Two Implementations of Shared Virtual Space Environments

Terrence L. Disz, Robert Olson, Michael E. Papka, Rick Stevens and Matthew Szymanski
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{disz,olson,papka,stevens,szymansk}@mcs.anl.gov

R. James Firby
Department of Computer Science
University of Chicago
Chicago, IL 60637
firby@cs.uchicago.edu

Abstract

While many issues in the area of virtual reality (VR) research have been addressed in recent years, the constant leaps forward in technology continue to push the field forward. VR research no longer is focused only on computer graphics, but instead has become even more interdisciplinary, combining the fields of networking, distributed computing, and even artificial intelligence. In this article we discuss some of the issues associated with distributed, collaborative virtual reality, as well as lessons learned during the development of two distributed virtual reality applications.

1 Introduction

The Futures Laboratory at Argonne National Laboratory has been exploring what is needed to support large-scale shared space virtual environments (VE) for wide-area collaborations. Our research has focused on the system architecture, software design, and features needed to implement such environments. In this article we discuss two prototype systems under development at Argonne.

Shared virtual spaces are complex multiuser online environments that use strong spatial metaphors for navigation, communication and interaction scoping, and object manipulation and may support 3D immersive displays. They allow a direct natural form of collaboration based on the real-world notion of spatial collocation (e.g., many people interacting in a room).

We believe that shared virtual space can be used to effectively support wide-area collaborations. Demonstrations of limited forms of shared space collaborative environments have shown both great potential and considerable limitations of current technologies [3].

A goal of each prototype system is to produce a research implementation that enables the exploration of the following capabilities:

- immersion
- sharing of objects and virtual space
- coordinated navigation and discovery
- interactive control and synchronization
- interactive modification of the environment
- scalable distribution of data

Motivation for these prototype implementations is generated by our interest in shared virtual environments and by the prospect of using these systems to support wide-area scientific collaborations. Additionally, these systems represent the next logical step after the work done on coupling large-scale computing to virtual environments [4]. By studying requirements of shared virtual environment spaces, we can expand our work to support collaboratories and collaborative design. We are already building on technology developed in these two prototype systems for the UbiWorld project [10] (UbiWorld is a shared virtual space that enables users to explore issues related to ubiquitous computing [5]).

Multiuser shared environments have been a topic of research for many years now, from text-based spaces to desktop graphics spaces to immersive virtual reality [2, 8]. Other groups have focused on what is needed for distributed collaborative environments [6, 9].

2 Experimental Environment

In this section we introduce and discuss the implementation of the Interactive Agent Environment and ManyWorlds. The discussions will include an outline of the architecture, implementation, and discoveries.

Both implementations use the CAVE family of display devices. The CAVE provides a wide variety of display options, ranging from the desktop to the fully immersive four-wall CAVE environment [1].

2.1 Interactive Agent Environment

The Interactive Agent Environment (IAE) system touches on each of the capabilities outlined in the introduction. In addition, this prototype implementation of IAE provides the architecture for testing and evaluating the separation of representation from behavior and computation. It allows for intelligent objects to be designed without concern for how they will be represented. It also adds a dynamic nature to the virtual world. While not designed as a collaborative space, IAE supports collaboration by the fact that it is a shared environment. The IAE system allows for an arbitrary number of display devices to connect to the world server and participate in the shared virtual space. An arbitrary number of computational entities are also allowed to connect to the world server, providing dynamic behavior within the virtual world.

Additionally, the IAE prototype environment allows for exploring the use of artificial agents within the virtual world. These agents could be used to annotate the virtual world and act as helpers to users. The artificial agent can act as a tour guide or help filter/navigate data.

2.1.1 Architecture

Figure 1 shows a high-level overview of the architecture of the IAE system. The IAE system has three major components: a world server, display devices, and active objects. Each of the three components can be run on different machines, and multiple instances of the display devices and active objects can be invoked.

The world server acts as the central connection point within IAE. The world server supports the loading of VRML representations of objects into the virtual world, communication, management of the environment, and sensing for active objects. An important notion within IAE is that of active and inactive objects. Active objects are objects associated with distributed computational processes and will be discussed in more detail below. Inactive objects are all other objects within the virtual world that add to the realism.

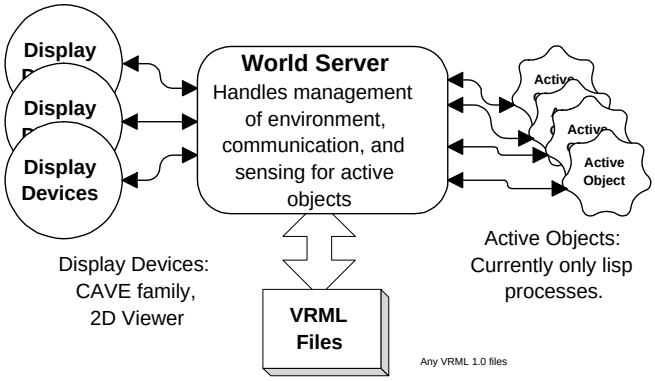

Figure 1. Architecture Overview of the Interactive Agent Environment

The display devices provide the users a view into the virtual world. Currently there are two supported display devices: the CAVE family of VE displays and a simple two-dimensional viewer. The CAVE viewers enable the user to experience the world in its full three-dimensional representation. The two-dimensional version is a top-down view on the world.

A unique feature of IAE is the use of computational entities to control the active objects. Active objects are processes running on the same system as the world server or on a remote system. Active objects implement behaviors associated with the graphical representations of objects in the world. This separation allows active objects to be associated at runtime with a variety of different graphical shapes.

Example: if the active object is a follower (an agent that follows some given object), it will—based on the size and shape of its graphical representation—be able to navigate in the appropriate manner (i.e., small object fits through narrow opening; large object goes around).

Active objects in IAE are currently written in Lisp, but other languages can be supported as long as they implement the world server connection model.

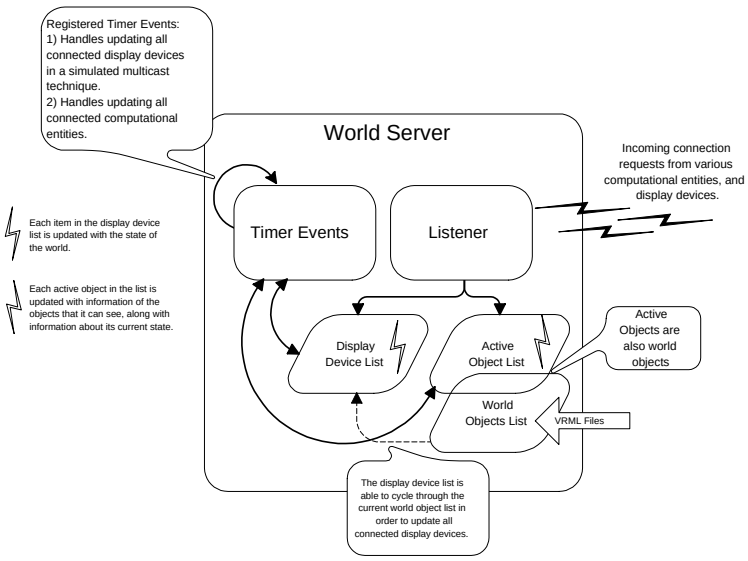

Figure 2. World Server Diagram

2.1.2 Implementation

The world server is constructed from a variety of different object-oriented patterns implemented in C++. These patterns represent various components within the world server. The world server handles the management of display and computation connections (see Figure 2). Additionally, the world server manages physical properties of the world and the sensing for active objects.

The distributed nature of the IAE system relies heavily on communication and on the ability to allow connections from a variety of sources at varying times. At the center is the ACE library: a toolkit that implements fundamental design patterns for communication software and, in particular, the reactor object [11]. The reactor object is a design pattern that supports the demultiplexing and dispatching of multiple event handlers [12]. It is an event-driven object capable of handling multiple connection requests by invoking the appropriate event handler at connection time. As each connection occurs, an individual communication channel is established between the server and the connecting client, be it a display device or a computational process.

The communication channel between the graphical display devices and the world server is principally one way, with the world server streaming the list of objects needing to be rendered each update. If the user wants to move from the role of passive observer to that of an active one, the display device must also stream information on position and orientation of the user back to the world server.

The communication channel between the computational processes (active objects) and the world server is a two-way channel, with information about the position and objects that an individual process can sense being sent out by the world server. The world server then receives back information for all the active objects' physical parameters. Currently these values are rotation and translation speeds, but in the future could include joint angles and speeds, etc.

The ACE reactor plays a role in the computation process of the world server. By using the event-handling capabilities of the reactor object, timer events are registered with the reactor at startup. These timer events, which happen at regularly scheduled intervals, update the graphics devices and the world simulation, as well as send the appropriate information to the active objects.

The active objects are driven by remote processes currently written in Lisp and running on a remote machine (see Figure 3). The world server reports to each active object, via its own private communication channel, the various objects (both active and inactive) it can see, as well as that active objects' location within the world. Based on this information and on its own goals, the active object process then decides what its next course of action should be. Currently, the active objects only try to avoid other objects while moving toward some predefined location. In general, the active objects could be any computational source that followed the specified format for connections and output. It should be noted that the information sent from

the world server to the active object described above is particular to that instance of an active object. Other active objects may need to know only if a user is within a certain range or if someone is holding an active object.

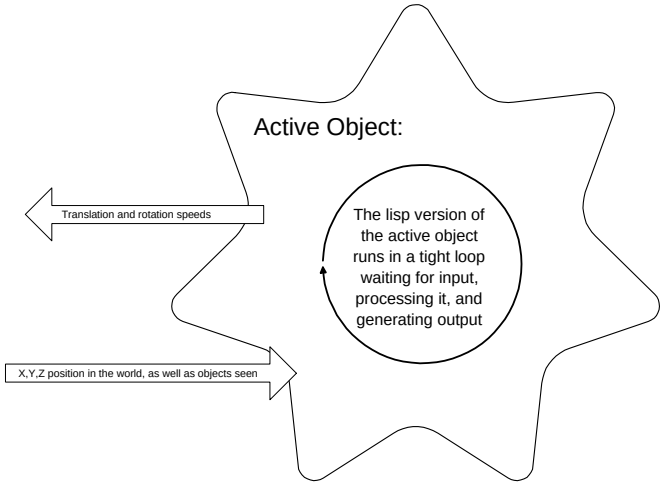

Figure 3. Active Object Diagram

The graphics capability of the IAE system is flexible, depending on the user's needs and capabilities. The different display devices that connect to the world server can be thought of as browsers. A user can connect to the IAE server via a two-dimensional browser that gives a top-down projection of the world. The two-dimensional browser is useful in debugging the system but also allows users with low-end hardware capabilities a way to observe the world and the interactions within it. A virtual reality browser is built on top of the CAVE library. The VR browser displays the objects in the world based on VRML files associated with each object. The server currently streams VRML files at regular intervals to each of the attached browsers. This process is extremely inefficient, particularly for the inactive objects, which remain fixed within the environment. A caching scheme is being developed to allow the server to update only those objects that have changed within the environment. In general, users maintain a passive representation within the virtual environment. This means the users are passive observers, not represented by avatars and not seen by other users connected to the virtual world. Users can be represented in the world by sending back their world coordinates and orientations to the world server, which then manages the user as a active object within the world.

2.1.3 Next Step

IAE depends on VRML for the three-dimensional representation of the objects and is currently using the VRML 1.0 specs. With a move to VRML 2.0, inactive objects could have simple yet interesting behaviors embedded in them by using the new 2.0 scripting features. Since the two-dimensional browser already generates its views based on the VRML files, a Java-based version can be built without much effort to allow connections from Java-enabled Web pages. Finally, in the the area of display devices, hooks could be added to allow for true Web-based VRML browsers to attach to the world server.

The most exciting future work will come in the area of active objects and their associated computational entities. Using the prototype system, one can start to build artificial agents that enhance the environment. One example would be a virtual librarian that is able to help users navigate a large database, leading the users to areas of interest. A second example would be a virtual cameraman, whose task is to record and archive the users' experiences within the environment.

Beyond the use of the active objects to model artificial agents, the active objects could be supercomputing simulations of various events that attach to the world server. Not only does this add to the richness of the virtual world, but it provides substance for the artificial agents to work on and explore.

2.2 ManyWorlds

The ManyWorlds system is a prototype implementation of an architecture for managing multiuser shared virtual reality experiences. The intent of the ManyWorlds architecture is to allow arbitrary applications to contribute content to a ManyWorlds session, to allow multiple users to connect to a ManyWorlds session using clients with a wide variety of capabilities, and to do so in a scalable manner. A long-term goal of ManyWorlds is to provide a scalable tele-immersion environment to support collaborative work.

2.2.1 Architecture

The basis for a shared space in ManyWorlds is an abstraction we call the stage. All visualization and interaction in ManyWorlds take place in a stage; a ManyWorlds session may have multiple stages, and users may switch from stage to stage at will. The entities visualized in the stage take the form of VRML objects, as in the Interactive Agents Environment. We will refer to these VRML objects as the content present on the stage.

A stage is viewed by a ManyWorlds browser. When a browser is directed to a stage, it will begin receiving updates of the content of the stage. Currently, two browsers are implemented in the ManyWorlds prototype. One is a CAVE browser, implemented by using Open Inventor and the CAVE library. It allows the user to navigate through the space managed by the stage, visualizing the objects in the stage in three dimensions. The other browser is a Web gateway, which allows connections from traditional Web-based VRML browsers.

Figure 4 provides an overview of a typical ManyWorlds session. At the center of the session is the stage. The stage mediates the transfer of data between content sources and sinks. Sources of data shown here include simulation, CAVE clients, static scenery, and an archival playback session. Sinks of data include the CAVE clients, a Web gateway, and an archival recording session.

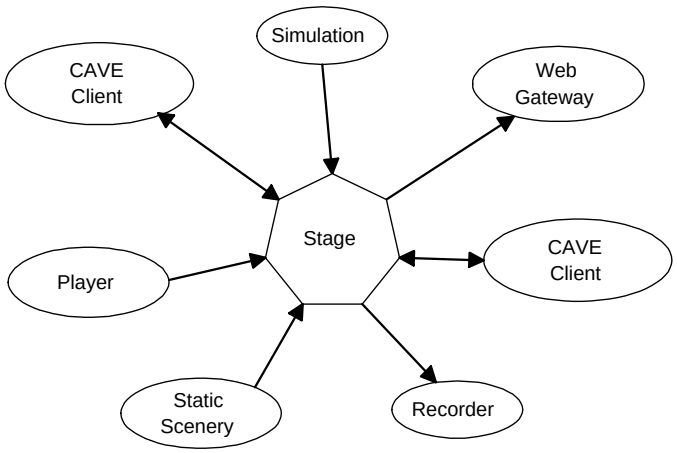

Figure 4. A Typical ManyWorlds Session

Clients connecting to a stage may contribute content to a stage, monitor the contents of a stage, or both. Examples of clients contributing content include scientific applications modified to generate their output as VRML data sets and applications that provide sets of static or slowly changing VRML objects as background scenery for a stage. Clients that monitor the content of a stage include the various browsers that users use to visualize a stage and world recorders that archive the interactions in a stage for later review.

2.2.2 Implementation

The current ManyWorlds prototype is implemented largely in Perl, making heavy use of the object-oriented programming mechanisms in the Perl language. Hence, it is natural to use the distributed object programming paradigm in the implementation of the communication between the components of the ManyWorlds system.

Communications between objects residing in different processes is achieved by the use of a Perl binding of the Nexus runtime library. We refer to this combination of Perl and Nexus as nPerl. The nPerl system handles the marshalling of method call arguments on the sending system and their unmarshalling on the remote system, as well as handling the actual invocation of the remote method call and the return of its return value to the caller. This marshalling leverages the support Nexus provides for passing data among a heterogeneous collection of computers.

The use of nPerl also provides a clean mechanism for supporting the dynamic nature of the ManyWorlds system. Through the Nexus dynamic attachment mechanism, a new observer or data provider can connect to a stage and take part in the action there. Through nPerl we can also cleanly and robustly handle abnormal termination of parts of the ManyWorlds system.

The communications involved in the distribution and update of VRML objects from content providers to observers is most naturally cast as a form of multicast. Each provider maintains a multicast group into which it injects updates of the data it is providing. Observers that wish to receive updates from a provider subscribe to the provider's multicast group when they join the stage. Information about the set of providers present on a stage is propagated via a multicast group managed by the stage itself.

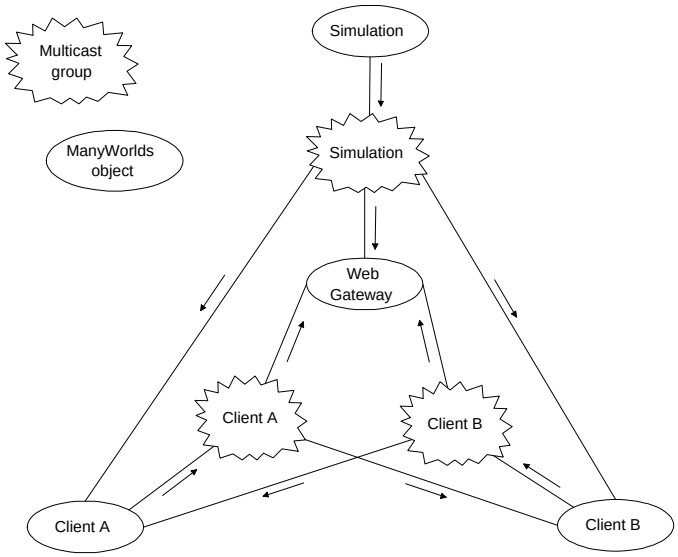

Figure 5. Data Flows in ManyWorlds

The current ManyWorlds prototype uses a simple,

replicated unicast implementation of multicast (see Figure 5). This method has the advantage that it makes no assumptions about the network infrastructure on which the system is executing. However, it scales poorly. The architecture has been designed such that alternative implementations of multicast communications can be added to the system with minimum disruption. Providers also have the freedom to specify the form of multicast implementation required for their particular needs. For example, the avatar provider in a CAVE viewer would likely want to have a reliable multicast distribution of the VRML description of the avatar, but the position updates of the avatar can use an unreliable multicast service.

The ManyWorlds architecture defers the actual computation involved in the application to the content providers. Content provider applications interact with a ManyWorlds session via an API that provides the functionality required for the application to join a stage, announce its presence, and supply data to any observers.

An example of a ManyWorlds content provider is the "world in a directory" client. This client scans a specified directory looking for VRML data files. When a file appears, it is added to the set of VRML data that is supplied to the ManyWorlds session. Changes to the file result in VRML data updates in the session.

The directory client has been used to interface a PETSc application [7], running on a multiprocessor workstation, to a ManyWorlds session. The PETSc application itself knows nothing of ManyWorlds; it is simply configured to place its output data in VRML format in the directory being scanned by the directory client. The result, as seen by a ManyWorlds browser, is the output of the simulation as it evolves over time.

We have implemented two different ManyWorlds browsers. The first is a CAVE library-based application with an nPerl front end that communicates with the rest of the ManyWorlds system, and a C++ and Open Inventor-based backend that handles the caching of VRML objects for rendering, as well as user input and navigation. This client acts as both a data viewer and a content provider. The content served to the rest of the system consists of a user-defined avatar, placed in virtual space according to the user's position in the CAVE itself, and the input of the navigation system within the application.

The second client is a World Wide Web gateway. The gateway acts as an HTTP server as well as a data viewer. When a request for the VRML page representing the stage arrives, the gateway composes the VRML objects currently in the world into a single VRML page suitable for viewing with a Web browser. It also inserts VRML camera definitions corresponding to the locations of any CAVE users, allowing the viewer of the stage via the Web to jump to the viewpoint of any of the CAVE users.

2.2.3 Next Step

Nothing in the ManyWorlds architecture, other than the display engines, restricts the data being shared to VRML. We anticipate using non-VRML media to augment the basic VRML structure of a shared VR session. For example, the CAVE browser could advertise audio and video streams to the world. Clients capable of viewing these media would negotiate with the browser to receive the streams. This is an example of the power of the abstract, application-level multicast scheme. In this case, the application would likely use IP multicast as the implementation of the abstract multicast, leveraging the existing multicast toolset.

An important method of visualizing scientific data sets is volume visualization. This is a very computationally expensive procedure to use, but is possible on the hardware used in the CAVE. Thus, we would like to add support for volume visualization as another alternative ManyWorlds medium. We anticipate that a scientific application might provide alternative forms of its output data: volume datasets for the high-end CAVE clients, and a less-detailed VRML dataset for other clients.

An important part of collaboration is the use of history. We have designed the ManyWorlds architecture such that it would be possible to transparently archive a ManyWorlds session. A recording application could join a stage as a data viewer and record all interactions in the stage. Later, a playback application could create a stage for the playback of the session and configure itself as proxy data provider for all of the original data sources. The viewers of the system would be able to navigate through the playback, observing the previous objects and interactions, as well as interacting with the other viewers of the playback.

Another important means of collaboration is what some call the show-me style: a knowledgeable user of an application can guide others through the intricacies of the application to find the areas of interest. We plan to implement a form of this capability in the CAVE browser as a flexible means of navigation, where the user can slave his viewpoint to that of another user. We will also investigate the use of active objects in the world that can have an effect on the navigation of the users browsing near the space. A simple example is gravity: we may wish to define a region in the virtual space that affects the navigation of users passing

through the space as gravity does. Another example would be to give the users of a flow-field simulation the ability to attach themselves to a particle in the flow and follow its path through the field.

We would like to allow a very rich interaction between users of the virtual space and the objects in the space. Because the architecture of the ManyWorlds system is based on a flexible distributed object system, we can define arbitrarily complex behaviors between objects on a stage and between the browser and objects on the stage. As a simple example, we could cause the CAVE wand's selection of an object to trigger a behavior defined by the object. We could also define shared user interface objects, such as popup menus, that have a form in the three-dimensional shared space. Support for the movement of objects in the space also falls into this category.

3 Conclusions

We have begun the design and implementation of two shared space systems. The two systems share many architectural concepts and features, and both are integrated with the CAVE environment; however, they have different goals. The Interactive Agent Environment supports development and experimentation with active objects and cooperative tasking and provides a linkage to intelligent systems technology that can be used to augment the shared environment. ManyWorlds is focused on scalability and exploring the communications and VR software infrastructure to support the rapid construction of spaces for collaborative data analysis, design, and learning. Neither system has all the features we envision for an ultimate production system.

Our near-term goal is to explore a variety of implementation strategies and mechanisms that can support large-scale collaborative environments. A longer-term goal is to develop a common software base for building a variety of shared space environments and to explore their use in large-scale scientific applications.

Acknowledgments

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38, and by Laboratory Director Research and Development funding.

References

[1] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In Computer Graphics (Proceedings of SIGGRAPH '93), pages 135–142. ACM SIGGRAPH, August 1993.

[2] P. Curtis and D. A. Nichols. Muds grow up: Social virtual reality in the real world. In Proceedings of the 1994 IEEE Computer Conference, pages 193–200, January 1994.

[3] T. A. DeFanti, M. D. Brown, and R. Stevens, editors. IEEE Computer Graphics and Applications (Special Report) – VR Over High-Speed Networks, volume 16 (4). IEEE Computer Society, 1996.

[4] T. A. DeFanti, M. E. Papka, and R. Stevens, editors. The International Journal of Supercomputer Applications and High Performance Computing (Special Issue) – I-WAY: Wide Area Supercomputing Applications, volume 10 (2/3). Sage Scientific, 1996.

[5] T. L. Disz, M. E. Papka, and R. Stevens. Ubiworld: An environment integrating virtual reality, supercomputing and design. In Proceedings of the Heterogeneous Computing Workshop, April 1997. to be published.

[6] I. Foster, M. E. Papka, and R. Stevens. Tools for distributed collaborative environments: A research agenda. In Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Compututing, pages 23–29. IEEE, IEEE Computer Society Press, 1996.

[7] W. D. Gropp and B. F. Smith. Scalable, extensible, and portable numerical libraries. In Proceedings of the Scalable Parallel Libraries Conference, pages 87–93. IEEE, 1994.

[8] O. Hagsand. Interactive multiuser VEs in the DIVE system. IEEE Multimedia, 3(1):334–232, 1996.

[9] J. Leigh and A. Johnson. Supporting transcontinental collaborative work in persistent virtual environments. IEEE Computer Graphics and Applications, 16(4):47–51, July 1996.

[10] M. E. Papka and R. Stevens. Ubiworld: An environment integrating virtual reality, supercomputing and design. In Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing, page 306. IEEE Computer Society Press, August 1996. extended abstract.

[11] D. C. Schmidt. The adaptive communication environment an object-oriented network programming toolkit for developing communication software. In Sun Users Group Conference, December 1993.

[12] D. C. Schmidt. Pattern Languages of Program Design, chapter Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching. Addison-Wesley, 1995.