Enhancing Collaboration in Virtual Reality Applications*

Víctor Theoktisto† and Marta Fairén
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract
We derive a complete component framework for transforming standalone VR applications into full-fledged multithreaded Collaborative Virtual Reality Environments (CVREs), after characterizing existing implementations into a feature-rich superset. Our main contribution is placing over the existing VR tool a very concise and extensible class framework as an add-on component that provides emerging collaboration features. The enhancements include: a scalable arbitrated peer-to-peer topology for scene sharing; multi-threaded components for graphics rendering, user interaction and network communications; a streaming message protocol for client communications; a collaborative user interface model for session handling; and interchangeable user roles with multi-camera perspectives, avatar awareness and shared 3D annotations. We validate the framework by converting the existing ALICE VR Navigator into complete CVRE, with experimental results showing good performance in the collaborative inspection and manipulation of complex models.

1. Introduction

Virtual Reality (VR) and Augmented Reality (AR) tools have been applied in all engineering fields in order to avoid the use of physical prototypes, to train in high risk situations, and to interpret real or simulated results. In medical applications they help patient monitoring, scanned data interpretation and surgery planning. In architectural settings enable designing, building, visiting and stress-testing upcoming facilities. Individual users inspect 3D scenes, navigate within models and manipulate objects and properties in these VR environments or VREs.

Most implementations of VREs begin as standalone applications, with collaboration requests arising from the natural desire of exchanging experiences. Allowing several clients to collaborate on the inspection of a model usually implies the development of a whole new application with distributed capabilities, adding network communications, and confronting portability problems due to the absence of a migration strategy.

We propose a “snap-on” superset framework for evolving complete Collaborative Virtual Reality Environments (CVREs) out of existing VR applications. The main contribution of our approach is a novel multithreaded architecture with a scalable peer-to-peer network topology that incorporates session layer management, a cross-platform message-passing communications library, and a hybrid collaborative interaction model with multiple avatar roles. The framework adjusts easily to working VR tools without affecting graphics performance.

In section 2 we review existing CVREs, and then use them in section 3 to build a feature superset taxonomy, with relevant collaborative interaction paradigms.

Later in section 4 we detail our contributions in the transformation of VR navigators into medium scale CVREs, using a generic framework for session management. The framework implements scene sharing using a scalable peer-to-peer topology, with a custom request broker and a corresponding message protocol. It is based on a collaborative user interface model for session handling, with callback hooks providing extensible functionalities, allowing avatar awareness with interchangeable user roles and the placing of shared 3D marker annotations.

In section 5 we describe a typical stand-alone VR tool used for the inspection and navigation of very complex models, the ALICE VR Real Time Inspector, developed at the Universitat Politècnica de Catalunya in Spain.

Section 6 shows the validation of the framework in the conversion of ALICE into a complete collaborative VR environment and a performance analysis of the enhanced clients using in a busy network set-up. Its new collaborative features allow users to share their virtual experiences and participate actively in the environment, showing highlighting. Exclusion control for the collaborative manipulation of objects in the environment is provided by shared object-locking service at the broker level. The same approach may be applied to similar applications, using a basic template that allows building new collaborative VR environments from scratch.

Finally in section 7 we plan for extending new capabilities into the framework, such as optimized scene-sharing and a software component to handle haptic interactions.

---


† On leave from Universidad Simón Bolívar, Caracas, Venezuela
2. Collaboration and Virtual Reality

Distributed environments exist since the introduction of the first networks. Scope and complexity have kept pace with distributed systems evolution, migrating towards distributed processing, data sharing, multiple execution threads and sophisticated display technology. Computer Support for Cooperative Work (CSCW) [1] is an umbrella term for distributed applications in which multiple users collaborate toward common goals, under a high level event notification and message passing architecture. When combined with several degrees of information sharing, 3D visualization and real world user-interaction metaphors they become Collaborative Virtual Reality Environments or CVRE’s. Remote participants using visual identities (called avatars) may navigate inside the virtual space, interact with other remote avatars, and propagate changes to neighboring objects. Figure 1 details all major components present in a complete CVRE, requiring sophisticated user interaction models for domain and object manipulation.

Among the first available distributed virtual environments was DIVE [2], in which all users were equal peers, communicating among themselves to synchronize state, environment and display information. In MASSIVE–3 [3], another early DVE, the approach used is the transmission of images and video in Virtual Reality (VR) or Augmented Reality (AR) environments. A review of simulation CVE’s developed at the US Air Force Institute, such as SIMNET and its successors is presented in [4], addressing issues in human-computer interaction, virtual reality, software engineering, 3D graphics, scene modeling/object modeling, and artificial intelligence. Duce et al. [5] characterize reference models for CVEs and the impact they have on the degree of collaboration, based on the increase of complexity in the collaboration. A different approach is presented in [6], where the CVE is included into a library called REPO-3D, which allows the migration/replication of graphic objects over the network.

Different network technologies can be used to enable distribution on a CVE (BSD-sockets, RPC, Java RMI, DCOM, CORBA, etc.). AVOCADO (later AVANGO) [7], DIVE, SIMNET, NAVL [8], NPSNET–V (Java-based) [9], and Distributed Open Inventor [10] apply different solutions, using multiple execution threads where each one has an image of the other participants in the interaction. COVISE [11], an OpenInventor implementation, uses a request broker to handle client connections, and a one master–several slaves approach using exec calls for local processes and rexec/rlogin/rsh for processes on remote computers. Audio channels and the use of 3D markers help steer the collaboration. In Distributed Open Inventor (DOI), each client has a copy of the global scene graph and the synchronization is done by using rsxp (replicated scene graph protocol). Some local variations on the graph are allowed, such as different levels of detail, “ghost” objects and local changes of color or texture.

Each client requires, thus, at least a partial image of the scene graph. Zeleznik et al. [12] use the scene graph as a communications bus instead of a tree, whose nodes are sites at different network nodes and are accessed by synchronized access mechanisms by heterogeneous applications. Diverse [13] uses remote shared memory and UDP network datagrams for a rapid memory interchange. Although it does not guaranty message delivery, the quick reception of almost complete information allows to maintain the coherence among participants.

The earlier treatment of arising temporal inconsistencies in distributed systems due to network delays was initially described by Lamport [14], while using logical clocks to resolve temporal causality relations between events [15], and their contribution to reduce shared scene corruption are detailed in [16] and the VOODIE system [17]. Greenhalg [18] describes a technique for embedding temporal links in the Massive-3 CVE, which allows recursive self-reference for models, such as showing a small version of the model (a maquette) as a 3D-map within itself.

The most sophisticated approaches delegate network management to components outside of the CVE. Some of them use customized solutions, like Octopus and Tweek for VR–Juggler [19] or CAVERNSoft for CAVElib [20]. Others are based on the use of the CORBA standard [21] for distributed services over the network, having a central object registry and a localization service. CORBA-based solutions need to implement an Object Request Broker (ORB) to resolve requests for object references, using an Interface Definition Language (IDL) to publish the interfaces of objects in a language-independent manner. This allows the development and integration of heterogeneous systems with adaptive services related to network performance, useful to insure QoS on high traffic environments, but this approach may be too complex for an expected short number of equal clients.

Blue–c [22] combines live video feeds in a distributed scene graph based on the OpenGL Performer toolkit and an...
API that minimizes synchronization overhead among local and shared objects, under the CORBA-based middleware. The NOMAD framework [23] additionally implements some real-time CORBA extensions to deliver environment state changes to a number of clients in a timely manner. Hubbold et al [24] in the GNU/Maverik/Deva VR system take a different approach, using a modular micro-kernel architecture with default callbacks and immediate-mode rendering, having the external Deva module in charge of the collaboration services. Yet another option is to tinker at a lower level with a custom communications protocol such as vrp (virtual reality transfer protocol) [25], implemented by a plugin architecture under the Bamboo API [26] for networked VE’s.

Uneven network speeds and workstation performance are a hindrance for truly massive CVRE’s. A solution currently used in P2P environments [27] establishes the SplitStream or bit torrent content distribution system. Objects are split in segments and load is distributed among all the participants, so the more clients there are, the faster distribution works. For truly massive architectures, VELVET [28] introduces an adaptive hybrid architecture through a filtering scheme based on multicasting that allows collaboration between supercomputers in high-speed networks with other clients in not-so-powerful systems and/or under slow connections.

3. A Taxonomy of CVREs

A taxonomy of the meanings and subtypes of copresence structuring human interaction in virtual environments is proposed by Zhao in [29], defined as the relationship between the physical conditions and the sense of being with others and how the former conditions the latter. Masa [30] describes the sensations users need to perceive for a complete experience in a CVRE:

- a shared sense of location in (3D) space
- a shared sense of (real) time
- a shared sense of co-presence (using avatars)
- a shared (external) communication channel (text, speech or video)
- a sharing mechanism for object manipulation

To achieve these perceived effects, the design of collaborative virtual environments must choose to implement some alternative to comply with the following orthogonal requirements. The first three apply to generic CVEs and the last two are specific to virtual reality environments:

i) session awareness
ii) scalable topology
iii) network transmission
iv) collaborative user interaction features
v) object complexity

This provides the designer with a recipe for creating a compelling sense of co-presence in virtual environments, no matter what the feature set is. All of the previously mentioned CVREs were profiled as part of this research [31] with the extended requirement set described next.

i) Session awareness: Persistence, the temporal or permanent effect of user interactions have in the CVRE system [32]; may be described as:
- Participatory: The CVRE exists only while the participants are in it, and shuts down when all participants leave the environment.
- Journaled: Session scripted for later state recovery, allowing recording and replaying of 3D temporal annotations to guide other clients (Massive–3).
- Continuous: The CVRE is always active. A simulation make change scene and objects even if no clients are connected (SIMNET).

ii) Scalable topology: Scene sharing schemes among participants [5] can range from:
- Homogeneous replication using broadcast
  Each client maintains a complete replica of the shared environment. Messages across the network maintain state information. No central control; a new client has to wait some time to get information sent by other clients broadcasting changes (SIMNET, DIVE);
- Shared-centralized on a server
  Classical client/server model one with one scene being shared by all, residing at a central server (CAVERN, NPSNET–V). When the server fails it brings down all the clients.
- Shared-distributed with client/server groups
  Several groups of servers and clients. Uses same scheme as mobile phones cells, in which clients are connected to the nearest or least busy server (DIVE, Massive–3, Octopus, NOMAD, VELVET);
- Shared-distributed using P2P actualization
  Peer-to-peer connections among all participants, either directly or using a third party relay (broker). Changes are atomically broadcasted to all participants. It comes in two flavors:
  P2Pr: Replicating the same scene graph at each node (DOI, COVISE), with objects stored locally. Synchronization is done by callbacks managing of session coherence.
  P2Ps: Sharing objects across the network in a distributed scene graph with remote objects (Diverse, Blue–c, GNU/Maverik)

iii) Network transmission: Using an appropriate protocol to the expected message flow, such as choosing UDP or TCP/IP packets; use of broadcast in a LAN (as in SIMNET), unicast (one-to-one, all) or multicast (one-to-many) addresses (CAVERN, DIVE,
Diverse, DOI, GNU/Maverik, Massive–3, NOMAD, NPSNET–V, Octopus, VELVET); procuring reliability, bandwidth and minimizing network latency.

iv) Collaborative user interaction features: the collaborative set of desirable manipulation and visualization interfaces, teleconference capabilities (chat, video and audio), flexible support for model construction, synchronous and asynchronous collaboration modes, adaptive multi-resolution strategies, interoperability standards, and virtual space shared utilization. Crucial features for CRVEs are: 3D annotation and action indicators for remote event notification, multiple alternate views, selectable avatars, and expected low latency response times.

v) Object complexity: determines the network broadcasting cost of object and scene changes [25], including LoD and multi-resolution models, as

- **Light objects**, Short messages containing event state and control information, requiring low latency, high-speed networks, such as trackers, sensors, status information (All systems).
- **Remote references**, local network references shadowing remote objects (All but SIMNET).
- **Heavy objects**, Big objects requiring reliable transmissions, but small enough to reside in local memory, (e.g. object 3D geometry, avatars or cameras) (All systems).
- **Real-time streams**, large-segmented data. Data so big it has to be transmitted in pieces and/or continuously, (e.g. big geometric objects, volume information, textures, video, audio, etc.). (CAVERN, Blue–c, Massive–3).

From the above, it is evident that many CVREs use multicast addresses for UDP or TCP/IP communications. The most recent ones lean towards P2P or small client/server topologies, with replicated or shared scene graphs. Only CAVERN, Blue–c and Massive–3 integrate segmented data such as video feeds, while some of the others resort to variable resolution schemes or out-of-core segmentation.

Massive–3 is the lone provider of a journaling mechanism for interaction recovery. Avatars are a common feature, but none allow multiple perspectives. Only DIVE, Massive–3, VELVET, GNU/Maverik seem capable of handling large user loads or huge data models.

As far as the former reviews show, there is no clear strategy allowing an orderly and easy migration path from standalone VR applications to collaborative ones.

3.1. Collaborative Interaction Models

A special attention must be provided to interaction issues in distributed setups. There are two widely used conceptual paradigms in the design of user interfaces: the Model-View-Controller paradigm, known as MVC, and the Abstraction-Link-View paradigm or ALV, shown in Figure 2. The classical “Model, View, Controller” (MVC) triad of classes for user interface design (GUIs) was first introduced as part of the Smalltalk-80 language at Xerox PARC by Reenskaug [33], refactoring all application objects in three categories according to their functional roles in the user interface:

- **Model** objects residing in an algorithmic layer.
- **View** objects located in a visualization layer.
- **Controller** objects, user interface widgets’ layer that translates interaction into actions.

Communication among layers is achieved by an internal messaging system that feeds user actions into a switchboard event loop with a dispatcher. Callbacks connect each switch hook with the corresponding widget object(s), which in turn effect changes in the domain.

In the Abstraction-Link-View (ALV) paradigm [34], objects are refactored in abstraction, view and link layers.

- **Abstraction** objects are models shared by all users.
- **View** objects handle user interaction and visual rendering
- **Link** objects are constraint sets synchronizing abstraction and view objects.

The ALV’s Abstraction layer is equivalent to the MVC’s Model layer, while the ALV’s View component layer merges both View and Controller layers of MVC. The ALV’s links connect abstractions and their views, using references to remote objects. Consistency in ALV is kept tracking local state changes at a central repository, while the MVC’s Controllers handle communications among all its objects.

For CRVEs, the decoupled MVC approach proves insufficient because it does not provide for a common persistence layer to hold the shared state properties of remote interactions. The ALV model does provide a method for keeping track of object and session changes, but it is heavily slanted toward a client-server distributed model. In subsection 4.3 we propose a more suitable session management model to render reliable object flows at high speed rates.

![Figure 2. Side-by-side correspondence between the MVC and ALV collaborative UI paradigms.](image-url)

Many virtual reality applications begin as scene and object visualization environments, having special user interface metaphors for navigation and manipulation, and shown on display devices ranging from CRTs to immersive stereo projection systems. Most science disciplines (and the entertainment industry) use VR techniques to enhance user experiences. As research shows, users always desire to share these virtual experiences, either by showing models to prospective audiences, or by having an active remote participation in the environment.

Evolving collaboration at this stage usually entails the redesign and development of a (new) application, inserting a networking infrastructure under the environment, and other software-porting problems. Issues such as synchrony overheads, concurrent user load and system lags may degrade interaction and adversely affect graphics performance. There are generic API libraries for implementing shared scene graphs [35] that could be used for building multithreaded CVREs.

In the following subsections we describe the collaborative features of the proposed superset framework. The rationale behind our approach is that the object-oriented nature of current standalone VR applications, usually having rendering and user-interface components, facilitate their transformation into complete CVRE’s, by allowing the seamless attachment of a network-based component to enable collaboration. Given that the different VR tools may spread across platforms and support varied output display systems, the ideal solution should not compromise current designs or imply extensive recoding of components. Massive or large-scale implementations were discarded due to user administration performance considerations, although the proposed framework has scaled well for a reasonable number of (less than twenty) participants.

Based on the features described in Section 3, our solution involves the implementation of
- A multithreaded software components architecture,
- A scalable P2P sharing topology,
- A session awareness management layer, and
- A network cross-platform transmission library.

We have left for a future implementation the treatment of real-time streaming, since the framework does not modify the current object granularity of the target application. On the practical side, it is a portable generic framework (not an API), requiring only the instantiation of a custom message-parsing class for the shared session.

4.1. Multithreaded Software Components

We assume that a good VR tool is the final product of a sound systems design, developed under a classical MVC paradigm. A standard software engineering practice in Computer Graphics is the refactoring of application objects into at least two weakly cohesive software functional components, graphics rendering and user interface. We decouple the Graphics Rendering (GR) and User Interface (UI) parts and instantiate them in separate threads. The same approach is taken with the new network communications component (NC), launched in its own separate concurrent thread. In this way, advantage is taken of the underlying operating system’s context switching, loading the new software components without altering functioning code. This extensible approach allows the addition of more component threads, such as one dedicated to tracker data acquisition or interaction with haptic devices.

A snapshot of a working framework model is shown in Figure 3, detailing each software component. The NC component thread handles communications and message parsing; the top Shared Session (SS) management layer (see the MVCS model in subsection 4.3) launches all concurrent threads, tracks users’ avatars, propagate state changes to the UI and GR components using callbacks, and is in general responsible for the emerging collaborative behavior; the GR and UI components are mostly untouched except for the binding “glue” to the Shared Session layer.

This setup is implemented by means of an abstract class wrapper incorporating network awareness and a corresponding message protocol. An appropriate set of mutexes avoid shared state inconsistencies and race conditions when updating information.

4.2. Scalable P2P Sharing Topology

Fitting any of the client/server topologies would have implied the creation of at least one central server from scratch and compromised the applications’ standalone behavior. We chose instead a peer-to-peer scalable topology, the most adequate for equal participants with separate access to their models. There are two possible topologies available in the framework: P2P [Peer-to-Peer with scene replication] and P2Ps [Peer-to-Peer with scene sharing].
In a P2Pr topology, each client has its own local scene replica. Since only a few scene objects are modified in the session, collaboration starts as soon as all clients have loaded their common model, and situated themselves within it. If there are no other participants in the environment, it defaults naturally to the standalone behavior.

A P2Ps topology must build a shared scene graph first, with each individual client adding whole chunks. For a particular client, scene graph objects are labeled local or remote depending on whether they are cached internally or need to be fetched elsewhere. If a client fails, its part of the shared scene must be reconstructed by the others.

**Thin broker for session administration**

With no central server, both approaches require a third party to locate clients willing to enter in a session. In our proposal, this third party is called a message broker, tracking session interaction, as seen in Figure 4. It is loosely based on some CORBA facilities, but without the associated overhead of an IDL implementation. Shared state information is kept through the following services:

- A name service for location and client registration.
- A session management service.
- A session/client state report and mirroring service.

Since the broker is not a bridge, client messages must go directly to their destiny. Each client keeps track of other participants, and periodically may send its current state to the broker for shared session recording purposes.

The broker can be easily extended to cover other collaborative functionalities that require some session control over them. An example is explained in subsection 6.1 for the control of collaborative manipulations in ALICE.

**Message protocol**

The message protocol is short and simple. Its main objective is synchronizing session state across participants.

There are two kinds of expected message flows:

- **Continuous Session Updates**, such as participants’ position and orientation, moveable objects, video and audio streams; uses fast [or real-time] multicasting.
- **Discrete Session Updates**, high-level changes in object properties (such as texture), manipulation, text chat; uses two channels: reliable unicast (for peer-to-broker, or P2B) and multicasting (for peer-to-peer, or P2P).

Location and orientation messages may be the avatars’ camera coordinates being continuously broadcasted among all participants as they move about. Session messages are the ones exchanged between the broker and the peers: connecting and disconnecting, reporting internet addresses and ports, number of active cameras, avatar appearance, global scene file, and other relevant data. Manipulation messages (such as a local client touching, grabbing, adding or modifying an object) are sent to remote users by the callback system to maintain scene coherence among all participants.

A message parser class in the NC thread listens asynchronously in three separate [configurable] ports, one for each of the message channels (continuous location messages, peer-to-broker communications, and peer-to-peer callback diffusion). All messages are of the form:

<table>
<thead>
<tr>
<th>MsgID</th>
<th>Session (P2B) or Callback (P2P) ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target</td>
<td>Peer IP, Broker IP, All</td>
</tr>
<tr>
<td>Flags</td>
<td>Session context properties</td>
</tr>
<tr>
<td>Parameters</td>
<td>&lt; parameter, value &gt; tuples</td>
</tr>
</tbody>
</table>

To avoid parsing overhead, no metadata information about the parameters (such as type specification or semantics) is included in the messages. Since participants are all homogeneous, the MsgID determines exactly the expected number, order and type of the received parameters for all peers (and the broker). Thus, no IDL is necessary.

**4.3. Session Awareness Management Layer**

After analyzing the desirable characteristics exhibited by existing CRVEs, we concluded that the optimal feature set for session awareness is described by the following:

- Collaborative user interface model.
- Client awareness using avatars.
- Session management with differentiated user roles.
- Shared annotation and 3D marker highlighting.
- External real-time verbal communication channel.

For a client in this scenario, there must be perceptual evidence that other entities (human or otherwise) are participating, so 3D client embodiments (avatars) are used to dynamically reflect their position and state in the scene. Clients may want to call others to attention by placing special 3D signals, leaving trails in the scene or modifying the environment. Some users could just browse through the model, while others could have object editing privileges. A collaborative interface metaphor allows the remote manipulation of objects, and session tasks may keep a journaled record of the interaction.
Collaborative user interface model

The problem to solve when recasting existing VR navigators as CVREs is how to implement the maximal collaborative feature set with the least possible implementation cost, and without affecting the original standalone behavior. We pick from each category of Section 2 the items that better support awareness under a hybrid Model-View-Controller-Session (MVCS) approach, tying the ALV’s links as network pipelines to MVC objects, in which:

- MVC objects may not reside together at the same network node, having their Model (structure and behavior) defined at one client, many different Views elsewhere (renderings, at least one for each client), and flow control effected by all. Nodes may have several viewpoints (cameras), allowing for multiple perspectives and resolutions of the same scene.
- Controllers operate using a callback mechanism, routing to the corresponding network nodes for non-local objects, as shown on Figure 5. Session layer coherence is maintained by existing network-aware controllers at each node, who also notify the broker. It does not matter whether objects are shared or replicated, so it allows either P2Pr or P2Ps approaches.

Client awareness using avatars

Each client has its own 3D representation traversing the environment, having several active camera perspectives at any time. Avatars broadcast a number of state attributes, such as position, orientation and velocity camera vectors for dead reckoning calculations.

Session management with differentiated session roles

So far we have identified five different collaborative user behaviors: standalone, peer, incognito, slave, and master. A standalone client is not aware of other clients. It defaults to the original isolated behavior of the application. Peers are clients that communicate among themselves using the common message protocol. Users traveling incognito may observe scene interaction in “voyeur” mode without other clients knowing it. A slave is a peer that is bound to another, correspondingly called a master, in the sense that the master’s current state is continuously replicated by the remote slave(s). These client roles are voluntary and changeable during a session, leaving open the possibility of adding more roles. A self-explanatory three bit code catalogues their functional role results in the following user codification (from left-to-right):

- bit 2 whether the client broadcasts messages to others
- bit 1 whether the client listens to remote messages
- bit 0 whether the client binds to another

<table>
<thead>
<tr>
<th>standalone</th>
<th>incognito</th>
<th>slave</th>
<th>peer</th>
<th>master</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>010</td>
<td>011</td>
<td>110</td>
<td>111</td>
</tr>
</tbody>
</table>

Shared annotation and 3D marker highlighting

Users must not only be aware of each other, they must be able to call the attention of remote participants to some feature or object in the environment. This is accomplished by temporal 3D markers such as arrows, billboards or banners that are pinned at interesting locations.

External real-time verbal communication channel

Collaborative environments use at least one real-time communication channel to allow the human users behind the workstations to exchange impressions about the virtual experience. The framework does not provide this service, but external suitable open-source cross-platform alternatives such as Gaim, Gnomeeting and others have been used with equivalent ease.

4.4. Cross-platform Network Transmission

Since communication is what enables collaboration, the new NC software component handles network communication capabilities. This is done by a cross-platform networking class that allows either datagram-oriented (UDP) and connection-oriented (TCP) communications in IPv4 or IPv6 multicast networks. The NC thread, under a common message protocol, implements the following basic services, each one running on its own separate listening socket:

- Shared event pipeline for sending environment state changes and callback messages
- Continuous streaming of client properties or data, such as camera position and orientation
- A notifying service for the Broker.

When a client reports to the broker, it posts its network address and listening ports. A configurable setup accounts for external firewalling rules, allowing several clients to run concurrently on the same machine by choosing unique port numbers. This enhances performance tests, because it permits the simulation of heavier client loads independently of available workstations. Network traffic is generated only for broker requests, for position or orientation changes, and for shared callbacks (such as object manipulation).

System synchronization

The framework avoids hosting a central time server by
keeping relative time differences for every peer-to-peer connection at the client’s side. The local event time or timestamp is included in each network message. Clients at the other end may process incoming messages as either

- **Immediate:** messages are processed at once, or
- **Buffered:** messages are queued by timestamp.

When using the first approach, high network traffic may produce jumpy updates and short temporal inconsistencies. The second is more suitable for replaying events in exact time sequence, at the expense of bigger time delays.

5. The ALICE Virtual Reality Navigator

The ALICE VR Real Time Inspector and Navigator [36] is a standalone VR software platform for the real time inspection and navigation of very complex virtual models, developed at the Universitat Politècnica de Catalunya. It has been used in a number of applications such as navigation in urban environments or interior ship design. In order to allow the users of these applications to be able to navigate and inspect complex 3D models in several VR systems, ALICE offers the following features:

- **Stereoscopic visualization:** works either with active stereo, such as Head Mounted Displays [37], or passive stereo, for low cost VR systems [36], [38].
- **User position and orientation tracking:** allowing implicit interaction by following the user’s movements and making him feel he is inspecting a real object.
- **Different VR modes of execution:** able to work over different VR display systems like stereoscopic tables, the CAVE, etc.
- **Use of multiple interaction devices:** being able to follow orders from mice, joysticks or VR gloves.

Apart from these external features, ALICE implements internally an extensible system of callbacks and many advanced computer graphics algorithms, in order to be able to work interactively with highly complex scenes. It uses internally a hierarchical object scene graph, keeping also for each element non-geometrical information, allowing, for example, multi-resolution textures. Among these advanced algorithms are the following:

- **Simplification techniques:** ALICE maintains different levels of detail (LoD) for all objects in the scene, allowing a faster visualization of complex models by choosing the right level of detail depending on the distance between the object and the observer [39] (further objects can be visualized with less detail without loosing image quality).
- **Visibility culling:** This technique eliminates from the visualization process those parts of the geometric model that will not be visible from the current observer’s (camera) position [40]. The technique may be combined in ALICE with the multiple LoDs [41].
- **Collision detection:** The collision detection is a key component for any VR system, being the base for object manipulation (such as using a VR glove to select objects by virtually “touching” them [42]), robotics, vehicle simulators, etc.

6. Performance analysis of the framework

The ALICE application is already refactored into two software components, Graphics Rendering and User Interface. The User Interface component is provided by Qt, an object-oriented user interface cross-platform toolkit (using the MVC paradigm) for MS Windows, several UNIX variants, GNU/Linux and MacOS X. The Graphics Rendering component is an OpenGL class frame (which may expand to the complete screen space).

The decoupled callback hooks system in ALICE connects user events to the graphics pipeline by means of a indexed command list. Each element of the lists stores a settable reference (the “hook”) to some object’s method (the “callback” function). When an UI event triggers a particular command, its corresponding callback hook is executed with the provided event information and current environment state.
6.1. Framework validation in ALICE

Given all the above, it was considered a suitable candidate for enhancing its collaboration features. Just changing some flags at the compiling phase allows the UI and GR components to load in separate threads. Next, the following steps were taken to fit ALICE into the framework:

1) Instantiate the shared session (SS) layer class, holding all common state awareness attributes, such as the scene graph, avatars, remote references for the broker and the list of participants.
2) Choose a scalable topology (P2Pr, for this version).
3) Devise the peer-to-peer and peer-to-broker message protocols.
4) Instantiate the message parser class to process event messages, and place it in the networking communications (NC) component.
5) Wrap the GR, UI and NC software components as SS layer class attributes, and launch each of them in a separate thread.
6) Add one method call to provide a callback hook linking the message parser class in the NC component to the session-update method of the SS component.
7) Add one method call in the UI’s main method to provide a callback hook to the SS layer.
8) Add one method call in the GR to provide the callback hook syncing the cameras and states of network peers just before rendering.
9) Instantiate the broker class, adding the necessary services.

The whole setup comprises just five classes: session-peer-broker communications, thread management, message parsing, callback serialization and remote camera handling.

Peer and Broker instantiation

A scalable P2Pr topology was initially chosen [43], given that all clients already function with local scene replicas and it would not change much ALICE’s behavior. In shared mode, the broker indicates the remote reference of the current scene, so hopefully everyone would be placed in the same model. The broker must be active for a session to be initiated by at least two subscribing participants. Each client may choose a session role (usually peer) and an avatar representation (from a menu), as shown in Figure 7, while keeping a list of the current active interactions with other users. Every time a new client is connected, it gets from the broker the name of the current scene and the list of currently blocked objects. As they navigate, clients may chat, speak or video-conference among themselves using unrelated helper applications, while placing 3D markers to call each other’s attention to particular features in the scene.

Clients can also take the role of “voluntary slaves” for some other user, which now becomes a camera server. The slave shuts down its own cameras and reflects the master’s camera viewpoint and actions, the latter effectively taking possession of the slave’s remote display devices. This feature may also be used to “teleport” a participant to the position of another, which is very useful to avoid losing virtual eye-contact among peers. Each node does independent renderings, which allows a client to show a wireframe representation while another fully renders the same scene.

Message protocol implementation

The message protocol for communicating with the broker is the same for all implementations, since it is application-independent. The continuous communication channel for
location/orientation messages may be used as implemented for avatar presence notification, although it may be extended to add continuous streaming of data acquisition from haptic input devices. The message parser class must be extended to handle the manipulation messages proper to each new VR tool. Callbacks that affect model integrity or are subjected to user interaction (such as a local client touching, grabbing, adding or modifying an object or the scene) were fitted with serialization and de-serialization methods to encode and decode the callback replication message, which is immediately multicasted to all peers in the session (plus the broker, if the session is being recorded). Out of the growing callback set of ALICE (around 100), only a subset of 14 affect model integrity, shared scene state and object appearance, although more may be defined in the protocol.

Semaphores in the NC thread activate and deactivate the UI and GR threads when it is modifying incoming packets such as clients’ cameras, so to avoid the race conditions so common to concurrent programming.

6.2. Emergent ALICE collaborative features

A practical side arising of an implementation based on abstract wrapper classes, is that it is platform-agnostic and extensible, which makes it quite portable. Each application only needs to inherit from the message parsing class, add its own protocol processing code and provide the hooks for the UI and GR components. Given the new features provided by the framework, ALICE was enhanced visually and structurally with three new important additions

i) Remote user windows

This additional feature to the collaborative capabilities of ALICE allows the user to overlay several small frames echoing the rendering of remote collaborating users on top of the application’s own rendering. In Figure 8 the main window shows the avatar of the remote user whose rendering is being visualized in the small north-east corner.

Since the purpose of this feature is to know how the remote user is seeing the scene at any time, some information about the local rendering of that user is also required to pass through the network (apart from the information of position and orientation of his camera). This information includes, for example, the position of far and near clipping planes.

ii) Extending the callback mechanism

When a user interacts with ALICE, he can cause some changes in the scene or in his representation in the collaborative session (changing the avatar or becoming stand-alone, for example). These changes should be broadcasted to the rest of the participants on the session, so they can be aware of what the changes were.

As already stated before, an ALICE callback is a code entity hooked to a certain task that has to be done in the application. Callbacks that imply a network communication, causing the replication of its task in all other participants in the collaborative session, require the implementation of methods for serializing, sending, receiving and de-serializing the information associated with it. Since each callback may involve different number and type of parameters, its serialization and de-serialization is known and performed by the same callback, at both ends of the network. This also has the advantage of packing the parameters as one stream of data, avoiding sending redundant metadata overhead. The only information that has to be sent with the callback network message is the unique ID of the callback method able to de-serialize the message.

Examples of callbacks included in the collaborative ALICE application are the following:

- change avatar representation, the user selects a different representation for its own avatar;
- add a new light to the scene;
- select/deselect an object;
- change a property of the selected object;
- eliminate the selected object;
- move/scale/rotate the selected object;

The last four callbacks listed are directly related with the collaborative manipulation of objects explained next.

iii) Collaborative object manipulation

ALICE, as a navigator and inspector, has limited modeling options in the scene, but it allows the user to move and delete objects and also to change several object attributes like color, for example.

Since this collaborative object manipulation requires some control at session level, the broker’s framework (explained in subsection 4.2) has been extended to support a locking hierarchy for object selection. When a user wants to manipulate an object he has to select it first by asking the broker whether the object is already selected by another user. The requests for selecting an object arriving to the broker are served by timestamp, so the oldest request is the one who locks the selection. When the broker accepts a
selection coming from a user, this selection is multicasted (via the callback) to the rest of the users in the session. In order to increase user awareness of other participants, objects held by remote users are labeled as remotely selected in the local copy of the scene, and the visualization routine overlays a red wireframe around them. Objects labeled locally selected are overlaid with a white wireframe. This information of remote selections is also kept in the list of remote users, to be able to recognize the user currently holding a certain object, useful when performing the rendering of that user’s viewpoint in the remote frames. For example, in Figure 8 the finger being manipulated by the local user is being shown as remotely selected in the small frame of a separate peer.

Object locking prevents two or more clients to modify the same object. However heavy traffic conditions may give raise to a well known relativistic effect in networks. High and irregular latency conditions may cause that earlier messages from a client, such as a position change, arrive later than they are supposed to logically occur, and produce short visual inconsistencies. The effect is reduced at each peer by the use of a time-ordered event message queue based on relative timestamp differences between peers, or may be ignored altogether by ignoring all network events whose timestamp signature is earlier than the current time.

6.3. Evaluation of Results

ALICE has been successfully used as a collaborative platform over different VR systems, such as HMDs, Stereoscopic tables [36] or the portable system [38] (see Figure 6). Some of the applications are the following:

- Inspection and navigation in the interior of a ship. Ship designers cannot use real prototypes to evaluate their designs, so they need virtual prototypes to be evaluated. Different users may discuss about the resulting design without needing to be at the same place at the same time.
- Training in medicine. A surgeon can show the students, for example, how the scalpel should be used for a certain surgery incision.
- Inspection and modification of architectural design. The virtual design made by the architects can be shown to potential clients in order to know their opinion. A client, by collaborating with the architect (see Figure 7), can also make small changes such as moving furniture or walls in order to propose other possibilities not far away of the initial design.

We have tested ALICE’s remote collaboration and navigation services in the several VR systems in our lab, and also in sessions with the Girona University (located 100 Km. from the Barcelona campus) through a 10Mb wide area network connection. In our lab we have available HMDs, a stereoscopic table, a CAVE, a MiniVR system and flat displays; and a similar setup at the Girona campus. The results obtained from our tests can be seen in the following table. The scene used on these tests (the interior of a ship, see Figure 9) contains over 50,000 polygons (300,000 rendering triangles), but on purpose does not have complex textures that could skew graphics performance. The table shows the results obtained in the communication of 2, 4 and 8 workstations using unicast addresses from both sites.

<table>
<thead>
<tr>
<th>Participants</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. # of messages</td>
<td>–</td>
<td>2539</td>
<td>8067</td>
<td>14331</td>
</tr>
<tr>
<td>Avg. Net Time (ms)</td>
<td>–</td>
<td>35</td>
<td>31</td>
<td>46</td>
</tr>
<tr>
<td>Avg. Latency (ms)</td>
<td>–</td>
<td>13</td>
<td>46</td>
<td>57</td>
</tr>
<tr>
<td>Avg. Framerate (fps)</td>
<td>47.3</td>
<td>45.2</td>
<td>44.2</td>
<td>42.7</td>
</tr>
</tbody>
</table>
In the table we observe the average total number of messages sent through the network in a series of repeated navigation trials, each test lasting 4 minutes. The total network time (in milliseconds) gives information about how much time ALICE spent in the transmission of messages during these 4 minutes tests (this means that only around 0.1-0.2% of total time was spent in network communications). The roundtrip time is also indicated in milliseconds. Since for this test we use unicast addresses, roundtrip time increases as more peers participate in the session. Finally, the table shows the average rendering framerate achieved for each case, which indicates that increasing the number of nodes affects graphics performance very slightly compared to the standalone performance, and is comparable to similar setups in the studied environments.

As already stated in subsection 6.1, the migration of ALICE to a CVRE was fast and uneventful. Based on the fact that the application was already designed considering graphics rendering and user interface as separate components, its porting to our framework only required to define a message protocol, connect the appropriate callback hooks, and add two method calls and corresponding code hooks in order to attach the application to the new network and session parts. Following the same migration scheme, it should be easier to transform any other VR application into a collaborative VR application, given that the peer-to-broker protocol won’t change much, if at all. In fact we are presently porting another application built in our lab which addresses inspection and management of medical models.

Some fine-tuning must be performed to adjust threaded execution. A highly textured model may take a while to render, making timely interaction slow and difficult. Although this can not be avoided, it may be reduced by changing thread priorities to model complexity and network traffic. To measure performance in a clogged network, we created high traffic conditions by loading multiple ALICE instances at each workstation, resulting in clients falling out-of-sync due to packet losses. To minimize these latency problems, there is an option to process only the most recently received (by timestamp) packet from each peer in the environment, at the expense of a somewhat jumpier navigation.

The proposed mechanism for camera management and sharing is reasonably easy to learn for users and seem to be adequate for collaboration tasks. We want to make some experiments with untrained users soon in order to have a more accurate perception of ease-of-use.

6.4. Developing new VR applications

The component-based nature of the framework allows it to be applied to other stand-alone applications with ease. At a very low level it works using a POSIX threads (**threads**) implementation, which is API-compatible with several open-source and commercial thread implementations. Since the rendering component is not modified in any way, as long as it runs in its own thread it should not be adversely affected.

If what is desired is the development of a completely new application, then there are several options available. Using our framework from scratch is very straightforward, which allows for several teams working separately into each component. A ready-made template, providing just a main rendering window, a minimal user interface and networking capabilities with basic broker support provides the basis for a new development. However each development must provide its own message protocol and interpreter.

7. Conclusions and Future Work

Based on a characterization of generic collaborative features for VR systems, we have proposed a versatile framework for evolving collaborative capabilities in stand-alone VR navigators. Our approach incorporates a hybrid distributed user interaction model, multithreaded software components, network communications under a peer-to-peer scalable topology, message passing channels with a custom protocol, and changeable user roles in a multi-camera subscription model.

The framework’s development has been validated by a fast porting of the ALICE VR Navigator. The generic cross-platform design allows an easy migration of similar VR applications into complete collaborative virtual reality environments. Most of the reviewed CRVEs toolkits use a proper API to build a new application. Our approach, being a collaboration framework embedded in a “snap-on” component, needs only minimal adjustments on the original application, avoiding the burden of a major code refactoring and development time required by an API approach.

Several enhancements have proposed for increasing the network performance of the communications component, switching to a real-time network protocol such as RTP. This will reduce or eliminate altogether the need for clients to keep track of each other current latencies. Also, all network messages (with the possible exception of camera coordinates’ broadcasting) may be encoded into XML messages, using SOAP-based communications, which may allow for a more standard and extensible implementation of the message protocol.

We are working on extending the collaborative breadth of the framework by including in the Session layer a fourth thread for handling haptic devices, adding high frequency force-feedback events to the interactive session repertoire, and haptic texture streaming. Given the huge scene size of current VR scenes and objects, we plan to migrate applications towards a peer-to-peer with sharing scheme
(P2Ps), and also to allow the incremental streaming of multi-resolution objects to improve rendering performance and scalability.

Acknowledgements

We want to thank professor Isabel Navazo for many fruitful discussions on different aspects of this research. This work has been partially funded by from the Ministry of Science and Technology of Spain, grant TIN2004–08065–C02–01.

References


