VIRTUS: A Collaborative Multi-User Platform

Kurt Saar
Institut für Betriebs- und Dialogsysteme
University of Karlsruhe
Kaiserstrasse 12
D- 76128 Karlsruhe

ABSTRACT

VRML is a file format for the description of dynamic scene graphs containing 3D objects with their visual appearance, multimedia content, an event model, and scripting capabilities. It is designed to be used on the Internet and on local system and to be used as an exchange file format. Although equipped with sophisticated techniques for user interaction the current VRML standard still lacks direct support for sharing virtual worlds that can not only be visited but also manipulated by multiple users distributed over the network. Several multi-user technologies have been developed in the past and some use VRML as the rendering and interaction vehicle.

This paper gives a short review of design considerations for distributed virtual environments and approaches taken so far in the development of multi-user technologies. We present the design and implementation of VIRTUS, a multi-user platform that allows multiple geographically separated users to enter and manipulate shared VRML scenes.

CR Categories and Subject Descriptors: C.2.4 [Computer-Communication Networks]: Distributed Applications - H.5.3 [Information Interfaces and Presentation]: Collaborative Computing – I.3.2 [Computer Graphics]: Distributed/Network Graphics - I.3.3 [Computer Graphics]: Picture/Image Generation - Viewing Algorithms; I.3.6 [Computer Graphics]: Methodology and Techniques - Interaction Techniques; J.6 [Computer Applications]: Computer-Aided -Engineering

Additional Keywords: VRML, Virtual Worlds, Virtual Environments, VRML Event Model, Multi-User Technologies, Distributed Environments, Living Worlds, Collaborative Virtual Environment (CVE), Computer Supported Collaborative Work (CSCW), Architecture Construction Engineering (ACE), Dead Reckoning

1 INTRODUCTION

Essentially he WWW is a solitary place where many people see the same information but have no perceptible support for interaction or awareness of one another. Multi-user worlds or DVEs (Distributed Virtual Environments) have the potential to evolve the Web from a pure information space to a social space and to radically alter the way we learn, work, consume, and play from isolated pursuits to collaborative activities.

Steady progress has been made over the last 20 years of research in DVEs and many current systems can portray large, complex, rapidly changing environments containing a large number and variety of actors with some degree of fidelity. But two factors have delayed public interest in DVEs: real time graphics, audio hardware, and networks have not been accessible to everyone. These barriers disappear with the availability of powerful graphics accelerators for PCs, high-speed modems, ISDN-adapters and ADSL. The user interfaces of many traditional and new applications can now take the next step in the evolution from command-line to desktop metaphor to a real 3D-community metaphor which allows us to change virtual worlds and observe changes made by other geographically separated people. Large scale DVEs may turn out to be a new Grand Challenge for computer science.

Most of the DVE systems developed so far have been limited to certain platforms, proprietary applications, or network protocols. As the Living Worlds [17] initiative and Open Community indicate there is increasing movement towards open standards. VRML is central to this movement. Since the current VRML standard does not offer language constructs for direct multi-user support most VRML scenes today run on a single machine, have limited interaction and respond to a single user's input. The implementation and authoring of multi-user environments still requires skilled programmers with thorough knowledge of VRML, Java, JavaScript, and the EAI.

We present VIRTUS, a multi-user platform based on VRML2.0, Java and TCP/IP, which eases the development and authoring of distributed environments with a special focus on collaborative work. In a case study we use VIRTUS for architectural construction and engineering purposes.

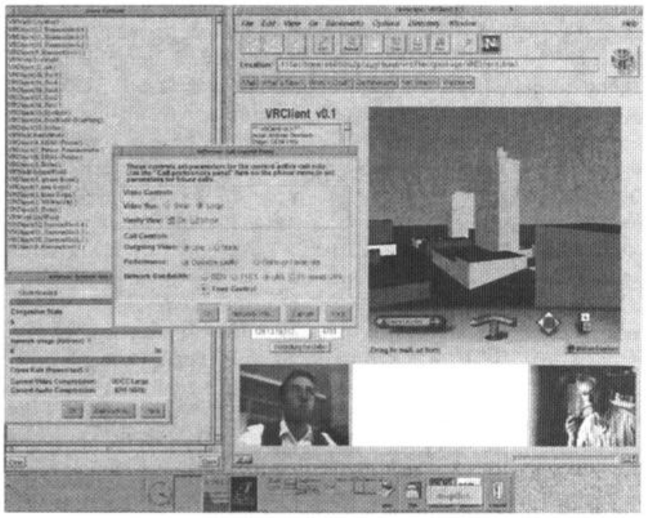


Figure 1: Screenshot of the VIRTUS user interface

The rest of this paper is organized as follows: *Section 2* offers a general overview of multi-user developments with or without use of VRML and gives a classification of distributed virtual environments. *Section 3* reviews related work and summarizes some of the hightlights of DVE development. *Section 4* discusses the architectural design of our multi-user system VIRTUS while *section5* focuses on some implementation specific details. The remaining sections provide lessons learned from the use of VRML in the context of architecture, some experimental results and an insight into our current and future efforts in this field.

2 DESIGN CONSIDERATIONS FOR DVE SYSTEMS

This section discusses the issues anyone is confronted when trying to implement the key features of DVE systems: Several, if possible thousands of geographically separated users are allowed to meet and interact in real time in a dynamically and persistently modifiable 3D virtual scenery. Verbal communication should not only be supported via text but also via voice chat. These requirements immediately raise a number of questions concerning the division of data and functionality into atomic parts and packaging each of these as coherent code structures:

- What is distributed?
- How is it distributed?
- Why is it distributed?

The answers to these questions illuminate advantages and deficiencies of different concepts and lead to a classification of DVE systems.

2.1 Fields of application

More and more applications are enjoying networked 3D computer graphics: military simulations, virtual surgery, engineering, architecture, CBT, gaming, product presentation and virtual shopping malls. The development of DVEs has been driven forward in three fields:

Military Applications: The benefits of virtual environment for military purposes have been realized early: no danger of life or destruction, no real damage, strategic simulations in arbitrary terrain and landscapes, simulation of vehicle prototypes. More possibilities are added if the environment is distributed: training of teams, scalability of the number of participants, installation in different separated locations, simulation with a combination of several military forces and semi-automated forces (SAFs). The most prominent developments in the military realm have been SimNet[23] and DIS[16]. Many DVE concepts trace their foundations to the SimNet project.

Entertainment: the entertainment sector offers a potentially large marketplace either for *home-based* or *location-based* entertainment. In the 1970s games like Adventure or Dungeon & Dragon spawned a new genre of role-playing games. MUD (Multi-User Dungeon) and their object-oriented versions MOO became the generic descriptions for multi-user games. Home-entertainment devices fall in two categories: game consoles and PCs. Game consoles have little support for multi-user playing. The history of multi-player games and teaching systems goes back to the 1960s (PLATO) and culminated 1993 in Doom. More and more games with support for TCP/IP and protocol tunneling reach the market. Some current DVEs have evolved out of earlier MUDs (Habitat, Worlds Away, Pueblo). The notion *location-based entertainment* describes multi-player gaming with specialized equipment in BattleTech centers or amusement parks.

Research and commercial systems: Most of the money has been spent by and for the specialized high-end military applications and low-end networked games [32]. The widespread use of the WWW makes computer supported collaborative work (CSCW) and virtual shopping applications more and more interesting for research and commerce. Goals and principles of the Living Worlds proposal include the rapid implementation of strictly VRML-based multi-user environments and respect for the role of the market. Multi-user VR tends create immersive

2.2 Performance

Performance has at least two aspects: **rendering performance** must be improved locally by using specialized hardware for coordinate transformations, HLHSR, shading, texture mapping and making the graphics hardware accessible to applications in a comfortable way via low and high level APIs like OpenGL, Direct3D, Java3D or Fahrenheit. This is not a DVE-specific problem and the boost of graphic hardware performance we currently experience alleviates the problem from day to day.

The much bigger challenge is **communication performance**. While research labs may have access to Gbps-networks consumer links are usually 28.8-modems or ISDN-channels. The limited bandwidth requires a reduction of the amount of network traffic (textual, motion, state, geometry, sound, video) and limiting the number of objects and avatars (and thus polygons) to be rendered. Approaches reducing the amount of necessary communication have implication on the scalability and consistency of a DVE.

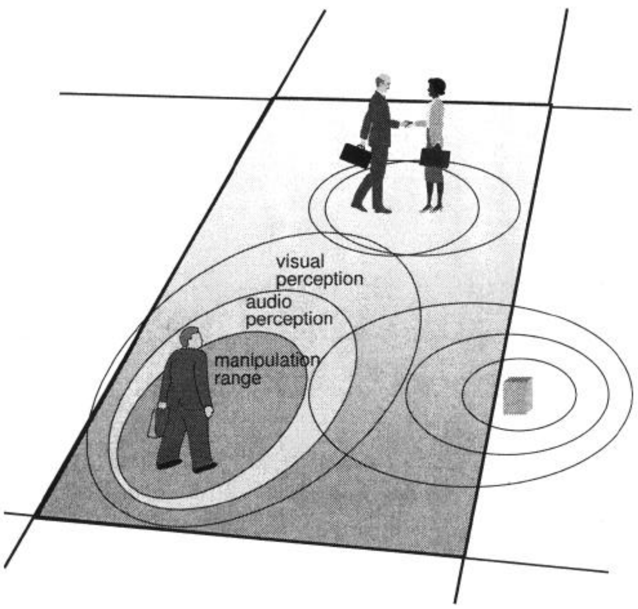


Figure 2: Environments are partitioned into polygonal regions; avatars and objects may have several horizons for visual, acoustic perception and radiation and for areas of interest.

2.3 Scalability

The number of possible interactions between n simultaneous users in a multi-user system is of order $O(n^2)$ at any moment. Ideally network traffic should be almost constant or grow near-linear with the number of users.

Usually there are large regions and associated data in the environment that may not be relevant to a particular user at a given time. This suggests the idea of partitioning the environment into regions (or zones, locales, auras)[2,15] that may either be fixed or bound to moving avatars. Events and actions in remote zones need not be distributed and remote objects need not be visualized or might be visualized at a lower level of detail. Most of the traffic is isolated within zones.
Consider an indoor scene with different rooms separated by walls and connected by doors and hallways: Walls may be marked as boundaries of regions, doors may be marked as portals to adjacent rooms. When the user leaves one region to enter a different one, some kind of region management has to detect the situation and present the new region to the user. This might involve a sudden download of very large scene description files. Predictive systems might even anticipate user movements and a just-in-time scheduler might load regions in advance to avoid interruptions and delays when entering.
Partitioning the world is the key to scalability. Some system use the notion aura or area of interest surrounding the user at his current position. This might cause bizarre effects in crowded aggregations: users in close face-to-face proximity might not see each other just because they are in different aura groups.
There are several variations to the partitioning paradigm.

2.4 Consistency

Since the database is shared by all clients accessing and maybe updating the data, the issue of distributed consistency must be solved by any DVE to ensure the same view to all participants. Solving consistency means satisfying user predictions and providing a basis for *causality* in an environment.
Since the number participants is not fixed during the run time and users may enter the environment after it has changed from its initial state there needs to be *support for late-comers*.
There is always some degree of uncertainty concerning the current state of a DVE and we can define different **levels of consistency**: *Strict consistency* requires immediate propagation of all actions. Conflicts between users must be prohibited or resolved.
The *best effort approach* relaxes the guarantee for consistency in the temporal or value domain to reduce communication costs.
The issue of consistency is tightly coupled with the issues of data replication and communication protocols via the question: *Who needs to know what and how frequently?* Crucial aspects to the efficiency of any consistency algorithms are the number of participants in the consistency algorithm and the degree to which consistency will be guaranteed to these participants. Since the complexity of the problem is quadratic in the number n of participants, reducing n significantly reduces the number of messages to be exchanged. The number of participants in a consistency group may be reduced by partitioning environment.
CSCW oriented systems have strict requirements for consistency throughout the DVE they focus on sophisticated group interaction models. Large scale simulation platforms on the other hand put their focus on simple state updates not on complex.
SimNet uses *dead-reckoning* algorithms to extrapolate positions of entities based on their last known position, velocity and elapsed time since the latest information receipt. The protocol data units (PDUs) distributed contain just enough information to approximate the current state.
Many DVEs reduce computing and communication costs by updating scene states only in areas relevant to users. Does this mean the system may cull all dynamics and autonomous actions in areas without observers? This may be appropriate in some applications, but not in simulation systems relying on causal relationships[11].

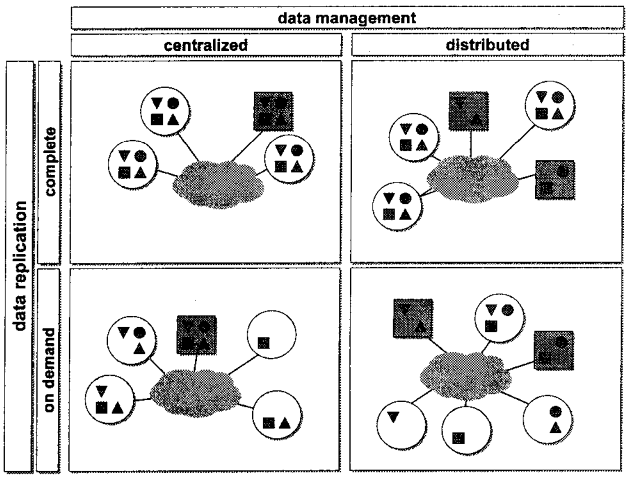


Figure 3: Data management and data replication

Data describing the DVE contain information about the world and its objects. The data may be managed either by a central host with guaranteed data consistency or a set of decentralized hosts. The latter method raises the question how much of the data should be distributed and how and when to update the distributed copies.

2.5 Connectivity

Connectivity has various dimensions including bandwidth, capacity, protocols and topologies[8]. One peculiarity of DVEs is the heterogeneity of data to be transferred: there is a mixture of continuous high-bandwidth real-time transmission of data streams (>32 kBps audio or >1 MBps for video), occasional burst transfers of large files (graphical data, geometry, texture) and frequent small data packages (position updates, state changes).
The heterogeneity of data types, required bandwidth and reliability suggest the use of heterogeneous transport protocols using several different basic Internet protocols rather than a single fixed distribution scheme. Since there is no established infrastructure for efficient distribution at hand multi-user developers are still dependent on low-level protocols. The complexity of network details should be hidden to the user and the application programmer. Future **Dial-a-Behavior-Protocols** (DBPs) promise to modify the operation of protocol on-the-fly with respect to syntactic and semantic packet contents.
Today we must be satisfied with the protocols available on the different layers:
The basic layer-3 Internet protocol in its current version **IPv4** and next-generation **IPv6** may be run on a variety of physical media and link control protocols. The transport layer offers the connection-oriented **TCP**, suitable for reliable transfer, and the connectionless and unreliable **UDP**. Since each UDP-datagram may take an individual route, there is no guarantee for arrival of datagrams and the correct sequence of arrived datagrams. Less overhead and lack of retransmissions make UDP a suitable protocol for the transmission of continuous data streams
UDP may be used in unicast or multicast mode.
MBone, the experimental subnet of the Internet uses multicast UDP. Hosts can subscribe of ignore multicast packet down at the hardware level by informing the network adapter which multicast addresses to monitor. This way, high-bandwidth streams can reach a large group of hosts identified by a single multicast-address

without producing load on hosts not in the multicast group. However: not every machine is equipped with multicast-capable hardware. The major bottleneck is located on the application layer: **HTTP** in combination with **FTP**, **gopher**, **telnet** is optimized for serving hypermedia documents from a server to a single client in a reliable way. The IEEE **DIS**-protocol (Distributed Interactive Simulation), which was initially developed for military use (SimNet, NPSNET) specifies the interactions of physical entities by exchanging state information in its PDUs. This state includes position, linear and angular velocity and acceleration, articulation, etc. Other data units include simulation and logistics management, sensor-weapon interactions, radio communication which are of questionable use for many applications. Transferring state information frequently provides a certain degree of consistency and allows for connection and disconnection of participants at any time, but these heartbeats generate permanent network traffic. The DIS-Java-VRML working group is currently exploring the DIS-Java interaction in large scale virtual environments.
More recent proposals **VRTP** (Virtual Reality Transfer Protocol)[7], **ISTP** (Interactive Sharing Transfer Protocol)[31], or **DWTP** (Distributed Worlds Transfer Protocol)[6] assure to provide a frameworks for the optimized combination of underlying protocols. The **CBone** (CyberBackbone) is intended to improve MBone and offer a guaranteed QoS by partitioning traffic within and between networks with the help of area of interest managers (AOIMs). All these proposals indicate the need for a suitable distributed virtual reality communication infrastructure.

Information Type	Current Protocols (Layers 4-8)
Files *(scene and avatar descriptions, scripts)*	HTTP, FTP, TCP
Events *(small updates)*	UDP(uni-/multicast)
Messages *(small but important data0*	TCP
Streaming Data *(audio, voice chat, video)*	RTP, UDP

Table 1: Information types in DVE and more or less suitable network protocols

2.6 Distribution Concepts

2.6.1 Distribution of Data

Another attribute of DVEs is the way data are distributed to the participant.

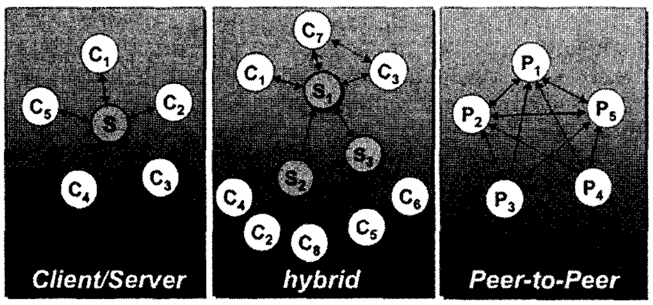
Unicast: the sender sends a message to exactly one receiver which is known to the sender. (1:1 point-to-point)

Broadcast: the sender sends a message that may be received by all instances currently present in the system. Each receiver can individually decide if the message is relevant in the current situation. (Broadcast packets are filtered by routers to avoid network pollution.)

Multicast: the sender sends messages to a well defined subset of instances present in the system.

2.6.2 Distribution of Processes

The critical choice of the process distribution subsystem has implications on design decision in other subsystems. The communication architecture can be hierarchical (client/server-model) or decentralized (peer-to-peer-model). Many discussions deal with the question "client/server or peer-to-peer? "



Client-Server: the majority of small-scale DVEs are realized as client-server architectures because this is conceptually and practically the simplest and provided possibilities for accounting and security management. The scene database is centralized in one location: the server. Clients request scene descriptions from the server and provide any relevant changes originating at their site to the server. Inconsistencies in the world description can occur if the clients hold local caches of the world. If any changes to the cache should be necessary, the server has to invalidate the cached data and replace it by an up-to-date master copy. Obviously the server easily becomes a bottleneck with increasing number of clients. Empirical evidence proves the scalability of sophisticated client/server systems to several hundred users. Many commercial DVEs use the client server model, not only for technical but for administrative and financial reasons: clients are free, while servers and services are sold.

peer-to-peer: This architecture does not provide a single database or master copy of worlds. Each client hold its own copy or replica of data. Of course, changes to scenes used by several clients have to be distributed to all participating browsers. The maintenance of consistency is much more difficult here but in general peer-to-peer system scale much better to thousands of simultaneous users.

hybrid systems: Single server architectures have at least two drawbacks: the server may easily become a performance bottleneck and failure of the single server crashes the whole DVE. Hybrid systems merge client-server and peer-to-peer architectures by replicating and updating the world database on several servers and still providing client-server communication links from the browsers to the servers.

The existence of a broad spectrum of functionality between the endpoints client/server and peer-to-peer suggests that this discrimination is not a proper dichotomy. The VRTP protocol is designed for efficient data transfer and monitoring in client/server as well as in peer-to-peer environments.

2.7 Persistency

DVEs should be modifiable at run-time by accepting the contribution of new objects and new behavior and optionally saving these even after the user who created them has disconnect from the system. This issue also involves the creation of user profiles, persistent roles and access rights for users.

2.8 Ownership and Operation Locking

Multi-user worlds offer lots of opportunities for conflicts. One user tries to open a door while another user tries to close it. These conflicts must be avoided or resolved. One method to determine which objects may be modified by which users is to assign temporary ownership to objects. Manipulation of objects may include a change of the object's coordinate system and changes in the scene graph: users may grasp objects and take them to a different world. Operations like these are essential for virtual shopping applications. If owning an object means owning the complete subtree (child objects) in the VRML scene graph, then it might be difficult for user A to present a box of chocolates to user B and ask to grab a chocolate. In this situation a locking mechanism on an operation or interaction basis is more appropriate.

2.9 Realism

Besides a degree of visual realism that is acceptable to the user DVEs should aim towards sensory fidelity concerning *visual*, *aural* and maybe even *tactile* information presented to the user that replicates the real world to a maximum extent possible.
One familiar example for missing realism is the lack of collision handling[18], Objects in the real world to not tend to penetrate one another unless they break into pieces. Objects in VRML-scenes and most other virtual environments or modelers do not hesitate to do so. VRML-Browsers offer an approximate collision detection between avatar and collision groups, but inter-object-collisions are ignored. VRML-browsers should provide some visual cue of contact and penetration by geometrical intersection tests. Grasping and assembly operations would benefit enormously from contact determination with visual or acoustic feedback and from collision avoidance with constraints. Robust and accurate collision determination is a time-consuming feature usually sacrificed for the sake of rendering speed.
Some applications might even require the insertion of real-world phenomenology like weather, time of the day, location of the sun[28]. On the other hand simulations in virtual reality allows us to ignore and go beyond the limitations of reality imposed by the laws of mechanics (kinematics, dynamics, electrodynamics, energy conservation).

2.10 User Interface

User interfaces for 3D virtual multi-user environments demand several interesting additions to traditional WIMP-interfaces consisting of screen, keyboard and mouse. Movements change the user's visual and auditory perspective. For immersion into the VE we have to offer three dimensions to eye, ear, and hand. (Auto)stereographic displays for graphic output, spatialized sound and real-world mock-ups or gesture-capturing input devices are components of expensive VR-Equipment.
There is still a deficit of HCI theories for merging multi-sensory data streams into a single UI, most of the problems have their roots in time lags: input device lags, processing lags, rendering lag, synchronization lags, and frame-rate induced lags.
In certain applications (flight simulators) inappropriate user interfaces may even induce negative training. We also lack a theory for the estimation of the indispensable levels of fidelity.

2.11 Language

The implementation of DVEs may vary in some other aspects which we just list here:

- programming language of the system
- scene description language: VRML, VRML variant, proprietary language
- object behavior description language: Java, JavaScript, VRMLscript, Tcl, etc.
- implementation: API, toolkit, or integrated system.

3 REALIZED SYSTEMS

This section summarizes some features of a set of popular DVE systems. We classified them according to the heterogeneity and distribution concept of their processes. The list is not exhaustive.
Client-server systems

NVR, the Networked Virtual Reality is a C-API for the development of multi-user VR applications. Client and server communicate via TCP/IP. Objects are kept in client-side database, objects and interactions are classified as local or global.

Open Community [11] is an proposed open standard API callable from Java and C for multi-user support in the Internet. It is designed to integrate with the Living Worlds and Universal Avatars specifications as successor for **Spline** (Scalable Platform for Large Interactive Network Environments). The API is independent of graphical description languages. Open Community is based on several servers providing different services: a *Session Server* manages multicast addresses for a local area, a *Locale Server* is responsible for a certain region (locale) of the world and manages all objects within that region, a *Persistence Server* manages ownership, and a *Beacon Server* provides the necessary multicast addresses to client when they initiate the connection to a Locale Server. The partitioning of the *WorldModel* into several, possibly overlapping locales is the primary way Open Community achieves scalability to thousands of users. Messages are limited to those coming from the current and adjacent locales. Each locale has two multicast addresses: one for static objects and one for streaming data (voice chat, positional audio). Communication is accomplished via ISTP.

dVS/dVISE (distributed Virtual System) and its authoring extension dVISE are successful commercial client-server systems composed of several actors. Each actor has a specialized task (like I/O or simulation) and a database of relevant information. Dead reckoning is used to keep update communication at a low rate.

Community Place[20]: The server process (bureau) acts as a position tracker and message forwarder and uses an areas of interest algorithm to select the clients to be informed: any objects intersecting a user's aura are candidates for influence or interaction, focus represents the degree of interest one user brings to bear on another and nimbus represents the degree of attention one user pays to another. Static data and scripts are replicated locally in the clients.

Two more systems are the commercial **Blaxxun Community Server** capable of serving up to 5000 simultaneous users [3] and **VNET**.

3.1 Peer-to-peer systems

SimNet, the Simulation Network was initiated in 1983 by DARPA and is in use as a military battle simulation environment until today [23]. There are no central instances. State changes of objects are distributed by broadcast. SimNet's communication protocol DIS was established as IEEE-standard in 1993. SimNet and DIS are designed for large-scale environments with some 10^5 participants.
The **NPSNET** project [21] at the Naval Postgraduate School in Monterey started in 1986 and has been influenced by SimNet since 1988. NPSNET is compatible to DIS and can use IP Multicast over MBone. NPSNET divides the world statically into chunks and associates different multicast addresses to different regions. Changes performed by any client in a region are distributed to clients within the same region via its multicast

address. SimNet and NPSNET use dead-reckoning algorithms to reduce network traffic and repair packet losses. **GSnet** [22] is the implementation of a collaborative development environment in the framework of the Greenspace project at Fujitsu Ltd. and the Human Interface Technology Lab at the University of Washington. It uses IP-Multicast by default. GSnet worlds are composed of *chunks* representing objects and methods. Chunks may be grouped and maintained in distributed databases. Groups are connected by specialized chunks and may be exchanged from host to host. Modification of chunks is allowed only to the current chunk owner. **VEOS** (Virtual Environment Operating Shell) [4] has a special focus on rapid prototyping, portability, and distributed computing. The VEOS kernel is composed of three components: SHELL is responsible for process initialization, memory allocation and scheduling of computing time. TALK coordinates the interprocess communication via unicast. NANCY is a transaction manager for a distributed database. A virtual environment is a collection of autonomous entities (sets of lightweight and heavyweight processes). Communication between entities is handled by a separate component named FERN. The Distributed Interactive Virtual Environment. **DIVE** [10,30] was developed at the Swedish Institute of Computer Science. Each host is part of exactly one world and has a complete copy of this world which is replicated on demand. High-resolution traffic is limited to dynamically associated auras. The aura concept is refined by the notions of focus, nimbus, and awareness. Experiments identified the size of auras and the maximum number of users in one aura as two parameters crucial to system performance. **MR** [25,26], the MR Toolkit and the MR Toolkit peers package are developments for handling a variety of input devices and provide mechanisms for UDP based communication between VR applications over the Internet. They are intended for a small number of users, five or less.

3.2 Hybrid systems

SpaceFusion [29]: objects may be moved, shared, and change their owner. SpaceFusion adopts a multi-server approach: multiple server ensure scalability and stability and incorporate security and filtering mechanisms. Filtering is based on the concept of regions. A client-side fusion manager combines information from different servers before presenting it to the user. Other prominent system are **WAVES** [19], **AVIARY** [27], **BrickNet**, **MASSIVE**[13,14] and **CAVERNsoft**. Community Place, although inherently client/server, has elements of hybrid architecture.

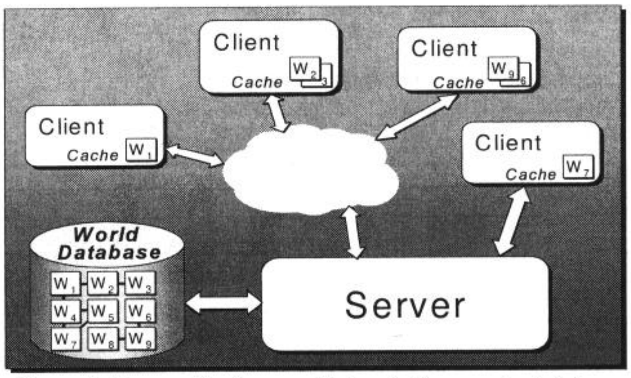
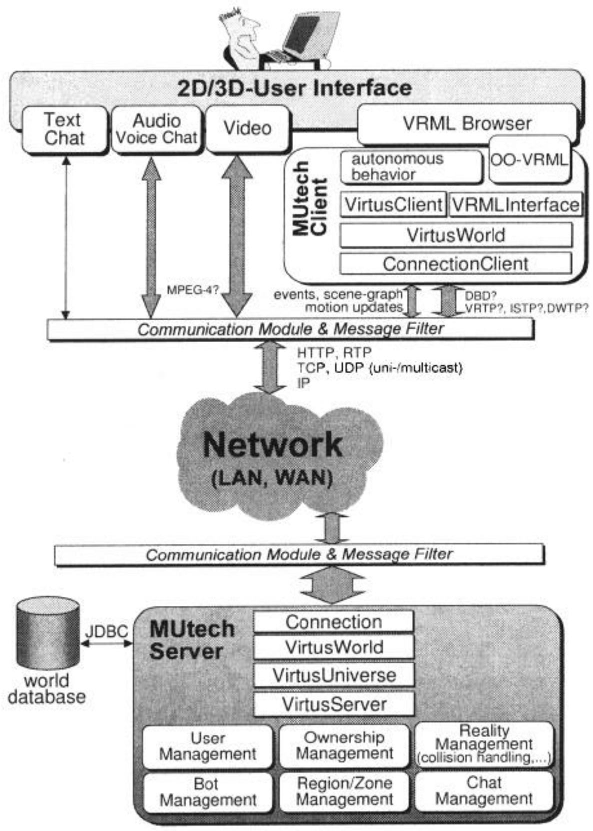


Figure 4: Data replication: world descriptions are loaded from a centralized world database, replicated and updated locally.



4 ARCHITECTURE OF VIRTUS

Our interest in distributed multi-user environment was stimulated by a general interest in VRML and then by its practical application in architectural education. We outlined VIRTUS, our VRML-based multi-user system and finally implemented a first prototype. Our approach was guided by the following requirements:

Openness: the system should be open to as many hardware and software platforms as possible. It should not be designed for any specific application area.

Consistency: all distributed users in the same world are supposed to have the same view at the world. Manipulation by any users should have immediate effect on the world seen by others.

Dynamics: users can enter and leave the virtual environment at any moment. All enter and leave events are distributed to all instances. New users have an up-to-date view of the world.

Persistence: user manipulations within the world can have effects on the world after the user has left the world and disconnected from the system. The current state of the world can be saved at any moment.

Partitioning: complex worlds should be partitioned to minimize the amount of necessary communication.

Extensibility: the system should be implemented in a way that allows for easy extensibility concerning new communication protocols, new object behaviors and new functionality like inter-object collision handling.

Database-Interface: the system should have an interface to a database of objects, avatars and behaviors.

Complex behaviors: object behavior should not be limited to simple movements. A database of complex behaviors for objects like doors, windows, switches, and any kind of devices, 3D widgets and avatars is required.

4.1 Communication Architecture

We chose the client/server model to distribute functionality for the following reasons: the software structure is less complex and easier to maintain. Collision handling, user and ownership management, and the guarantee of consistency generate less problems. The server may act a message filter and keep unnecessary communication to a minimum. As a client/server system VIRTUS can be extended to a hybrid or multicast system with less effort than pure peer-to-peer systems. Since there is direct communication only between clients and the server, no modifications can be missed by the server. VIRTUS employs centralized data management with data replication on demand. The server knows all worlds in the environment and all objects in all these worlds, handles ownership and guarantees consistency. It is notified of changes in any of the worlds and supports persistency of object states after client termination. Late-coming users immediately have an up-to-date view at the environment.
The server manages a database of worlds (VirtusWorlds $W_1,...,W_n$) comprising the universe (VirtusUniverse) and organizes the communication to the clients. Each client knows at most one VirtusWorld at any moment. When a client switches from one world to another, it receives all necessary information about the new world from the server. Centralized data management does not limit the creation of objects to the server side. Clients may also create new objects and insert them to the scene. They are registered by the server and presented to other clients. A special case of necessary client side object creation is the insertion of *avatars*.

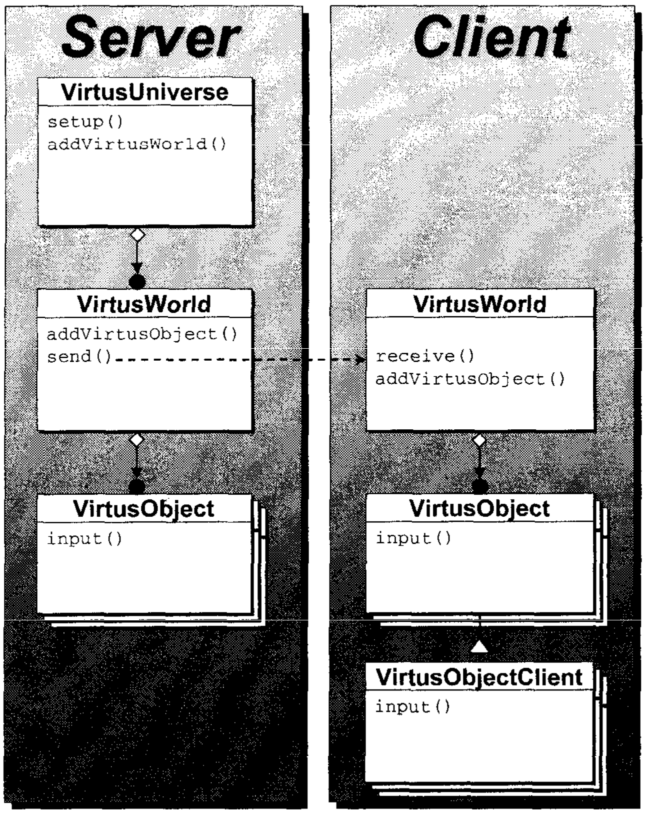


Figure 5: Data objects and their instatiation.

4.2 VirtusObject

Any VirtusWorld is composed of VirtusObjects. Each object is described by a triple *(shape, state, behavior, state)*. The *shape* of an object covers all visual and acoustic characteristics: geometry, appearance, color, texture, sound. All of these may be represented at different levels of detail (LODs). *state* describes the current values of the shape, its relation to the environment (position, orientation, weight, etc.) and some internal state variables. *behavior* describes all actions that may influence the object's shape or state. VirtusObjects may be simple or complex. Complex objects should be structured into a hierarchy of container and child objects.
VirtusObjects contain the following information: A unique *object identifier* assigned by the server. Object *relations* describing parent-child relationships between objects and owner information. Object *transformation* containing position, orientation, size, center, and bounding volume information for efficient collision handling. Object *behavior*: objects fall in several categories:

- *Passive* objects do not create any events.
- *Active* objects create events.
- *Static* objects do not react to external events.
- *Dynamic* objects react to external events.

Two values describe the event sending and receiving behavior of objects. Additionally a *standard behavior* as reaction to specific user actions may be defined for objects. This behavior is inherited by child objects. *Object descriptions* contain the visually and acoustic representation, behavior-defining scripts, and textual annotations. More MUtech-specific attributes (locks, messages, nicknames) are likely to be added in the near future.
The functionality of an object is encapsulated in the Java-class `VirtusObject`. Table 2 gives an overview of the current set of variables in this class. All variables are set and read via methods and most of them have default values set at instantiation time. The subclass `VirtusObjectClient`. specifies additional variables and methods necessary for rendering on the client side.

variable name	type	description
	Object Identification	
`id`	`Integer`	unique identifier
`objectName`	`String`	name of the object
	Object Relations	
`owner`	`Integer`	identification for owner
`objectParent`	`VirtusObject`	reference to parent object
`objectChild`	`VirtusObject[]`	reference list of child objects
`transferable`	`Boolean`	permission of owner change
`persistent`	`Boolean`	indicator for persistence state
	Object Transformation	
`translation`	`Float[]`	values for translation
`rotation`	`Float[]`	values for rotation
`center`	`Float[]`	coordinates of rotation center
`scale`	`Float[]`	scaling values
`scaleOrientation`	`Float[]`	scaling orientation
`bboxCenter`	`Float[]`	coordinates of bounding box center
`bboxSize`	`Float[]`	dimensions of bounding box
	Object Behavior	
`sending`	`Boolean`	indicator for event sending
`receiving`	`Boolean`	indicator for event reception
`movable`	`Boolean`	indicator for object translation
`turnable`	`Boolean`	indicator for object rotation
`scalable`	`Boolean`	indicator for object scaling
	Object Description	
`textdescription`	`String[]`	textual object description
`vrmldescription`	`String[]`	geometric object description
`scriptdescription`	`String[]`	behavioral description

Table 2: Variables of `VirtusObject`

4.3 VirtusWorld

A VirtusWorld contains essentially a list of VirtusObjects. Each VirtusObject may be a member of one VirtusWorld. Table 3 illustrates the variables in class `VirtusWorld` and Table 4 lists the methods operating on these variables.

variable name	type	description
	World Identification	
`id`	`Integer`	unique world identifier
`worldName`	`String`	name of the world
	Container	
`VirtusWorld`	`VirtusObject[]`	list of VirtusObject instances

Table 3: Variables of class `VirtusWorld`

method name	description
	World Management
`getId()`	returns unique world identifier
`setName()`	sets or changes the name of the world
`getName()`	returns the name of the world
`getSize()`	returns number of objects in the world
	Object Management
`addVirtusObject()`	adds an object to the world
`removeVirtusObject()`	deletes an object from the world
`getVirtusObject()`	returns reference to the object of the specified id
	World Transmission
`send()`	send contents of a world
`receive()`	receives contents of the world

Table 4: Methods of Class `VirtusWorld`

4.4 VirtusUniverse

A VirtusUniverse contains a list of several VirtusWorlds and represents the complete virtual environment. An example: VirtusWorlds may represent individual rooms of a building, the VirtusUniverse represents the building. Users may walk from one VirtusWorld to the next by leaving rooms through doors.

We omit a detailed description of variables and methods for class `VirtusUniverse`. The class diagram in Figure 4 illustrates the creation process of the virtual universe and the dependencies between the classes of the data objects. Method `setup()` initializes the universe, reads world descriptions from a database, creates a VirtusObject for each object found in the world. `addVirtusObject()` adds the object to the list `VirtusWorld`.

`addVirtusWorld()` inserts the world into VirtusUniverse as soon as finished loading all objects for the world from the database. The identifiers `worldId` for the VirtusWorlds and `objectId` for the VirtusObjects are generated by the server and are guaranteed to be unique. `send()` transmits a VirtusWorld from server to client where `receive()` reads the description from the network connection. VirtusWorlds can be transmitted vice versa from client to server.

4.5 The Server Side

The server provides each client with its VirtusWorld and delivers all updates within these worlds. It keeps all the necessary state information about every connected client. Other duties of the server include the initialization during the system start, saving operations, the guarantee of world-wide consistency, and conflict avoidance among objects and users by granting or refusing ownership. The server is composed of two threads: VirtusServer and Connection.

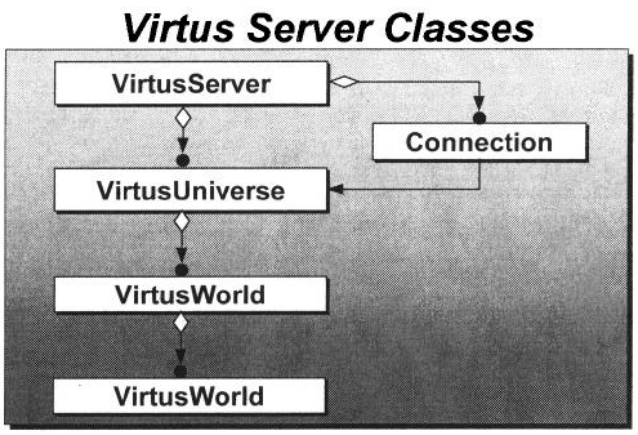


Figure 6: Server-side class diagram.

4.5.1 Thread Virtus Server

VirtusServer initializes the system by creating an instance of class VirtusUniverse and by listening for incoming client requests on a well-defined port. As soon as a client asks for participation a Connection thread is started with an id number, a port number, and a reference to the VirtusUniverse. Since this Connection thread is responsible for the communication between client and server it is assigned an id, a socket, and a reference to the universe. The VirtusServer thread also realizes the saving operation for persistent objects to the database, either at termination or on user demand.

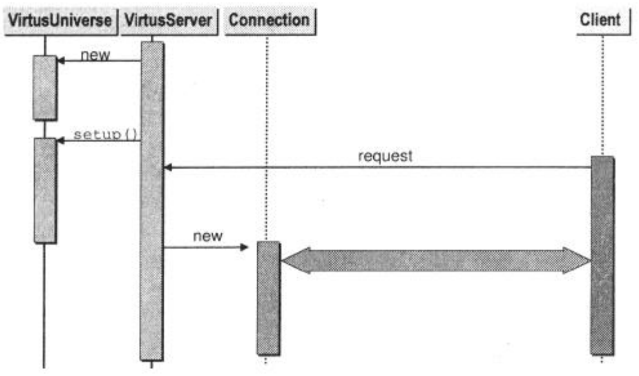


Figure 7: Communication and interaction during start of VirtusServer

4.5.2 Thread Connection

For each client there is a dedicated Connection thread that handles the communication between client and server. It provides the client with information about its world and takes events from the client to the other Connection thread. The lifetime of a Connection thread expires with the termination of the client-server connection. After receiving an acknowledgement of successful transmission of the requested world description the Connection thread creates a visual representation for the user: an avatar is inserted to the VirtusWorld and all clients interested in this world are notified about the existence of a new user. This completes the initialization phase.

The Connection thread now listens for messages coming from the client or from other Connection threads. Potential incoming messages from the client side are:

- User interactions with objects: object creation, deletion, transformation, pushing buttons, using SensorNodes, etc. These actions issue events that are sent to the Connection thread and forwarded to all Connection threads managing the same world.
- User switches from one world to another: The new world is transmitted to the client, the avatar is deleted in one world and a new avatar is created in the new world. All Connection threads managing the old and the new world are notified.
- The client requests ownership for a specific VirtusObject. If the server currently owns the VirtusObject and if its ownership is transferable, then the Connection thread changes ownership and returns acknowledgment. In other cases the request for ownership is refused. Not all objects may change ownership: avatars are always owned by the server.
- Ownership is returned by the client. Ownership is intended to be bound to a user only for a limited period of time. Having completed their manipulations users return ownership to the server. Connection resets ownership and informs other Connection threads that may have claimed ownership in the meantime.
- A client terminates: Connection deletes the avatar, informs other Connection threads, returns all ownership from the terminated client to the server, informs the VirtusServer thread about the termination of the client and terminates itself.

4.6 The Client Side

The client side is designed as an applet and acts as the interface between the user and the virtual environment. It is composed of six elements. Data objects are managed by three classes: `VirtusWorld`,`VirtusObject`, `VirtusObjectClient`. `VirtusClient` implements the user interface by processing user input and by displaying information. `ConnectionClient` manages the communication with the server. `VRMLInterface` is the interface between `VirtusClient` and the VRML-browser.

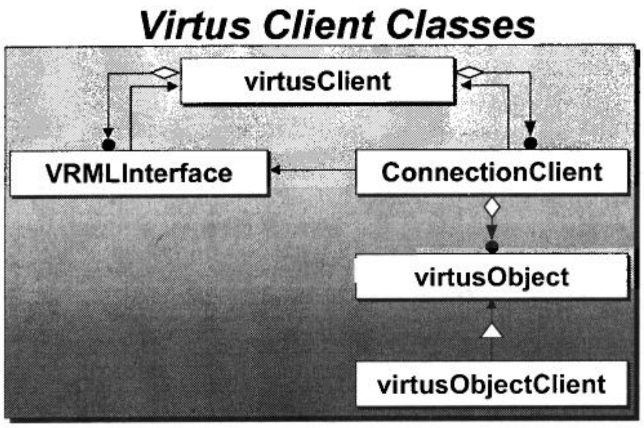
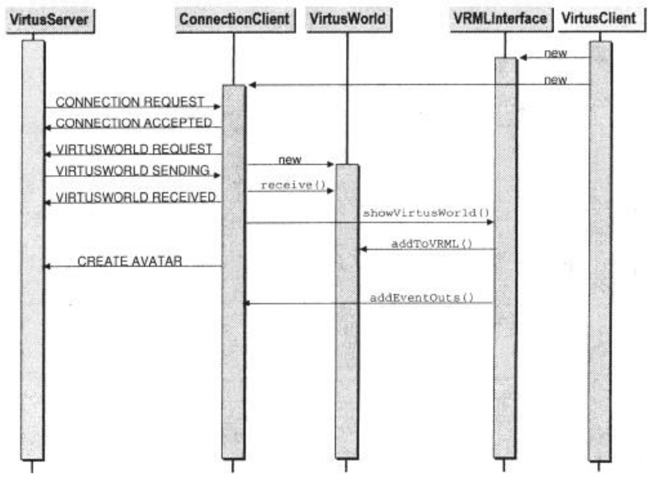


Figure 8: Class diagram of the client side.

4.6.1 Thread VirtusClient

This Thread is the main thread on the client side. It offers the graphical user interface for input and output of system. and initializes `ConnectionClient` and `VRMLInterface`.



4.6.2 Thread ConnectionClient

ConnectionClient handles all the communication between client and server. At initialization time it receives a socket and port number, establishes a connection to the server and requests a user selectable scene description (a VirtusWorld) from the server. VirtusWorld's method `receive()` reads the data sent by the server and assigns them via `input()` to VirtusObjects. The VirtusWorld is propagated to VRMLInterface and rendered by the browser. The request for an avatar and the notification of the current viewer position complete the initialization phase.

From now on the ConnectionClient waits for messages coming from the server and for events issued either by the VRMLInterface or the user. Here are some examples of possible actions:

- A user enters a new world (or region in the universe): ConnectionClient receives the world description and forwards it to VRMLInterface.
- VRMLInterface requests ownership for a VirtusObject. ConnectionClient forwards this request to the server and forwards acknowledgment or refusal to the interface.
- VRMLInterface returns ownership of a VirtusObject. The server is notified and may grant ownership to different users.
- Viewer position has changed: ConnectionClient forwards an event with coordinates of the new position to the server.
- VRMLInterface returns a general event. An event with event type, VirtusObject identifier, event parameters and timestamp is sent to the server.
- User requests the end of the session: ConnectionClient sends this requests to the server, notifies the VRMLInterface, deletes the local VirtusWorld and terminates.

4.6.3 VRMLInterface

VRMLInterface acts as an interface between VirtusClient and ConnectionClient on one side and the VRML-browser on the other. It has the capability to manipulate the scene graph in the browser and to create and read events in the scene graph. Method `showVirtusObject()` adds an object to the scene, `showVirtusWorld()` adds a complete world to the scene graph. `removeVirtusObject()` and `removeVirtusWorld()` delete components from the scene. Events in a world may be traced by adding them to a list of interesting events `addEventOuts()` and assigning callback functions. Events created by the user or some remote client may be sent to the VirtusObjects via `setEvent()`.

4.6.4 VIRTUS Communication Protocol

The decision for an appropriate network protocol useful for scalable distributed environments is still an open issue. The communication between the two threads Connection (server side) and ConnectionClient (client side) is ruled by a very simple application layer protocol. Figure 9 presents the synchronization between the thread during VirtusWorld transmissions. Table 5 lists the PDUs currently specified by the protocol. *EVENTTYPE* is a generic name representing a variety of event types: position changes issue TRANSLATION events, a proximity node issues PROXIMITYSENSOR events, etc. Since the number of parameters varies between event types, it is represented as $x_1,\ldots,x_n$. Each event comes with a timestamp.

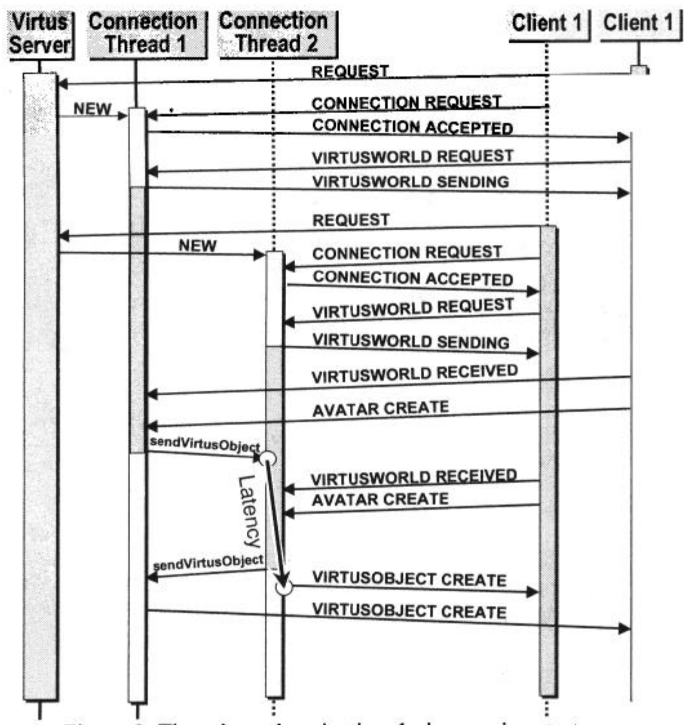


Figure 9: Thread synchronization during session start.

Sequence	Parameters	Direction
Connection Establishment		
CONNECTION REQUEST	-	S ← C
CONNECTION ACCEPTED	-	S → C
CONNECTION REFUSED	-	S → C
Initialization Phase		
VIRTUSWORLD REQUEST	VirtusWorld ID	S ← C
VIRTUSWORLD NOT AVAILABLE	-	S → C
VIRTUSWORLD SENDING	VirtusWorld ID	S ↔ C
VIRTUSWORLD RECEIVED	-	S ↔ C
AVATAR CREATE	-	S ← C
TRANSLATION	(x y z)	
ROTATION	(x y z α)	
Working Phase		
OWNERSHIP REQUEST	VirtusObject ID	S ← C
OWNERSHIP ACCEPTED	VirtusObject ID	S → C
OWNERSHIP REFUSED	VirtusObject ID	S → C
OWNERSHIP RELEASE	VirtusObject ID	S ← C
VIRTUSOBJECT CREATE	VirtusObject	S ↔ C
VIRTUSOBJECT DELETE	VirtusObject ID	S ↔ C
EVENT	VirtusObject ID	S ↔ C
EVENTTYPE	$(x_1, \ldots, x_n)$	
TIMESTAMP	*time*	
Disconnection Phase		
CONNECTION END	-	S ↔ C
CONNECTION CLOSED	-	S ↔ C

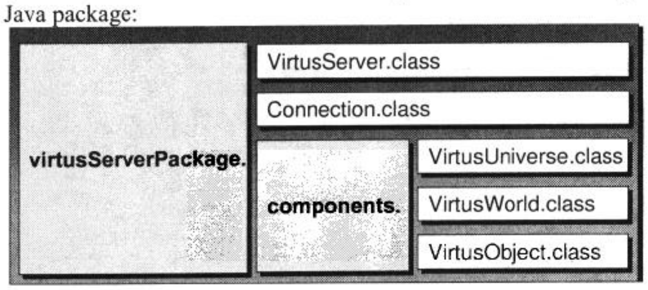
Table 5: VIRTUS' simple application layer protocol

5 IMPLEMENTATION OF VIRTUS

VIRTUS has been implemented entirely with VRML97, Java 1.0, Java1.1, and the External Authoring Interface EAI.

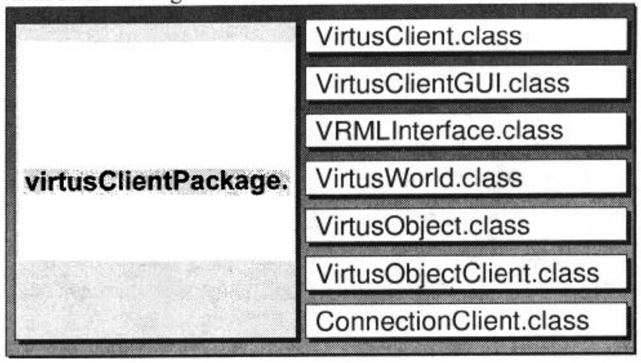
5.1 The Server Side

The server code is responsible for reading world descriptions from a database of predefined worlds, for the initialization of a VirtusUniverse, for the management of clients, and for the creation of avatars. The server-side functionality is bundled in a single Java package:



5.2 The Client Side

The client side is implemented in Java-package virtusClientPackage which contains seven classes:



`virtusClient` and `virtusClientGUI` implement the user interface. `ConnectionClient` control all communication between client and server. `VRMLInterface` communicates between Java applet and VRML-plugin. A HTML-File VirtusClient.html contains tags to load and embed several user interface components:

- A Java applet with AWT widgets to control and monitor the communication, request worlds and view status and log messages during the session. The applet contacts the VRML-Browser via thread VRMLInterface.
- The VRML-Plugin is initialized with a simple VRML-File "root.wrl" containing essentially a named entry point (ROOT) for the External Authoring Interface
- The bottom area of the user interface is reserved for future integration of additional Applets and controls (text, audio chat, video)

The thread VRMLInterface is responsible for three tasks: it creates 2D and 3D textual information (status and error messages) in the Applet-Window and in the VRML scene. It manages user coordinates during navigation and informs the server-side Connection about position changes. Finally it manages all instances of VirtusObjects and triggers the rendering of their position, orientation and appearance changes. The necessary updates in the VRML scene graph, insertion and deletion of VirtusObjects are accomplished via standard EAI methods (createVrmlFromString, getEventIn) and documented in the Java applet's log area. User position updates VRMLInterface implements an EventOutObserver to catch user interaction in the scene.
The activation of any Touch-, Plane-, Cylinder-, Sphere-, or ProximitySensor in the world is recognized by the EventOutObserver and initiates the request for ownership from the server. If it is granted

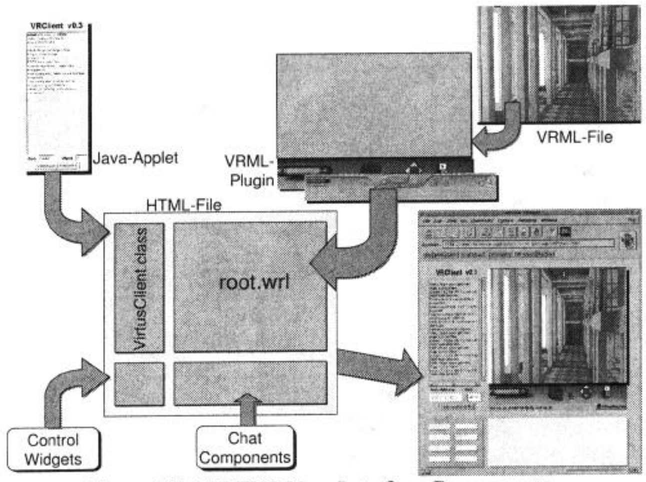


Figure 10: VIRTUS User Interface Components

6 CASE STUDY: ARC ITECTURAL CONSTRUCTION

In 1995 we implemented a system for 3D virtual walkthroughs which was also capable of simulating the building construction in 4D, with the additional parameter of time. This system was based on SuperScape. As also stated by [9] VRML offers today similar features to distribute an architect's design intentions to students, clients, contractors, and fabricators. Multi-user environments additionally offer the ability to meet many times in a virtual collaborative building to review progress, plan and discuss changes.

The benefits of using 3D online techniques in architecture are obvious: there is no need to produce and distribute numerous views on paper and the models may much easier be checked for layout conflicts by visual inspection than large sets of 2D drawings. VRML supports the modeling in several levels of detail which is a traditional concept in architecture. Furthermore, hyperlinks on the 3D components may immediately reveal construction specifications, e.g. in HTML.
However, several deficiencies of the current VRML standard and tools will prohibit the adoption of VRML as a primary medium for design by the AEC industry in the near future:
VRML modeling software does not provide the sophisticated modeling techniques used in professional CAD systems. VRML-export filters are not suitable because they generate polygonal approximations of models (large IndexedFaceSets). VRML generally provides rather qualitative data for rendering rather than quantitative and analytic data for engineering purposes and construction processes. Methods to query sizes and object distances are required as well as new spatial metaphors for the display of analytic data.

7 EXPERIMENTAL RESULTS AND CONCLUSION

The VIRTUS system has been implemented in first prototype mainly as a proof-of-concept within 12 weeks by a single programmer. Since it is completely written in Java, it runs everywhere. The client side has been successfully tested on IRIX 6.x and Windows NT 4.0, with Netscape Navigator and Microsoft Internet Explorer and CosmoPlayer 1.0.2, 1.1, and 2.1. We chose to install installed the VirtusServer on a Sun SPARC10 running Solaris 2.6 and working at the same time as our HTTP- and FTP-server. The clients run primarily on SGI O2 and Indigo^2 running IRIX6.x and PCs running Windows NT4.0 and Linux. Most of the machines are located in the server's 10Mbps-Ethernet segment, a PC is connected via ISDN. The clients use Netscape Navigator 3.1S, 4.05 and 4.5b, Microsoft Internet Explorer, CosmoPlayer 1.0.2 and 1.1 under IRIX, and CosmoPlayer 2.1 under NT.
We ran the system with up to ten simultaneous users, more than typically would participate in our architectural design scenario. Experimental results indicate only little rendering slowdown when users are merely navigating through the scene. Large amounts of sensor-interaction however swamp the communication channels and cause jumpy motion. We assume the reason for this in the quite primitive communication protocol which uses exclusively TCP in its current version and performs no filtering or controlled sampling of events. Quantitative measurements of network load under different conditions will follow.
As the use of UDP does not require dedicated sockets, setup and take down cost or retransmission in case of failure in busy networks, it will be used for motion updates and events. The transfer of more functionality and intelligence to the client side is a promising approach to improve the system.
Consistency in the VirtusWorlds in guaranteed by ownership management in the server. The system is dynamic: users may enter and leave the world and any moment, may create new objects, which are added as persistent components to the world. The current state of the environment can be saved and reloaded. The environment (universe) is partitioned (manually) into separate worlds. The modular design of VIRTUS provides extensibility with little effort.

8 CURRENT AND FUTURE WORK

Currently we are working on a database for more complex, sophisticated behaviors, object-oriented paradigms[24] and a set of VRML widgets with callback mechanisms. This will include dynamics, the use of constraints [12] for assembly (Snap & Move concepts). We are working on the inter-object collision handling, on server-side and on client side and we are investigated methods for automatic partitioning of complex environments.

ACKNOWLEDGMENT

We wish to send our thanks to Andreas Steinbuch, who conceived and implemented most of the VIRTUS code and contributed to the concept in a very prolific way.

REFERENCES

[1] Yoshiaki Araki. VSPLUS: A High-Level Multi-user Extensions Library For Interactive VRML Worlds. *Proceedings of the VRML '98 Symposium*, ACM.

[2] J.W. Barrus. R.C. Waters and D.B. Anderson. Locales and Beacons: Efficient and Precise Support for Large Multi-User Environments", IEEE Virtual Reality Annual International Symposium, March 1996, pages 204-213, IEEE Computer Society Press

[3] Cyberhub, Blaxxun Interactive. [www] http://www.blaxxun.com

[4] W. Bricken and G. Coco. The VEOS Project. *Presence*,3(2):111-129, Spring 1994.

[5] Wolfgang Broll. Populating the Internet: Supporting Multiple Users and Shared Applications with VRML. *Proceedings of the VRML '97 Symposium*, pages 33-40, ACM, Feb. 1997.

[6] Wolfgang Broll. DWTP – An Internet Protocol for Shared Virtual Environments. *Proceedings of the VRML '98 Symposium*, ACM, 1998.

[7] D. Brutzman, M. Zyda, K. Watson and M. Macedonia. Virtual Reality Transfer Protocol (VRTP) Design Rationale. In *Proceedings Sixth IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 179-186. MIT, Cambridge, Massachusetts, IEEE Computer Society Press, June 1997.

[8] D. Brutzman. Graphics Internetworking: Bottlenecks and Breakthroughs. In Clark Dodsworth ed.: *Digital Illusions*, pages 61-97, Addison Wesley, 1997.

[9] D. Campbell. VRML In Architectural Construction Documents: A Case Study.

[10] C. Carlsson and O. Hagsand. DIVE – a Multi-user Virtual Reality System. *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 394-400, September 1993

[11] S. Chenney, J. Ichnowski, D. Forsyth. Efficient Dynamics Modeling for VRML and Java. *Proceedings of the VRML '98 Symposium*, ACM, 1998.

[12] Stephan Diehl. Extending VRML by One-Way Equational Constraints. *Workshop on Constraint Reasoning on the Internet*, 1997

[13] C. Greenhalgh and S. Benford. MASSIVE: A Distributed Virtual Reality System Incorporating Spatial Trading. *Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 27-34, Mai 1995.

[14] C. Greenhalgh and S. Benford. MASSIVE: A Distributed Virtual Reality System for Tele-Conferencing. *ACM Transactions on Computer Human Interface*, 2(3):239-261, September 1995.

[15] Olof Hagsand, Rodger Lea and Mårten Stenius. Using Spatial Techniques to Decrease Message Passing in a Distributed VE System, *Proceedings of the VRML '97 Symposium*, pages 7-15, ACM, 1997

[16] R. Hofer and M. Loper. DIS today. *Proceedings of the IEEE*, 83(8):1124-1137, August 1995

[17] Y. Honda, Y. Mitra, B. Rockwell, B. Roehl. Living Worlds, Draft 2.0, April 13 1997, [www] http://www.living-worlds.com/draft_2/index.htm

[18] T.C. Hudson, M.C. Lin, J. Cohen, S. Gottschalk, D. Manocha. V-COLLIDE: Accelerated Collision Detection for VRML. *Proceedings of the VRML '97 Symposium*, pages 117-123, ACM, Feb. 1997.

[19] R. Kazman. Making Waves: On the Design of Architectures for Low-end Distributed Virtual Environments. *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 443-449, September 1993.

[20] R. Lea, Y. Honda, K. Matsuda and S. Matsuda. Community Place: Architecture and Performance. *Proceedings of the VRML '97 Symposium*, pages 41-50, ACM, Feb. 1997.

[21] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zeswitz. NPSNET: A Network Software Architecture for Large-Scale Virtual Environments. *Presence*,3(4):256-287, Fall 1994.

[22] J. Mandeville, T. Furness, M. Kawahata, D. Campbell, P. Danset, A. Dahl, J. Davidson. K. Kandie, and P. Schwartz. Greenspace: Creating a Distributed Virtual Environment for Global Application. *IEEE Proceddings of the Networked Reality Workshop*, Oktober 1995.

[23] D. Miller and J. Thorpe. SimNet: The Advent of Simulator Networking. *Proceedings of the IEEE*, 83(8):1114-1123, August 1995.

[24] Sungwoo Park, Taissok Han. Object-Oriented VRML for Multi-user Environments. *Proceedings of the VRML '97 Symposium*, pages 25-32, ACM, 1997

[25] C. Shaw and M. Green. The MR Toolkit peers package and Experiment. *Proceedings of IEEE Virtual Reality Annual International Symposium*, pages 463-469, September 1993.

[26] C. Shaw, M. Green, J. Liang, and Y. Sun. Decoupled Simulation in Virtual Realtiy with the MR Toolkit. *ACM Transactions on Information Systems*, 11(3):287-317, July 1993.

[27] D. Snowdown and A. West. AVIARY: Design Issues for Future Large Scale Virtual Environments. *Presence*, 3(4): 288-308, Fall 1994.

[28] M.R. Stytz. Distributed Virtual Environments. IEEE CG&A, May 1996, pages 19-31

[29] H. Sugano, K. Otani, H. Ueda, S. Hiraiwa, S. Endo, Y. Kohda. SpaceFusion: A Multi-Server Architecture For Shared Virtual Environments. *Proceedings of the VRML '97 Symposium*, pages 51-58, ACM, Feb. 1997.

[30] Swedish Institute Computer Science: DCE Group Research Projects. [www] http://sics.se/dce/group-research/group-research.html

[31] R. Waters, D. Andersen, and D. Schwenke. Design of the Interactive Sharing Transfer Protocol. In *Proceedings Sixth IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 140-147. MIT, Cambridge, Massachusetts, IEEE Computer Society Press, June 1997.

[32] M. Zyda and J. Sheehan. Modeling and Simulation: Linking Entertainment and Defense. National Academy Press, 1997.