integrate these tools as well.

FUTURE PROSPECTS

Worthy applications for VET are real, and are here today [10,11]. However, there is much engineering work needed to realize the potential. The immediate challenge is to construct an infrastructure that supports a wide variety of VET investigations. VEA does this by taking a modular, object-oriented programming approach. Input and output devices are embedded in filter and manager layers, which encourages device and application independence, and flexibility. Simulation and modeling are supported through the integral real-time clock, and the knowledge base. Rapid prototyping is supported through the use of flexible configuration files, which allow each VEA session to be tailored for a different use. High performance is obtained with built-in support for concurrent multiprocessing.

Power Utilities

MITRE is actively participating in the development of a simulated control panel for a fossil-fueled power plant control room. Power utility companies throughout the country must train and rehearse power plant operators in correct procedures. Currently, control room mockup trainers are constructed full-scale from the actual devices. Such trainers are very expensive to build, operate, and maintain. Virtual environments are a low-cost alternative that can be tailored to the needs of individual power plants. In this application, integration with a proven numerical simulation is critical.

Another application targets training for high-voltage switch and transformer yard repair procedures. Severe accidents, resulting in deaths, are currently a reality for many power utilities. Training and refresher courses in proper safety procedures are essential to reduce these occurrences. Virtual environments may offer a way to perform such training effectively, without exposing the operator to hazardous conditions.

Health

A recent conference focused on how VET could help persons with various disabilities [12]. For example, a person with Parkinson's disease often has an involuntary shaking in the hands, but is unaware of this motion, unless he looks at his hands. A glove device, however, can be programmed to filter out the involuntary motion, and present a stable representation of the hand. Thus, the presentation of the hand would match the users mental image. This could enable such persons to perform normal activities in the virtual environment, that would be difficult to perform in a real environment.

A related application for VET is as a flexible platform for psycho-physical experiments, on earth as well as in space. A well-designed and integrated suite of hardware and software could replace several specially-designed experimental apparati.

ACKNOWLEDGMENT

The work was sponsored in part by The MITRE Corporation, in addition to primary sponsorship by the Rome Laboratory, Griffiss AFB, NY, and the Electronic Systems Center, Hanscom AFB, MA, under contract #F19628-89-C-0001.

REFERENCES

[1] D. A. Southard (1992), Transformations for Stereoscopic Visual Simulation, *Computers & Graphics* 16(4) 401-410.

[2] R. B. Mitchell, J. L. Segal (1992), A Virtual Maintenance Trainer, *SID Digest 23* 906-908.

[3] P. J. Hezel, H. Veron (1993), Head Mounted Displays for Virtual Reality, *SID Digest 24*, Paper No. 41.3 (to appear).

[4] G. G. Grinstein, R. B. Mitchell, D. A. Southard (1993), Virtual Reality: An Interface Architecture for Interactive Simulations, *Proc. Soc. Computer Simulation* (to appear).

[5] D. A. Southard, J. P. Lee, R. B. Mitchell, G. G. Grinstein (1993), Case Study: A Virtual Environment Architecture (submitted to *IEEE Symposium on Research Frontiers in Virtual Reality).*

[6] H. C. Masterman, G. G. Grinstein (1993), Software Requirements for Virtual-Environment Applications, *SID Digest* 24, Paper No. 35.1 (to appear).

[7] G. Coco, *VEOS 2. 0 Tool Builders Manual,* Human Interface Technology Laboratory, University of Washington, May 1992.

[8] *Provision Software Overview,* Division Inc., Bristol, UK, 1991.

[9] P. A. Appino, J. B. Lewis, L. Koved, D. T. Ling, D. A. Rabenhorst, C. F. Codella (1992), "An Architecture for Virtual Worlds," *Presence: Teleoperators and Virtual Environments* 1(1).

[10] P. T. Breen (1992), *Near-Term Applications for Virtual Environment Technology,* M92B0000011, The MITRE Corp., Bedford, MA.

[11] P. T. Breen (1992), The Reality of Fantasy: Real Applications for Virtual Environments, *Information Display 8(1 1)* 15-18.

[12] *Proc. Birtual Reality and Persons with Disavilities Conf.,* California State University Northridge, 18-21 March 1992, Los Angles, CA. 10.