The Virtual Object System: An Open Platform for Internet Multiuser Virtual Reality

Peter Amstutz and T. Reed Hedges
Department of Computer Science
University of Massachusetts, Amherst
Amherst, Massachusetts 01003
amstutz@cs.umass.edu, reed@zerohour.net

Abstract—**Interactive Virtual Reality (VR) on the Internet has been a dream long promised, but despite a great deal of work few technologies today fulfill this promise and none are in wide use. Previous efforts have been unavailable to the public, floundered as academic prototypes, or focused too much on the graphical aspects of the problem — ignoring the true challenge of designing a robust infrastructure on which virtual environments may be built. This paper considers the factors that must be taken into account when designing a multiuser virtual reality system, and then presents the Virtual Object System (VOS), an open infrastructure designed to support networks of distributed, collaboratively built virtual worlds. VOS is a message-passing core concerned with the management of "vobjects". Vobjects are network-accessible objects organized with incoming and outgoing hyperlinks to other vobjects which form dynamic network-spanning structures. individual vobjects may be extended on the fly to support new object types, allowing a single vobject to support several different interfaces. This infrastructure has been used to implement the Abstract 3D Layer (A3DL), a set of object types for describing 3D scenes. A rendering application realizes these worlds on the screen, allowing a user to join, move about in and interact with a world and its contents (including other users). VOS has been successfully used for several nontrivial projects including an immersive interface for monitoring and operating mobile robots at the University of Massachusetts, Amherst.**

I. Introduction

Few technologies have had so much exposure in the popular media as the promise of fully interactive multiuser Virtual Reality (VR) on the Internet. Yet — despite the pervasive portrayal of three dimensional interfaces in books, movies, and television — such technology has failed to materialize on the average user's desktop with the exception of online games. Why is this, and what might be an appropriate design for true shared online spaces? This paper will first discuss some of the technologies that have been developed to support online virtual environments but have thus far failed to take hold, and examine some of the challenges facing the implementation of online spaces. It will then present the Virtual Object System (VOS)[1], our contribution as an open platform for development of multiuser virtual environments. We discuss the design of VOS, how VOS is used for implementing VR systems, and our uses of VOS to support other research.

Fig. 1. A conversation in cyberspace

The use of VOS for VR has many interesting applications. Perhaps most promising and exciting is the expansion of online social spaces to a richer medium of communication, allowing user movement, gestures, and expressions to improve the interaction experience. Another fascinating potential is in interactive online artistic expression. More practically, teleoperation and telepresence, in which users exert remote control over robotic systems, would benefit greatly from a common platform for collaborative VR. Directly tying incoming data feeds to update three dimensional models in realtime would have many interesting uses in shared visualization of complex, changing data. Ultimately it is impossible to predict all the potential applications of a common multiuser platform, but one

[1] *Not* to be confused with the "Virtual Operating System", an entirely unrelated product offered by Stratus Technologies, http://www.stratus.com/products/VOS

envisions a scenario much like the World Wide Web where the creativity of the wider community results in a vast number of applications never foreseen at the systems inception. The key to these applications is in the creation of a living space, where any aspect is changeable and such changes propagate immediately to all the users of the space.

Although there are many different ideas about its meaning, in this paper the following definition for "Virtual Reality" will be used: Virtual Reality is a persistent three-dimensional space fundamentally designed to use computer networks for communications among many users; users are represented by articulated three-dimensional models called *avatars* and interact with each other in real time for social, entertainment, educational and economic purposes; users are able to contribute to the world by creating, modifying, deleting or otherwise interacting with any feature of the world for which they have permission to do so.

II. Previous Work

There has been a great deal of work in building shared virtual reality spaces. Unfortunately, most attempts at building general-purpose VR for the Internet have been either laboratory prototypes or commercial endeavors. Prototypes often lack ongoing development and are eventually discarded due to limited functionality or outdated technology. Commercial systems are closed to outside development and are often driven more by business considerations than seeking to create the best possible technology.

The most well known effort is Virtual Reality Markup Language (VRML) [1] by the Web3D Consortium. The VRML standard consists primarily of a file format for 3D geometry and specifications for client-side scripting and events. While it has been successful and widely adopted, it solves only one piece of the VR puzzle. Notably lacking is a standard network protocol to tie the users of a world together to enable collaboration. While several proprietary multiuser systems based on VRML exist, notably Contact [2], DeepMatrix [3] and VNet [4], these systems are limited and incompatible. Unfortunately the relevant Web3D working groups for Virtual Reality Transport Protocol [5] and Living Worlds [6] have had little activity since 1999.

Online games are easily the most common form of online virtual reality experienced today. Games such as Id Software's Quake [7] and Sony's Everquest [8] allow for realtime online interaction in three dimensions, but do not enable the user to contribute to the world in a unique or substantive way. Some MUD (Multi-User Dungeon) systems [9] give great freedom to the users to interact with and add to the world, but these systems are limited to text-based communication. Most online games rely on a central server.

The United States Military has developed the Distributed Interactive Simulation (DIS) [10] system for simulating military operations. DIS is highly scalable and has seen extensive use in soldier training and simulations with hundreds of individual actors. Unfortunately most components are not not available to the public. The system is also quite specialized and requires a high performance network.

Deva [11] is an architecture for supporting a distributed virtual environment server so that a single world may scale to support many users. It provides object management and migration enabling a world to be distributed among various servers. It does not provide services for structurally or categorically organizing objects, and is based largely on the notion of centralized world servers.

Distributed Interactive Virtual Environment (DIVE) [12] is a multiuser system that has been in development since 1991. DIVE replicates objects and their behaviors across every node in the system. When the user interacts with an object, the changes are sent to every other user of the system using a multicast group. Although the aggressive decentralization frees the system from any single point of failure, it also prevents any client from exercising absolute control over a particular entity. Additionally, because the system relies on each client making a decision about entity behavior, there is significant potential for abuse as well. DIVE is also not realistic to use in many situations due to its reliance on multicast, which has not yet been widely adopted on the Internet at large.

The Columbia Object-oriented Testbed for Exploratory Research in Interactive Environments (COTERIE) [13] is a system for multiuser virtual environments based on a shared memory model using distributed, replicated data objects and protocols for updating those objects. The system is written in Modula-3 and Obliq, and supports hierarchical object directories. This system may be the most similar to VOS and although very interesting, it has not been made available to the general public.

III. Design Issues

There are many challenges to building a system suitable for pervasive online VR. These issues include but are not limited to: performance of 3D rendering, network latency and bandwidth, resource discovery and organization, single points of failure, security, privacy, trust, ease of use (by users, content creators and programmers), extensibility, and the insufficiency of existing protocols and software in addressing most of these issues.

It is our opinion that the first challenge, rendering performance, has become far less of a problem than it once was. With the proliferation of hardware-based 3D acceleration in standard consumer PCs, beautiful high resolution 3D graphics can be rendered on nearly any system bought within the last three years.

Network latency and bandwidth are by far two of the most difficult issues with multiuser VR. There are two fairly effective ways of handling latency: the first is to "push" updates where interested parties are notified immediately when something happens, rather than having to poll for changes; the second is to move more processing to the client side and make the network protocol describe action on a higher level. These measures also help reduce bandwidth usage.

Resource discovery is the problem of knowing what things (objects, avatars, sounds, etc.) are in a world and how they relate to one another. For example, when you load a web page that contains a hyperlink to a second web page you have not seen before, you have "discovered" that second web page. Furthermore, you understand that it is somehow relevant to the web page you are currently viewing. Without such hyperlinks the World Wide Web would be thoroughly useless. Similarly, the elements of a VR space must be able to interlink with each other such that the whole is more meaningful than the sum of its parts.

For a VR system to be robust, it must not contain a single point of failure. The idea of a single massive server farm running *the* virtual world is not only bad design, but quite antithetical to the fundamental principles of the Internet, that are based on decentralized processing. The World Wide Web is again instructive here: if a single web server goes down, the effects are isolated to the immediate content hosted on that server, and the rest of the Internet continues on its way.

There is also an important philosophical point: the technical values of the system will influence the social values of the system. Do we want a dictatorial central server, or a democratic distributed system? The web democratically allows anyone with a bit of connectivity to host his or her own web site, and VR should be the same way.

Security, privacy and trust are absolutely crucial elements of VR, yet are often overlooked. To prevent people from snooping in on a conversation, it must be possible to encrypt communications. This is especially necessary for transactions such as e-commerce. Encryption gives users a sense of privacy as personal information can be protected. By using public key cryptography and digital signatures, users can verify the identity of another user — incredibly important in an environment where it is trivially easy to otherwise masquerade as another user for sinister purposes.

That VR needs to be easy to use in order to be popular is self-evident. To users, this means a seamless experience of moving from world to world engaged in activities such as talking to other avatars. To content creators, this means ease of rapidly creating and arranging the elements of a world in an intuitive fashion. To programmers, this means ease of understanding the building blocks of the world, which should follow the principles of object-oriented design.

Extensibility is possibly the single most important feature of a VR system. If cyberspace cannot grow to accommodate changes in technology, what's the point? The VR technology will be obsolete even before it achieves widespread use.

A VR protocol should be object-oriented, distributed, secure, extensible, and simple. It should integrate with existing infrastructure as much as possible, and minimize latency by immediately sending updates to interested parties when the world changes.

It is with these and other design requirements in mind that we present the Virtual Object System (VOS), an infrastructure designed to support development of open Virtual Reality on the Internet.

IV. Terminology

This section briefly describes some terms necessary for understanding VOS.

The most important term is *Virtual Object* or *vobject* (pronounced "vee-object"). Because the term "object" is extremely broad in the English language as well as having a variety of technical meanings, we use the term "vobject" to unambiguously refer to the abstraction provided by VOS. Vobjects are formally defined in the next section, but one important detail to note is that in object-oriented terms vobjects are *instances*, not classes.

Vobjects interconnect to one another with logically oriented hyperlinks. From the perspective of a particular vobject, incoming links are termed *parents* and outgoing links are termed *children*.

Publish/subscribe change notification is termed *listening* and is an important part of programming for VOS. This programming technique enables code to register interest in a particular resource in a separate part of the system. When that resource changes state, all listeners are automatically notified with messages or function callbacks. This obviates the need to periodically poll the resource to see if anything has changed.

A single program will usually create and host a number of local vobjects. These vobjects must have some way of communicating over the network and in particular it is desirable to multiplex the access to these vobjects over a single socket. This is accomplished by binding normal vobjects to a special vobject called a *site*. A site acts as a bridge between local and remote vobjects by exchanging VOS messages over the desired network channel.

V. The VOS Design

The VOS networking framework supports resource discovery, decentralization, security and extensibility in a general way with a particular inclination toward network VR. The high-level abstraction presented is that of a distributed, network-transparent, highly interconnected system of virtual objects that, by their structure and properties, may be used to describe the contents of a virtual world.

A. Goals

The purpose of the VOS core is to support the following features:

- abstract, object-oriented message passing,
- methods for naming, addressing and interconnecting vobjects,
- the ability to add listeners to all aspects of vobjects,
- a uniform naming system, based on URLs, and
- a polymorphic object typing and extension system

B. The Vobject Model

The basic units of VOS are vobjects, which are formally defined by the following object model:

1. The vobject exists with a unique name as a direct child of exactly one site.
2. The vobject can be a source and destination for exchanging messages with other vobjects, with sites handling intermediate transport as necessary.
3. The vobject has a set of type strings. These may be used to describe what interfaces this vobject supports, how to interpret the meaning of child vobjects, or tag the vobject for other special treatment by an application.
4. The vobject has an ordered, associative list of outbound links to other vobjects. Each link has a position and is tagged with an additional contextual name. This is used by the application to determine how to interpret the child (linked-to vobject) in the context of the parent (linked-from vobject). The contextual name may be used more than once in a given child list, and may be blank (in which case the link is characterized only by its position).
5. Both sides of a parent-child link are aware of one another. A vobject may have multiple inbound parent links.

C. Sites

Sites are a special type of vobject that, in addition to following the above object model, act as the connection point between vobjects that are not in the same address space. Sites encapsulate some networking medium used to transport messages to their destination; this medium could be TCP/IP, ATM virtual circuits, shared memory backplanes, etc. [2] Each participant of the virtual world (clients and servers) must have a contact point (such as an open TCP port); it is the resposibility of the site to manage this contact point. Sites also act as security domains, and the connections between sites may be authenticated and encrypted; VOS supports Transport Layer Security (TLS v1) and Secure Sockets Layer (SSL v3) for this purpose. Using X509 [14] certificates, it is possible to authenticate the identity of other sites.

As will be discussed below, a vobject may be referred to through many possible paths, making it difficult to determine if two given paths actually refer to the same vobject. Because of this, every vobject is required to be bound to exactly one site as a direct child with a unique contextual name. This restriction allows us to give objects short unique names so that one needs only the site contact information and vobject's site name to address any vobject.

D. Messages

Messages are used to transmit all information between vobjects. This includes structural information (maintaining parent-child connections) position updates, describing the appearance of things in the world, etc. Messages consist of the following information:

1. The message type,
2. To and From fields,
3. A Method field,
4. A Nonce (unique identifier) field,
5. A Dependency field,
6. A Time field, and
7. After the above header fields there is an ordered, associative list of parameters. Each parameter has a key (which may not be blank but may be repeated in the same message) and a value.

Items 1-6 are headers containing message meta-information that must be treated specially, item 7 is the message "payload", which is split into user-defined fields.

Messages with the update type are a special sort. These messages indicate that a requested or listened-to bit of information has changed. It is desirable to direct these cache updates to the code handling that particular remote vobject. As a result, an "update" message type suggests that the message be delivered to the *local stub* (also called a proxy) of the remote object,

[2] As of this writing (September 2002) VOS only supports TCP/IP for transport.

that contains the code for handling request replies and cache updates.

E. Types

The core of the VOS extension mechanism is a simple and flexible type model for vobjects. Unlike formal class hierarchies found in most object oriented languages, a type string simply expresses that the vobject conforms to a certain *Object Type Definition (OTD).* An OTD is a document that specifies a vobject's structure (what meaning should be attached to child vobjects in particular), what messages it accepts and emits, and any special or domain specific handling or interpretation of the object that is expected in implementing applications. A vobject has a set of type strings, meaning that a single vobject may implement multiple interfaces or standards. All vobjects support the basic vobject linking API.

Type strings are almost free form, with only one important exception. If a type contains one or more period characters ('.'), it means that the type extends each type that would be produced if the portion of the type string following each period were removed in turn. For example, a vobject with the type "object3D.cylinder" inherits all the aspects of "object3D" specified in the appropriate OTD. Were there to be another type "object3D.cylinder.foo" it would be expected to inherit all aspects from "object3D" as well as "object3D.cylinder". A corollary from this is that if an application encounters a type it does not understand (such as the example "object3D.cylinder.foo") it may work its way up the inheritance until it finds a type that it does understand (perhaps "object3D.cylinder" in this case).

Allowing a single object to have multiple type definitions is extremely useful. A common example is the case of user avatars, where the avatar vobject supports (at minimum): "a3dl:object3D.model", "misc:avatar" and "misc:talkative". The first type defines a static geometry model (such as the lego man shown in the first page) with position in 3D space. The second type specifies several fields such as nickname, real name, user information, and presence (such as ready for chat, occupied, away from keyboard, etc). The third type specifies that the vobject accepts and emits the "say" message for text messaging and chatting. It is quite convenient and intuitive to combine these abilities in a single vobject, and makes it easy for vobjects to interact even if each side only understands a subset of the other's interfaces.

F. Linking and Naming

The parent-child linking system was originally inspired by scene graphs, but was later expanded out to support general-purpose structures. Vobject links can form any directed graph, although typically vobject links are used to group vobjects in "contains" or "describes" relationships, which form trees or DAGs (Directed Acyclic Graphs). Links can actually be thought of as data structures in their own right. They list the parent (source) and child (destination) vobjects, as well as the contextual name and position in the parent's child list. Each link in the child list is characterized by its position; the name (which may be repeated or blank and is often meaningful in particular type contexts) is used to determine the context in which to interpret the particular child vobject.

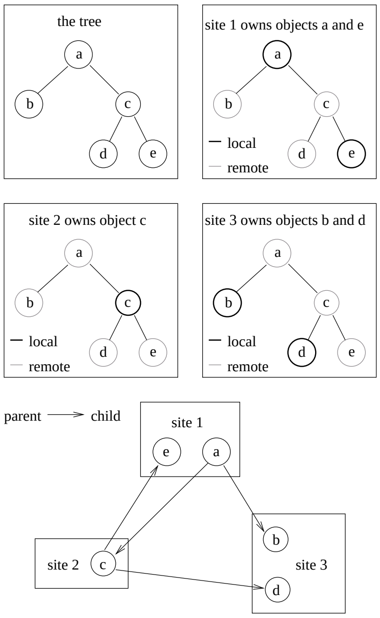

Fig. 2. A graphical example of how the objects on various sites might interconnect. Vobjects which are distributed among several sites can be linked together to form a tree structure.

Vobjects are addressed using URLs in the format "vop://hostname[:port]/objname" Consider the following examples, all of which ultimately refer to the same vobject.[3]

1. vop://interreality.org/world/ball/position
2. vop://interreality.org/1474612399/position
3. vop://interreality.org/1264095060

Although pointing to the same vobject, each path may be interpreted differently. The first path shows the entire scene graph in which the position vobject describes the position of the ball in the world coordinate space. The second path points to the ball directly; the

[3] There is nothing special about the fact that site names are numeric except that it is easy to generate randomly and guard against collisions. We are considering changing the implementation to make these names slightly more useful than long strings of digits.

position still describes the ball but lacks a coordinate frame. The third path points directly to the position vobject with no context at all, it is simply a three dimensional vector. It is entirely possible to rediscover context by asking a vobject for its parents (incoming links) and processing the graph in reverse order.

Because the basic vobject linking model is standard across all vobjects, the same tools to access and modify the vobject structure can be used for a variety of applications. For example, we have written a command line shell called *mesh* which allows one to browse vobject structures using unix-style commands like "cd" and "ls" as well as send and receive messages interactively. This is very useful for inspecting the underlying structure of a virtual environment and providing a layer of transparency to the system to programmers and content creators.

It is this distributed structure that is at the heart of VOS. Failure is isolated to the immediate vobjects that are located on the site that has failed. A particular vobject may be a world root describing the contents of a space, but should that root become suddenly unavailable, users are still capable of communicating with one another directly. Indeed, on one occasion during the testing of this system the world server crashed, and for several minutes the users logged into the world continued to move about and talk with each other, unaware of what had happened!

G. Listeners

Vobjects essentially consist of three dynamic data structures: the children list, the parent set, and the type set. Listeners enable the programmer to request that a particular vobject send a notification update message whenever the state of one of these lists changes. The notifications are: child add/replace/remove, parent add/remove and type add/remove. The concept of listeners is also used for some object types as well (most notably the "property" type). Setting up listeners between sites reduces latency in the system as the interested parties are sent update messages immediately when a change occurs. It also allows for event-driven programming, and many of the behaviors of a VOS application may be based on reacting to changes in the vobject structure.

VI. How VOS is Used for VR

Layered on top of the VOS infrastructure are the services and object types used for a virtual environment.

The Abstract 3D Layer (A3DL, pronounced "ideal") is a set of object types for various 3D primitives and concepts. These include spheres, boxes, cones, cylinders, static 3D models (loaded from external file formats such as .3DS), vertex/polygon geometry directly specified using vobjects, lights and viewpoints. Most of these types extend the type "a3dl:object3D". This type specifies that a 3D object has three child vobjects, named "position", "orientation" and "scaling". Each of these vobjects is a property supplying relevant information about the logical 3D object. Because these properties are separate vobjects, it is possible to seamlessly use a different and remotely located vobject such as one supplying position from a GPS unit, or orientation from a head tracker. Extensions to "a3dl:object3D", such as "a3dl:object3D.cube", include additional child properties appropriate to that type (such as coloration or material).

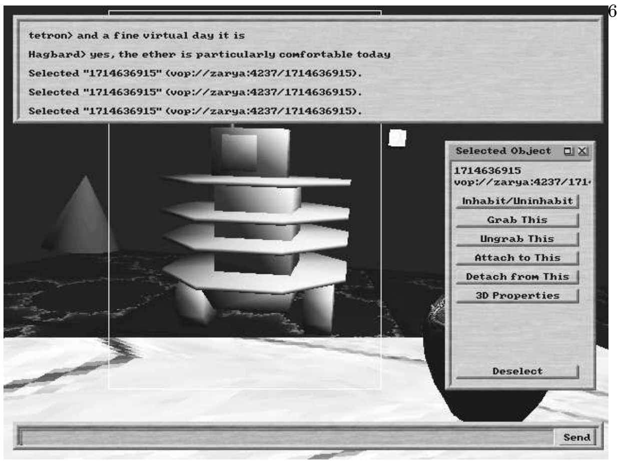

Fig. 3. Selecting an object in Ter'Angreal

The 3D rendering application that realizes a world described with A3DL on screen is named Ter'Angreal[4]. This is the main user interface to the virtual world. This client allows the user to join and move around the world, to manipulate objects with the mouse and keyboard commands and to talk to other users. To simplify implementation, we have made use of the 3D engine Crystal Space [15]. Using this preexisting code base has allowed us to focus on the VOS networking infrastructure and on application-level development.

Network lag is one of the most difficult problems affecting the user experience of virtual worlds. The most common manifestation is seeing jerky, inconsistent movement of other users in the virtual world because it is impossible to send out position updates anywhere close to the framerate at which the client is rendering graphics. The usual solution to this is known as "client side prediction" or "dead reckoning". In VOS this is implemented as a generalized

[4]A *Ter'Angreal* is a magical artifact from Robert Jordan's *Wheel of Time* saga. Some of the *Ter'Angreal* most pivotal to the plot are gateways to other worlds.

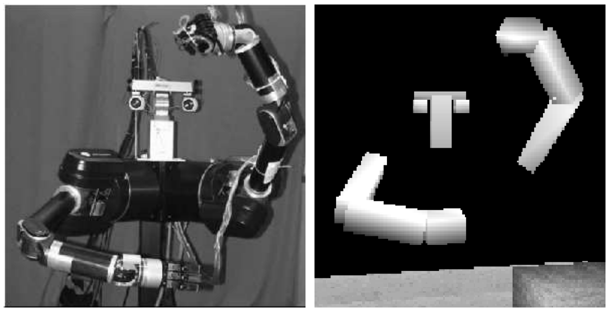

Fig. 4. The virtual representation of the arm mirrors, in real time, the position of the real arm. Figure is from Amstutz, Fagg:2002 [15].

vector extrapolation along a quadratic curve. When an object moves, the update which is sent out supplies the coefficients and initial value for t. On the client side, position is calculated over time t (in seconds) as $< x_2t^2 + x_1t + x, \quad y_2t^2 + y_1t + y, \quad z_2t^2 + z_1t + z >$ presenting the user with smooth linear or parabolic motion of the desired object. In addition, this facility can be used to extrapolate other vector quantities including rotation and scaling.

VII. Applications of VOS

Several applications have been implemented using this system. One such application is a network-accessible three-dimensional visualization of the state of a jointed robotic arm. The real robotic arm in question is part of a torso here at the University of Massachusetts, Amherst. The torso consists of a pair of seven-degree-of-freedom jointed arms, each with a three-fingered hand (the hand itself is fairly complex and not presently modeled in the visualization). Communication with the arm controllers is accomplished through a dedicated shared-memory network, giving real-time information about joint positions (as well as rotational velocity and other aspects of the state). Real joint angle information is used to update the positional state information of the 3D model. This 3D model represents the actual upper arm, forearm, and hand in a way that can be easily rendered. This work was reported in detail in "Real Time Visualization of Robot State with Mobile Virtual Reality" [16].

VOS was chosen to develop an augmented reality game. Players are equipped with Xybernaut MA 4 wearable computers, wireless networking (802.11b), a GPS unit, and a 3-DOF Intersense IS-300 head tracker [17]. A game server is run on a computer accessible via the wireless network. Players go out into the field and place goals by walking around in the real world and registering the goals based on their GPS coordinates. Each player is presented with an augmented reality 3D overlay view used to find the goals set by other players in order to score points. To make the game more interesting, players may also lay down traps which detract from the score of any players who tread upon the affect area (again based on GPS coordinates). VOS was used to manage all the objects in this virtual space and Ter'Angreal (which did not need to be modified to support any game-specific features) was used to present the augmented reality interface.

VOS has been used to build a VR interface to the Player-Stage distributed robotic control and simulation system [18]. Models representing mobile robots (both real and simulated) are inserted into the world and track with the robot's movements. Operators using Ter'Angreal can monitor and issue commands to the robots such as adding goals and waypoints through typed and spoken commands.

VIII. Future Work

VOS has come a long way and has many features for supporting virtual environments. We have plans for many new features which will greatly enhance the experience of using the virtual world.

VOS currently lacks any sort of physics simulation. Physics is usually expressed in virtual environments through collision detection and gravity. Presently, users are unconstrained and may move anywhere in the world. This brings up difficult questions of control, for in a decentralized system there is the question of what part(s) of the system may be responsible for doing the necessary physics simulation. Our tentative design is to make physics the responsibility of the site that owns a particular object. A user's client program would determine the gravitational acceleration vector (perhaps based on policies that vary from world to world); when two objects collide, each object would calculate its own bounce trajectory independently.

Presently, animating the movement of a model requires breaking it up into individual vobjects, as was done for the robotic torso. Unfortunately this has a couple of disadvantages: it greatly increases the number of vobjects to be downloaded and it is not possible to "skin" a single texture seamlessly across several independently movable segments. One possible solution, widely used in three-dimensional games, is to have a pre-set list of animation key frames. The user would then indicate that the model is to assume a particular posture, or move through a particular sequence, linearly interpolating vertices between key frames. Although fairly easy, this technique has several significant disadvantages. The user is necessarily limited to set actions, which is often unsatisfying when there is no available action expressing the user's intent. This technique also suffers from the problem that the animation is entirely independent of the current context and suffers from problems such as "ice skating" where the strides of a walking model very obviously do not match up with the movement of the model as a whole. Specific movements, such as an avatar reaching out and picking up an item, are nearly impossible. For these reasons, we intend to investigate forward and inverse kinematic techniques as a way of allowing arbitrary motion of articulated figures, and for reducing network traffic (by sending only a few control points, and doing the kinematic calculations on the client side).

Finally, an important aspect of virtual worlds is the aural environment. Ter'Angreal does not presently support sound. We intend to add support for effect presets (downloaded ahead of time, which may then be triggered) and as well as streamed sound for uses such as background music.

IX. Conclusion and Final Notes

In this paper we have considered the work so far and challenges in building multiuser virtual reality systems. We presented the Virtual Object System, our design for an infrastructure to support distributed, peer-to-peer virtual environments. VOS vobjects have shown to be a valuable framework for creating and maintaing the dynamic distributed data structures necessary to describe virtual worlds. We have described how this infrastructure addresses some of the specific challenges of multiuser virtual reality and the applications it has been successfully put to so far. We concluded with a discussion of some of our future plans to build on to this system; VOS continues to be under active development.

The VOS core library and its applications are written in C++ and have been primarily developed in a Debian GNU/Linux environment with ports to Macintosh OS X and Microsoft Windows. A complete set of Perl bindings are also available. VOS is free software, available under the terms of the GNU Lesser General Public License [19]. It may be downloaded from http://interreality.org/software. Additional documentation is available at http://interreality.org/docs. A demonstration and live screenshot of the running test world server through the "virtual web cam" is available at http://interreality.org/software/vr. The toplevel VOS project page is http://interreality.org. We invite interested members of the community to collaborate with us in our goal to create Internet Reality — Interreality.

X. Acknowledgments

The work described in this manuscript was supported in part by NSF #EIA 9703217, DARPA/ITO #DABT63-99-1-0022 (SDR) and DARPA/ITO #DABT63-99-1-0004 (MARS). The authors wish to thank Andrew Fagg, Brian Levine, Jack Wileden and Sarah Morrison for their comments and support during the development of this work and the writing of this manuscript.

References

[1] ISO/IEC Joint Technical Committee 1 Information technology Subcommittee 24 Computer graphics, image processing, and The VRML Consortium Inc (http://www.vrml.org) with the VRML moderated email list (www–vrml@vrml.org), "The virtual reality modeling language," ISO/IEC 14772-1:1997.

[2] Blaxxun Interactive, "Contact," http://www.blaxxun.com.

[3] Geometrek, "Deepmatrix," http://www.geometrek.com/products/deepmatrix.html.

[4] Jeff Sonstein and Stephen White, "Vnet 1.1b1," http://ariadne.iz.net/ jeffs/vnet/.

[5] The VRTP Working Group, "Virtual reality transport protocol," http://www.web3d.org/WorkingGroups/vrtp/.

[6] The Living Worlds Working Group, "Living worlds: Making vrml 97 applications interpersonal and interoperable," http://www.web3d.org/WorkingGroups/living-worlds/.

[7] Id Software, "Quake," http://www.idsoftware.com.

[8] Sony, "Everquest," http://www.everquest.com.

[9] P. Curtis and D. A. Nichols, "Muds grow up: Social virtual reality in the real world," in *In Proceedings of the Third International Conference on Cyberspace*, May 1993.

[10] Naval Postgraduate School, "Distributed interactive simulation standard," IEEE Standard 1278.1 (series).

[11] Steve Pettifer, Jon Cook, James Marsh, and Adrian West, "Deva3: Architecture for a large scale virtual reality system," in *Proc. ACM Symposium in Virtual Reality Software and Technology 2000 (VRST'00)*, October 2000.

[12] Emmanuel Frécon and Mårten Stenius, "Dive: A scalable network architecture for distributed virtual environments," *Distributed Systems Engineering Journal*, vol. 5, 1998, Special Issue on Distributed Virtual Environments.

[13] B MacIntyre and S. Feiner, "Language-level support for exploratory programming of distributed virtual environments," in *Proceedings of 1996 ACM Symposium on User Interface Software and Technology*, November 6-8 1996, pp. 83–95.

[14] ITU-T Rec. X.509 (REVISED), *The Directory Authentication Framework*, International Telecommunications Union, ISO/IEC 9594-8:1994.

[15] J. Tyberghein, A. Zabolotony, E. Sunshine, T. Hieber, S. Galbraith, M. Geisse M. Voase an S. Humphreys, A. Pfaffe, M. Ewert, R. Bate, G Haussmann, and P. Wyett, "Crystal space manual," 2002, http://crystal.sourceforge.net/docs/online/manual/.

[16] Peter Amstutz and Andrew Fagg, "Real time visualization of robot state with mobile virtual reality," in *Proceedings of 2002 IEEE International Conference on Robotics and Automation*, 2002, pp. 241–247.

[17] E. Foxlin and M. Harrington, "WearTrack: A self-referenced head and hand tracker for wearable computers and portable VR," in *Proceedings of the Fourth International Symposium on Wearable Computers*, 2000.

[18] Brian P. Gerkey, Richard T. Vaughan, Kasper Støy, Andrew Howard, Gaurav S. Sukhatme, and Maja J Mataric, "Most valuable player: A robot device server for distributed control," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, October 29 - November 3 2001, pp. 1226–1231.

[19] Free Software Foundation, "Gnu lesser general public license," http://www.gnu.org/licenses/gpl.html.