CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments

Jason Leigh (jleigh@eecs.uic.edu), Andrew E. Johnson
Thomas A. DeFanti
Electronic Visualization Laboratory
University of Illinois at Chicago

Abstract:

CAVERN, the CAVE Research Network, is an alliance of industrial and research institutions equipped with CAVE s, ImmersaDesks, and high-performance computing resources all interconnected by high-speed networks to support collaboration in design, training, scientific visualization, and computational steering, in virtual reality.

CAVERNsoft is the common collaborative software architecture for CAVERN. CAVERNsoft uses light-weight distributed data-stores as the mechanism for managing a wide range of data volumes (from a few bytes to several terrabytes) that are typically needed for sustaining persistence in virtual environments. Multiple networking interfaces support customizable- latency, data consistency, and scalability that is needed to support a broad spectrum of networking requirements.

This paper begins by describing a number of collaborative virtual reality scenarios developed by our laboratory as well as our collaborators, with the goal of identifying the key issues of networking and database management that are unique to collaborative virtual reality. From these the CAVERNsoft architecture is presented including a report on the current status of CAVERNsoft's development.

Keywords: Scalable Persistence Collaborative Virtual Reality

1 Introduction

Collaborative Virtual Reality (CVR) is currently one of the most challenging areas of research in Virtual Reality (VR.) Collaboration adds a new dimension of complexity to human-factors, networking and database research in VR. For example, human-factors research in VR has traditionally focused on the development of natural interfaces for manipulating virtual objects and traversing virtual landscapes. *Collaborative* manipulation, on the other hand, requires the consideration of how participants should interact with each other in a shared space, in addition to how co-manipulated objects should behave. Other issues include: how participants should be represented in the collaborative environment; how to effectively transmit non-verbal cues that real-world collaborators so casually and effectively use; how to best transmit video and audio via a channel that allows both public addressing as well as private conversations to occur; and how to sustain a virtual environment even when all its participants have left.

Naturally CVR poses new challenges to traditional areas of networking and databases as well. A Collaborative Virtual Environment (CVE) requires an unconventionally broad range of networking, database and graphics capabilities. This vast range makes the rapid construction of complex CVEs difficult. Current attempts at building networking and database architectures for CVEs have resulted in ad-hoc solutions that are specifically designed to solve a small range of problems. These ad-hoc solutions typically use only a single protocol for data transmission- either reliable or unreliable but rarely both. There is no uniform means (network and database) for interoperability between virtual environments and non-virtual environments (e.g. Java-based desktop workstations.) There is little middleware to provide higher-level services for collaborative VR (avatars, audio/video teleconferencing, database) that can allow developers to ``jump-start'' the construction of a

collaborative environment.

CAVERNsoft is a software architecture for supporting CVR that attempts to be cognizant of these diverse human-factors, networking and database issues. Its main goal is to explore the problem of supporting persistent CVEs for collaborative, education, and scientific and engineering visualizations that involve supercomputers as well as massive data stores.

In this paper we will describe a number of scenarios that have helped motivate the design of CAVERNsoft by identifying a number of key issues that a software architecture for CVR must support. We will then propose an architecture for CAVERNsoft and conclude by describing its progress at the present time.

2 Representative Collaborative Virtual Reality Scenarios

The following scenarios describe representative CVEs in several domains. Although these scenarios may not fully represent all possible scenarios that will arise in the future of CVR, they are chosen because they are either historically illustrative- or are currently illustrative- of problems that are actively being researched.

One constraint this paper attempts to impose on the set of scenarios, is that they involve tasks that would benefit from a solution in CVR over simply non-collaborative VR or 3D workstation computer graphics. For example, simple audio/video teleconferencing alone is not considered a scenario that can significantly benefit from the use of CVR. However collaborative work that depends on the spatial qualities of VR (such as collaborative architectural design) in addition to teleconferencing as part of its solution (teleimmersion) *is* considered a good candidate for a CVR solution.

2.1 Collaborative Design and Engineering

Collaborative design work in VR typically involves a small group of users, either synchronously or asynchronously, engaged in the construction, and manipulation of objects in the virtual world. Since the interfaces for three-dimensional modeling in VR are still relatively imprecise compared to 2.5D CAD packages, most of the collaborative tasks in collaborative design involve evaluations of the design, and to a lesser degree, redesign or brainstorming for new design possibilities[Leigh et al., 1996b, Leigh et al., 1996a, Leigh and Johnson, 1996].

The National Computational Science Alliance (NCSA) has been working with Caterpillar Belgium S.A., to develop a system to allow remotely located engineers to work together on vehicle design review and redesign[Lehner and DeFanti, 1997]. Remote collaboration is necessary here because the eventual system will be used by Caterpillar engineers in the U.S. and Europe who must jointly design Caterpillar vehicles so that they meet customer demands and safety requirements for both markets. For example, European safety standards require a roading fender to be added to the basic vehicle design. The collaborative VR system allows engineers to evaluate rearward visibility from a viewpoint in the virtual cab of the vehicle. Virtual co-presence allows one designer to manipulate the fender while another designer watches for its effect on visibility.

To support user-to-user communication, publicly available audio and video teleconferencing tools (**vat** and **nv** respectively) were modified to work with the CAVE virtual environment[Cruz-Neira et al., 1993]. Video images from each participant were texture-mapped onto the surface of a rectangular box to establish their presence in the environment. The 3D models of the Caterpillar vehicles that are used in the collaboration are first duplicated at every site. Then an unreliable multicast data stream is used to distribute information about the participants and changes in the models to all the other participants.

2.2 Collaborative Training

The earliest CVR systems were military-based applications such as SIMNET and NPSNET [Locke, 1995, Macedonia and Zyda, 1995, Macedonia et al., 1995]. SIMNET is a standard for distributed interactive simulations developed by DARPA beginning in 1985. The purpose of SIMNET was to facilitate early phases

of training at a cost far below the expense of conducting real battlefield exercises. A SIMNET participant may be wearing a head-mounted display and standing on a tread-mill to train as a foot-soldier. Alternatively another SIMNET participant may be sitting in a tank simulator. Typically, SIMNET expects hundreds of participants to be engaged in the simulation at the same time. To reduce the bandwidth and the effects of latency needed to sustain this degree of scalability SIMNET uses a technique called dead-reckoning to predict the location of participants at any instant in time based on their previous reported position and velocity and acceleration.

As SIMNET was designed primarily for military simulation, its underlying unit of data transmission (called a Protocol Data Unit- PDU for short) specifically contains encodings for military entities (such as tanks and airplanes.) DIS (Distributed Interactive Simulation) is a newer and more ambitious simulation standard (IEEE 1278) that is based on SIMNET but allows for greater complexity and realism. For example: SIMNET uses a flat terrain whereas DIS accounts for the curvature of the Earth. SIMNET is oriented towards terrain and the sky above it whereas DIS encompasses all areas of potential military activity including below the ocean and in space.

2.3 Collaborative Scientific Visualization

A typical scenario in collaborative scientific visualization is for a small group of scientists that are remotely located, to enter a virtual environment to discuss a data set that is being visualized. This data set may originate from a database or may be computing simultaneously on a supercomputer, in which case the virtual environment can be used to steer the computation[Roy and Cruz-Neira, 1995]. The importance of collaboration in this environment is not so much in allowing the remote participants to perform different tasks simultaneously as it is to allow them to offer their different opinions over what is observed in the visualization. The default assumption is that all the participants should share a homogeneous view. However for a complex data set that spans numerous dimensions, it may be more useful to partition the dimensions so that different virtual environments observe different dimensions during the simulation.

Argonne National Laboratory (ANL) in collaboration with Nalco Fuel Tech have built an immersive interactive engineering tool for designing pollution control systems for commercial boilers and incinerators[Freitag et al., 1995]. Using ANL's CAVEcomm library multiple CAVEs could synchronously connect with an IBM SP supercomputer to steer the interactive simulation of flue gas flow in the boiler. Control of the simulation was strictly via turn-taking. One participant could initiate the flow from one viewing location while another participant could simultaneously view the flow in a different chamber of the boiler. Participants could communicate with one another via a conference telephone call.

As with most CVR applications, this system is only in a prototypical phase. Additional capabilities that may be useful in enhancing work in the environment include:

1. Discovery Recording - the ability to annotate (perhaps using voice recording) to mark points of interest in the data set- storing the annotation, and the state of the environment when the ``snapshot'' was taken. This will allow the engineers to return to the time of the event and re-observe it.
2. Storing Computed or Raw Data Sets - typically the data generated by a simulation or gathered from data-gathering devises are too large to fit into the physical memory of the computer performing the visualization. In this case some scheme of hierarchically storing this data is needed to allow querying for smaller subsets of the data for visualization.

2.4 CALVIN and NICE - the Pre-history of CAVERNsoft

CAVERNsoft is a continuation and generalization of the work we have done at the Electronic Visualization Laboratory on several collaborative virtual environments in recent years. Two of these environments: CALVIN and NICE, provided testbeds to prototype several of the ideas that would eventually form parts of CAVERNsoft.

2.4.1 CALVIN - Collaborative Architectural Layout Via Immersive Navigation

CALVIN [Leigh et al., 1996a, Leigh et al., 1996b, Leigh and Johnson, 1996] is a CVE that allows multiple users to synchronously and asynchronously experiment with architectural room layout designs in the CAVE

(Figure 1.)

Participants are able to move, rotate, and scale architectural design pieces such as walls and furniture. Participants may work as either ``mortals'' who see the world life-sized (classically known as an ``inside-out'' view), or as ``deities'' (``outside-in'' view) who see the world as if it were a miniature model. Deities by virtue of their enlarged size relative to the environment, tend to tower above the scene and are better at performing gross manipulations on objects. Mortals on the otherhand are at the same scale as the environment, and are hence better able to perform fine manipulations.

Asynchronous access allows designers to enter the space whenever inspiration strikes them, rather than requiring them to wait to schedule formal meetings, which can be particularly difficult if the participants are located at opposite parts of the world with significant timezone differences. In fact CALVIN already provides interfaces for bilingual (Japanese and English) interaction.

Participants are able to save versions of the design as the collaboration progresses. When participants re-enter the environment at a later time, the most recently saved version is automatically loaded. If on the otherhand the participant re-loads a different version of the design, CALVIN will record successive designs as a new branch in the version tree.

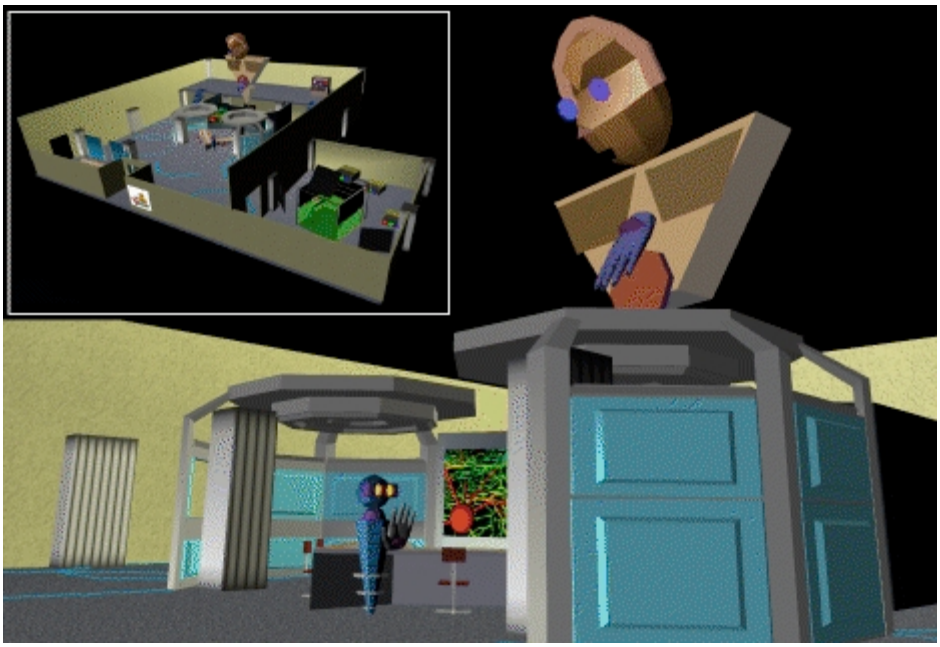

Figure 1: CALVIN: a collaborative design environment for architectural layout. The scene shows two avatars (a tall one and a short one) viewing the space at different perspectives. The top lefthand inset of the top image is a zoomed-out view of the entire design space. The lower image shows one of the avatars using CALVIN's Japanese interface.

CALVIN employees a shared variable model of a distributed shared memory (DSM) system to eliminate the need of the programmers to develop specific protocols for network communication. The DSM itself uses a reliable protocol and a centralized sequencer to guarantee consistency in all clients. C++ classes representing networked versions of floats, integers and character arrays are provided so that assignment to variable instantiations of these classes automatically shares the information with all the remote clients.

These networked variables are used to send data such as the state of objects in the world and user-tracker information. Tracker information is sent so that avatars can be drawn in the place of participants in the virtual scenes. Position as well as orientation data from the user's hand and head are transmitted so that fundamental gestures such as nodding, pointing, and waving can be communicated through the avatars.

Although the task of world synchronization is greatly simplified by the centralized sequencer, the transmission of tracker information over such a reliable channel can introduce latencies- especially when synchronizing between the participant's real location and their avatar's location. This is acceptable for small relatively closely located working groups where the network traffic and latency is relatively low but is unsuitable for larger and more distant groups of participants dispersed over the internet. In fact, to transmit audio/video signals between sites, the shared memory system is bypassed with point-to-point raw ATM streams which are able to support teleconferencing at NTSC resolution and at 30 frames per second.

Finally, in CALVIN when two or more participants simultaneously modify an object, a ``tug-of-war'' occurs where the object appears to jump back and forth between two positions, eventually remaining at the position given to it by the last person holding onto it. This problem can be alleviated by using a locking scheme, but this was intentionally **not** done. In VR, where emphasis is placed on natural interaction, it would be unnatural if the user had to lock an object before picking it up. The presence of avatars in combination with audio communication (the most important of the communication channels to provide) compensated for the lack of strict floor control and database locking. For example, the declaration: ``I'm going to move this chair'' combined with the visual cue of an avatar standing next to a chair and pointing at it, alerts other users that this user is about to grab that chair.

CALVIN's centralized topology and distributed shared memory implementation informed us on the potential and usefulness of this architecture for supporting CVR. Our next goal was to investigate the issues involved in

developing CVEs that use highly distributed techniques. These investigations are embodied in the architecture developed for the NICE project.

2.4.2 NICE - Narrative Immersive Constructionist/Collaborative Environments

The NICE group is building a collaborative environment in the form of a virtual island for young children (approximately 6-8 years of age)[Roussos et al., 1996, Roussos et al., 1997]. In the center of this island the children can tend a virtual garden. The children, represented by avatars, collaboratively plant, grow, and pick vegetables and flowers. They ensure that the plants have sufficient water, sunlight, and space to grow, and need to keep a look out for hungry animals which may sneak in and eat the plants. The children can shrink down to the size of a mouse and crawl under the garden to see the root system, and can talk with the other remotely located children or other characters in the scene. The children are able to modify the parameters of this small ecosystem to see how it affects the health of the garden (Figure 2.)

Figure 2: NICE: a narrative immersive collaborative environment for education. The top scene shows an avatar handing a flower to another avatar in the NICE garden. Below is an image of a child interacting with an avatar in the CAVE.

NICE's architecture is based on the techniques derived from CALVIN in that a central server is used to maintain consistency across all the participating virtual environments. Whereas CALVIN solely used a reliable connection to synchronize state information, NICE used an unreliable protocol (either multicasting or UDP) to share avatar information from magnetic trackers, and a reliable socket connection to share world state information and to dynamically download models from WWW servers using the HTTP 1.0 protocol.

Both multicasting and UDP were provided to deliver tracker data, as it was not always possible to acquire the administrative privileges to conveniently erect multicast tunnels between distant remote sites. Hence a number of interconnected NICE ``smart-repeaters'' were deployed at various remote sites that allowed the use of multicasting amongst clients at localized sites but UDP for repeating packets between remote locations. In addition, to prevent faster clients from overwhelming slower clients with data, the smart-repeaters performed dynamic filtering of data based on the throughput capabilities of the clients. Using this scheme we have participants running on high speed networks have been able to collaborate with participants running on slower 33Kbps modem lines.

NICE's virtual environment is persistent. That is, even when all the participants have left the environment and the virtual display devices have been switched off, the environment continues to evolve; the plants in the garden keep growing and the autonomous creatures that inhabit the island remain active.

Interactions with the NICE garden are not limited to users with VR hardware. The garden in NICE can be experienced either by entering VR, a basic WWW browser (http://www.ice.eecs.uic.edu/~nice), a VRML2 browser, or in a Java applet. Participants using a mouse can interact with participants using VR hardware where

the desktop user's mouse position is used to position an avatar in the 3D virtual world, and the bodies of the VR users are used to position 2D icons on the desktop screen. This kind of scalability will be important for increasing the breadth of possible collaborations.

3 The Particular Requirements of Collaborative Virtual Reality

The above scenarios illustrate the broad spectrum of human-factors, graphics, networking, and database requirements that are needed to support CVR. These requirements are elaborated in the following sections.

3.1 Avatars

When collaborating in VR, virtual representations are needed to uniquely identify each participant. The popular default assumption for representing avatars in VR is to place texture-mapped, live video images on the head of a three-dimensional model of a human. The problems with this scheme are manifold: the bandwidth required to transmit video may be too high to ``waste'' on sending facial images; most VR experiences require the participant to wear some form of head gear which will typically occlude most of the participant's face making video images of faces impractical; attaching a video camera and light source, however small, in an optimal position to capture images significantly increases the incumberances already inherent in the VR gear.

The elaborateness of the avatar should vary with the task being performed. Hence it is important to identify the minimum elements of representation needed to afford recognizability and to convey non-verbal information such as body language and gesture. In our experience we have found a minimum of head position and orientation, body direction, and hand position and orientation to be adequate for many CVR tasks. To afford recognizability, we have found it easier to distinguish avatars based on geometry rather than color. Hence the commonly used, homogeneously shaped avatars with varying colors and overlayed name tags, do not make good avatars.

To support the minimal avatar, a bandwidth of approximately 12Kbits/sec (at 30 frames per second) is needed. Theoretically this implies that 10 avatars can be supported over a 128Kbits/sec ISDN connection. In practice however, our experiments have shown that it is able to support a maximum of four avatars with an average latency of 60ms using UDP as the transmission protocol. Although this is not a scalable solution, it is a cost effective means of transmitting VR avatar data with the quality of service of a dedicated connection.

3.2 Suitable Interfaces for Collaborative Manipulation and Visualization

High-level virtual interfaces must be developed to allow collaborative manipulation of shared objects. In addition, these manipulation tools require some form of locking to occur so that consistency is maintained across all the virtual environments sharing the virtual space. The goal is to provide mechanisms for acquiring distributed locks (possibly through predictive means) so that the user does not realize that locks have had to be acquired before objects could be manipulated. This is particularly important over high latency networks where there might be a noticeable delay between the time when a user physically picks up an object (and hence attempting a lock on it,) to the time when the VR system confirms the lock on the object. Lag similar to this has been shown to significantly degrade human performance in a VR environment[Ware and Balakrishnan, 1994]. In our experience we have found that for coordinated VR tasks involving two expert VR users, performance begins to degrade when network latency increases above 200ms[Park, 1997]. Other research has found acceptable latencies to be much lower (100ms)[Macedonia and Zyda, 1995]. The acceptable latency is expected to be lower for inexperienced users and for coordinated tasks involving very fine manipulation of shared objects. In the latter situation tracker inaccuracy will also begin to affect human performance.

3.3 Audio/Video Teleconferencing

Audio (voice telephony) is one of the most important channels to provide in a collaborative experience[Fish et al., 1990, Tang and Isaacs, 1993]. It has been shown that latencies of greater than 200ms will result in degradations in conversion[Fish, 1996]. As the latencies continue to increase the amount of time spent in confirming conversion increases, and the amount of useful information being conveyed in the conversation decreases. Video conferencing is useful in instances where it is important for the participants to see each other face to face for negotiation tasks[Argyle and Cook, 1976, McGrath, 1984, Short et al., 1976]. In traditional conference-room video conferencing, video provides a means to convey a sense of co-presence[Olson and Olson, 1995]. In VR however co-presence is directly created through the use of avatars and hence we believe video may play a less significant role in the collaboration.

3.4 Flexible Support of Various Data Characteristics

The design of the CVR library is dependent on two interrelated factors: the characteristics of the data being distributed and the distribution scheme employed. The four attributes that characterize CVR data that most greatly affect the mode of transmission, management and storage of CVR data, are: quality of service, data size, persistence and queueing.

3.4.1 Quality of Service

For closely coordinated work in CVR, minimum levels of network bandwidth, latency and jitter are desirable. In addition, both reliable and unreliable protocols of unicast, broadcast and multicast transmission are needed to optimally transport different classes of CVR data (3D tracker data, state information, streamed audio/video feeds, geometric models, large scientific data sets.)

Unreliable protocols are suitable for the transmission of tracker data because: 1. the loss of a packet of tracker data is usually followed shortly afterwards by newer ones, and 2. unreliable protocols have a lower latency and utilize lower bandwidth than reliable protocols.

Multicasting has the additional benefit that clients that subscribe to a multicast group need only send one message to the group, rather than having to send the same message individually to each participant in the collaboration. The multicast protocol will automatically propagate the single message to all the other subscribers. The main disadvantage however is that multicast is based on unreliable UDP. Work however, is currently underway in developing reliable multicast protocols [Lin and Paul, 1996]. Reliable transmission is important in CVR for the delivery of accurate state information as well as models and scientific data sets. Here the loss of a packet could produce an unwanted artifact in the visualization that is not representative of the original data set.

A flexible solution to networking for CVR should include both reliable and unreliable forms of transmission. However it is interesting to note that only a few provide both capabilities simultaneously[Mandeville et al., 1995, Roussos et al., 1996]. This is perhaps due to the following reasons three reasons: First, the main concentration of VR libraries in the past has been in providing tools to allow programmers to quickly build interactive non-collaborative VR environments (e.g MRToolkit[Shaw and Green, 1993]). Support for collaboration was generally an after-thought and hence reliable TCP is used as the default, safe and generic solution. Secondly, most CVR implementations are still experimental technologies undergoing significant change. For example DIVE[Carlsson and Hagsand, 1993] initially used a transaction-oriented, object-oriented database called ISIS and a reliable TCP connection to synchronize all state information in the CVE. They are now using a peer-to-peer connection with a replicated database that synchronizes data via a reliable multicast connection. Finally, the implementations may be highly customized for specific problem domains. For example NPSNET[Macedonia and Zyda, 1995] uses multicasting to deliver information for military simulations. Other researchers have attempted to extend the underlying DIS protocol to allow the delivery of ``non-ballistic'' information. But because it uses an unreliable protocol additional mechanisms for retransmitting packets had to be devised. In addition, since the notion of military weapons are directly embedded in the specification of the protocol it does not serve as a generic protocol for non-military simulations such as collaborative engineering or scientific visualization.

3.4.2 Data Size

There are essentially three categories of CVR data sizes: small-event, medium-atomic, and large-segmented.

These divisions are created because they affect the manner in which they are optimally transmitted and manipulated.

- **Small-Event data** are data such as unreliable tracker data, and reliable state and event data. These typically require priority transmission with low latency.
- **Medium-Atomic data** are data that are small enough to fit in the physical memory of the client because it must be processed as one atomic ``chunk.'' Examples of these are 3D geometries representing individual objects in the VR scene.
- **Large-Segmented data** are data that are too large to fit in the physical memory of the client and hence can only be accessed in smaller segments. Large scientific data sets and long pre-digitized video streams fit this category. These data sets usually need to be ``abstracted-down'' first before they can be visualized, as the amount of data that can potentially be visualized can easily exceed the graphics rendering capabilities of the VR system.

3.4.3 Queued/Unqueued Data

Data that are sent to clients or servers, regardless of whether they are stored in a database or not, need to be either queued or unqueued. For example, world state information may be unqueued since only the latest information is necessary. Queued data are data which must all arrive at a client or server in order. This implies the use of a reliable protocol. There are however instances where a queued, unreliable protocol may still be useful- specifically for audio conferencing, long, unreliable data streams are transmitted to all participating clients.

3.4.4 Persistent/Transient Data

Persistent data characterizes data that needs to be stored in a database or file system for later use. This data remains in the database after all the clients leave the CVE. All state data that is crucial to the resumption of a client in a CVR session must be persistent. Models and scientific datasets that will be loaded into CVE are also prime candidates for database storage.

Transient data are data that are not stored in a database. An example of this kind of data are command messages that might be sent between clients to effect events or audio/video data streams. An exception to this definition is when transient data is stored in a database to allow re-play of events at a later time. In this case the data is more accurately characterized as persistent rather than transient.

3.5 Scalable and Flexible Topological Construction

No single interconnection of distributed resources will perform optimally for all CVR applications. The number of participants expected to work in the environment, the amount and form of the data being shared, the geographic distance, and the intervening networks connecting participants, have profound effects on the design of a suitable distributed topology. Systems that are designed to scale well with respect to connectivity (connection scalability) typically must sacrifice strong data consistency. Most currently existing systems prioritize connection scalability over data scalability (ability of CVEs to handle enormous amounts of data.)

It is our belief that data scalability is of greater importance to the development of engineering and scientific applications than connection scalability. Data sets in these problem domains are typically enormous in size however the number of people simultaneously collaborating is unlikely to exceed 6 or 7.

The three main classes of distributed topologies used in CVR include: replicated homogeneous, shared centralized, and shared distributed[Macedonia and Zyda, 1995]. These are described below.

3.5.1 Replicated Homogeneous

Replicated Homogeneous topologies are classical of military VR simulations (as in SIMNET, NPSNET, DIS)[Macedonia and Zyda, 1995]. In such topologies each client holds a completely replicated database of the shared environment and state information is shared by broadcasting messages to all participating clients. This system has no centralized control whatsoever, hence any new client joining a session must wait and gather state

information about the world that is broadcasted by the other clients.

3.5.2 Shared Centralized

In this approach all shared data is stored at a central server. The main advantage of this scheme is that it greatly simplifies the management of multiple clients, especially in situations requiring strict concurrency control. However, its role as an intermediary for the delivery of data can impose an additional lag in the system. Another disadvantage is that if the central server fails none of the connected clients can interact with each other. Despite these disadvantages, this architecture is still useful for supporting small groups of collaborators.

3.5.3 Shared Distributed with Peer-to-peer Updates

This approach simulates a wide-area shared memory structure [Carlsson and Hagsand, 1993, Shaw and Green, 1993, Mandeville et al., 1995, Wang et al., 1995] in which objects that are instantiated at one site are automatically replicated at all the remote sites. This logical abstraction simplifies the CVR application development at the cost of performance. Typically in these implementations, a newly connected client must form point-to-point connections with all the participating clients. Hence for n participants the number of connections required is $n(n\text{-}1)/2$. In addition if the environment involves the sharing of enormous scientific data sets, the data set will be fully replicated at every site. Unless the data sharing policy is modified to account for large datasets this scheme will not be scalable.

3.5.4 Shared Distributed using Client-server Subgrouping

This topology distributes the database amongst multiple servers. Clients connect to the appropriate server as needed. A classic approach is to bind the servers to unique multicast addresses. Clients then subscribe to different multicast addresses to listen to broadcasts from the servers[Barrus et al., 1996, Funkhouser, 1996]. This is a particularly effective way to handle large numbers of connected clients distributed over a wide virtual space. Each geographic region of the virtual space can be maintained by a separate server. The servers share the load of sustaining the state of the virtual world by handling only the subset of the connected clients that are in their geographic region.

3.6 Synchronous and Asynchronous Collaboration

The main focus of most CVR applications has been on synchronous collaboration. That is, all participants are working together in the environment at the same time. However in trans-global collaborations the timezone differences make routine synchronous collaboration highly inconvenient. In this case it is important to also provide a means for distributed groups to work asynchronously in a shared virtual space. The support of asynchrony will require the use of distributed databases to maintain the states between the remote sites.

3.7 Persistence in Collaborative Virtual Reality

Persistence in Collaborative Virtual Reality describes the extent to which the virtual environment exists after all participants have left the environment. Persistence can be divided into three major classes: participatory persistence, state persistence, and continuous persistence.

3.7.1 Participatory Persistence

This is persistence in which the VE only exists in the brief amount of time that participants are in it. When all participants leave, the environment is extinguished with no record of the state of the environment before it was extinguished. When the environment is started at a later time, it always begins at the beginning. Most virtual environments are still only participatory persistent.

3.7.2 State Persistence

This is where the state of the virtual environment may be saved at any given time to be recalled later. Either

intermittent snapshots can be created or entire collaborative experiences can be recorded for later review.

In a scientific visualization environment that involves simulations that are running on supercomputers a recording should include either the entire state of the virtual environment as well as the state and output of the simulation, or the state of the virtual environment and only the geometric representation of the simulation. The advantage of the latter is that it simplifies the mechanism for re-play. Re-play will only require a rendering of the geometry which can easily be encapsulated to work even in external viewers such as VRML2 browsers. However the disadvantage is that the geometry data itself cannot be re-used to further query the output of the simulation.

On the otherhand the advantage of saving the state of the environment and the simulation is that during re-play the participant can choose to dynamically re-involve the supercomputer. This is motivated by the following:

To ensure accuracy in computational science simulations, the simulation steps are kept relatively small. However the computed results are collected at every *n* time steps due to, surprisingly, disk space limitations (the output can occupy between several hundred megabytes to many gigabytes)[Roy and Cruz-Neira, 1995]. If by viewing the results a feature of interest is found, the scientist would normally re-execute the simulation from the beginning but only begin recording the output in the region of the feature and at each time step rather than at every *n* time steps. In this scenario persistence may be used to offer some assistance in reducing the amount of re-computation time. State Persistence may be invoked during the initial course-grained recording where, instead of simply recording the output, the states of the computation are also recorded. When the region of interest has been isolated, rather than returning computation to the beginning as is typical, the state of the computation can be retrieved from the persistent database and computation can be resumed from that point. Recording can then resume at a finer granularity.

In general, as part of the recording of persistent experiences it may be useful to also record the actions of the avatars so that on re-play they may be re-positioned in the scene to serve as reminders of which particular area of the visualization was being observed or manipulated at the time of the recording. In fact it may actually seem rather unnatural to watch an event transpire without being able to see the effector of the event. One could also imagine that the re-play procedure may also be recursively recorded so that a participant could observe him/herself observing him/herself. It is not entirely clear if this capability is of any value but the idea is at least somewhat intriguing.

3.7.3 Continuous Persistence

This is where the state of the virtual environment remains extant even when all the participants have left. Hence when participants re-enter the environment the state of the world may have changed. Such environments are liken to MUDs (Multiuser Domains/Dungeons) which by their popularity, have shown to encourage the spontaneous use of collaborative environments.

Although this may seem to be an extravagant use of computing power, it is anticipated that in future generations of CVR environments the notion of persistence is merely an extension of the existing idea of the operating system or the WWW server. These are essentially, already continuously persistent environments.

3.8 Interoperability with Heterogeneous Systems

The varying domains in which CVR is applied requires connectivity between heterogeneous resources such as external databases, supercomputers, desktop workstations, and miscellaneous VR systems. For example, Argonne's incinerator simulator connects the CAVE VR system to an IBM SP supercomputer. The supercomputer performs the computation while the CAVE visualizes the results. In NICE, the system allows CAVEs, ImmersaDesks, desktop workstations, WWW browsers and Java programs to collaborate simultaneously.

3.9 Application Specific Servers

These are unlike traditional networking and database servers in that they do not simply store and forward data. Application specific servers in VR also possess semi-graphical capabilities as they may need a local

representation of the virtual space for their operation. For example, an application specific server simulating the movement of autonomous agents through a virtual landscape may also use the same graphical routines that model and visualize the terrain to perform operations such as collision detection.

4 CAVERN

CAVERN (CAVE Research Network) is a collection of participating industrial and research institutions equipped with CAVEs, ImmersaDesks, and high-performance computing resources all interconnected by high-speed networks for the purpose of supporting collaborative: engineering and design; education and training; and scientific visualization and computational steering, in virtual reality.

CAVERNsoft is the collaborative VR middleware to facilitate the construction of persistent CVEs within CAVERN. CAVERNsoft uses light-weight distributed data-stores as the mechanism for managing a wide range of data volumes (from a few bytes to several terrabytes) that are typically needed for sustaining persistence in virtual environments. Multiple networking interfaces support customizable- latency, data consistency, and scalability that is needed to support a broad spectrum of networking requirements. Although CAVERNsoft is being developed to support the CAVERN partnership, it is not a CAVE/ImmersaDesk-specific library. Only the higher level template layers (described below) of CAVERNsoft are VR-platform specific. The underlying capabilities of CAVERNsoft are designed to work on heterogeneous computing facilities (such as workstations, supercomputers, desktop PCs) that are not VR specific. It was conceived to create a common software middleware with which heterogeneous applications may communicate with VR applications.

4.1 The Information Request Broker

The Information Request Broker (IRB) is the nucleus of all CAVERN-based client and server applications. An IRB is an autonomous repository of persistent data that is accessible by a variety of networking interfaces. The goal is to develop a hybrid system that combines the idea of the distributed shared memory model in CALVIN with database technology and realtime networking technology under a unified interface to flexibly support data distribution in collaborative VR.

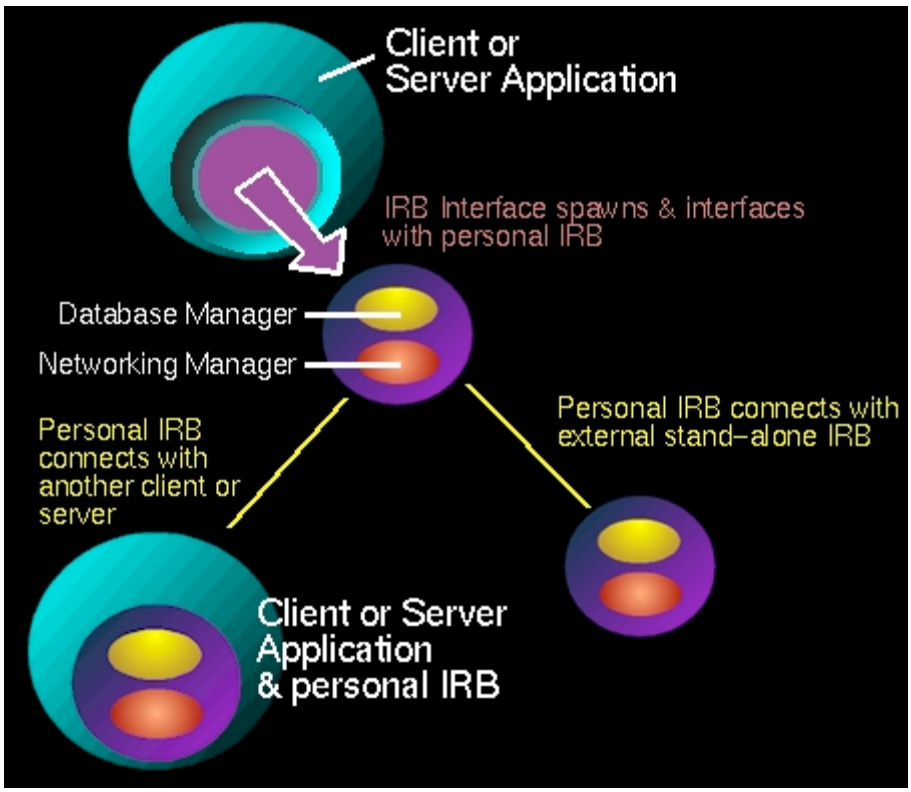

Figure 3: Clients/Servers use the IRB interface to spawn personal IRBs with which to communicate with other

clients/servers or standalone IRBs.

A client application is built by using an IRB interface (IRBi) which, on invocation, will spawn the client's ``personal'' IRB. This IRB is used to cache data retrieved from other IRBs during the operation of the client. An application-specific server is similarly built using the IRBi. Hence there is little differentiation between a client and a server (Figure 3.) Using the IRBi a client can arbitrarily form a connection, after having acquired the proper permissions, with any other client or server to access its resources. The IRBi will communicate the request to the client's personal IRB which will then communicate with the remote client's or server's IRB. It is the IRBs' responsibility to negotiate the networking and database services requested by the client/server applications. This form of flexibility and symmetry will allow all of the main CVR topologies to be quickly constructed (Figures 4.) Figure 4a. shows IRB-based clients with possibly fully replicated databases (as in NPSNET) sharing updates via a multicast group. Figure 4b. shows the use of IRBs in a shared, centralized database. Figure 4c. shows IRB-based clients in a fully connected configuration to support a shared, distributed database with peer-to-peer updates. Finally Figure 4d. shows IRB-based clients and servers that are connected to form a shared, distributed client-server database. The clients may arbitrarily connect to any of the servers using any desired communications protocol to retrieve information. Since there is no distinction between a client or a server, an IRB-based program may be a client running on a supercomputer, or a server interfacing with a large database of scientific data.

The mechanisms for facilitating the IRB's capabilities will be described in greater detail in the following sections.

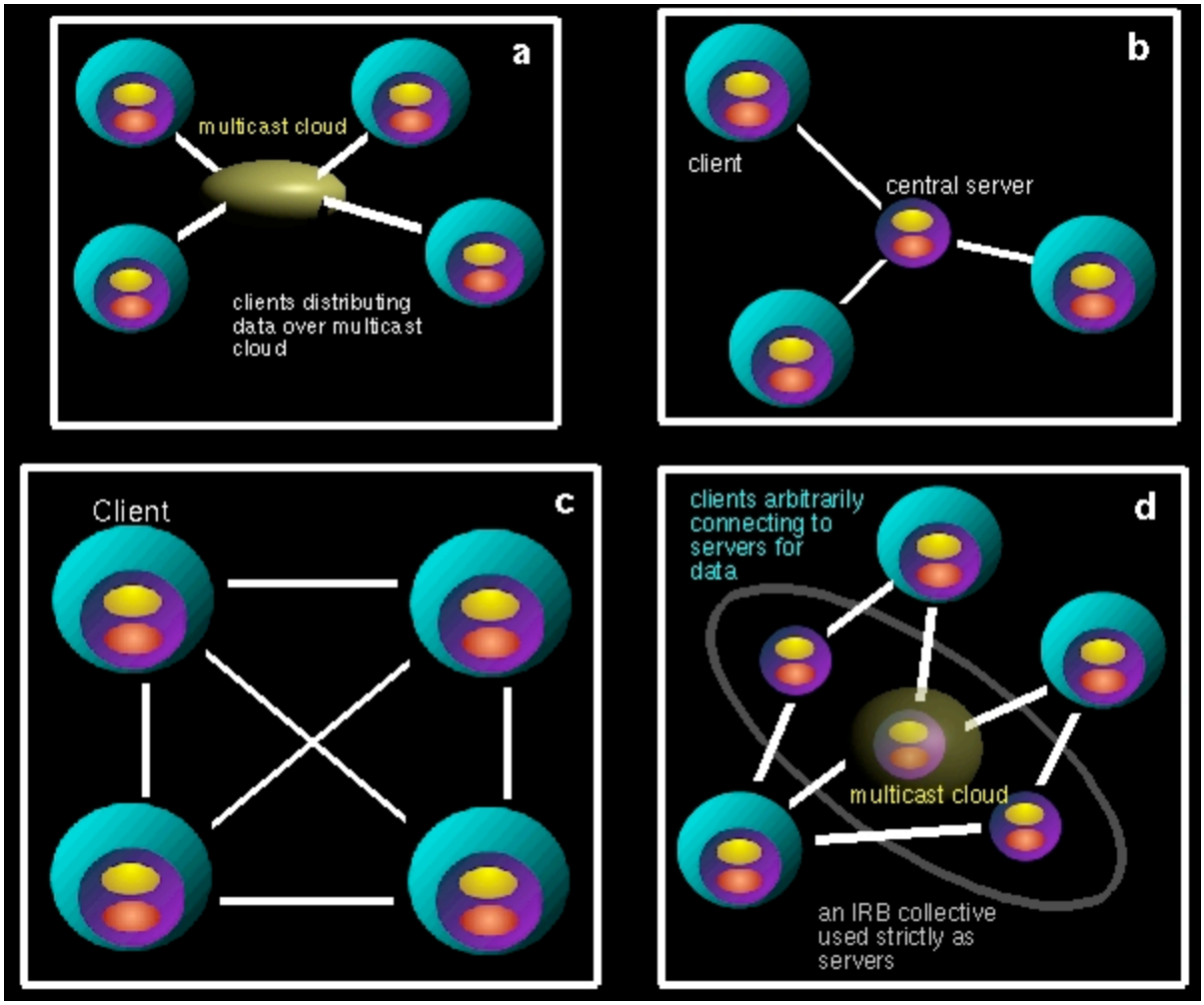

Figure 4: Use of IRBs to construct all the major classes of CVR topologies. (a) Fully replicated databases sharing updates via a multicast group/cloud. (b) IRB clients connected to a shared centralized database (also an IRB.) (c) IRB clients in a fully connected configuration to support a shared, distributed database with peer-to-peer updates. (d) IRB clients and servers connected to form a shared, distributed client-server database.

4.2 The IRB Interface

The IRB interface (IRBi) is the client and server's interface to the IRB. The IRBi provides a networking interface, a database interface and a high-level template interface. The IRBi is tightly coupled with the IRB as they are merely threads that share the same address space. This reduces the need for creating artificial message passing schemes to invoke functionality between the client's application and the IRB.

A client's handle to their personal IRB is used to activate dynamic connections with remote IRBs. A client wishing to share information between its personal IRB and a remote IRB begins by first creating a communication channel and declaring its communication properties. Then any number of local and remote keys may be linked over the channel. A key is a handle to a storage location in an IRB's database. The database is used to cache data received from remote keys. Keys are uniquely identified across all IRBs and can be hierarchically organized much like a UNIX directory structure. Each local key may be linked to only one remote key. This is the link explicitly invoked by the user to share information with a remote IRB. However each local key can *accept* multiple linkages from other remote subscribing keys. The user is generally made unaware of these additional linkages as the personal IRB transparently manages data sharing with the remote subscribers. Any modifications that are made to one key will automatically be propagated to all the other linked keys.

4.2.1 Channel Properties

Channel properties allow clients to specify the networking service desired for data delivery. Clients may specify reliable TCP, or unreliable UDP and multicast. Large packets delivered over unreliable channels will automatically be fragmented at the source and reconstructed at the destination. If any fragment is lost while in transit the entire packet is rejected.

In addition to connection reliability clients may specify Quality of Service (QoS) requirements. Hence they are able to declare the desired bandwidth, latency, and jitter of the data stream. The personal IRB will attempt to obtain the desired level of QoS from the remote IRB, but if it fails, the client may at any time negotiate for a lower QoS. As in RSVP[Zhang et al., 1993] client-initiated QoS is used so that the client can specify the amount of data it can handle from the remote IRB.

4.2.2 Link Properties

Link properties allow clients to specify the actions taken when local and remote keys are linked. This includes being able to choose between active and passive updates and being able to select the initial and subsequent synchronization behavior.

In most CVR applications, world state information consisting of a few tens of bytes are actively distributed. That is, the moment a new value is generated it is automatically propagated to all the subscribers of the data. Passive updates occur only on subscriber request and usually involves a comparison of local and remote timestamps before transmission. For example, passive updates are typically used to download large volumes of 3D model data. Caching data and comparing their timestamps helps to reduce the need to redundantly download the same data set.

The initial synchronization behavior determines how the local and remote keys should be synchronized when the links are first formed. That is, clients are able to choose to synchronize automatically based on the keys' timestamps. That is the older key will be updated with information from the newer key. However the client may also choose to force synchronization from the local key to the remote key, and vice versa, regardless of timestamp. Of course clients may choose to perform no initial synchronization at all.

Subsequent synchronization behavior specifies the manner in which data is synchronized when local or remote updates to keys occur. The same options as for initial synchronization hold.

The default link property is to use active updates with automatic initial and subsequent synchronization.

4.2.3 Key Properties

Keys may be defined at a client's personal IRB or at a remote IRB provided the client has the necessary permissions. Keys may either be transient or persistent. Persistent keys are keys that will be stored in the IRB's datastore so that when a client or server re-launches, the data will still be retrievable by specifying the same key identifier. Clients determine whether a key is to persist by asking the IRB to perform a commit operation on the data. In addition simple locking functions are provided to allow clients to lock local or remote keys. Locking calls are non-blocking to prevent realtime applications from stalling when attempting to acquire locks on keys. Instead the locking call accepts a user-specified callback function that will be called when a lock has been acquired or when any relevant event pertaining to the lock occurs.

4.2.4 Asynchronous Triggering of Events

Many events may arise during the course of distributing data between clients and servers. The client/server may need to be notified so that appropriate actions may be taken in response to these events. It is inefficient for realtime VR applications to poll for such events. Instead the programs provide the IRBi with callback functions that the IRBi may call when the event arises.

Some examples of events include:

- **New Incoming Data Event**

  This event occurs when a key receives a new piece of data. For example a key could be subscribing to avatar state information (position and orientation of the avatar's head). When this information changes, a callback can be invoked to make the corresponding changes to the graphical representations of the avatars in the virtual world.

- **IRB Connection Broken Event**

  When a connection to an IRB has been broken (possibly due to a crash) clients will continue to function by accessing local versions of the subscribed data. The personal IRB may then periodically attempt to re-establish connection with the remote IRB or choose another client or server to take the place of the ``broken'' IRB. It is the responsibility of the application-specific IRBs to determine the policies for such situations.

- **QoS Deviation Event**

  This event occurs when the negotiated QoS falls below contracted levels. For example, if the latency negotiated for a stream of tracker data falls below acceptable limits, the VR client can be warned so that perhaps interpolative techniques such as dead-reckoning[Macedonia and Zyda, 1995] can be activated to reduce the impact of the increased latency. Alternatively the client can re-negotiate a different QoS, perhaps one involving a lowering of the bandwidth (by compression of data) in order to maintain the desired latency.

4.2.5 Recording Keys

In addition to declaring the retention properties of keys the IRBi allows the clients to declare keys that hold recordings of groups of keys. This facility is provided to support State Persistence in VR.

In these recordings close synchronization of remote system clocks is not absolutely necessary as recording is always made from one point of view and hence it is the point of view's time reference that all relevant information is recorded.

Recordings may consist of time stamping and storing every change in value that occurs at a key and recording the state of all the keys at wide intervals. The former is needed to track the gradual changes in the virtual environment over time. The latter is needed to establish checkpoints so that the recordings may be fast-forwarded or rewound without having to compute every successive state that led to the fast-forwarded/rewound location.

On playback the recordings will populate the appropriate keys and, if desired, trigger client callbacks. In some

instances it is useful to be able to playback only a subset of the recorded keys. This will allow the user to observe smaller subsets of events that occur in the VR environment. For example the Virtual Director[Thiebaux, 1997] (a VR application that allows users to record the path of a virtual camera through a virtual environment) allows playback of recordings of camera positions in each of the three X,Y,Z axes so that each of the paths in the axes can be edited independently.

Finally to synchronize the playback of experiences across multiple virtual environments each environment must constantly broadcast their frame-rate. This ensures that faster VR systems do not overtake slower systems while rendering the virtual imagery.

4.2.6 Direct Connection Interface

In addition to the many automatic networking capabilities provided by IRBs the IRBi must still support direct access to low-level socket TCP, UDP, multicast interfaces so that connectivity with legacy systems (such as WWW servers) can be supported. However CAVERNsoft adds value to the basic socket-level interfaces by providing automatic mechanisms for accepting new connections, and making asynchronous data-driven calls to user-defined callbacks.

4.2.7 Supplementary Concurrent Processing Facilities

Most of the networking and database operations performed in the IRB are executed concurrently and, if a multiprocessor system is available, in parallel with the VR system. It is therefore necessary to provide basic concurrency control primitives such as mutual exclusion and signals. These are implemented as macro definitions on top of the underlying threads library used by the IRB (for example POSIX threads.)

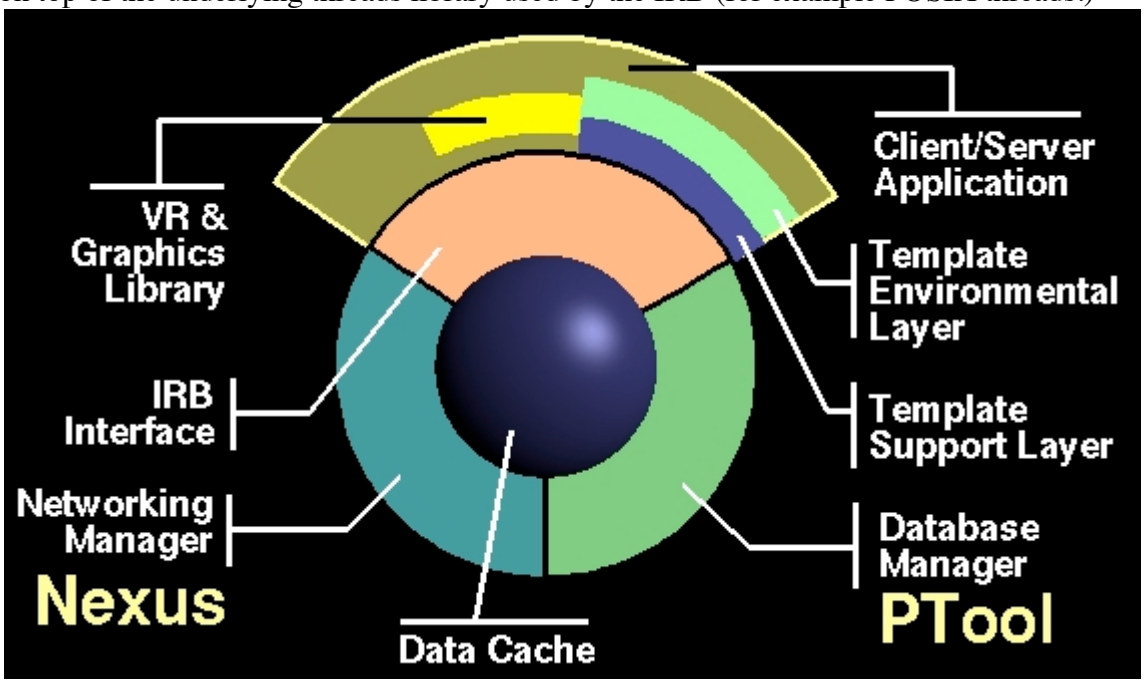

Figure 5: Software architecture of a CAVERN IRB-based Client/Server.

4.2.8 High-level Templates

In our experience with developing CVEs, we have found that it is substantially more difficult to retro-fit a single-user application with collaborative capabilities than to include these capabilities as part of the original design of the application. The IRB's high-level templates are designed to encourage VR application developers to think in terms of developing a collaborative application, by providing them with high-level tools that they are likely to all commonly need.

Templates are divided into two categories: support templates and environmental templates.

Support templates provide a collection of libraries to support various basic CVR component services such as:

encoding and decoding of audio and video streams for teleconferencing and management of avatars.

Environmental templates provide a suite of complete but extensible CVEs. For example an environmental template could be designed specifically to help domain scientists ``jumpstart'' the process of building collaborative scientific visualization applications. Such a template would automatically provide networking, visualization and recording components as well as basic collaboration components such as audio/video conferencing, and avatars.

The template layers of CAVERNsoft are the only layers that simultaneously interface with the IRB interface and the graphics interface (Figure 5.) For example, in order to support avatars, the IRB interface must be used to declare keys to hold avatar information. The graphics portion of the avatar template will be implemented with the CAVE library in conjunction with either OpenGL, OpenInventor, or Performer. This separation of the basic IRB interface and the graphics interface with the template layer allows the IRB system to be used in non-graphic computing systems such as supercomputers or workstation farms.

4.3 Implementation Notes

Figure 5 shows how all the various components of CAVERNsoft fit together to build a client/server application. The networking manager is founded on Nexus[Foster et al., 1996]. Nexus is an efficient multithreaded communications library developed by Argonne National Laboratory to connect client applications with remote supercomputing resources. Using Nexus the IRB's networking manager can negotiate networking protocols and quality of service contracts, and manage connections once they have been established. As Nexus was originally built to coordinate communications amongst multiple nodes of supercomputers this allows CAVERNsoft to leverage those capabilities in support of IRB-based clients that will run on supercomputers.

The database manager will be built using PTool[Grossman et al., 1995], a persistent object store developed by the Laboratory for Advanced Computing at the University of Illinois at Chicago. PTool's main use is in the efficient storage and retrieval of enormous persistent objects (typically occupying giga- to terra-bytes in size). A custom interface will be built on top of PTool to provide the abstraction of persistent keys. Strictly speaking the database that CAVERNsoft uses is a datastore. PTool achieves significant performance improvements over other object-oriented databases by stripping away the transaction management capabilities found in traditional databases.

Nexus and PTool are tightly coupled to minimize the latency involved in propagating incoming data from the network to the client application. Incoming data from the network is deposited directly into the datastore's cache. The client, on accessing the data, may either make a single copy of the data, or choose to temporarily lock the key (and hence the affected region of the cache) to directly access the data via a pointer. By traditional database standards, this is potentially ``dangerous,'' as a client may forget to unlock a key after using it and hence prevent its subsequent update; but when dealing with time-critical data as in CVR, this is acceptable. Essentially this means that incoming data from the network is at most copied once by the time the data is stored in the datastore and the client uses it. This is typically not the case with traditional distributed-object implementations such as Orbix 1.3 and ORBeline 1.2[Schmidt et al., 1995].

As there are potentially multiple streams of incoming data destined for a key, each key holds a number of temporary ``buffers'' (one per link) which all contain pointers to independent regions of the datastore's cache. When data arrives, it is deposited directly in the appropriate temporary buffer, and a pointer switch occurs which makes the buffered data, the current data in the key. Again, this guarantees that at most one memory copy occurs from the time the data arrives at the IRB to the time the data is used by the client.

4.3.1 Using CAVERNsoft to Implement a Collaborative VR Application

This section will describe how a CVR application such as NICE can be constructed using CAVERNsoft. The first noticeable characteristic of building the NICE architecture in terms of IRBs is that many different solutions are possible. The description below will only present one of these solutions.

The main distributed components of NICE consist of the garden simulation server, the WWW servers from which models are downloaded, and the NICE ``smart-repeater.'' In the CAVERNsoft model all these components are replaced by IRB-based clients and servers.

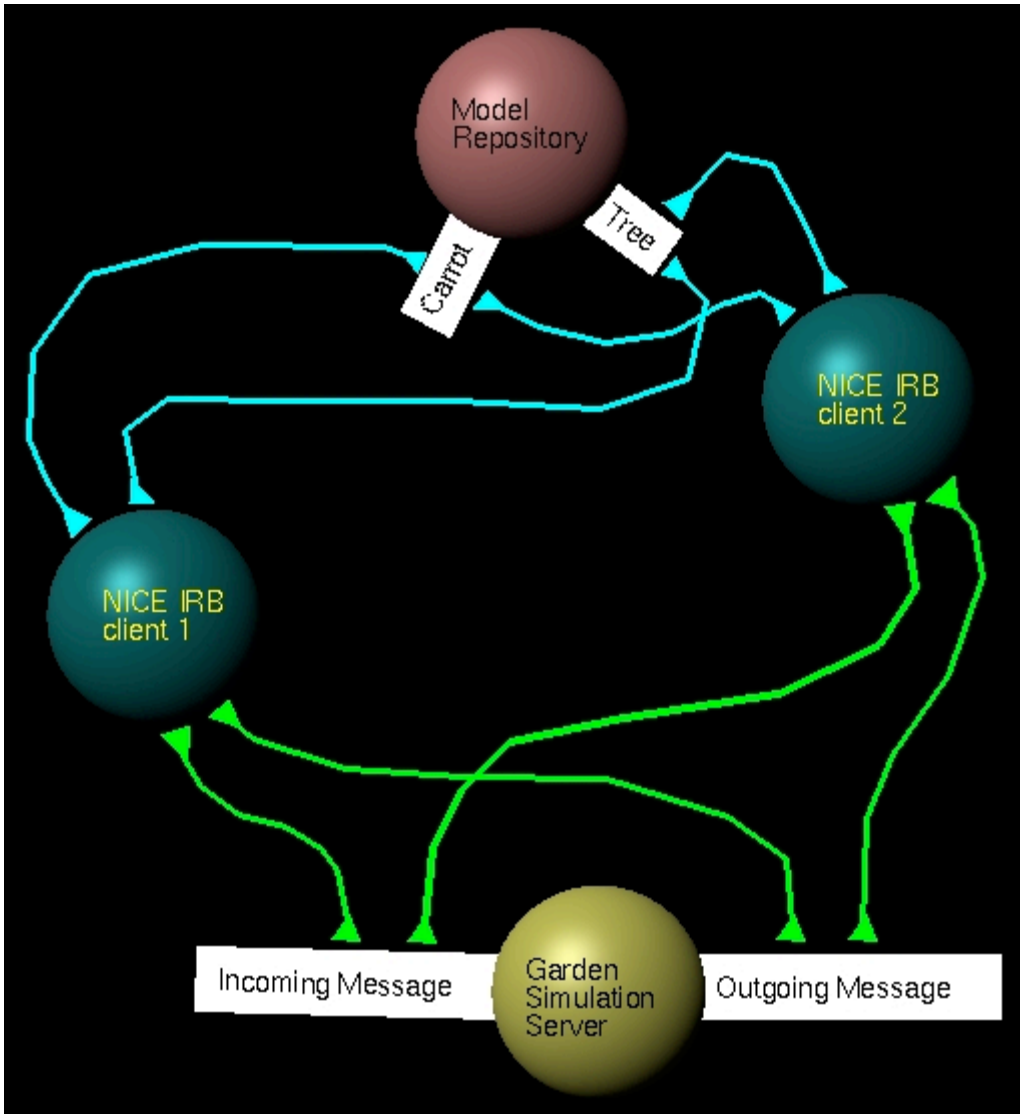

Figure 6: IRB-based NICE clients connecting to the garden server to determine which 3D models to download from the model repository.

The garden server can be built out of an IRB with two main keys- an incoming message key and an outgoing message key (Figure 6.) NICE clients, which are also built from IRBs, can start up by first linking to the garden message key to ask for information on the state of the garden. This is done by sending a message to the garden server's incoming key and receiving state information on the outgoing key. The state of the garden can actually be distributed amongst multiple keys, but for simplicity garden message keys will be treated as message-passing conduits. Based on the messages from the garden server the NICE client can decide which 3D models it needs to download from the model repository server. The model repository server behaves like the WWW server in the original NICE implementation. The repository is of course implemented with an IRB. Since keys in all IRBs are timestamped, the client's download of the 3D models only occurs if the repository has a newer version of the model.

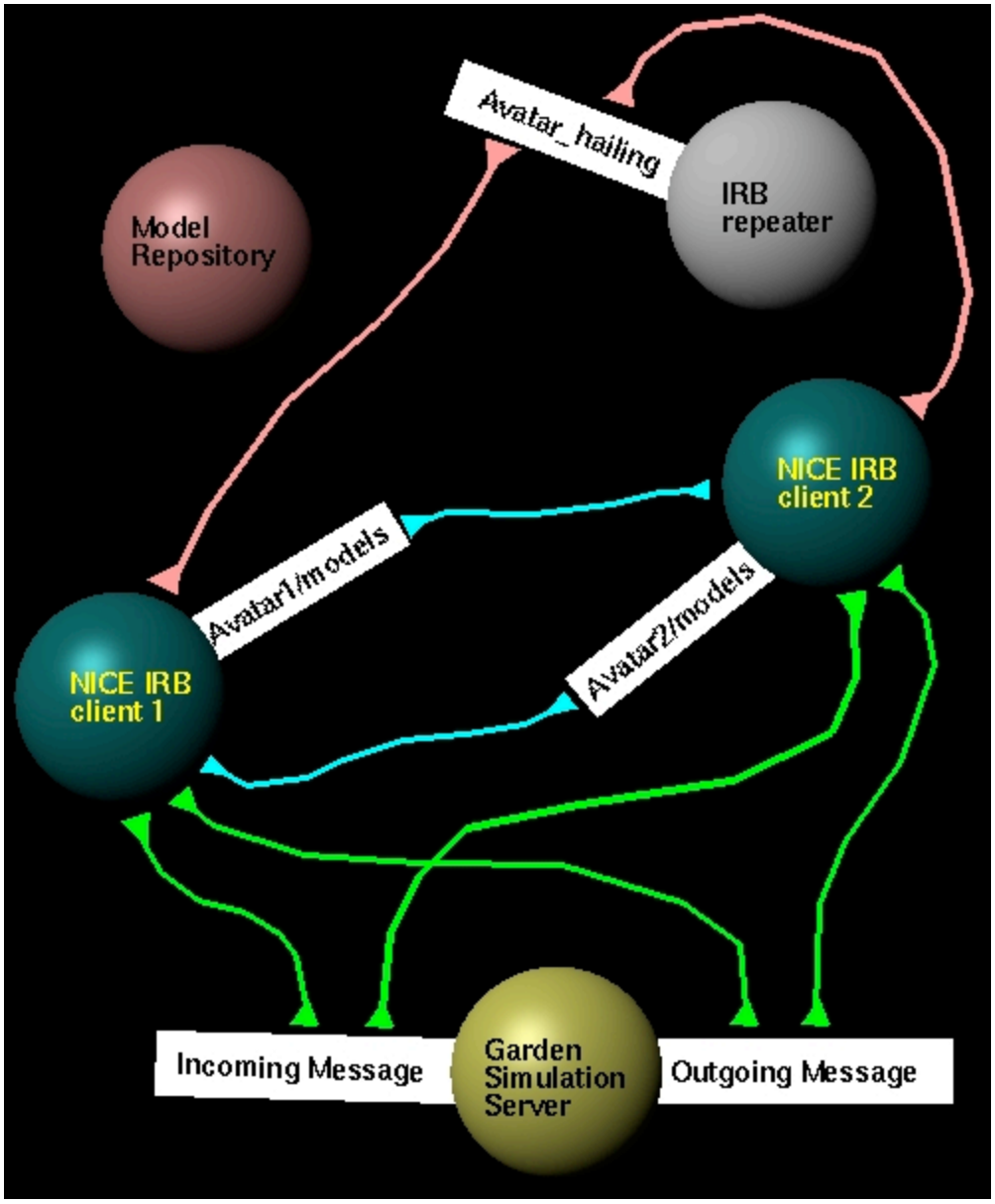

Figure 7: IRB-based NICE clients communicating with the IRB repeater to discover each other's avatars.

While the models are being downloaded, the NICE client may send a message to the avatar hailing key on the IRB repeater server (Figure 7.) This will inform any other NICE client that may already be participating in the environment, that a new participant has arrived. This information can consist of the address of the remote NICE client's IRB and the key under which the avatar models are stored (e.g. avatar1/models). Hence one client can directly access the models from a remote client. Alternatively all the avatar models can be stored in the model repository, in which case the clients simply give each other the key identifiers of the avatar models they are assuming. The clients can then consult the repository for the models.

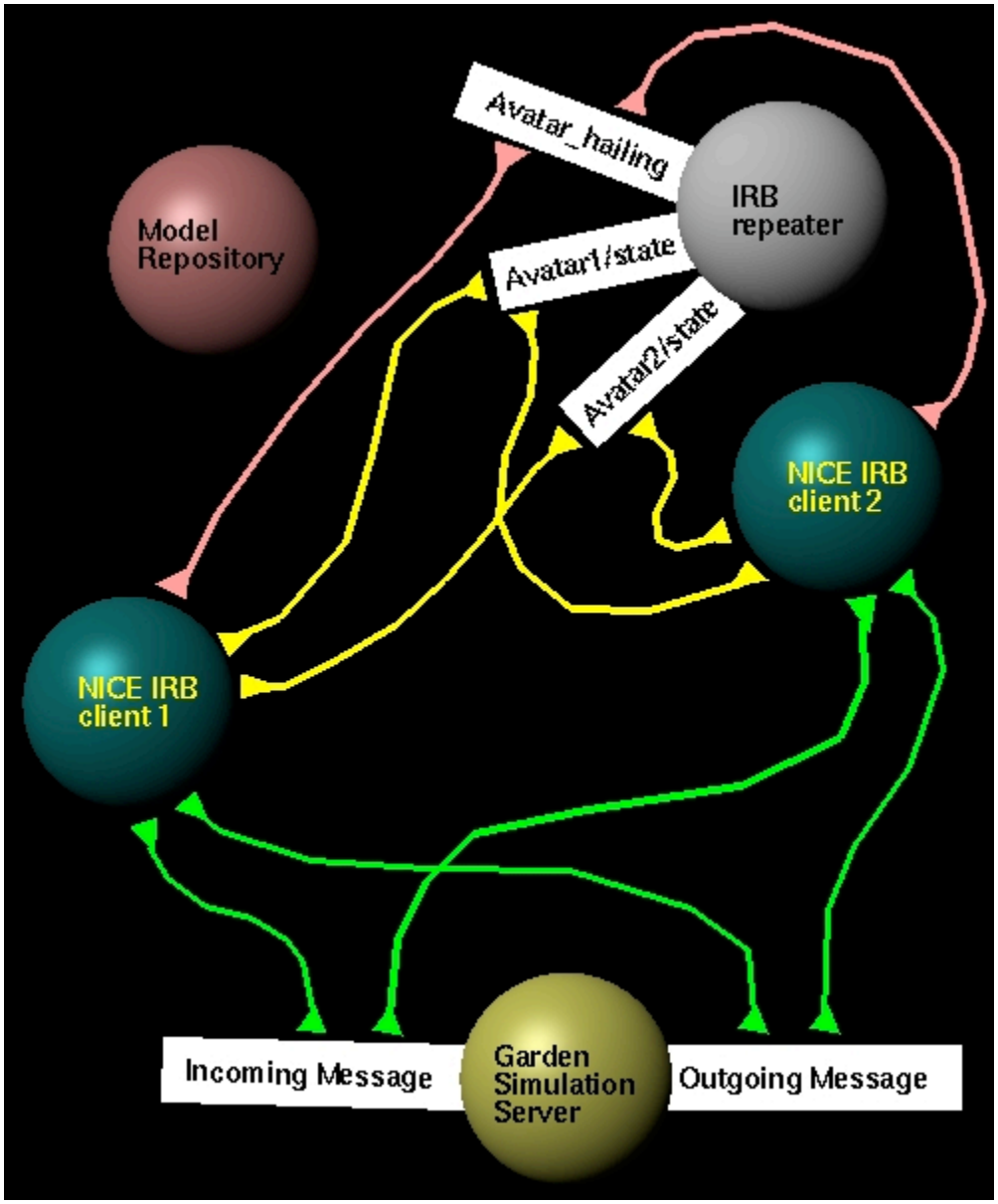

Figure 8: IRB-based clients subscribing to keys holding avatar state information on the IRB repeater.

Each client that connects to an IRB repeater can create a key on the repeater to hold its avatar state information (e.g. avatar1/state; Figure 8.) Hence any changes to client avatar1 will be first sent by the client to the repeater which will hold the data under that key. Another avatar (e.g. avatar2) will learn about the key through messages sent along the avatar hailing key by avatar1. Avatar2 can then discover the state of client avatar1 by subscribing to the avatar1/state key on the repeater. To reduce the latency involved in first sending avatar information to a repeater and then to the individual subscribing clients, avatars can choose to directly subscribe to each other's avatar/state keys. Yet another approach would be to link the avatar/state keys to one or more multicast addresses (Figure 9.) Again the hailing message key would be used to inform all participating clients of the proper subscription addresses.

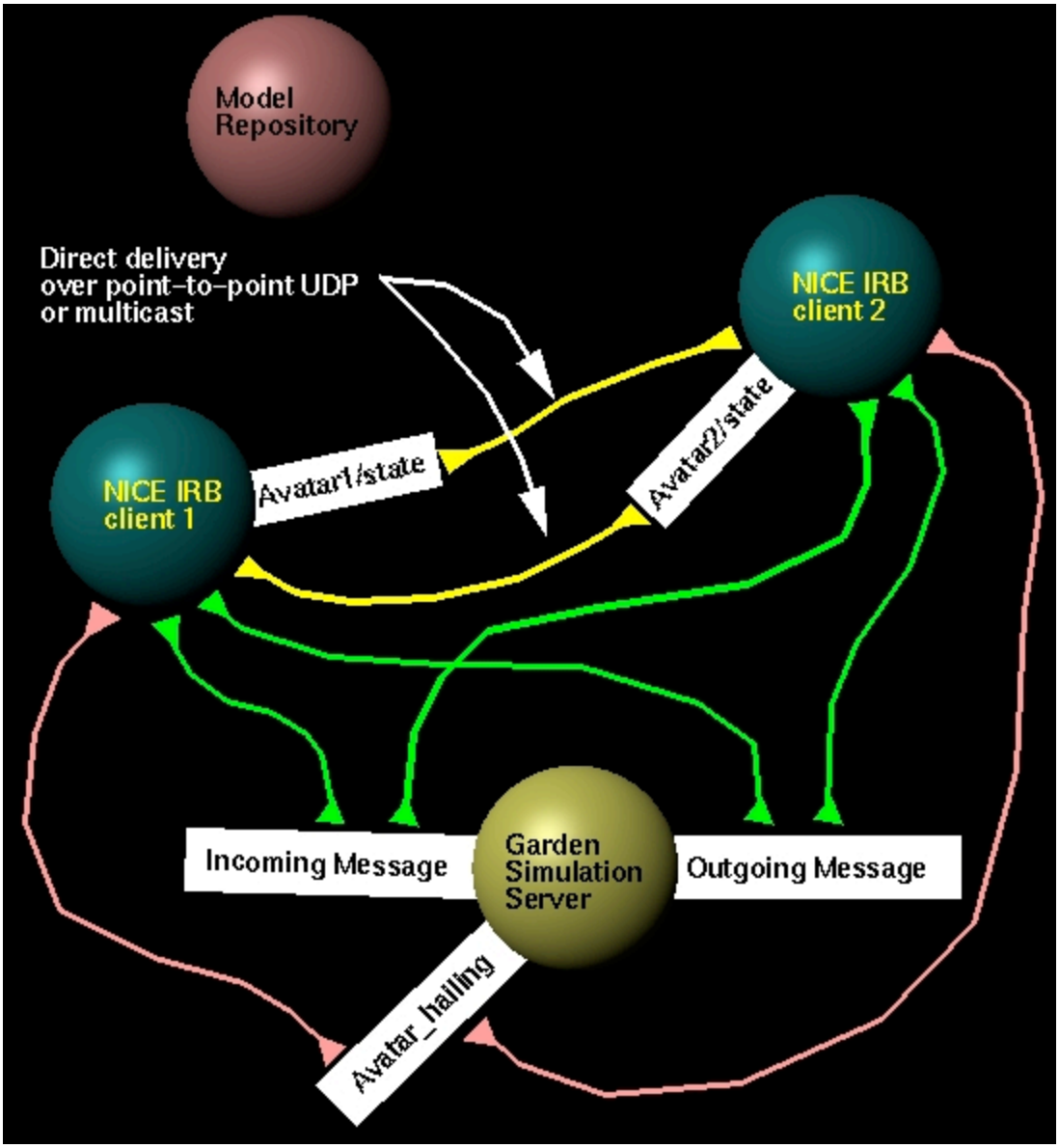

Figure 9: Example of an IRB-based NICE implementation using either direct links between avatar state keys or multicast links.

5 Work in Progress

The first prototype of CAVERNsoft reflecting the minimal capabilities of the IRB has been implemented. This IRB currently supports the following capabilities:

1. The ability to arbitrarily define local and remote keys on networked IRBs.
2. The ability to create multiple communications channels between arbitrary IRBs. The QoS capabilities of Nexus are still currently being specified and hence will appear in later editions of CAVERNsoft.
3. The ability to link keys across communication channels and have them automatically propagate data on a first-come-first-served basis. Clients are able to select different initial and subsequent synchronization mechanisms.
4. A prototype persistent key datastore has been implemented. This preliminary implementation caches all accessed persistent data in main memory and maps the hierarchy of the key datastore to a UNIX directory and file system. The finally version will be developed by the PTool group and will use the advanced caching algorithms of PTool.

The design of CAVERNsoft has attempted to take into consideration a broader perspective of collaborative virtual reality that includes both human-factors issues as well as engineering issues. It consolidates our research

and experience[Leigh et al., 1993, Vasilakis, 1995, Leigh et al., 1996b, Leigh et al., 1996a, Leigh and Johnson, 1996, Roussos et al., 1996] from collaborating with artists, experts in human-factors, graphics, networking and database research, as well as application developers, domain scientists and end-users.

In doing so one of our primary and largely practical goals is the production of CAVERNsoft to facilitate the rapid construction of complex CVR applications. To achieve this, we believe that at the present time, given the realtime performance demands of virtual reality, the most flexible approach is to adopt a symmetric architecture that uses light-weight, fast, distributed data-stores interlinked by customizable networking connections.

As CAVERNsoft is a long term solution to CAVERN's CVR needs it will continue to develop and evolve in the next few years as it is used by the CAVERN partnership. It is expected that by the end of 1997 the first applications built using CAVERNsoft will emerge. After the development of the underlying CAVERNsoft infrastructure, it is anticipated that future work will focus on the construction of multiple reusable CAVERNsoft environmental templates to further facilitate the rapid construction of collaborative virtual environments in numerous problem domains.

Acknowledgements

We would like to thank the members of the PTool group for their support in developing the KeyTool persistent heap system for this project. The PTool (http://matisse.eecs.uic.edu) group consists of Stuart Bailey and Robert Grossman (grossman@eecs.uic.edu) at the Laboratory for Advanced Computing at the University of Illinois at Chicago.

We would also like to thank the Nexus group for their networking support. The Nexus (http://www.nexus.org) group consists of Steve Tueke and Ian Foster at Argonne National Laboratory.

Major funding is provided by the National Science Foundation (CDA-9303433.)

References

Argyle and Cook, 1976
Argyle, M. and Cook, M. (1976). *Gaze and Mutual Gaze*. Cambridge University Press.

Barrus et al., 1996
Barrus, J. W., Waters, R. C., and Anderson, D. B. (1996). Locales and beacons: Efficient and precise support for large multi-user virtual environments. In *Proceedings of the Virtual Reality Annual International Symposium*. *VRAIS'96*, pages 204-213. IEEE Computer Society, IEEE.

Carlsson and Hagsand, 1993
Carlsson, C. and Hagsand, O. (1993). DIVE - a multi-user virtual reality system. In *Proceedings of the IEEE Virtual Reality Annual International Symposium.*

Cruz-Neira et al., 1993
Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. (1993). Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In Kajiya, J. T., editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 135-142.

Fish, 1996
Fish, R. (1996). Bellcore cross-industry working team workshop XIWT. personal correspondence.

Fish et al., 1990
Fish, R. S., Kraut, R. E., and Chalfonte, B. L. (1990). The videowindowsystem in informal communication. In *Proceedings of Computer Supported Cooperative Work*, pages 1-11.

Foster et al., 1996
Foster, I., Kesselman, C., and Tuecke, S. (1996). The Nexus approach to integrating multithreading and communication. *Journal of Parallel and Distributed Computing*, (37):70-82.

Freitag et al., 1995
Freitag, L., Diachin, D., Heath, D., Herzog, J., and Plassmann, P. (1995). Remote engineering using cave-to-cave communications. *Virtual Environments and Distributed Computing at Supercomputing'95: GII Testbed and HPC Challenge Applications on the I-Way*, page 41.

Funkhouser, 1996
Funkhouser, T. A. (1996). Network topologies for scalable multi-user virtual environments. In

Proceedings of the Virtual Reality Annual International Symposium. VRAIS'96, pages 222-228. IEEE Computer Society, IEEE.

Grossman et al., 1995

Grossman, R. L., Hanley, D., and Qin, X. (1995). PTool: A light weight persistent object manager. In *Proceedings of SIGMOD'95*, page 488. ACM.

Lehner and DeFanti, 1997

Lehner, V. D. and DeFanti, T. A. (1997). Distributed virtual reality: Supporting remote collaboration in vehicle design. *IEEE Computer Graphics and Applications, in press.*

Leigh et al., 1993

Leigh, J., De Schutter, E., Lee, M., Bhalla, U. S., Bower, J. M., and DeFanti, T. A. (1993). Realistic modeling of brain structures with remote interaction between simulations of an inferior olivary neuron and a cerebellar Purkinje cell. In *Proceedings of the SCS Simulations Multiconference*, Arlington, VA.

Leigh et al., 1996a

Leigh, J., Johnson, A., and DeFanti, T. A. (1996a). CALVIN: an immersimedia design environment utilizing heterogeneous perspectives. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems '96*, pages 20-23.

Leigh and Johnson, 1996

Leigh, J. and Johnson, A. E. (1996). Supporting transcontinental collaborative work in persistent virtual environments. *IEEE Computer Graphics and Applications*, pages 47-51.

Leigh et al., 1996b

Leigh, J., Johnson, A. E., Vasilakis, C. A., and DeFanti, T. A. (1996b). Multi-perspective collaborative design in persistent networked virtual environments. In *Proceedings of IEEE Virtual Reality Annual International Symposium '96*, pages 253-260.

Lin and Paul, 1996

Lin, J. C. and Paul, S. (1996). RMTP: A reliable multicast transport protocol. In *Proceedings of IEEE INFOCOM'96*, pages 1414-1424.

Locke, 1995

Locke, J. (1995). An introduction to the internet networking environment and SIMNET/DIS. Master's thesis, Naval Postgraduate School. http://www-npsnet.cs.nps.navy.mil/npsnet/publications/DISIntro.ps.Z.

Macedonia et al., 1995

Macedonia, M. R., Brutzman, D. P., and Zyda, M. J. (1995). NPSNET: A multi-player 3D virtual environment over the internet. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 93-94. ACM, ACM.

Macedonia and Zyda, 1995

Macedonia, M. R. and Zyda, M. J. (1995). A taxonomy for networked virtual environments. In *Proceedings of the 1995 Workshop on Networked Realities.*

Mandeville et al., 1995

Mandeville, J., Furness, J., and Kawahata, T. (1995). Greenspace: Creating a distributed virtual environment for global applications. In *Proceedings of IEEE Networked Virtual Reality Workshop.* IEEE.

McGrath, 1984

McGrath, J. (1984). *Groups: Interaction and Performance.* Englewood Cliffs, NJL Prentice-Hall.

Olson and Olson, 1995

Olson, J. and Olson, G. M. (1995). What mix of video and audio is useful for small groups doing remote real-time design work. In *Proceedings of SIGCHI'95*, pages 362-368. ACM, ACM Press.

Park, 1997

Park, K. S. (1997). Effects of network characteristics and information sharing on human performance in cove. Master's thesis, Electronic Visualization Laboratory, University of Illinois at Chicago.

Roussos et al., 1997

Roussos, M., Johnson, A., Leigh, J., Barnes, C. R., Vasilakis, C. A., and Moher, T. G. (1997). The nice project: Narrative, immersive, constructionist/collaborative environments for learning in virtual reality. In *ED-MEDIA/ED-TELECOM 97: World Conferences on Educational Multimedia and Hypermedia and on Educational Telecommunications.*

Roussos et al., 1996

Roussos, M., Johnson, A., Leigh, J., Vasilakis, C., and Moher, T. G. (1996). Constructing collaborative stories within virtual learning landscapes. In *Proceedings of the European Conference on A.I. in Education*, pages 129-135.

Roy and Cruz-Neira, 1995

Roy, T. and Cruz-Neira, C. (1995). Cosmic worm in the CAVE: Steering a high-performance computing application from a virtual environment. *Presence*, 4(2):121-129.

Schmidt et al., 1995

Schmidt, D. C., Harrison, T., and Al-Shaer, E. (1995). Object-oriented components for high-speed network programming. In *Proceedings of USENIX Conference on Object-Oriented Technologies*, Monterey, CA.

Shaw and Green, 1993

Shaw, C. and Green, M. (1993). The MR toolkit peers package and environment. In *Proceedings of the Virtual Reality Annual International Symposium. VRAIS'93*. IEEE Computer.

Short et al., 1976

Short, J., Williams, E., and Christie, B. (1976). *The Social Psychology of Telecommunications*. Wiley and Sons.

Tang and Isaacs, 1993

Tang, J. C. and Isaacs, E. (1993). Why do users like video? In *Computer Supported Cooperative Work*, pages 163-196. CSCW.

Thiebaux, 1997

Thiebaux, M. (1997). The Virtual Director. Master's thesis, Electronic Visualization Laboratory, University of Illinois at Chicago.

Vasilakis, 1995

Vasilakis, C. (1995). CASA- Computer Augmentation for Smart Architectonics: Masters of Fine Arts thesis and performance art piece at Electronic Visualization Event 4, University of Illinois at Chicago.

Wang et al., 1995

Wang, Q., Green, M., and Shaw, C. (1995). EM - an environment manager for building networked virtual environments. In *Proceedings of the Virtual Reality Annual International Symposium. VRAIS'95*, pages 11-18. IEEE Computer, IEEE.

Ware and Balakrishnan, 1994

Ware, C. and Balakrishnan, R. (1994). Reaching for objects in VR displays: Lag and frame rate. *ACM Transactions on Computer-Human Interaction*, 1(4):334-356.

Zhang et al., 1993

Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D. (1993). RSVP: A new resource ReSerVation Protocol. *IEEE Network*.