

Dialog Manager Support For Real-Time Conferencing In Virtual Worlds

*Lawrence Koved, Christopher Codella, Reza Jalili, J. Bryan Lewis,
Daniel T. Ling, James S. Lipscomb, David A. Rabenhorst, Chu P. Wang*

Veridical User Environments
IBM T. J. Watson Research Center
Computer Science Department
P.O. Box 704
Yorktown Heights, New York 10598

Abstract

Virtual Worlds are interactive three-dimensional computer-generated multi-sensory environments where people interact with the computer as they would with the real world. The motivation is to create a human-computer interface that is more natural and intuitive. Real-time computer conferencing enables two or more people to work and interact in a coordinated fashion. Virtual Worlds systems need to provide means for allowing people to work collaboratively.

Several requirements drive the design of multi-person Virtual Worlds: multiple I/O devices operating concurrently, independent views, and interaction with real-time simulations. A dialog manager provides the means of coordinating I/O devices with the simulation. When all participants have their own I/O devices and dialog manager, it is then possible to provide each participant with an independent view of the simulation. The separation of the application's *style* from its *content* through the use of a dialog manager has facilitated the creation of a multi-user application from a single-user application.

Keywords: CSCW, real-time conferencing, synchronous conferencing, Virtual Worlds, Virtual Reality, dialog manager.

Introduction

An emerging area in computer-supported cooperative work is real-time computer conferencing. This technology allows two or more people to see a shared view of a computer application. That is, when any person moves a mouse, uses the keyboard, or uses other input devices, the results of these actions are immediately shown to all people in the conference. This technology is especially useful in consulting, education, programming, meetings, marketing, technical writing and a variety of other applications.

The ability to support real-time conferencing within a virtual world provides the opportunity for multiple participants to share multi-sensory, three dimensional (3-D), computer-generated environments. This leads to applications which go far beyond simply reproducing or augmenting face-to-face conferences. Virtual worlds provide a computer environment that reifies the object of collaboration, whether it be a car, a model of the US economy, or the medical images of a patient. The virtual world becomes a shared laboratory for the participants to rapidly test hypotheses and explore alternatives. The technical challenges are particularly severe because these laboratories are typically based upon interactive real-time¹ simulations where the ordering of events and the time elapsed between events affect the simulation results.

Collaboration in virtual worlds implies that all of the participants in the collaboration need to be able to work and share information about a common virtual world while it is running. Providing for collaboration in interactive real-time simulations in virtual worlds places new requirements and constraints on existing techniques. Among these are the need to:

- share real-time simulations without significantly slowing down the simulation,
- provide each participant with an independent view of the virtual world,

¹ "Real-time" is intended to mean that the simulation is able to compute its new state, after receiving new input, at interactive speeds. It is not intended to imply real-time programming techniques.

- maintain interactive response time as the number of participants increases,
- provide a flexible strategy for participants' access to shared objects, and
- run on a heterogeneous collection of computers.

To meet the above requirements, we have found the need to separate application *style* from its *content* [Wiecha et al., 1989]. The style is the user interface, including the I/O devices. The content is the operational characteristics of the application, independent of the user interface. To achieve this separation of content from style, a *dialog manager* is employed to map the content of the virtual world into the style [Rhyne, 1988, Lewis et al., 1991]. By making this separation, and by applying a hierarchical organization to the writing of dialog rules (code run in the dialog manager), we are able to readily convert an existing single-user application lacking an interactive interface into a multi-user interactive virtual world application.

The next section will describe previous work in real-time conferencing. The balance of the paper will discuss the application of the VUE architecture to a specific example, called Rubber Rocks. Issues surrounding modification of the application into a multi-user virtual world will be followed by a discussion of future research issues.

Related Work

Some of the earliest work in real-time computer conferencing can be traced to Engelbart's shared-screen conferencing in NLS/Augment [English and Engelbart, 1968, Engelbart, 1975, Engelbart, 1982]. Many early real-time conferencing systems are based on the concept of *terminal-linking* through the use of a software bridge or pseudo-terminal. A pseudo-terminal connects to an application and each user's terminal connects to the pseudo-terminal (see Figure 1). Examples of terminal-linking have been extended into the windowing system arena (c.f., Patterson [1990]). Our own work based on terminal-linking is a system called *Cooperative Viewing Facility* (CVIEW), which separates the conferencing functions from the application [IBM, 1985, Koved, 1990].

Sarin and Greif [1985], Sarin and Greif [1988], Greif and Sarin [1988] identified many important issues in real-time conferencing:

- allowing participants to join and leave conferences at times other than at the beginning and end of the conference,
- allowing each participant to have independent (different) views or interfaces to the application, yet provide mechanisms to align of the views, when desired, to achieve WYSIWIS (What You See Is What I See [Stefik et al., 1987]),
- separation of the conference controls from the shared application,
- interface to a shared application is through the specification of operations on objects,
- participants can have different roles, or privileges and responsibilities, within the conference which may imply the need for access control on the conference's shared objects,
- heterogeneity of hardware and software used by the participants' user interface (e.g., workstation).

Lantz [1986] amplifies these issues by identifying a number of key technologies and providing a model to make real-time conferencing easier to implement. These include the separation of the user interface from the application by using a *dialog manager*, using a *conference manager* external to the shared application to provide access and management of conferencing functions, and using servers (e.g. window managers) to provide hardware independent access to shared resources such as computer displays, audio and video resources.

Lantz [1986], Lauwers et al. [1990] use existing single-user applications and turn them into multi-user applications by replicating the computing environment on each of the participants' workstations (see Figure 1). A replicated architecture eliminates the bottleneck found in a centralized architecture where the output from the application needs to be multicast to all of the conference participants' workstation. The state of the replicated application is synchronized by starting the application with identical state (e.g., files, environment variables). The input from each workstation is multicast to the other workstations and synchronized by the conferencing support environment so the application on each of the workstations computes the same results and generates the same output. This approach is partially successful, although maintaining synchronization of the input stream across many processes and handling timing-dependent

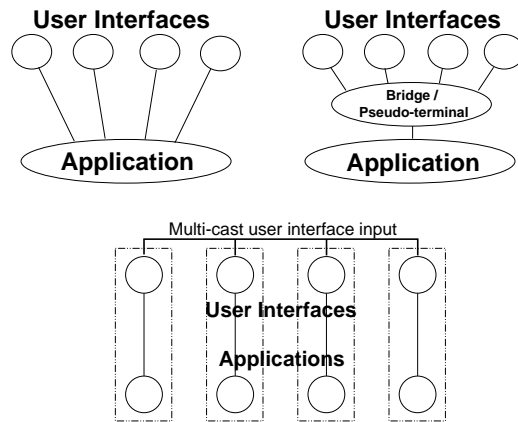


Figure 1. Centralized and replicated architectures: The figure in the upper left shows a centralized multi-user application. The user interface for each of the conference participants connects directly with the application that handles all user interface interactions. The figure in the upper right shows a single-user application connected to a bridge (or pseudo-terminal), and the bridge connects to each of the user interfaces. This is the approach usually used in *terminal-linking*. The figure on the bottom shows the application (lower circles) and user interface (upper circles) replicated on four computers (the rectangles). The input from each of the user interfaces is multi-cast to the other workstations in the conference.

interactions (such as continuous window scrolling), among other issues, turns out to be fairly difficult using existing technologies. These difficulties are also reported by Crowley et al. [1990].

Ahuja et al. [1990] compares both single-site execution (centralized architecture) and multi-site execution (replicated architecture) approaches. They find that it is too difficult to maintain consistency across the multiple execution sites when they replicate the application. In addition, their applications are not I/O intensive, so the networking overhead does not create substantial bottlenecks as had been expected. Thus, they adopt a centralized model for their future work.

Patterson et al. [1990], Patterson [1991] have implemented collaboration-aware applications where the application is separated from the user interface through the use of a User Interface Management System (dialog manager). The application is considered to be a set of shared objects, including user interface objects, upon which operations can be performed. The user interface can then be customized for each of the participants, and can take into account their role in the conference, including the access rights they have to the application objects. The separation of the application from the interfaces allows conference participants to have different views, or input and output representations, of the conference objects. Patterson also discusses performance issues in developing multi-user applications and ways to improve interactive response time.

Stefik et al. [1987], Stefik et al. [1988] discuss methods for access to shared conference objects (shared database). These methods include centralized data, centralized-lock, a "cooperative method", dependency-detection, and roving-locks. In addition to the locking methods, they also advocate the partitioning multi-user applications into *user input*, *semantic actions*, and *display actions*. The semantic actions correspond to what we call the application content, and the user input and display actions correspond to what we call style.

Virtual World Application

A flexible object simulator that integrates Newton's second law ($F=ma$) over time was converted from being a non-interactive program into a multi-user virtual world [Codella et al., 1992]. The simulator models objects as a network of point masses and springs (or bonds) [Bacon et al., 1989, Norton et al., 1991, Sweeney et al., 1991]. The dynamics of the system include external forces such as gravity, friction and repulsive forces due to collisions. Collision forces occur when objects collide with each other, room boundaries, or any of the users' hands.

Figure 2. Rubber Rocks: Two users are interacting with Rubber Rocks. Each is wearing a baseball cap with a position sensor attached for head-motion parallax. Gloves and position sensors are used for gesture input. The two users are looking at a room from opposite sides. Within the room are a number of flexible objects. For this photograph, the simulator is frozen at a time when both users are tugging at a single brick-shaped object. Hands are also displayed in the scene reflecting both the position and the gesture of the users' own hands. Note that one user is viewing the scene on a twenty-three inch display while the second is using a large-screen projection system. Both views are monoscopic for purposes of the photograph. (Reprinted from [Codella et al., 1992])

For the CHI'91 Interactive Experience and SIGGRAPH'91, a playful version of the system called *Rubber Rocks* was created. A number of objects are simulated, including octahedra, cubes, rectangles and sheets. User interaction with these objects results in perturbation of the simulated dynamical equations through the imposition of either forces or absolute positions. The interactions include slapping, carrying, and squirting² objects (see Figure 2).

The simulation program, running as a process under the AIX³ version of Unix⁴, has been modified to accept commands via a TCP socket. These commands include the ability to create and delete simulation objects, and include the concept of a "hand," modelled as a rigid object with a location and volume in space. The interface also includes the ability to apply external forces to and change the absolute coordinates of simulated objects. The addition of this new interface to the application allows a user to interactively steer the simulation's computations by slapping, grabbing and squirting the simulated objects.

VUE Architecture

The VUE architecture is based on an event-driven dialog manager acting as a coordination layer between the virtual world applications and the device servers [Lewis et al., 1991, Appino et al., 1992, Codella et al., 1992] (see Figure 3). Each input and output device is implemented as a separate process, and the dialog manager processes the stream of events from these devices in separate threads, or dialogs. For example, a glove input device for recognizing hand gestures sends events to the dialog manager. Similarly, the dialog manager receives the hand's location from the 3-D position tracking device. A piece of dialog code (a rule) combines the two pieces of information into a "hand" event which it relays to the output devices (e.g., graphics output device), and also relays the hand position and other state information to the simulation.

² Squirting is the application of a force on an object in the direction the user is pointing.

³ AIX is a trademark of IBM

⁴ Unix is a trademark of AT&T.

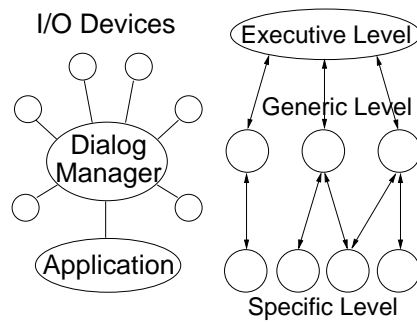


Figure 3. Architecture of a virtual world and hierarchical organization of dialog rules: On the left is a diagram of a dialog manager connected to multiple I/O devices and the application. The righthand figure represents the logical hierarchy of dialog rules in the system. For a further discussion of rule hierarchies, see Appino et al. [1992].

In addition to coordinating and transforming information between the input and output devices, the dialog manager allows for the concurrent operation of multiple I/O devices and applications operating at different rates. For example, the 3-D tracking device sends events to the dialog manager at a higher rate than the graphics output device can use the data. Dialog rules buffer the data from the tracking device until the graphic device needs it.

Conceptually, dialog rules are partitioned into a three-level hierarchy (see Figure 3):

- The *specific* level is the lowest in the hierarchy, and interfaces to I/O devices or applications. The purpose is to encapsulate low-level details associated with interfacing to devices, including communication flow control and data/noise filtering.
- The *generic* level accepts events from the specific level and translates them into more general interaction techniques independent of specific I/O devices. For example, events from 3-D tracking devices and a hand gesture input devices can be combined into a generic "hand" event. The "hand" event does not contain device dependent information. Also, events from the executive level can be transformed into events to be handled by the specific level.
- At the top is the *executive* level that defines the scenario or interaction sequences in the virtual world. The scenario contains two components -- the sequences of actions the user can perform, and the goals the user is trying to achieve.

These three levels are conceptual and afford substitution of one input device for another since the events generated by the generic level are device independent. Similarly, at the executive level, it is possible to change the interaction scenario of the virtual world. For example, at CHI'91, Rubber Rocks was created as a competitive multi-user game. A substitution of executive level rules resulted in a non-competitive version of the system for SIGGRAPH'91 [Codella et al., 1992].

Real-time Conferencing & Multi-User Application Issues

A simplified view of the process organization for single and two-user Rubber Rocks is shown in Figure 4. The figure on the left represents a single-user version of the system without the I/O devices. The user's dialog manager is at the top, and attaches to the application to send commands and receive results of the simulation. The figure on the right shows how the second user's dialog manager is connected to the application and to the first person's dialog manager. The set of commands and responses between the second user and the application is the same as for the first person's dialog manager interaction with the application. However, with a second person, information about the user interfaces (the style) is exchanged between the dialog managers.

The above approach to organizing the software for Rubber Rocks was decided after a number of real-time conferencing issues were considered. These issues are discussed below.

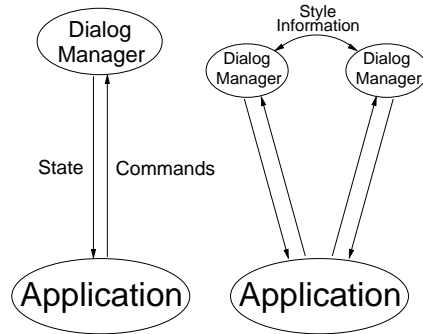


Figure 4. Process organization -- one and two-user: Commands and state information flow between the dialog managers and the application. When there are two or more dialog managers coordinating to create a multi-user system, then style (user interface) information is exchanged between the dialog managers. In Rubber Rocks, the style included such information as the user's hand location and gestures, and the object colors.

Centralized / Replicated architectures

A basic issue: does the application program run in one place, or do all users run their own copy (see Figure 1). In the case of Rubber Rocks, the application is a fairly complex simulation where the complexity of the computation is $O(n^2)$, where n is the number of point masses in all of the objects being simulated. As a practical matter, it is not cost effective for each user to run a copy of the application since a single copy easily consumes the entire CPU of a workstation.

A second consideration is that the application has *timing-dependent behavior*, where the ordering and the time elapsed between events affect the results of the computation. As users' hands move about the simulation space, objects collide with the hands. The computation for collision detection and resolving the differential equations is performed every simulation time step. Each simulation time step is approximately 14 milliseconds, roughly the same amount of time to send a round-trip message to a process on another computer. Synchronizing multiple copies of the application to maintain the relative order and timing of the hand position changes would drastically slow the simulation.

For the above reasons, it was clear that a single copy of the simulation would be shared among the conference participants. Typically, the simulation is run on a dedicated workstation.

Collaboration-unaware / aware

To interface with the dialogs, a layer of communication software was added to the simulation to make it an interactive program. As part of the new interface, a user was given the ability to use two hands (left and right) to interact with the simulated objects. To make the application collaboration-aware, it was necessary to include additional hands for each of the conference participants. This was done by extending the array of hand information to be processed by the simulation.

To be collaboration-unaware, a bridge or virtual terminal would be needed to make the multiple conference participants appear to the application to be a single user. The hard part would be trying to take multiple pairs of hands, one pair for each user, and converting the data so that these hands appeared to the program as a single pair of hands. If the simulator received only one pair of hands, then whose hands would be used? Taking the average position of all hand would lead to unintuitive behavior since the exact location of the average position would be hard for a person to accurately predict from their own hand movements. Interaction with the simulated objects would become cumbersome. Alternatively, if a different participant's hands were used every simulation step, then the hands would appear to be rapidly moving, and would throw off the simulation results. Thus, making the flexible object simulator collaboration-unaware was impractical.

Floor control

Typically when we think of floor control, we are talking about taking turns when interacting with the objects upon which we are collaborating. In virtual worlds, we need to relax the constraints on access to

the shared objects because interactions with the objects need to be continuous, and traditional database locking strategies could make interactions stilted and non-intuitive. The primary issues in floor control in virtual worlds are the access to shared objects in the simulation (content) and the exchange of interface information (style). In Rubber Rocks, both a centralized and distributed approach are used. The centralized case is that object creation and hand-object interactions, including the determination of what to do when multiple hands are manipulating a given object, is mediated by the application through simple algorithms that do not require database locking protocols. The distributed case is the determination of when to create new objects and deciding the object colors. This is managed by the dialog managers communicating amongst each other, without having to resort to locking protocols. In general, floor control is not an important factor in the design of shared this virtual world.

Abstraction

Our early decision to separate the application style from content makes it possible to define the application and user interface as a set of operations on abstract objects. In the case of the application, there are a number of operations the user can perform, and in turn the simulation returns results and state information. Similarly, in the user interface, when elements of the interface are considered to be abstract objects, the dialogs can process requests to perform operations on these interface objects. For example, when a hand changes location, the dialog performs an operation to change the location of the rendered hand. When the simulation determines that an object has shattered, then the information returned to the dialog will result in deletion of the object from the interface.

The VUE architecture further refines the general idea of abstraction into layers. In a multi-user environment, the dialogs exchange event information at the highest layers whenever possible. In the case of Rubber Rocks running as a competitive game, the rules and scoring for the game are located in the executive level. The dialogs exchange point scoring and other game related information rather than the details about when objects fractured. For each participant to see the other participants' hands, the dialogs exchange "hand" events rather than the 3-D tracker and glove information. The goal is to use the higher levels of abstraction (executive and generic level events). By doing so, there is greater device independence and the amount of network traffic may be reduced. Device independence also affords independent views.

Views

One of the challenges of building real-time conferencing systems has been the desire to allow participants to have *independent* views of the application. Several factors in the VUE architecture make it fairly easy for participants to have independent views. First is the partitioning of content from style, so the application does not have knowledge of the user interface. The style, or user interface, is localized in the dialog rules and I/O devices. All participants interact with their own dialogs and I/O devices. The information participants share is application data and high-level (generic and executive level) events exchanged among the dialogs. As long as the data exchange among the dialogs remains abstract⁵, then it is straightforward to allow all users to have their own independent views of the application data.

In the case of the VUE architecture, and specifically Rubber Rocks, all participants have their own view of the virtual world. The application sends state information (e.g., collision reports, object fractures, object creations, etc.) to each of the participant's dialog managers, and graphics data to each participants' graphic output device. The dialogs exchange hand events and object color information amongst themselves. Each participant views the simulation from a different perspective⁶. Also, one person can be looking at Gouraud-shaded graphics objects, while the other person may be looking at wire-frame renderings. Each participant's head location is independently tracked to get head-motion parallax in the graphics output.

It is fairly easy to switch from independent views to shared (or *aligned*) views. The technique for sharing is essentially the same as the process for exchanging hand position information between dialogs. If participants want to align their object rendering technique (wire-frame vs. shaded graphics), then the dialogs exchange events whenever the rendering technique should be changed. To align the viewing location for

⁵ For example, abstract hand information would include current position, orientation and gesture. In contrast, a non-abstract form of hand information would be a graphic rendering list for a specific piece of graphics hardware.

⁶ In the two person configuration, participants view the simulation from opposite sides of the simulated room.

all participants, the dialog managers exchange the viewing location whenever it is updated, and the dialog passes the position on to the graphics renderer.

A radically different view of the system is provided by window-based menus. These menus provide a superset of the operations afforded by the glove, 3-D tracker and speech input devices by generating events at each of the hierarchy levels -- specific, generic and executive. It was straightforward to add the menu interface without having to modify the dialog rules for Rubber Rocks because of the hierarchically organized event-based structure of the VUE Architecture. The menu interface is typically used for system start-up and diagnostics.

Concurrency

Interactive response time is an important issue in both virtual worlds and real-time conference design. In a centralized architecture, the performance bottleneck is the application attempting to communicate with multiple clients/terminals. It is unacceptable for a real-time conference to experience long pauses whenever it is waiting for input from a participant. It is also undesirable for participants to experience pauses whenever one of the other participants generates input or receives output from the application.

In the case of the VUE architecture, concurrent processing and rapid non-preemptive processing are employed to avoid unacceptable pauses and delays in a conference. All participants have their own dialog manager and I/O devices, so an interaction performed by one person does not impact the interactive response time of the other participants. The I/O devices and the simulation process all service requests from the dialog managers in a non-preemptive FIFO fashion, without blocking, to minimize delays. Since the processes are loosely coupled and event driven, the users do not perceive delays due to additional participants in the virtual world.

Session management

An important set of issues in conference design is when and how do participants join and leave conferences, when are shared applications started, and are the conference controls an integral part of the application. One issue for joining a conference is determining whether a participant can join after the conference has begun. If state information about the application and user interface can be transmitted to a new participant, then it is possible for a person to join a conference after it has started.

In the VUE architecture, the application is a process running independent of the user interface. In general, the dialogs and the application's communication interface are designed to be robust enough to recover should either fail and need restarting. Therefore, it is possible to start the application either before or after the conference begins. Whenever a dialog manager connects to another dialog manager, any shared state information is exchanged so the new participant can join a conference. By default, when all conference participants depart, the application is left running. Subsequently, new participants can connect to the application and resume interaction.

Homogeneous / heterogeneous hardware

The event-based design of the VUE architecture has enabled us to use computers with different clock speeds. The processes asynchronously pass events amongst each other to exchange data. Since the processes are event driven, each process is able to run at its own speed. For example, the 3-D tracking device runs at a higher rate (40 samples/second) than the graphics rendering device (8-20 frames/second).

It is possible to extend the heterogeneity to include different processor architectures. X Windows [Scheifler and Gettys, 1986] is able to run across multiple hardware platforms by using a standardized event protocol that allows data interchange between heterogeneous systems. Similarly, the VUE architecture is also event based, so applications can be developed to run on an heterogeneous collection of computers.

Future Issues

Rubber Rocks is an interesting study of how to create multi-user virtual worlds. Three areas have been identified as needing further research: tools to assist in migrating applications from single-user or batch

processing to multi-user, partitioning of public (shared) and private information, and additional support for joining/leaving a conference.

Two different categories of tools are needed to convert event-based single-user to multi-user applications. For collaboration-unaware applications, a general-purpose bridge is needed to coordinate events from multiple interface dialogs to the application. This bridge should be fairly easy to construct. The other tools are for assisting the construction of multi-user applications, such as real-time simulations, where the application needs to be collaboration-aware. General techniques for handling floor control (access to shared objects) need to be generalized and made available to application developers.

Mechanisms for sharing information need further exploration. There is a general problem of how to determine when information is to be shared among all users, and when it is private to a specific user. In Rubber Rocks, most of the application (content) state information is shared among all participants. This may not be true in other applications where the participants have different roles and the access to the application's state should be restricted based on the participant's conference role. At the interface level, dialogs may choose to share hand information (e.g., location and gesture). However, in some circumstances it is inappropriate to exchange information, such as when the participants want to have independent viewing locations.

Joining and leaving conferences involves the exchange and update of state information, both in the application (content) and the user interface (style) processes. When joining a conference, or connecting to an application that is already running, it is necessary for the user interface to determine the current state of the application. It is desirable for the interface between the application and user interface to automatically negotiate the exchange of state information needed to synchronize the user interface with the application. Also, when a new participant joins an existing conference, the participants' dialog managers should automatically exchange shared style information. The methods and mechanisms for the automatic exchange of state information need to be further developed.

Participants can drop from a conference voluntarily and involuntarily (i.e., a network connection fails). If the participant drops voluntarily, then their dialog can gracefully initiate cleanup before it disconnects from the conference. If the drop is involuntary, the application and other dialogs may need error recovery code to clean up lingering state information. For example, if a user's dialog terminates, that person's hand should be removed from the simulation and the other participants' user interface. Methods for making it easier to handle voluntary and involuntary drops from a conference are needed.

Conclusions

Supporting real-time interactive conferencing in virtual worlds with real-time simulation has been simplified in the VUE architecture by separating the application content from its view through the use of a dialog manager. The application becomes the focal point for sharing application content information. In the case of Rubber Rocks, we converted a non-interactive application into a multi-user real-time simulation. All shared access to the application's content was provided through a new layer of communication software. The centralized approach to running the shared application did permit flexible floor control, and we found that sufficient interactive performance could be maintained. While the addition of the new software layer was necessary for this particular application, the VUE architecture can support sharing of an application through a software bridge so that the application can be collaboration-unaware. The most important aspect for application sharing is that application does not contain user interface (style) information.

The user interface is provided by a number of I/O devices communicating through a dialog manager. Each user having their own dialog manager helps maintain good interactive response time. To share interface information, the dialog managers pass high-level events amongst themselves. Because the interface between dialogs, and between the dialogs and the application, is a set of operations on abstract objects, it is fairly easy to allow each participant to have an independent view of the application.

A number of advantages are accrued by using the VUE architecture that are not available in most prior multi-user applications and real-time conferencing systems. Typically, the applications' user interface input and output data streams are used to synchronize the interface of the conferencing systems. In contrast, the VUE architecture does not scattered the user interface code throughout the application program.

This easily permits changing of the user interface without having to modify the application, and can be run on a set of heterogeneous computers. Since all participants have their own dialog managers and I/O devices, it is possible to allow all participants to have their own view, or interface, to the application. Since communication between the application and the interface is through operations on abstract objects, it is easy to convert applications from being a single-user to being multi-user.

Acknowledgements

We would like to acknowledge Alan Norton, Paula Sweeney and Greg Turk for their work on the flexible object simulation used in Rubber Rocks, and Paula's help in adapting the simulation to be an interactive application.

We would also like to acknowledge Wayne L. Wooten and Jeremy Stone for developing system components. In addition, we would like to thank Ron Frank for his contributions to making the Rubber Rocks demonstrations at CHI'91 and SIGGRAPH'91 possible.

References

- Ahuja, S. R., Ensor, J. R. and Lucco, S. E. A comparison of Application Sharing Mechanisms in Real-Time Desktop Conferencing Systems. *Proceedings of the Conference on Office Information Systems (SIGOIS Bulletin)*, 11(2,3):238-248, ACM, Cambridge, MA, April 1990.
- Appino, Perry A., Lewis, J. Bryan, Koved, Lawrence, Ling, Daniel T., Rabenhorst, David A. and Codella, Christopher F. An Architecture for Virtual Worlds. *Presence*, 1(1), 1992.
- Bacon, R., Gerth, J., Norton, A., Sweeney, P. and Turk, G. Topsy Turvy. *SIGGRAPH Electronic Theater*, 1989.
- Codella, C., Jalili, R., Koved, L., Lewis, J. B., Ling, D. T., Lipscomb, J. S., Rabenhorst, D. A., Wang, C. P., Norton, A., Sweeney, P. and Turk, G. Interactive Simulation in a Multi-Person Virtual World. *Conference Proceedings of CHI'92*, ACM, Monterey, CA, May 1992.
- Crowley, T., Milazzo, P., Baker, E., Forsdick, H. and Tomlinson, R. MMConf: An Infrastructure for Building Shared Multimedia Applications. *Conference Proceedings of CSCW'90*, 329-342, ACM, Los Angeles, CA, October 1990.
- Doug C. Engelbart. NLS Teleconferencing Features: The Journal, and Shared-Screen Telephoning. *Proceedings of the 1975 COMPCON*, Washington D.C., September 1975.
- Douglas C. Engelbart. Toward High-Performance Knowledge Workers. *Office Automation Conference Digest*, 279-290, April 1982.
- W. K. English and Doug C. Engelbart. A Research Center for Augmenting Human Intellect. *Proceedings of the National Computer Conference*, 395-410, IFIPS, 1968.
- Greif, I. and Sarin, S. Data Sharing in Group Work. In Irene Greif, editor, *Computer-Supported Cooperative Work: A Book of Readings*, 477-508, Morgan Kaufmann Publishers, San Mateo, CA, 1988.
- IBM. *Cooperative Viewing Facility: Version 2 General Information*. International Business Machines Corporation, Cary, N.C., 1985.
- Koved, L. CVIEW: A Real-Time Interactive Conferencing System. IBM Research, RC 16138, Yorktown Heights, NY. September 1990.
- Keith A. Lantz. An Experiment in Integrated Multimedia Conferencing. *Conference Proceedings of CSCW '86*, 267-275, Austin, TX, December 1986.
- Lauwers, J. C., Joseph, T. A., Lantz, K. A. and Romanow, A. L. Replicated Architectures for Shared Window Systems: A Critique. *Proceedings of the Conference on Office Information Systems (SIGOIS Bulletin)*, 11(2,3):249-260, ACM, Cambridge, MA, April 1990.
- Lewis, J.B., Koved, L. and Ling, D.T. Dialogue Structures for Virtual Worlds. *Conference Proceedings of CHI '91*, ACM, New Orleans, April 1991.
- Norton, A., Turk, G., Bacon, B., Gerth, J. and Sweeney, P. Animation of fracture by physical modeling. *The Visual Computer*, 7:210-219, 1991.
- Patterson, J. F. The Good, the Bad, and the Ugly of Window Sharing in X. *Proceedings of the Fourth Annual X Technical Conference*, Boston, MA, January 1990.
- Patterson, J. F. Comparing the Programming Demands of Single-User and Multi-User Applications. *Conference Proceedings of UIST'91*, 87-94, ACM, Hilton Head, SC, November 1991.
- Patterson, J. F., Hill, R. D. and Rohall, S. L. Rendezvous: An Architecture for Synchronous Multi-User Applications. *Conference Proceedings of CSCW'90*, 329-342, ACM, Los Angeles, CA, October 1990.
- Rhyne, J.R. Extensions to C for Interface Programming. *Proceedings of ACM SIGGRAPH, Symposium on User Interface Software*, ACM, Banff, Alberta, Canada, October 1988.
- Sarin, S. and Greif, I. Computer-Based Real-Time Conferencing Systems. *Computer*, 33-45, IEEE, October 1985.

- Sarin, S. and Greif, I. Computer-Based Real-Time Conferencing Systems. In Irene Greif, editor, *Computer-Supported Cooperative Work: A Book of Readings*, 397-420, Morgan Kaufmann Publishers, San Mateo, CA, 1988.
- Scheifler, R. W. and Gettys, J. The X Window System. *Transactions on Graphics*, 5(2):79-109, ACM, April 1986.
- Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S. and Suchman, L. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. *Communications of the ACM*, 30(1):32-47, ACM, January 1987.
- Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S. and Suchman, L. Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings. In Irene Greif, editor, *Computer-Supported Cooperative Work: A Book of Readings*, 335-366, Morgan Kaufmann Publishers, San Mateo, CA, 1988.
- Sweeney, P., Norton, A., Bacon, R., Haumann, D. and Turk, G. Modelling Physical Objects for Simulation. *Proceedings Winter Simulation Conference*, Phoenix, Arizona, 1991.
- Wiecha, C., Bennett, W., Boies, S. and Gould, J. Generating Highly Interactive User Interfaces. *Conference Proceedings of CHI '89*, 277-282, 1989.