An Overview of the COVEN Platform

Abstract

A central aim of the COVEN project was to prototype large-scale applications of collaborative virtual environments (CVEs) that went beyond the existing state of the art. These applications were used in a series of rea-scale networked trials that allowed us to gather many interesting human and technological results. To fulfill the technological and experimental goals of the project, we have modified an existing CVE platform: the Dive (distributed interactive virtual environment) toolkit. In this paper, we present the different services and extensions that have been implemented within the platform during the four years of the project. Such a presentation will exemplify the different features that will have to be offered by next-generation CVE platforms. Implementation of the COVEN services has had implications at all levels of the platform: from a new networking layer through to mechanisms for high-level semantic modeling of applications.

1 Introduction

Collaborative virtual environments (CVEs) are networked virtual reality systems that support group activities. Their central concept is one of a shared virtual world, a computer-generated space where participants can meet and interact. Users are embodied in the world by an avatar, a three-dimensional object that represents their movement, action, and presence in the space. The contents of the shared virtual space may represent data, other users, or interfaces to autonomous computer programs. Because users’ actions are embodied within the space through their avatar, it is possible to visualize what the other participants’ activities are, and what they are focused on. Consequently, an essential achievement of CVEs is that they combine the participants and the information that they access and manipulate in a single common space, and thus facilitate users’ collaborating on shared tasks.

CVEs afford many novel uses as they can effectively present spatial data, such as a large city or landscape to a community of users, so that those users can interact with respect to this space. Although the advantages of CVEs have been heralded for a number of years, the supporting platforms and infrastructures have been restricted to application development. This is due to a number of factors, such as immaturity of the product, limited functionality (that is, the product will support some but not all of the technologically available features), poor application development support (a complex programming paradigm must be learned), and, generally, a lack of understanding of the nature of CVEs, due to the immaturity of the technologies (as compared to the variety of usability guidelines available for more-conventional interface technologies).
The COVEN project aimed at addressing these issues, not individually, but in concert, to provide a common model for application development (Normand et al., 1999). The result of this approach is a mature, reliable, and scalable platform with a common application development model that supports a number of application interfaces. The most popular interface is the built TCL scripting interface. Using scripts attached to objects in the world—which can communicate directly or through shared properties—provides an efficient environment for prototyping VR-based, single-user, or CSCW applications.

In order to achieve rich, interactive, extended, detailed, and lively environments populated by many users and applications, COVEN has addressed many different scalability aspects, and has realized these in the DivE platform. In this paper, we present the different extensions that have been applied to DivE to fulfill the demanding requirements of the COVEN applications.

2 Background

During the initial stages of the COVEN project, existing CVEs simply provided a number of users with a controllable avatar in a fairly static 3-D space. The major concerns of such systems were the low-level technical problems of basic reliability and consistency of the shared world. Applications written in these systems were often hand-crafted, usually in a low-level language, and required in-depth knowledge of the actual CVE system itself. These are common problems of any system that can be characterized as a stable laboratory prototype. The underlying mission of the COVEN project was to investigate existing CVEs to determine the type of services required by a next-generation CVE platform.

Through network trials and application prototyping, we hoped to understand the fundamental types of service that a CVE would need to supply in order to support collaborative work. The driving application areas that were selected for the exploration of platform requirements included virtual teleconferencing, collaborative object arrangement, and virtual travel rehearsal. Although the aim was to choose areas that were thought to impose different demands on the software system, all applications shared a common characteristic, that of involving multiple users.

From the studies of CVE services, a platform was defined in two stages (COVEN, 1997, 1998), and this was implemented through extensions to the DivE system. Perhaps the most significant result of the COVEN project is the platform itself, an enhanced VE supporting a broad range of collaborative services. In this paper, we detail the extensions and enhancements made to DivE. They range from low-level network enhancements, such as semantic message passing, to high-level usability enhancements, such as individual, subjective views of the shared world.

3 Approach

Our approach has been to take an existing CVE platform and make orthogonal extensions in several areas: support for large numbers of users, world persistence, support for large and complex models, media integration, and support for rapid prototyping.

To support an increasing number of participants and objects within virtual environments, the COVEN platform incorporates a number of scalable networking techniques. The common theme of these solutions is that they utilize the properties of occlusion and attenuation with distance to reduce the number of participants that truly share a region of the environment. (See subsections 5.1 and 5.2.) Additionally, particular attention has been paid to the most common network operations, so as to reduce their size and frequency, whenever possible. (See subsection 5.3.)

COVEN addresses two other significant aspects of CVEs, namely keeping the environments persistent over time (see subsection 5.4) and establishing an infrastructure that will allow network traffic and human-factors analysis. (See subsection 5.5.)

COVEN has also provided support for many of the media types that have been made popular by the Web, such as still and animated pictures, video, and composite documents (subsection 5.7). In particular, to improve the quality of the simulation, point audio sources (that
is, object and avatar sounds) and ambient sounds are mixed and spatialized to render a soundscape that corresponds to the three-dimensional structure of the environment. (See subsection 5.6.) Presenting environments that are geometrically detailed and extended, as well as visually appealing, is key to certain applications. However, interactivity is a requirement of any real-time system, and the COVEN platform incorporates scalable rendering techniques that provide a continuous experience by maintaining a constant frame rate. (See subsection 5.8.) To provide detailed models, support is provided for international standards such as VRML (the Virtual Reality Modeling Language). Additionally, the COVEN platform is tightly integrated with a three-dimensional modeler that supports import and export of many file formats including the native DIVE file format and VRML. (See subsection 5.9.)

CDEV applications need to be dynamic and responsive to user interaction. DIVE already offered a low-level C programming language interface when the project began. COVEN has developed several complementary ways of implementing applications. These include scripting support for rapid application prototyping (see subsection 5.11) and external programming interfaces to offer lightweight mechanisms to allow external applications control over the environment. (See subsection 5.11.2.)

Finally, COVEN has enriched the user’s interface to the virtual environment both by supporting various types of multimodal interface (such as voice recognition) and by giving users and applications the ability to tailor an individual’s user interface. (See subsection 5.10.)

4 A modular summary of DIVE

In this section, we describe the different modules that compose the COVEN DIVE platform. This description gives an overview of the platform and of its different components, together with the basic functionality that they implement.

![Figure 1. The different modules that compose the DIVE system, together with their interfaces](image)

4.1 System Services

At the conceptual and programming level, DIVE is based on a hierarchical database of objects, termed entities. Applications operate solely on the database abstraction and do not communicate directly with one another. This technique allows a clean separation between application and network interfaces. Thus, programming will not differ when writing single-user applications or multiuser applications running over the Internet. This model has proven to be successful: DIVE has changed its interprocess communication package three times since the first version in 1991, and existing applications did not require any redesign.

Whereas the hierarchical model is inherited from traditional scene graphs, as used in the computer graphics community, the DIVE database is semantically richer. For example, it contains structures for storing information about other users, or nongeometric data specific to a particular application. In DIVE, the database is partially replicated at all participating nodes using a top-down
approach; that is, mechanisms are offered to control the replication of subbranches of a given entity. Other platforms have chosen different mechanisms (Singhal & Zylla, 1999), such as using proxies, to represent distant objects in the local database (Community Place, 1999) or more object-oriented biased approaches (Greenhalgh & Benford, 1995). It is worth noting that our conceptual model is very similar to that of the Spline system (Waters et al., 1997), even though both systems have emerged from distinct efforts and developed separately.

In Dive, an event system realizes the operations and modifications that occur within the database. Consequently, all operations on entities such as material modifications or transformations will generate events to which applications can react. Additionally, there are spontaneous and user-driven events, such as collision between objects or user interaction with input devices. An interesting feature of the event system is its support of high-level, application-specific events, enabling applications to define their content and utilization. This enables several processes composed of the same application (or a set of applications) to exchange any kind of information using their own protocol.

Most events occurring within the system will generate network updates that completely describe them. Other connected peers that hold a replica of the concerned entities will be able to apply the described modification unambiguously. Network messages are propagated using the multicast mechanisms that are built in the system. Dive uses a variation of SRM (scalable reliable multicast (Floyd, Jacobson, Liu, McCanne, & Zhang, 1997)) to control the transmission of updates and ensure the consistency of the database at all connected peers. The SRM approach requires the transport layer to be able to ask the application (in this case Dive as a whole) to regenerate updates if necessary. Update regeneration is necessary when gaps are discovered in the sequence numbers that are associated with every entity in the database. Gaps imply that network messages must have been lost along the path from a sender to one of its receivers. In addition to SRM, COVEN extensions to Dive make it possible to access any document using more common network protocols (HTTP and FTP) and to integrate these documents within the environment by recognizing their media types (such as VRML and HTML).

In any application, the content of the database must be initialized. Dive uses a module that manages several three-dimensional formats and translates them into the internal data structures that best represent their content. Usually only one peer will load and parse a particular file, and the resulting entity hierarchy will be distributed to other connected peers through a series of (multicast) updates that describe the resulting entities. This specific mechanism differs from many other systems that rely on being able to access the description files from all connected peers, such as dVS from Parametric Corp.

Dive has an embedded scripting language that provides an interface to most of the services of the platform. (See subsection 5.11.) Scripts register an interest in, and are triggered by, events that occur within the system. They will usually react by modifying the state of the shared database. Moreover, these modifications can lead to other events, which will possibly trigger additional scripts. A series of commands allow the logic of the scripts to gather information from the database and decide on the correct sequence of actions.

The Dive runtime environment consists of a set of communicating processes, running on nodes distributed within both local- and wide-area networks. The processes, representing either human users or autonomous applications, have access to a number of databases, which they update concurrently. As described earlier, each database contains a number of abstract descriptions of graphical objects that, together, constitute a virtual world. A typical Dive application will, upon connection to a virtual world, introduce a set of objects to the environment that will serve as its user interface and start listening to events and react accordingly. One essential application of the system is the 3-D browser, called vishnu. Vishnu is the application that gives its user a presence within the environment. It introduces a new entity called an actor to the shared environment, which is the virtual representation of the real user.

### 4.2 User-Oriented Services

The services described previously are independent of any Dive application. This section focuses on the dif-
ferent modules present within the *vishnu* application that render a visual and aural space and provide the users with an interface that allows them to explore and interact with this space.

The primary display module is the graphical renderer. Traditionally, the rendering module traverses the database hierarchy and draws the scene from the viewpoint of the user. This module has been subject to a number of enhancements during the lifetime of the COVEN project. These include provision of geometry and material types beyond those found in common scene description languages, and support for a constant frame-rate rendering mode. (See subsection 5.8.)

**Dive** has integrated audio and video facilities. Audio and video streams between participants are distributed using unreliable multicast communication. Audio streams are spatialized so as to build a soundscape, where the perceived output of an audio source is a function of the distance to the source, the interaural distance, and the direction of the source (Adler, 1996). The audio module supports mono-, stereo-, or quadraphony audio rendering through speakers or headphones connected to the workstation. Input can be taken from microphones or from audio sample files referenced by a URL. Similarly, the video module takes its input from cameras connected to the workstations or video files referenced by URLs. Video streams can either be presented to remote users in separate windows or onto textures within the rendered environment.

Users may also be presented with a two-dimensional interface that offers access to rendering, collaboration, and editing facilities. The interface itself is written using the same scripting language as offered by the world database. Consequently, CVE applications can dynamically query and modify the appearance of the 2-D interface. For example, the London Traveler application (Steed, Frécon, Avatere Nõu, Pemberton, & Smith, 1999) exploits this feature by adding an application-specific menu to the regular interface of the Dive browser.

Finally, a MIME (Multimedia Internet Mail Extensions) module is provided to better integrate with external resources. It automatically interprets external URLs. For example, an audio stream will be forwarded onto the audio module where it will be mixed into the final soundscape.

### 5 The COVEN extensions to Dive

This section describes the extensions and enhancements made to the existing Dive system during the lifetime of the COVEN project.

#### 5.1 The DiveBone

To allow the number of participants and applications to grow, many CVE platforms incorporate scalable networking techniques. The recurrent theme of these solutions is that they utilize the general principles of occlusion and attenuation with (virtual) distance to reduce the number of participants that truly share a region of the environment. This is demonstrated by the use of regions in various platforms such as MASSIVE-2 (Greenhalgh, 1996; Benford & Greenhalgh, 1997), Spline (Waters et al., 1997), NPSNet (Macedonia, Zyda, Pratt, Brutzman, & Barham, 1995) and the Dive toolkit (Frécon & Stenius, 1998).

To minimize traffic within each region, these platforms use one or several IP multicast groups for data, audio, and video distribution. Through experimentation, these platforms have shown that IP multicast is one of the keys to the success of large-scale and media-rich CVEs. Indeed, used in conjunction with techniques such as partial and active replication of the shared environment, IP multicasting allows a system to minimize the amount of information that needs to be communicated during interactive simulations, and ensures that network packets are only duplicated when absolutely necessary.

For long-distance connections, most multicast-based distributed VR platforms rely on the existence of the multicast backbone: the MBone (Erikson, 1994). Experiments and network trials at the beginning of the COVEN project highlighted some weaknesses of the MBone, which hinder the establishment and testing of large-scale CVEs. Despite its wide acceptance within the research community, the MBone spreads slowly be-
tween subnetworks, especially at the corporate level. Also, multicast is not yet supported fully on all carriers (such as ATM and ISDN) and some older computer systems. Furthermore, any modification of the network parameters and configuration require the intervention of a system administrator. This is in strong opposition to the trial-and-error strategy that still characterizes most CVE experiments.

To tackle these obstacles, COVEN has developed an application architecture that acts as an extension to the regular MBone and is built as a standalone part of the DIVE toolkit. The DIVEBone (Frécon, Greenhalgh, & Stenius, 1999) is an application-level backbone that can interconnect sub-islands with multicast connectivity. The DIVEBone offers a framework for future facilities, not available at the MBone level, such as the ability to perform compression at the application level. (This technique is outlined in subsection 5.3.1.) Furthermore, the DIVEBone allows for visual analysis of the connection architecture and network traffic, and for remote maintenance operations. The DIVEBone capabilities have been demonstrated and successfully used in a series of large-scale pan-European tests over the Internet, as well as in various experiments involving IP over ISDN and ATM (Frécon et al., 1999; Greenhalgh, Lloyd, Bullock, Frécon, & Steed, 2001). These trials have proven the qualitative and quantitative adequacy of the DIVEBone in heterogeneous settings in which multicast connectivity is limited or nonexistent.

5.2 World-Structuring Techniques

DIVE uses hints within the entity hierarchy to control the replication mechanisms and to indicate regions of the virtual environment that can be allocated to different multicast groups. This is achieved through the association of multicast groups with branches of the hierarchy. Consequently, regions generally have a geometrical position that is determined by the entity that carries the multicast information. However, unlike most other platforms, DIVE does not enforce any extent to regions and leaves extent to be determined by application semantics. Within DIVE, multicast communication is bound to the world hierarchy (see figure 2) in two ways:

- The top-most entity of the hierarchy is associated with three multicast groups that will be used by default for all data, audio, and video communication within the environment.
- Any entity of the hierarchy can be associated with a new multicast group, called a lightweight group. When a modification message concerning an entity is to be sent, the hierarchy is traversed upwards until a multicast group is found. If no lightweight groups are found, the default groups are used. If a null multicast group is found, this part of the hierarchy will be local and modification messages will not be distributed.

Lightweight groups can be associated independently of the structure of the entity tree, enabling several branches to be associated with the same group. Joining and leaving the groups is left to the application, which can exploit group membership functionality through any of the programming languages supported by the platform. Lightweight groups are sufficiently flexible to

---

**Figure 2.** Lightweight groups \( (G_1 \text{ and } G_2) \) and the default group \( (G_w) \). Regular Dive objects are represented as circles, while hierarchies are simplified as triangles, and entities carrying group information are drawn as trapezoids. Note that \( G_1 \) spans the hierarchical structure.
serve as a basis for experiments with different platform techniques such as aura intersection (Hagasd, Lea, & Stenius, 1997) or subjective views (Smith, 1996).

The only operations possible on lightweight groups are join (start receiving updates about entities within that group) and leave. It is not possible to initialize the content of a branch in this way. However, in many applications, it is necessary to be able to initialize the branch under an entity with adequate content, such as from a file or a URL. Therefore, DIVE incorporates a world-structuring technique termed holders. Holders encapsulate part of the database hierarchy under a multicast address and associate it with an initial file description. Holders require explicit connection to retrieve the contents of the branch that they encapsulate. They can be used in two ways:

- **Local holders** introduce isolation. All participating processes will independently load the contents of the holder. Modifications to the subhierarchy will not generate network messages and will not be propagated to other connected peers. Local holders are especially useful for:
  - Describing parts of environments that are guaranteed to never change, such as the walls of a large building. They reduce the size of the dynamic part of the environment that has to be exchanged when new participants enter an environment and allow for quicker initialization of the database through the reading of files.
  - Distributed script execution—the execution of parallel, independent, and synchronized scripts, triggered by high-level events. High-level events are distributed to all connected sites by being sent to the holders themselves, whereas low-level events are kept beneath holders in the database hierarchy and are thus not distributed. This technique can be used to drastically reduce network traffic. This technique is exploited by a semantic messaging editor (Frécon & Smith, 1999) that enables an external command interface to be constructed for complex, but encapsulated (that is, within a holder) objects within the DIVE system. As none of the events generated by the objects in the holder is distributed, these semantically encoded objects are accessed through a discrete, application-defined, programming interface.
- **Distributed holders** offer a framework for world nesting, such as rooms within buildings. Network traffic is restricted only to the processes that are members, typically a subset of the total number of connected sites.

Holders are essential to the achievement of scalable worlds. Indeed, they allow the platform to address only the parts of the database that are necessary to the user. A consequence of this is reduced network traffic. By desynchronizing part of the hierarchy from the standard DIVE network mechanisms, holders also provide a framework for experimenting with new, mixed-distribution techniques, such as using a client-server approach only for certain parts of the hierarchy that require distributed locking.

### 5.3 Data Distribution Tuning

Networking scalability is one of the key problems addressed by COYFN. The structuring techniques described above restrict the number of interested parties to a subset of the processes involved in the environment. Together with the use of SRM, they contribute to dramatically reduce network traffic.

In this section, we describe two approaches that focus on the underlying network mechanisms of DIVE. Our goal here is to be able to decrease bandwidth use within a region of the environment. Both approaches can be seen as a semantically rich compression of the different packets that are sent within networked virtual environments. They focus on reducing the size of geometrical transformation messages. This approach is grounded in the network trials, which found that most of the events in a virtual environment are avatar movement updates (Greenhalgh, 1997). We end this section with a brief discussion of the DIVE network behavior as observed during one such series of trials.
5.3.1 Application-Driven Network Message Aggregation. This technique allows the content of several network messages to be encoded into a single (bigger) message. This feature reduces the number of messages necessary to describe database modifications. Consider the case of animating a human body. Without this technique, each part of the body has to be addressed separately at each frame of the animation. The required transformations of the limbs will map onto a separate network message, as depicted in figure 3. These messages typically contain as much data as protocol-dependent header information.

To optimize the available bandwidth it is possible to exploit an application-level aggregation mechanism. Here, the different transformations applied to the different limbs are considered to be a single unary operation and the individual messages are combined into a single network message (figure 4).

Other approaches to this problem exist, such as incorporating the animation descriptions into the platform, where each frame can be deterministically calculated at any time. However, our technique is not restricted to predefined deterministic animations. For example, when incorporating crowds of virtual humans to the platform, the crowd-control mechanism was attached to only one of the processes, which updates the other peers (Pandze et al., 2001). As the behavior of the crowd is dependent on the (dynamic) content of the environment, it is difficult to predict movement and frames in advance. The technique described above allowed us to use the available bandwidth more judiciously.

Using the DiveBone, message aggregation can be implemented more generically. Because many applications will run simultaneously on a given network, the Dive routing daemons face a continuous and dense flow of messages that they can aggregate into meta-messages. These can even be compressed using standard compression methods, such as using a timer that regularly ag-

---

1. Currently, Dive uses regular data compression when a consequence amount of data has to be transferred among peers. The implementation has proven that increased CPU load for compressing data is of little significance compared to the reduction in bandwidth use and transmission time.
gregates all received and not-yet-sent messages into a single meta-message. Redundant messages such as a series of absolute transformations for a given object can also be confined to an aggregation period.\(^2\) Because the DiveBone is aimed at connecting distant networks on which the roundtrip time can be up to several hundreds of milliseconds, this aggregation technique would not have significant implications on packet delivery time.

5.3.2 Application-Driven Dead Reckoning. Dive supports a series of user-defined functions that may be applied to any entity. The functions are invoked each time the position and orientation of the object is required by the platform, letting the functions compute new values at the time they are called. This supports a means of implementing dead reckoning at the application level, by providing each of the connected processes with the same functions (based on the same algorithm). This is exploited in the Dive platform, which implements a velocity model for dynamic objects. Each peer then animates the object based on this information. This mechanism can be extended to implement gravity and simple obstacle avoidance, and, with Dive/Tcl (see subsection 5.11), arbitrary application-driven dead reckoning can be implemented.

5.3.3 Network Traffic Analysis. During the COVEN project, a number of network trials were performed to assess the network behavior of various CVE applications. The trials were designed in such a way that they focused on shared activities, involving many users and often requiring the users to move around and communicate with each other to perform certain tasks.

One series of trials took the form of a networked game, similar to Clue (or Cluedo) (COVEN, 1999a; Greenhalgh et al., 2001). Here the users, grouped into a number of teams, were able to freely explore a virtual mansion. The participants were told to look for clues and question other team members for information to solve the murder mystery. These trials were undertaken with sixteen participants, working in teams of four. When the network data for each such trial was analyzed and combined, we found that the users were moving around 26.3% of the time on average, and speaking 8.1% of the time. The collected data allowed us to construct a simple network traffic model of a Dive multiuser session. According to this model, the expected bandwidth requirement for such a session with \(N\) users is

\[
14483 \times N + 2400 \text{ bits/sec. average}
\]

\[
128900 \times N + 2400 \text{ bits/sec. maximum}
\]

Of the average bandwidth per user (14483 bits/sec.), 9.2% of the traffic comes from a single TCP state transfer per session, and the rest from UDP multicast traffic. This multicast traffic consists of 56.6% audio data, 30.4% data updates (such as the introduction or removal of objects, and movements) and 13% from reliability and liveliness data (that is, data to ensure consistency), based on the SRM approach. The rest of the average traffic (2400 bits/sec.) comes from world multicast data not concerning any particular user (for example, world time data). From the equations, we can observe that a 10 Mbit/sec. Ethernet corresponds to 760 concurrent users based on average bandwidth, and 77 users based on maximum average.

5.4 Persistency

Many applications require access to a persistent virtual environment. That is, the environment can be augmented and edited by participants, but it will retain its state after all the participants have left and will restore that state when they return. COVEN has enhanced Dive with support for persistent environments. Persistency is managed using a neutral process (called Persistent), which keeps its own active copy of the whole world database. The process periodically archives its state to disk to minimize the impact of hardware, network, or software failures.

To strengthen persistency, several Persistent processes can be run in parallel on different hosts or different networks. This arrangement uses Dive’s active repli-

---

\(^2\) A variation of this technique is used in many server-based 3-D chat platforms, in which location updates are filtered to scale down to the low bandwidth of modem lines.
cation mechanisms to keep Persistent processes from reading their initial state from their associated file space on restart. Rather, they receive their initial state from any other Persistent process that may still be running.

The existence and activity of the persistency manager is controlled by another process called AutoPersistent. The AutoPersistent process automatically starts a Persistent process for each environment that is created, if such a process is not already running on the host that the AutoPersistent process runs on. Automatic starting can be restricted to a set of worlds, enabling institutions to offer persistency for the worlds that they wish to make public. Additionally, the AutoPersistent process supervises its associated Persistent processes and re-starts them on failure. (See figure 6.)

5.5 Logging and Replay

A central activity during the network trials (Greenhalgh et al., 2001) was analysis of the data traffic to produce a model that predicts the network bandwidth necessary to support a given number of users. For this purpose, it was essential to match the network packets that were actually sent during a session against what happened in the system. Consequently, DIVE was extended to include a logging facility that can actively record all events occurring within a DIVE session and archive them.

Once the logging system was in place, a session recorder and player was developed. This tool records and replays the contents of the recorded packets, including audio packets. Consequently, the application makes it possible to replay and review past sessions.

5.6 Audio Communication and Spatialization

DIVE supports real-time spatialized audio communication between participants and from objects to participants. Results from the COVEN trials have shown that audio is essential to collaboration and that spatialization improves collaboration and the sense of presence (COVEN, 1999b).

The audio sources may either be a microphone or a file containing recorded audio streams. Audio is introduced into the shared environment by one of the connected peers and is distributed to all other interested parties. For instance, if a microphone is connected to the workstation, the input will be distributed by the local process and played back by all receiving processes with appropriate audio hardware. Because many audio streams can be active simultaneously, a given process can be both a sender and a receiver at the same time. Receivers need additionally to mix all incoming streams to play them on the local audio hardware. Audio integration is described thoroughly by Adler (1996).

The distribution of audio streams within DIVE environments uses unreliable multicast. Currently, each world has a dedicated audio group that is used exclusively for audio communication and any number of additional lightweight audio groups. Any process that is interested in audio can subscribe to these groups and will thus receive all distributed audio packets. Different types of sound sources with different emission patterns can be defined. Before the sound block is spatialized, it is processed according to the sound source type, the position of the source, and the position of the listener. Examples of source types are omnidirectional and mega-
phone. Finally, audio packets are spatialized using the distance to the source and the interaural and filtering coefficients. All sources are mixed together, and packets make their way to the hardware using a double-buffering scheme.

5.7 Video Integration

Divi supports video by enabling live video images to be associated with entities of the environment and displayed as part of the rendering process. (See figure 7.) The technique used is to convert video images into textures that may be mapped onto 3-D objects. The video source may either be a camera, which allows for VR-based video conferencing, or a file that contains recorded video clips. The video is introduced into the system by one of the connected peers (the sender) and is distributed to all other interested parties (the receivers).

The process for managing video streams is similar to that for audio. Each image belongs to a specific stream and has a name chosen by the sending peer. The name is used as an identifier, which makes it possible to separate images from different streams as they pass through the different stages of coding and distribution on their way from the sender to the receivers.

The rendering portion of the video-streaming mechanism is intentionally video unaware. This enables the video-streaming mechanism to be used by any application or system module to distribute and display images. This has been used to embed X servers into Divi worlds. Previous work on this includes Dykstra (1994), in which shared memory was used to incorporate an X server into a virtual environment. With Divi, similar techniques combined with the dynamic video-streaming mechanisms can be used to incorporate X displays into a multiuser environment. In figure 8, we see the combination of an embedded X server with the virtual conferencing tools presented in (Frécon & Avatare Nou, 1998).

Because pictures are notably larger than encoded audio packets, the video module is able to handle both the distribution of whole pictures and parts of pictures, in a manner that is similar to many video compression techniques, such as MPEG.

5.8 Scalable Rendering Techniques

Presenting environments that are geometrically detailed and extended, and that are visually appealing, is a key issue in the design of CVE applications. However, it is important to maintain an acceptable frame rate in visually complex environments. Ideally, this frame rate
has to be close to the video frame rate and it should be kept fairly constant at all times. In terms of rendering, this means that the geometry passed to the graphics hardware should never exceed what it is capable of rendering within the desired frame time. Using appropriate geometry primitives and material definitions is one approach to this problem, because they relieve some of the rendering load. Under COVEN, DIVE has been extended to support primitives such as level of detail, billboards, switches, materials, and animated textures.

Furthermore, COVEN has experimented with more advanced rendering techniques that aim at providing a constant frame rate. The standard DIVE renderer follows a very simple scheme. (See figure 9.) The hierarchy is traversed in depth-first order using spherical bounding volume culling. When an entity containing a level of detail (LOD) is encountered, a simple range decision is made and then the geometry is passed to OpenGL.

However, in extended, massive, open-air environments such as the London model (Steed et al., 1999), many hundreds of thousands of polygons may be seen from any viewpoint. In such environments, the standard renderer can take several seconds to render a frame. Furthermore, the use of common LOD ranges is insufficient to provide a constant frame rate. Although LOD ranges do provide the renderer the ability to select only appropriate visual detail, the ranges cannot be created so that, from an arbitrary viewpoint on an arbitrary machine, a target rendering time is met. Typically, a minimum platform and minimum frame rate would be decided and the LOD ranges tuned to that platform. However, in some cases, with a faster machine, it is desirable to be able to render more of the model, rather than rendering at a faster frame rate.

We have experimented with renderers that aim to give a constant frame-rate experience, no matter the scene complexity, so that visual continuity can be maintained.

In the simplest version, we optimize the software culling distance so that a frame-rate target is met. This is done by computing the distance to objects during scene traversal. When the rendering target is overrun, the culling distance is rapidly brought nearer to bring the frame rate under control. When the target is met, the culling distance is slowly moved outwards. The change in depth is damped so that the frame time does not oscillate and objects do not pop on and off. In a more advanced version, we incrementally render the scene if the viewer is not moving, as shown in figure 10.

In order to perform this incremental rendering, it is necessary to render in three stages per frame. First, the background is progressively painted in with all static objects, and the scene image is stored. Secondly, depth buffer writing is disabled, all moving objects are drawn, and the resulting image is shown to the user. Finally, the image without the moving objects is restored to the off-screen buffer in preparation for another frame. The process is illustrated by figure 11. In this manner, it is possible for moving objects including other avatars to be correctly depth-buffered against the rest of the scene.

For the densely populated areas, a few crude visibility techniques can be integrated. Although we did not integrate computed visibility solutions such as those developed by Airey, Rohlf, and Brooks (1990), we have integrated tools into DIVE itself that allow objects to be grouped together by hand and identified as cells. These cells can be turned on or off depending upon the user's location using the subjective-view capability within DIVE. (See subsection 5.10.) Although the resulting visibility relationships are much more conservative than an analytic solution, they are quick and easy for the scene author to describe, and are more generally applicable to unstructured geometric models.

5.9 World-Modeling Techniques

Supporting several three-dimensional file formats is one of the keys to platform openness. In this section, we concentrate on how the result of 3-D file parsing is introduced into the environments, the different supported file formats, and a specific modeling tool that is integrated into the COVEN platform.
5.9.2 Parsing Techniques. File parsing in DIVE is based on the following philosophy: parsing is performed at one peer and its results (such as new objects and new behaviors) will be introduced to the shared database. The introduction is performed first locally, and then duplicated to all other interested peers using the internal replication mechanisms. The binary data corresponding to the new entities is streamed in a platform-independent format, possibly compressed, and sent as multicast or unicast messages.

The advantage of this approach is twofold:

- Parsing affects only the application that introduces new geometry and behavior to the environment. The parsing operation often consumes considerable time and processor resources. Thus, this approach puts the load on the application that performs the introduction, not the receiving applications.
- Only one peer needs to be able to access the files or URLs that will add one or more branches to the hierarchical database, avoiding network access problems. This also provides the invoking peer with continued control over the uploaded objects.

5.9.3 Geometry Manipulation—Enhanced AC3D. AC3D (AC3D, 1999) is a 3-D object (or world) modeling and construction application. It allows a user to manipulate and construct arbitrary 3-D structures and assign application-specific behaviors to individual entities. The user is presented with a control panel and four visualization windows: three ortho-
Figure 12. The AC3D modeler

Figure 13. Two subjective views of the same shared space, one unchanged (left) and another with a highlighted path and transparent buildings.

...graphic views and a 3-D rendered view. Users interact with 3-D models by manipulating the orthographic views that update the view in the 3-D window (See figure 12.)

AC3D integrates many of the basic features that are usually found in professional modeling packages. Within COVEN, the editor has been integrated with the Dive system in two different ways:

- AC3D allows application behaviors to be added to 3-D objects that are being modeled.
- The standard Dive visualizer and AC3D are able to exchange models on the fly. The current integration makes it possible to select an object hierarchy in the Dive viewer, send it for editing to AC3D, and put it back into the shared environment once editing is completed. This enables users to dynamically update the geometry of a runtime scene. Combined with the facility above, this allows for an integrated and dynamic behavioral and geometrical edition of the environment.

5.10 Cooperative Visualization Mechanisms

In traditional virtual environments, each user is presented with the same virtual world, albeit from a different viewpoint. Users cannot tailor their representa-

tion of the virtual scene or the degree to which they are aware of other users' activities. This is somewhat analogous to early 2-D shared multiuser interfaces in which users were each presented with the same application views. However, research in 2-D interfaces has shown a strong trend to support individual tailoring of the shared views to reflect user demands. These lessons from 2-D interfaces have been integrated into Dive, allowing individual users to have more control over their view of the virtual environment. This provides users with subjective views (Smith, 1996) of the virtual environment and provides them with the ability to tailor the environment to suit their working needs. It also enables applications to address users on a more individual basis. For example, consider a populated virtual town that augments the world with large directional arrows depicting a path between two user-selected locations. Without individual views, it would be difficult to differentiate individuals' selected paths. However, with subjective views, not only can the chosen paths be viewed by only selected users, but the starting and end points (in this case buildings) of the path can be highlighted by, for example, making the other buildings in the scene slightly transparent. (See figure 13.)

Populated virtual environments can be considered as three-dimensional spaces filled with representations of users. The need for a common 3-D frame of reference is required to allow users to feel they are sharing a common world. Therefore, the model devised to support subjective views embraces the need to maintain a common spatial frame of reference. The core feature of the approach is an access matrix, which defines the representation of individual objects for individual users. The ma-
<table>
<thead>
<tr>
<th>Notation for 1st Class Honors</th>
<th>Representations</th>
</tr>
</thead>
<tbody>
<tr>
<td>View Normally</td>
<td></td>
</tr>
<tr>
<td>Remove from View</td>
<td></td>
</tr>
<tr>
<td>Transparent</td>
<td></td>
</tr>
<tr>
<td>Wireframe</td>
<td></td>
</tr>
<tr>
<td>Dim</td>
<td></td>
</tr>
<tr>
<td>Bright</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 14.** A view matrix of an object, in which the view chosen is a transparent sphere.

Matrix defines an object’s view in terms of two independent factors:

- Differing geometric definition (such as different textual definitions), and
- Highlighting and de-emphasizing abilities (such as making an object appear as wireframe)

These two factors, termed appearance and modifier, may be arranged orthogonally into a view matrix that defines an object’s range of possible representations. (See figure 14.)

For example, consider an object that represents a first-class honors degree. This may be represented by either a large “1”, a sphere, or a cap. (See figure 14.) The modifiers apply orthogonally across each of the representations, allowing one user to view this object as a normal “1” while another sees a transparent sphere.

Although tailored views of the virtual environment enhance the usability of the system, different presentations of shared user interfaces, whether 2-D or 3-D, can cause problems in collaboration due to lack of “common knowledge” (Fussell & Krauss, 1992). This problem is also highlighted by Gutwin and Greenberg (1996), but is termed workspace awareness and defined as “the up-to-the-minute knowledge a person holds about another’s interaction with the workspace.” Therefore, there is an operational tradeoff between the benefits of interface tailoring and the amount of common knowledge required to facilitate collaboration. Generally, this fine-tuning of user interface coupling between users is the responsibility of the users themselves, but this activity is problematic as it is difficult for individual users to know the customized details of other users’ interfaces. Hence, the process of user interface recoupling (Smith & O’Brien, 1998) must be an automatic process without placing additional responsibility on the user. Rather, it should operate seamlessly, and reconfigure the different shared user interfaces on the users’ behalf. Recoupled user interfaces provide users with inter-subjective views of the shared world.

The core of our approach is the explicit identification of the awareness that users have of each other. The knowledge of this awareness is combined with the subjective information specified for each user to generate an awareness-sensitive subjective view. The knowledge of a user’s subjective view is combined within an awareness map that describes the awareness other users have of each object to create a new subjective view. This new subjective view is a second-generation subjective view derived from a combination of the users’ initial subjective configuration and the awareness map.

The recoupled view is manifested through increasing the visibility of objects in a given user’s view, if they are “more visible” to another, closely aware user. The issue of “more visible” is discussed in detail by Smith and O’Brien (1998); however, it can be implemented simply by making invisible objects appear increasingly solid (less transparent) as the users’ awareness increases. For example, consider an application of a CVE in which a number of users collaborate to define the arrangement of electronic and hydraulic components. A typical scene is depicted in figure 15a. To improve the usability of the tool, each engineer defines a subjective view, which provides him with only the information that he needs (for example, figure 15b). However, when the two engineers are working in the same space, they may be unaware of any changes made by one, which may affect the work of the other. To prevent this, they are provided with inter-subjective views of the shared space (figure 15c). Their views still concentrate on the aspects that are important to each user, but they provide them with sufficient awareness of the other’s work to reduce referential and confliction problems.
5.1.1 High-Level Behaviors

The ability to associate behavioral scripts with objects in the virtual environment enables objects to act autonomously, react to external stimuli, and give feedback to user actions in a number of scenarios. These considerations have led to the implementation of a behavior description language, based on the Tool Command Language (Tcl), as part of the DIVE platform. This language is named DIVE/Tcl and is described in detail by Frécon and Smith (1999).

DIVE is well suited for writing prototypes and other applications that are necessary to study various aspects of CVE technology. More generally, CVE applications are still not well understood, and much of their development relies on a trial-and-error approach. Having to write monolithic applications in a low-level language is in strong opposition to both the main purpose of the DIVE platform and the maturity of CVE technology. Therefore, it is necessary to extend the platform by offering new ways of writing applications.

In order to describe simple behaviors and animations, prior versions of DIVE have relied on being able to associate a state machine to any object of the virtual environment. This work is partially described by Carlsson and Hagsand (1993). Obviously, the state machine had limited possibilities and was aimed at writing simplistic behaviors and animations. A more powerful language is necessary to describe more complex behaviors. A beneficial advantage to a scripting language is control over a more common 2-D user-interface paradigm. Many applications such as word processors or spreadsheets are currently much simpler to use in a standard 2-D environment. Additionally, text-based media does not translate very well to VR. Hence, the ability to integrate 2-D windowing extensions in a VR platform can serve as an intermediary solution to this problem.

5.11.1 DIVE/Tcl Highlights. DIVE allows Tcl interpreters to be associated to any object in the DIVE database. These interpreters will register for system events and react accordingly. Examples of system events are interaction device input, object collisions, and timers. Scripts are able to use all of the programming constructs offered by Tcl, as well as an additional set of DIVER-specific commands, such as gathering information on objects and modifying their state.

Particular attention is paid to allow the programming and execution of truly distributed applications, that is,
applications for which code execution is allowed to migrate from one process to the other on demand and according to a controllable scheme. This is made possible by the fact that Dive is based on the concept of a shared database. Thus, all connected processes are made aware of modifications independently of which process initiated the modification. For events that are spontaneously generated by users, Dive executes the resultant code at the process through which users interact. Consequently, this approach ensures the results of the interaction to be visible first at the interaction process. This reduces any delay in user interface updates.

Because the implementation requires application code to be executed at different processes, it is essential to store application state within the scene database, so that application logic can behave similarly at any process. This is made possible through a specific Dive feature: the platform allows applications to associate their own data to objects of the database through a concept termed properties. Properties are triplets composed of a name, a type, and the actual data. They are distributed and replicated along with their associated objects. Consequently, it is possible for one process to modify the value of a property and for another connected process to read the modified content at a later time. Dive offers internal support for the most-common types such as integers, floating-point values and strings, together with a few internal Dive types. Additionally, any application process can register a new type, together with the functions that are necessary to store and retrieve the property to and from a network packet in a platform-independent way. This allows properties of unknown types to transit through a process, and reach another process that will recognize the type in question. The scripting integration provides two additional key aspects:

- It offers ways for applications to dialog with users using a two-dimensional user interface. This is achieved by enhancing the interpreters associated with the objects that represent each user within the shared database with Tk, the Tcl windowing companion.
- It enables existing, monolithic applications to interface with the Dive system. This is achieved by allowing applications to have a logical representation within the database hierarchy, which is accessed through additionally defined Tcl commands.

A progressive approach was undertaken for the implementation because we wanted to achieve an implementation that had sufficient commands for writing small to medium-sized applications without requiring the use of external plug-in or applications written in C or another low-level language. The resulting standard interface is described in (Dive, 1999a).

5.11.2 External Programming Interface. The Dive client interface, known as DCI (Dive, 1999b), offers an alternative way to interface Dive with external programming environments such as applications written in other languages, or complex programming systems in need of a clean separation. DCI enables external applications to connect to a running Dive application and execute commands within the environment on their behalf. Connected applications can send scripting commands to the Dive peer, receive replies, and be notified when given events occur. As any scripting command can be sent, the networked database shared by all peers can be modified and queried as needed. The client side of the DCI has been implemented in several languages, including C, TCL, Oz, Prolog, and Java. The Java interface, known as JDI, wraps the Dive class hierarchy into a Java class hierarchy. Each DCI command is wrapped in a method associated to the Java class onto which the command can be called.

6 Conclusions

This paper has presented the numerous extensions to the Dive system that have been developed during the COVEN project. The nature of these extensions ranges from low-level network performance gains and advanced graphical rendering techniques to user interface and usability enhancements. The wide range of extensions described in this document depicts the range of technologies that underpin complex CVEs such as Dive.

The additional functionality described in this document details the results of a number of years of work in
the development of a mature CVE. An important consideration is that DIVE supports these extensions orthogonally, which enables an application to exploit any of these features as required in a single system. Furthermore, due, in part, to the extensive usability and network trials across Europe, applications using DIVE are inherently multiuser and scalable.

DIVE is a flexible system. The range of application programming interfaces enables developers to integrate their application semantics into DIVE in a way that best suits their requirements. Additionally, the interpreted scripting support facilitates rapid prototyping of multiuser VR applications.

DIVE is currently being used as a research tool within a number of other national as well as European projects (such as eScape and KidStory), a fact that supports the validity and usefulness of many of the COVEN extensions. Several of these projects are introducing additional functionality and refinements, which ensures that the system will continue to prosper even though the COVEN project has come to an end. There are also plans to exploit the COVEN results by developing a commercial version of the system, based on the research platform but with an increased focus on stability and ease of use.

In summary, this paper has presented the technological innovations that have been consistently integrated into a CVE called DIVE. The range in functionality of these extensions, combined with the maturity of the system as a whole, provides application developers and users with a powerful multiuser VR system.

References


———. (1999a). COVEN Project Deliverable 3.6: Network assessment of the on-line applications. (Please contact the authors for a copy.)


