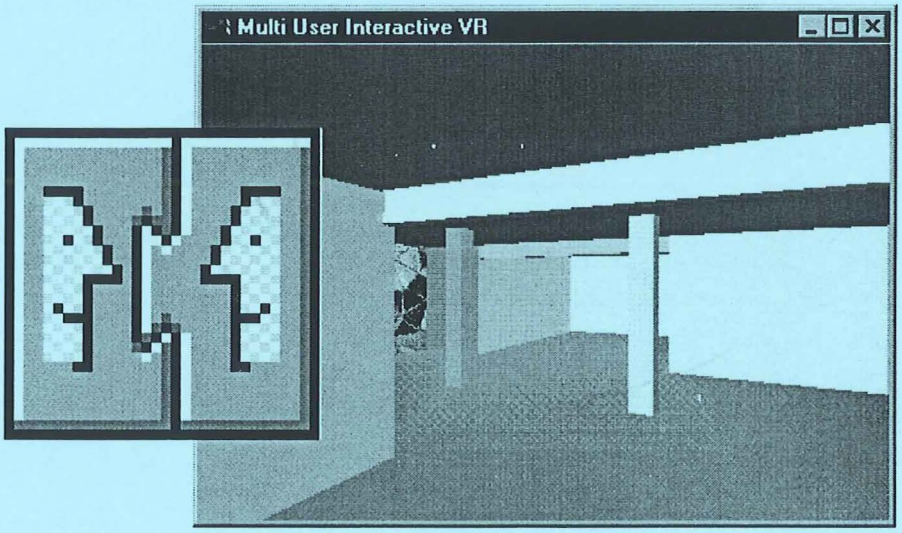
Tekniska Högskolan i Lund
Avdelningen för Bärande Konstruktioner

Lund Institute of Technology
Department of Structural Engineering

Lund 1996

Rapport TVBK-5081



Low cost distributed VR
LCD-VR

Jonas Lindemann

FÖRHANDSKOPIA

Rapport TVBK-5081
ISSN 0349-4969
ISRN: LUTVDG/TVBK--5081-SE

Low cost distributed VR
LCD-VR

EXAMENSARBETE TVBK-5081

Supervisor: Per Christiansson

LUND september 1996

Jonas Lindemann

Table of content

1 PREFACE

This report is the result of a master thesis which has been done at the Department of Structural Engineering, KBS-Medialab, Lund Institute of Technology. It has been created by Jonas Lindemann under the supervision of Assoc. Prof. Per Christiansson.

I also want to than Anders Follin, Tech. Lic. Of KBS-Medialab for his help.

Lund, September 1996

Jonas Lindemann

2 ABSTRACT

This report describes how a low cost distributed VR system can be created. It describes requirements and possibilities for a low cost VR environment. The use of the system is demonstrated as well as the system design process. Underlying IT-tools are analyzed and the client-server system components are described.

The projects goals were reached; to connect people regardless of time and space, collaboration in Virtual Reality using low cost components, respond directly to users with direct feedback, easy to use interface, representing users/visitor as artifacts in the VR.

3 INTRODUCTION

This project was created 1995-96 in cooperation with KBS-Medialab in the effort to see if it was possible to create a low cost distributed VR system.

The report starts with a short system description in which a short scenario is described step by step.

In chapter "Working method" the development process is described. Problems discovered along the way are covered. Milestones in the development are described.

The chapter "Conceptual model" covers the application, system and user models and how they relate to each other.

The "Data model" chapter describes the flow of data between different parts of the system. This chapter is a technical description and is not necessary for the understanding the system behavior.

In the chapter "Implementation model" the implementation environment is described. As well as development of the VR-server and user interface is also covered.

The "Discussion" chapter covers possible extensions and the potential market for a low cost distributed VR system.

4 BACKGROUND

The global net is growing and so is the services on the net (Internet). This creates new ways to communicate and cooperate in the construction process. You want to be less dependent of time and space and still be able to work together in a project.

The quality of the final product increases if it is easy to work on a common model over the net without time consuming travels. You can visualize problems and solutions in a three dimensional model and reduce misunderstandings. Client needs can be visualized in an early point in the development.

The goal of this work was to create a system that enables two people to work independent of time and space on the same geometric model on a part/object level. The users should be able to move objects, change colors and give the building objects different properties during visualization. The system should be able to run on an ordinary personal computer with moderate hardware demands.

If the system should be able to run on a normal PC the choice of rendering engine is of great importance. The following criteria was considered when the rendering engine was chosen:

1. High frame rate also with a complex model.
2. Goraud shading and texture mapping as a minimum.
3. A good programming interface API (Application Programming Interface).
4. Platform independence.

There are not many multiuser systems for PC system today. The systems that exists are often run on large and expensive workstations. One such example Silicon Graphics InPerson (Silicon Graphics, 1996). In this system you can work together independent of time and space and view 3D objects at the same time. Silicon Graphics also has larger systems for virtual reality that can be used on the internet. SICS (Swedish Institute of Computer Science) has developed an electronic conferencing system build around a virtual conference room. Their approach is to completely translate the real conference room into a virtual reality with 3D user interface. This system runs on Unix workstations (Swedish Institute of Computer Science, 1996).

My idea was to use the normal Windows user interface for tasks better implemented in 2D, for example dialog boxes and word processing. Today it is difficult to use the standard programs when you totally immersed in the virtual reality. This system described in this report combines the to worlds.

5 PROJECT GOAL

The goals of the project can be listed as follows:

- Connect people regardless of time and space.
- To demonstrate how collaboration can take place in Virtual Reality, VR, environment using low cost components.
- The system should respond directly to action of the users, giving them direct feedback.
- Create an easy to use intuitive interface.
- The sense of presence by representing the users/visitors as artifacts in the VR model.
- Create tools to collaboratively manipulate the VR model in real time. (Surface manipulation, annotation and editing, navigation controls, ...)

6 SHORT SYSTEM OVERVIEW

This chapter describes in a working example how the system works.

The first thing you meet is the startup screen. Here you can setup the system and the networking parameters.

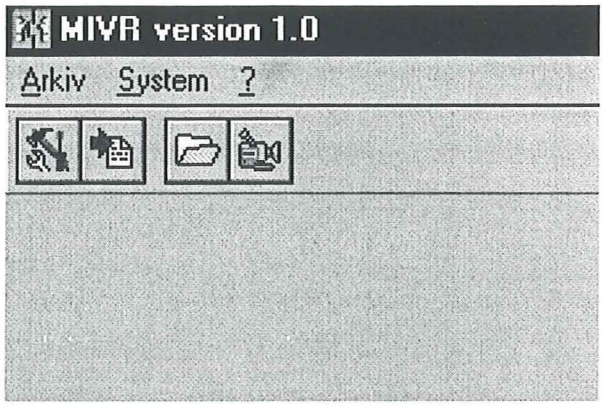


Figure 1 - Blow up of the toolbar and menu in the startup screen

Click on the  button to pick a model. A common dialog box is brought up from which we choose a file to load. When we load a model (for example a model of a house) the system brings up a control panel in the lower part of the screen. The view of the model is shown in the middle of the screen. The lower panel controls all the functions needed in the environment as movement, viewing and network connections.

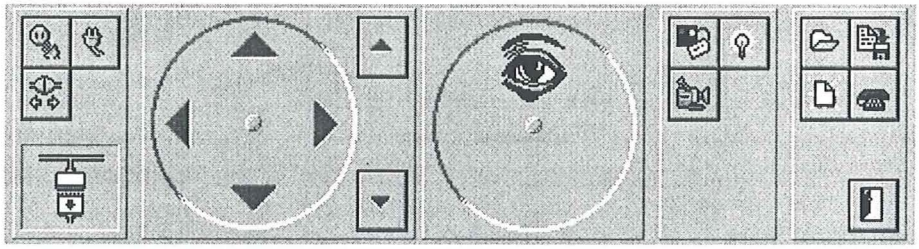


Figure 2 - The control panel

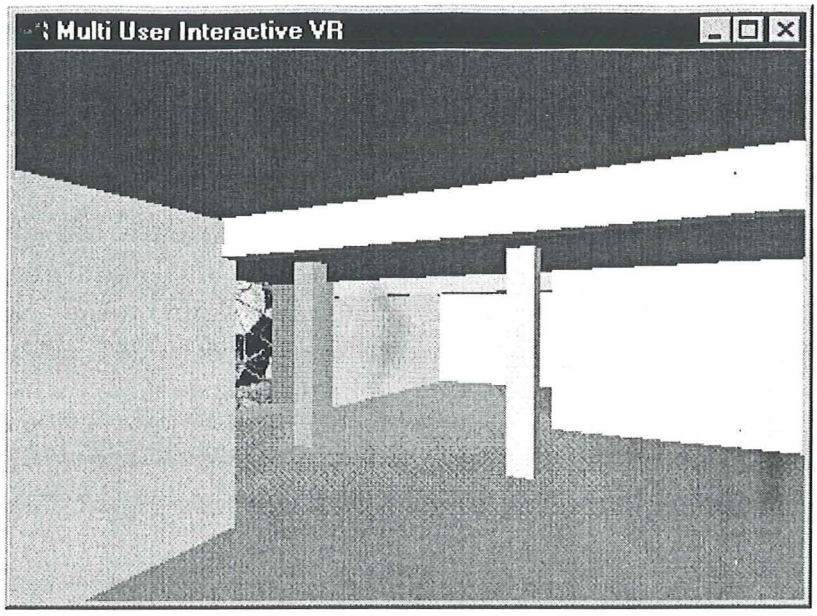
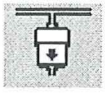


Figure 3 - The view window

Now we have loaded the model and want another user to enter the model thus click the  button which tells the system to wait for connections. This also implies our system is the server. The other user has setup his system with the address to our server. He then clicks on the  button and gets connected to our system.

When the connection is established the other user is shown in the view window and the following symbol is shown in the network part of the control panel:



This indicates that the connection is established.

We can save interesting views before the connection with the other user to quickly move to the interesting points when the connection is established. The saving and recalling of views is done using the view management window. We can also save views when the other user is connected.

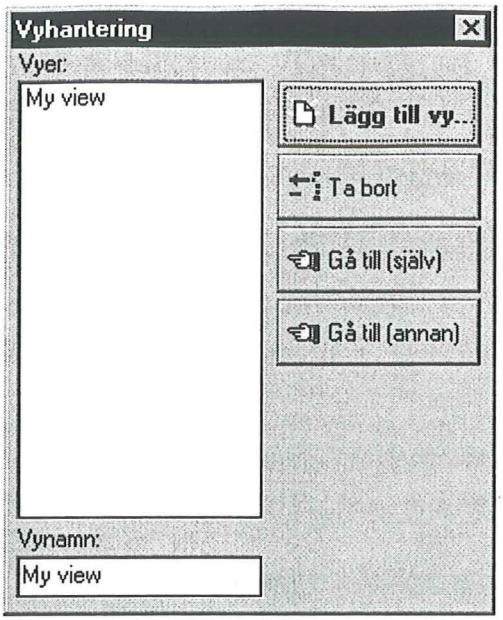


Figure 4 - View management window: (Add view, delete view, move user (self), move user (other)

In the view management window, see figure 4 there are functions for saving the current view, deleting saved views, moving to a certain view, moving the other user to a certain view.

Lets say that we moved the user to the view we are situated in, and we now want to add a comment to a pillar in the house. We click on the pillar in the view window. The pillar is darkened, see figure 5 and the object properties window is shown, see figure 6.

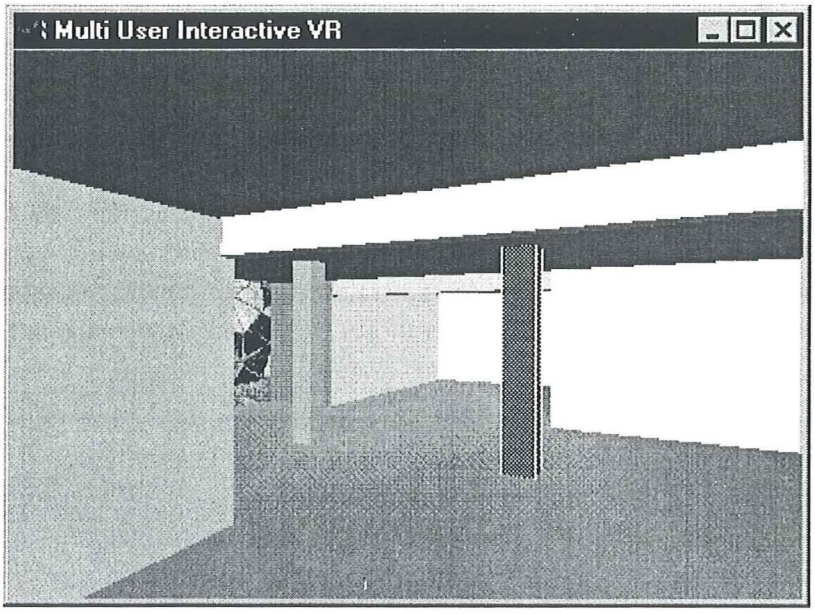


Figure 5 - View window with selected pillar

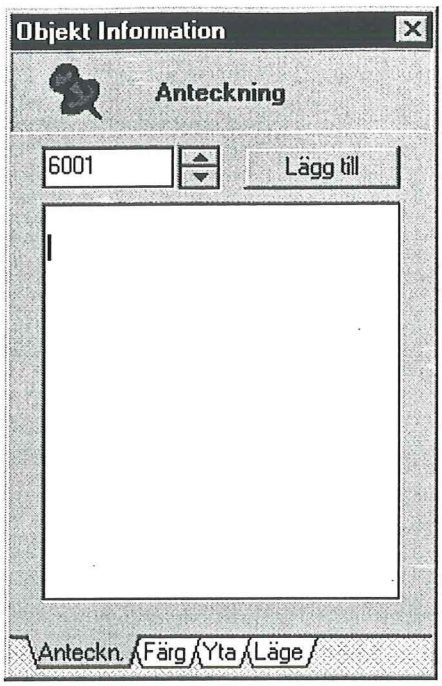


Figure 6 - Object properties window. Tabs for notes, color, surface and position

The object properties window contains a textbox in which we can add a comment to this object. When we are finished we click the **Add** button and the comment is added to the database. Red small cubes are attached to the object to show that it has comments connected to it, see figure 7.

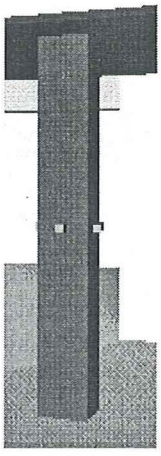


Figure 7 - Object with a comment attached

The color of the object is changed with the color sliders in the **Color** tab, see figure 8 in object properties window. Sliders for red, green and blue are shown in the window. Moving these will change the color of the object. We mix a yellow color to the object.

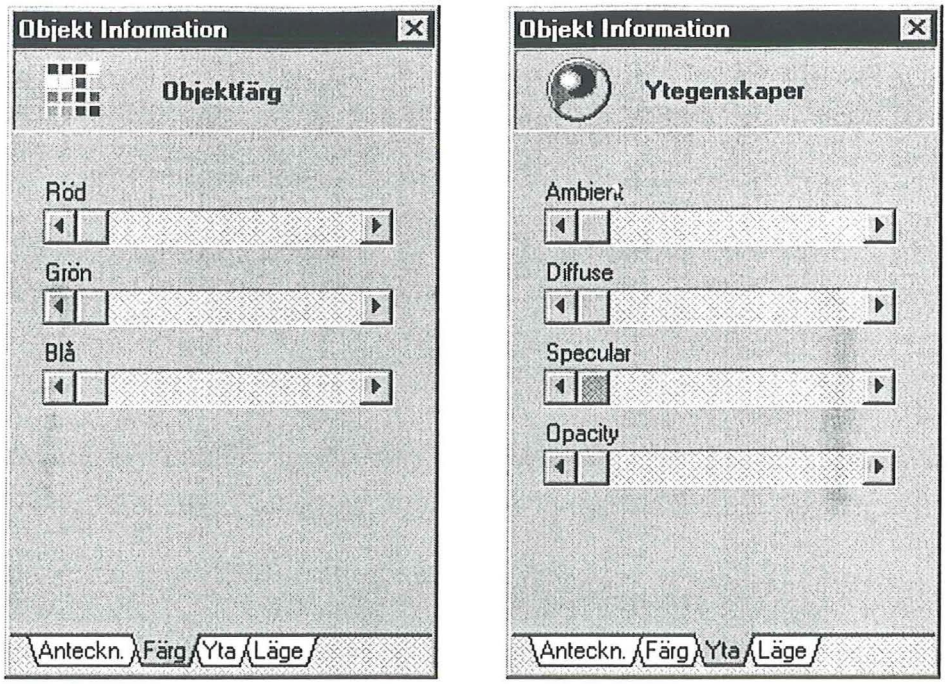


Figure 8 - The color, and surface tabs of the object properties window. Sliders for red, green and blue color components. Sliders for surface properties.

To change the surface properties of the object we click on the **Surface** tab, see figure 8 in the object properties window. Sliders for ambient, diffuse, specular and opacity are shown in the window. Moving these will change the surface appearance of the object. We make the pillar somewhat transparent with the opacity slider, see figure 9.

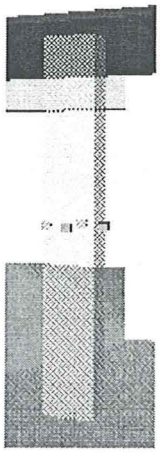


Figure 9 - Transparent object

With the last tab **Position** we can also move the object in the model.

The changes we make in the model also takes place on the screen of the other user. When we click on an object in the model, the properties window is also shown on the other users screen. Every action is duplicated in the client and server system.

A "virtual joystick" manages moving around in the model. The "virtual joystick" is a special user interface control that makes the mouse to a joystick. To use the control you move the mouse over the control and click one time with the left mouse button. This makes the normal mouse pointer disappear and the mouse is locked in the control mode. When the mouse is moved you also move in the model. Figure 10 illustrates the "virtual joystick":

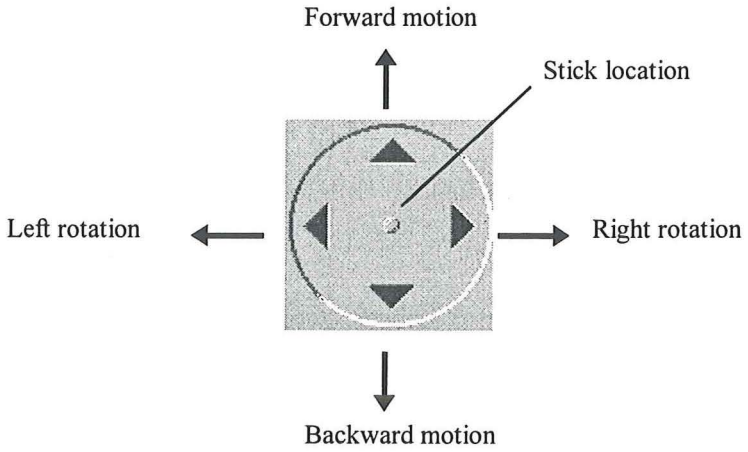


Figure 10 - Description of the virtual joystick

To exit the control we click the left mouse button again. The normal mouse pointer will appear and we can use the program in a normal way.

If we don't have any audio- or video communication channels, the system provides a chat window, see figure 11. To use the chat window click on the  - button. A window divided into two parts is displayed. In this window we can type messages to the other user, or receive messages from the other user.

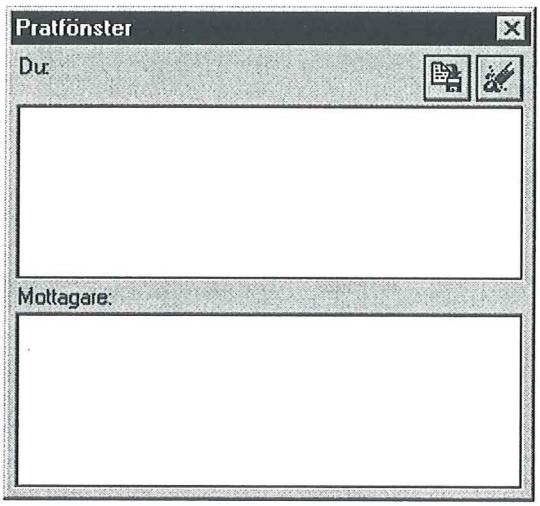


Figure 11 - The chat window

When we want to end the session we click the  button to terminate the connection. The  button saves the modified model, and comments database.

7 WORKING METHOD

The project was started in august 1995 when the Renderware library (Criterion Software, 1995), arrived. The first steps was to get the library working in the WATCOM C++ 10.0a (Watcom International Corp, 1995) environment, compiling some of the example programs, and testing out the structure of the renderware library.

The WATCOM C++ system did not have a visual environment for developing the user interface. I found that the best system for developing a user interface was Borland's Delphi 1.02 (Borland International, Inc, 1995). In the WATCOM environment I encapsulated the renderware system into an "VR-server" application that could respond to messages from a client application, in this case the Delphi application.

The first months were used to build an object structure in C++ around the renderware API. The application thus became less dependent of the renderware engine and also easier to program.

After the development of the object structure was finished a windows message interface was developed for communication with the renderware engine. A message interface has the advantage of queuing the messages and processing them in order. An alternative method is to make the "VR-server" to a DLL (Dynamic Link Library) in Windows. This approach is best when you have multitasking environment where you can create independent threads controlling the rendering or the communications. When you call the DLL library the program flow control goes to the DLL and interrupts the client program. This means for example that communication events can not be handled until the control is back to the client program. This problem occurs only in the Windows 3.x environment where we only have access to cooperative multitasking. The Windows 3.x environment was chosen because is widespread and it is a relatively low cost environment. At the project start Windows 95 or Windows 4.0 was not yet released and Windows NT was not a low cost environment.

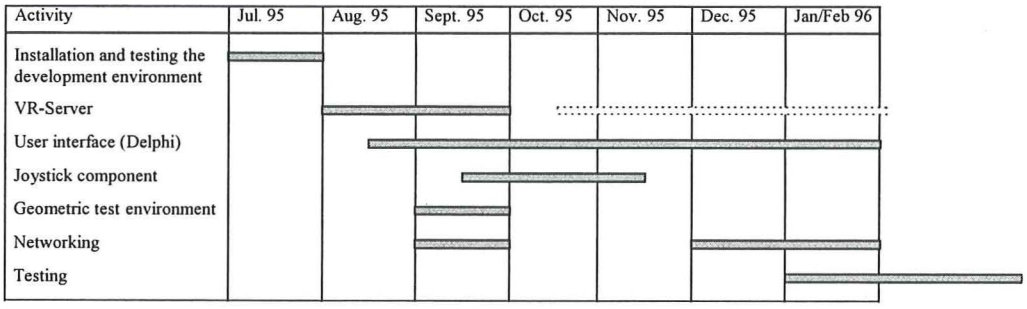


Figure 12 - Development process

When the "VR-server" was finished in a first version the primary steps of Delphi development started. The message interface was implemented as an object structure in Object Pascal. The object structure makes it possible to implement other 3D engines within the system. You just have to rewrite the implementation of the objects in Delphi. Procedures for starting up the server and establishing contact between then "VR-server" and then client application was written. Time out routines were written to handle situations when the "VR-server" won't start because of insufficient resources in the system. The work at the low level architecture of the system is complete.

A test world had to be created to test the environment. AutoCAD was used to create a simple building, with distinguishable parts. The file was exported as a DXF-file and converted to a renderware script file with a special conversion program. The script file was modified, so that every object in the model got a unique tag or handle. To make the system user friendly a user interface for the conversion program was needed. A small delphi application was written to control the parameters to the conversion utility.

The user interface in Delphi was constructed in the next step of the development. It consisted of a main window covering the entire screen, with a menu and a toolbar. When you pressed a button in the toolbar the "VR-server" and a floating control panel was shown. The control panel consisted of a number of buttons for moving around. This design of the control panel was just good enough. It was difficult to move around with the buttons on the toolbar. You constantly had to release the mouse button and move the mouse to another button and hold the button down when you wanted to traverse the model. Another design had to be created.

At the time we were a group working together with the different parts of the system. My brother Mattias Lindemann designed some of the interface for the object properties window, Anders Eriksson a friend of my brothers was looking for communication solutions. Asbjörn Ejsing was engaged in some of the controlpanel designs. I designed the low level parts of the system (renderer and delphi implementations). Due to employment and relocation Anders and Asbjörn couldn't participate very much in later development stages.

Asbjörn and I had discussed the possibility of using a "virtual joystick" that could be used in conjunction with the mouse. I designed a first joystick component in Delphi. This joystick was used by pressing on a "virtual stick" holding the mouse down and moving it. This triggered events in Delphi which could be used to control the movement with. The problem with this approach was that the mousebutton had to be held down all the time which seemed somewhat awkward. Asbjörn came with the idea to create a "sticky joystick". Which means to make the mouse stick to the joystick control when you click on it. The result of this design is that you can control the movement without having to concentrate on the control or the mousebutton all the time.

The goal of the project was to create a system that could connect people regardless of time and space. This in essence means use of the TCP/IP protocol and programming Windows sockets. I had never programmed this protocol, so I was a little worried. I felt that the best solution was to create a protocol that worked over a serial link using a communications VBX (Visual basic extension) with Delphi. The VBX first seemed to work fine under the Delphi environment, but it wouldn't send anything over the serial link. I had tested the VBX under Visual Basic and I knew it worked fine, so the problem had to exist in the Delphi implementation of the VBX support. Visual Basic was not considered in the project, because of its slow interpreting code, so a switch of development environment was out of the question. This meant that we had to find some other solution to program the serial link. The solution was programming the serial link with the Windows 16 bit API. Anders and I worked together on this and also created a serial protocol for sending messages. The problem was stability and error correction. As we tested the solution at high message rates it started to drop messages. We thought that the Windows API had built in handshaking procedures, but that was not the case. Writing a handshaking and error correcting serial link, was a project in it's own. We finally gave up the idea of a serial link, and tried to find some way to program a Windows socket solution. The problem was that none of us knew how to handle the Windows socket

API. After many hours of searching the internet after instructions and documentation on socket programming, I found a component DWinsock for the Delphi environment. This component encapsulated the entire Windows Socket API (version 1.1) in a couple of objects. The best part was that it was free. An old computer, and two network adapters was borrowed to test the component. In the Christmas holidays after hours of testing a connection between the two computers where established. Hard testing showed that the component worked and the communications link was stable and error free. Now one of the goals was reached, namely to have a TCP/IP link in the system.

Problems started to arise when the message protocol between the systems was implemented. Before the TCP/IP link a timer was used in the "VR-server" to render the picture. This solution didn't work together with the timer in the Delphi program which dispatched messages. The result was a slow and erratic behavior of the "VR-server". The solution to the problem was to use one single timer in the Delphi program, that also controlled the rendering of the "VR-server". The timer in the "VR-server" was removed. Messages was then continuously sent to the "VR-server" to re-render the picture. If no changes in the VR-world was made the "VR-server" didn't re-render the picture. This also had a positive effect on the overall performance of the system, selecting windows, using dialog boxes, and menus didn't slow down and could be used without problems. The problems discussed above comes from the fact that the Windows 3.x environment is a cooperative multitasker. If the implementation was done in the Windows 95- or Windows NT environment, threads could be created to handle the tasks of rendering and message dispatching. This is possible the next step in the programs evolution.

The next step in the development was to implement a message protocol between the systems. The development of the protocol was quite straightforward once the link was stable and error free.

The user interface created at the beginning of the project did not feel right. The floating control panel always had to be moved to fit the screen. The solution was to make the control panel a part of the program. When a model is entered the control panel i shown at the bottom of the screen, as a part of the main window.

A serious problem was found when the system was tested with 3Dstudio models. As the models where imported and loaded into the system, large holes appeared in objects. This was due to the fact that surfaces in the renderware system only are visible from one side. Which side depends on which direction the surface is drawn in the CAD/Visualisation program. Some kind of preprocessor for the scriptfiles, that could make the surfaces two sided was necessary. A processor was created in Delphi the problem was solved.

When testing the system with the "House-model" it was apparent that the select box developed earlier in the project couldn't be seen on certain objects. A new way to select objects had to be developed. The solution was to use a wireframe select box with "glass" sides as shown in figure 5. Testing also showed that there had to be a way to show that an object had a comment attached to it. This was solved with red cubes attached to each side of the object, when a note was attached to it (see figure 7)

8 CONCEPTUAL MODEL

A conceptual model was formulated based on (Christiansson, 1995)

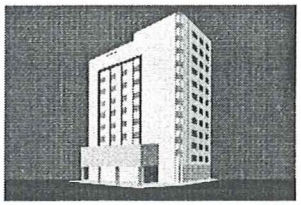
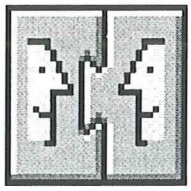
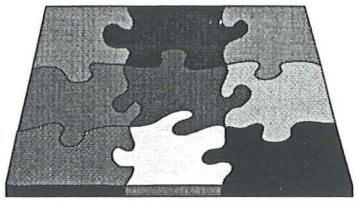
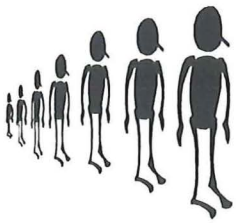
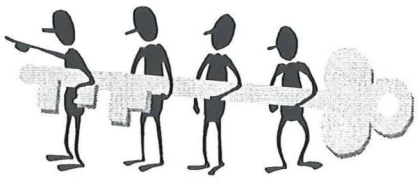
Domain	Product	Process
Application	 3D Geometric VR model	 Edit, change, move, annotate, walk arounds, preview alterations.
System	 Low cost distributed VR user interface description.	 Network handling, editing functions, control devices.
User	 Architects, engineers, systemengineers.	 Collaborative work on/in a 3D model

Table 1

8.1 Application model

The application model is based on a three dimensional geometric model. This model could for example be a building divided into parts or objects. Objects in the model should be able to be moved, edited, and tagged.

The geometric model can be derived from different CAD (Computer Aided Design) systems, for example AutoCAD, 3Dstudio, Microstation, and GDS (McDonald Douglas).

The CAD systems must support some kind of object hierarchy which could be implemented as layering or naming of objects.

The following figure illustrates a sample 3D model created in AutoCAD.

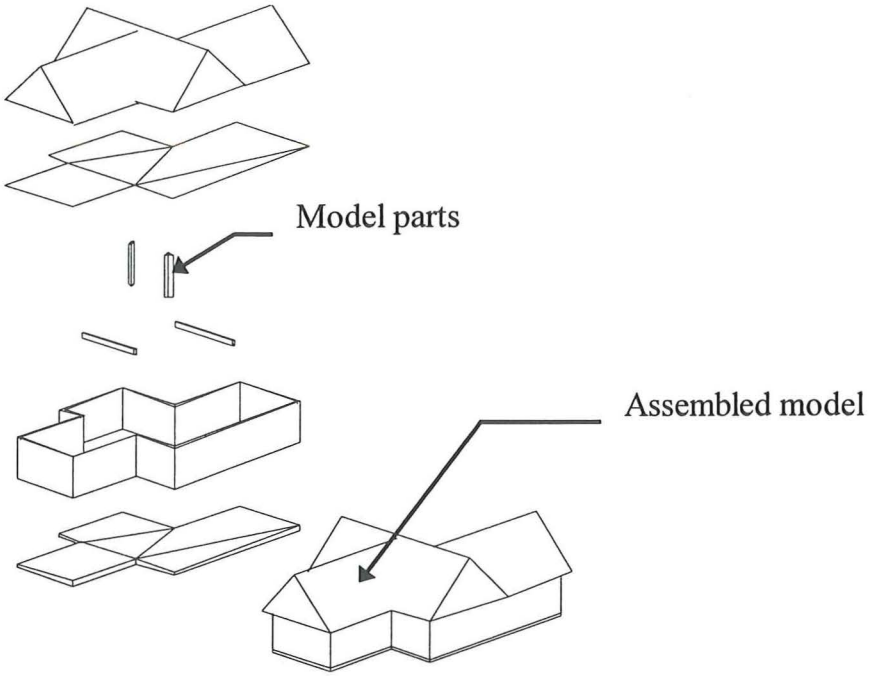


Figure 13 - Parts of a model

To be able to connect information to the different parts/objects in the model each part has a unique tag. The tag is an integer number. Table 2 shows an example of a tag numbering scheme that could be used for our house.

Tag numbers	Object types
0 - 999	System objects. Users. Ground etc.
1000 - 1999	Attached notes
2000 - 2999	Controls. Activators
	User defined object types
10000 - 19999	Supporting objects. Beams supports etc.
20000 - 29999	Walls objects.
30000 - 39999	Ceiling objects.
40000 - 49999	Roof objects.

Table 2 - Tag numbering scheme

8.2 System model

The system is designed as a direct manipulation system (Preece 1994, p 270) which means that it should have the following features

1. Visibility of the objects of interest

2. Rapid and reversible, incremental actions

3. Replacement of complex command language syntax by direct manipulation of the object of interest.

The LCD system is designed according to the above criteria. In the LCD system the objects are visible to us directly through the view window (1). The actions on the objects are rapid. When the color is changed through the object properties window the object is updated immediately. Reversible actions have not yet been implemeted (2). There is no command language in the LCD system instead you manipulate the objects directly by clicking on them and changing their properties (3).

The LCD system reads the geometric model and presents it to the user in a view window on the screen. Attached notes and comments are stored in a separate database with the same name as the geometric model.

Moving around in the model is done using a "virtual joystick". Clicking on objects in the view windows brings up the objects properties in a separate windows. In this window the user can change the object visibility properties, and annotate the objects. These comments are stored in the database.

All functions used when moving around in the model are presented in the form of a control panel in the lower part of the screen. Because the system uses a lot of windows there is a risk of clutter and disorganization (Preece 1994, p 303) and thus a floating control panel was discarded. The controls on the panel are divided into groups representing different type of controls, for example network controls and environment controls.

All the actions taken by one user is duplicated on the other system. This makes the system perform as one system even if the users are separated by great distances. In chapter 14.4 page 305 in Human Computer Interaction (Preece, 1994) some important points concerning feedback are covered

> "Imagine a situation in which two people are each using their own workstation to communicate with each other via a shared voice channel and desktop window. It is essential that these two media be kept synchronized, so that as one person comments on objects and points to them there is no lag while the desktop window catches up with the voice. The user making the comments and carrying out the manipulations must also experience immediate feedback."

This has been the goal of the design to make the system respond directly to action of the users, giving them direct feedback.

Communication between users can be done in the chat windows. A future enhancement could be to improve the chat window with audio and video.

Figure 14 describes in large the system architecture.

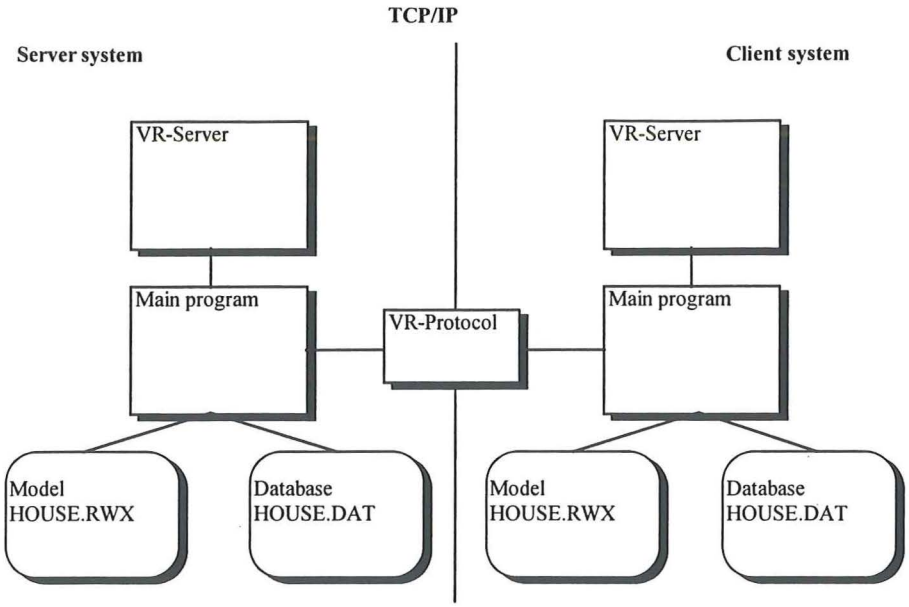


Figure 14 - General system architecture

8.2.1 MENU

The menubar is shown when the system is offline and at startup. The menubar provides means to control the systems overall settings before opening the model and connecting to another system. When the model is opened the menu is hidden and all the functions in the system is controlled from the control panel.

The menu itself is designed in a category fashion with menu options sorted according to what category they belong. (Preece, 1994, p 265)

The menu consists of a **File** menu and a **System** menu. The file menu contains functions for quitting and picking the geometric model and database. The **System** menu contains function for converting file formats and setting system properties.

In the file menu the only choice is to pick the geometric model and the database to use. The choice is done with a standard file dialog box.

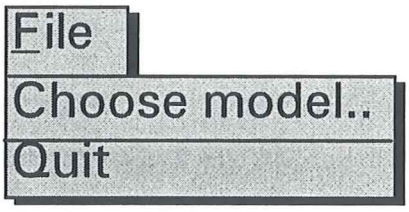


Figure 15 - File menu

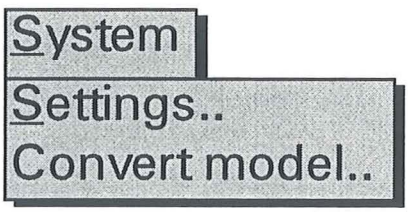


Figure 16 - System menu

8.2.2 TOOLBAR

The toolbar, figure 17 duplicates the menu functions, and experienced users can quickly control and setup the system by using the tool buttons instead of the menus. The tool buttons also have so called "hint boxes" that describes the button when the mouse cursor is located over the button.

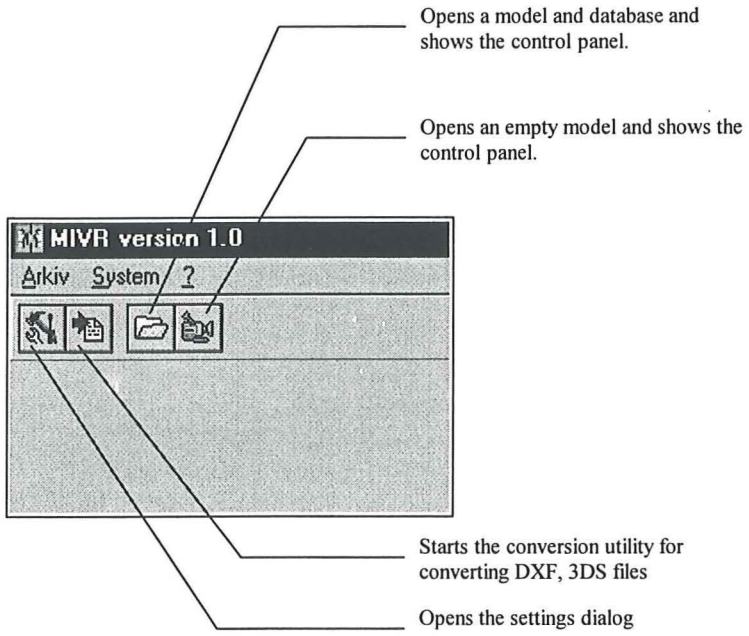


Figure 17 - The toolbar

8.2.3 SETTINGS DIALOG

The settings uses a dialog feature called TabSet in Delphi, see figure 18. The TabSet enables a dialog box to display multiple pages sorted under categories. (Borland, 1995b) One of the advantages of this feature is that you don't need a dialog box for each category. All settings in the program are handled using only one dialog box.

The settings dialogbox contains general settings and network node information for the entire system.

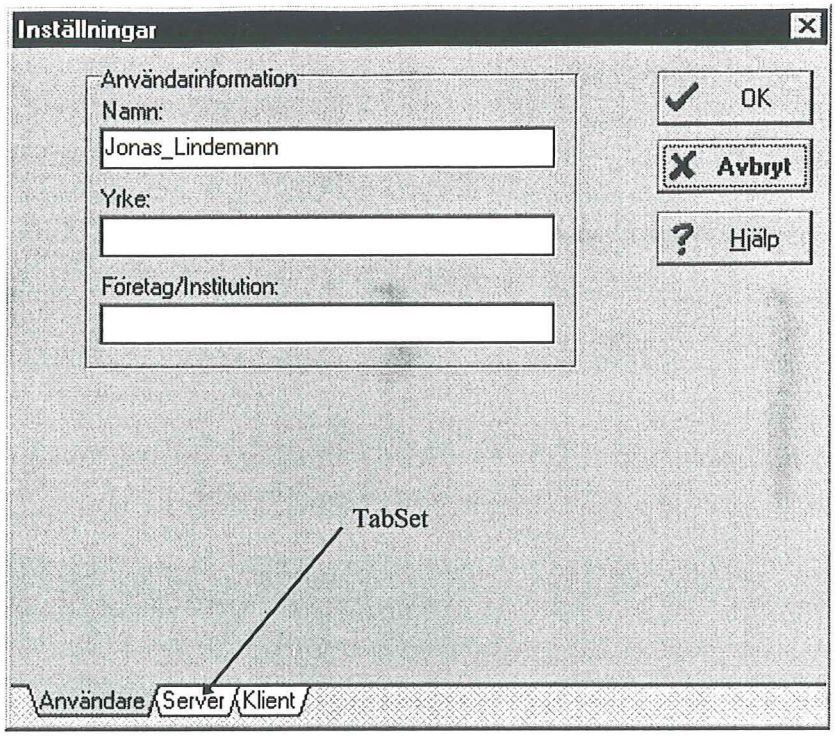


Figure 18 - The settings dialog

Table 3 describes the tabs and the contents.

Tab	Functions
User	General information such as name of user and user description
Server	Settings controlling parameters when the system is acting as a server: Server node IP adress Server port number Server start position Client start position Client rights
Client	Settings controlling parameters when the system is acting as a client: Server location IP adress Server port number

Table 3 - Setting dialog tabs

When the users clicks **OK** in the settings dialog the settings is written to a file. The file is automatically loaded when the system is started.

8.2.4 CONTROL PANEL

The control panel, see figure 19 is the central control of the system. All the functions of the system is placed in logical tool button groups.

It is important that the icons and buttons are intuitive and straight forward in their design in a direct manipulation system (Preece, 1994). The icons an symbols of the control panel was carefully chosen to be as intuitive and straight forward as possible to meet the demands for a direct manipulation system.

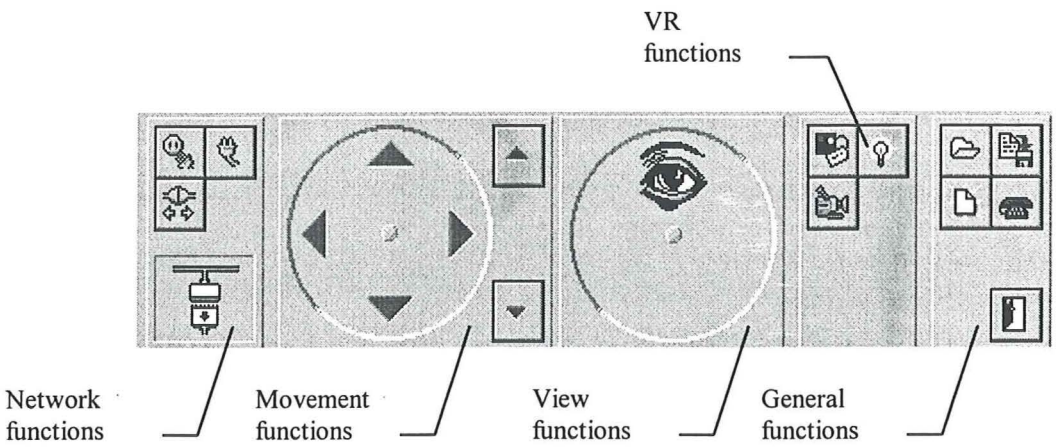


Figure 19 - The control panel

Table 4 describes the different control groups.

Group	Description
Network functions	Contains functions for connecting to another system over the network
Movement functions	Contains a "virtual joystick" for moving around in the VR environment
Viewing functions	Contains a "virtual joystick" for viewing in different angles.
VR functions	Contains functions manipulating the VR environment.
General functions	Contains function for saving the modified model, loading another model, and chat functions

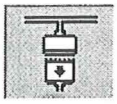
Table 4 - Function groups

8.2.5 NETWORK CONTROLS

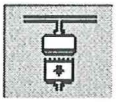
The network controls are used to establish system connections in the network. The control group consists of to three buttons. Visual feedback of the connection status is obtained through an icon in the network functions group.

A connection can be established in two different ways, (1) starting the system as a server and waiting for the other party to connect or (2) connect to the other system (server) as a client.

To start a session as a server the following procedure is done:

1. Click the  button to start the system listening for connections.
2. If a user connects to the system the  icon switches to  to indicate that a connection has been established. A representation of a user in the view window is also created.
3. If you want to terminate the connection at any point click the  button and the connection is immediately terminated. This button can also be used to stop listening for connections if nobody connects to the system.

To connect to a running server the following procedure is done:

1. Click the  to connect to the server indicated in the settings dialog.
2. If a user connects to the system the  icon switches to  to indicate that a connection has been established. A representation of a user in the view window is also created.
3. To disconnect from the server just click the  button.

Properly configured the user should not have to be concerned with IP addresses and network configurations. A more detailed description of the connection procedure is given in figure 20

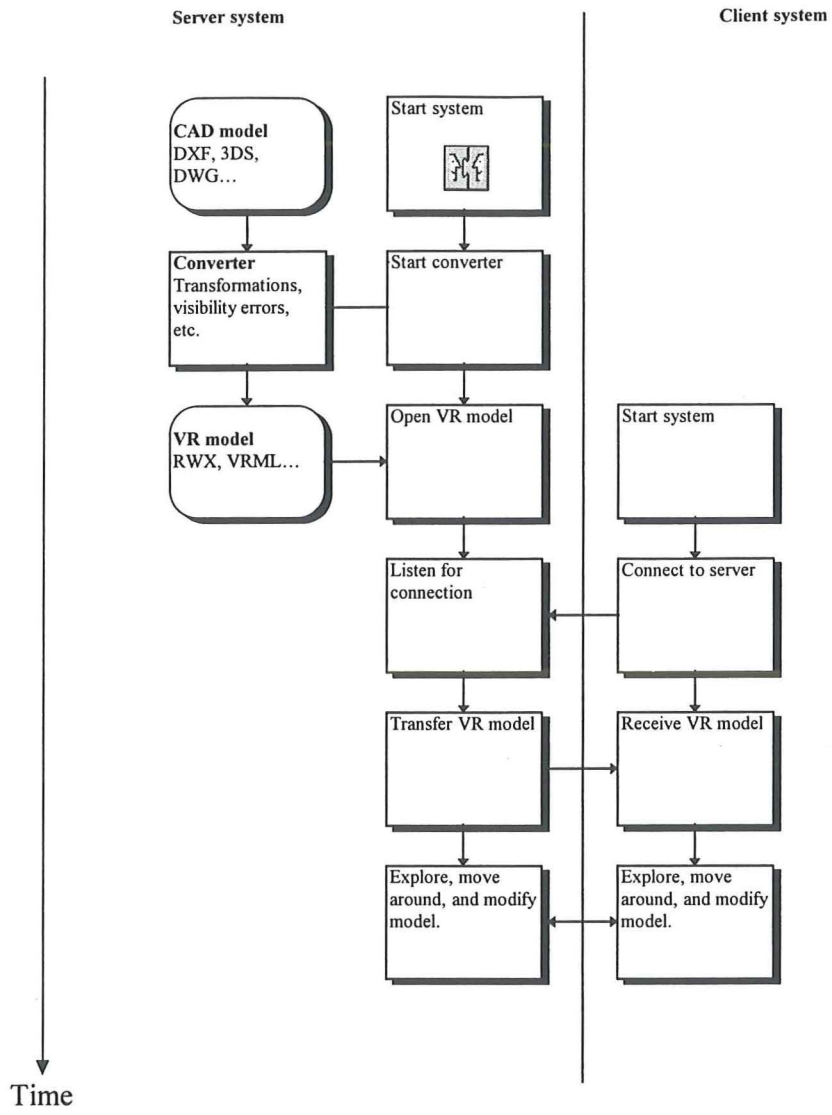


Figure 20 - The connection procedure

8.2.6 VR CONTROLS

The VR function group contains functions used to manipulate the VR system in some ways. The final version of the program will have more buttons in this menu.

Virtual note

This button drops a "virtual note" in the 3D model. This note can then be treated as any object in the model. If the note is clicked on in the view window the object properties window is shown and a comment can be attached to the note. The visual representation of the note is shown in the following figure:

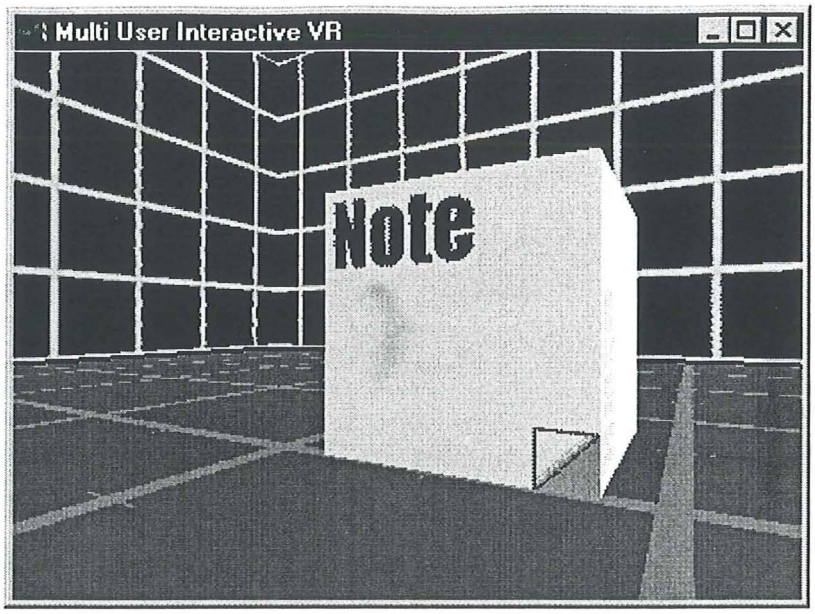


Figure 21 - The virtual note

 ***Basic environment***

This button hides and shows a basic 3D environment with walls and floor, see figure 22. The environment is used to create some references in the world if the model does not provide any environment.

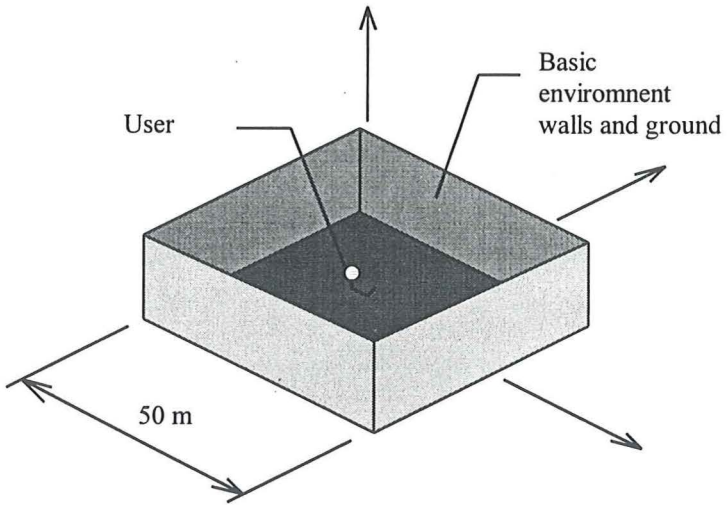


Figure 22 - The basic environment

 ***View manager***

Shows the view manager. The view manager is described in more detail in chapter 8.2.12

8.2.7 GENERAL CONTROLS

The general control group contains more general functions as save, change the active model, and show the chat window.

 ***Open model***

Opens a existing model. A standard file dialog box is shown and the user is prompted to choose a model. The model is then loaded and shown in the view window. This action is only available when the system is running as a server. When a server opens a model it is automatically transferred to the client system.

 ***Save model***

Saves the current model and database. A standard file dialog box is shown and the user is prompted to enter a filename. It is suggested that a different filename should be picked, because changes to the original model has been made.

 ***New model***

Erases the current model and database.

 ***Open chat window***

Opens the chat window. The chat window is described in a chapter 8.2.11

8.2.8 MOVEMENT CONTROLS

Movement in the 3D model is controlled with a "virtual joystick". This is a control designed specially for this system. It combines the mouse ease of use and the control aspects of the joystick. The following steps describes the usage of the "virtual joystick":

1 Click the left mousebutton inside the joystick control to use the control

2 Move the mouse to use the joystick

3 Click one more time on the left mouse button to release the mouse button from the control

Movement and rotation with the joystick can be described with the following figure:

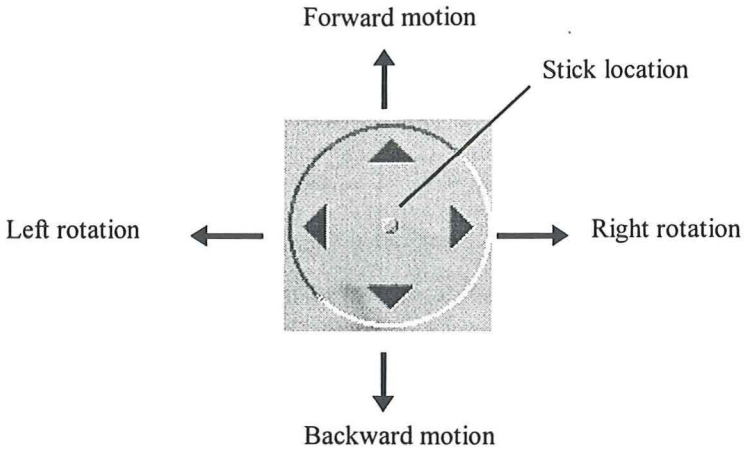


Figure 23 - The virtual joystick

It is possible with this control to get a smooth and continuous movement using only the mouse. Because the control simulates an analog joystick it is also possible to control the speed with the mouse. This gives a good feeling of control when moving and rotating.

The joystick also has the ability to apply different scaling factors. The following figure describes the different scaling options on the joystick control:

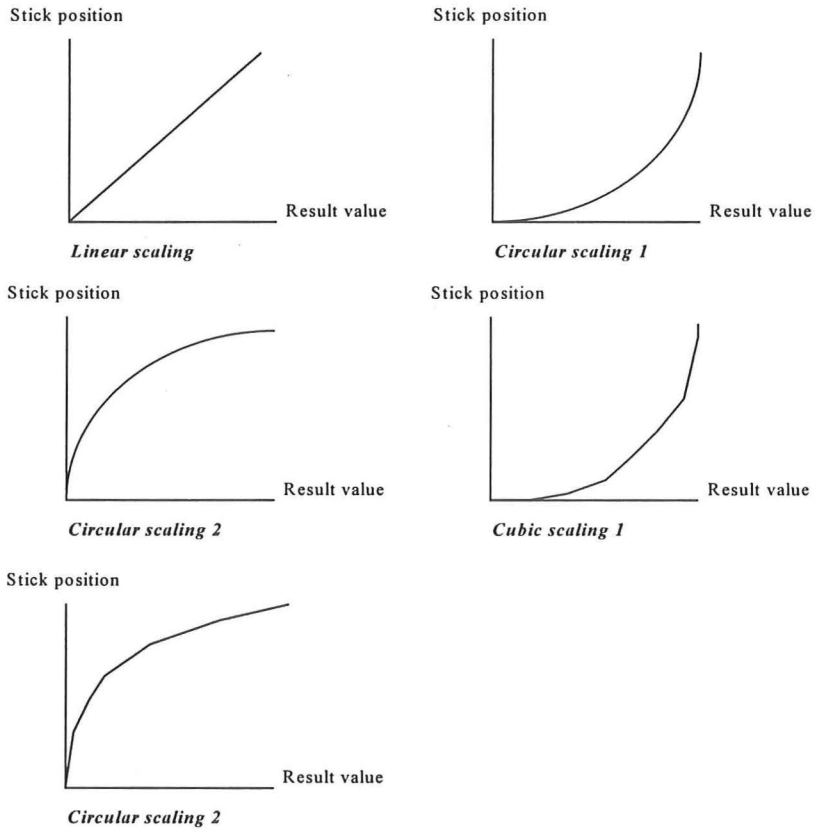


Figure 24 - Joystick scaling parameters

The scaling function gives different "feels" to the joystick movement. Thisw can be useful in certain cases. If you for example want better control at slow speed the circular 2 or cubic 2 scaling can be used.

8.2.9 SELECTING OBJECTS

Interacting with the 3D model is done using the view window and object properties window. The mouse pointer is moved over the object and clicked once to select an object in the model. When an object is selected the object properties window is automatically shown and the object marked with a special bounding box. The properties window is described in chapter 8.2.10.

A selected object is shown in the view window as a white wire frame bounding box with shaded "glass" sides, see figure 26. This approach was chosen because some objects are difficult to see when only a wire frame bounding box is used. A wall in a building for example would not show the wire frame bounding box well enough. The shaded "glass" sides shows the wall in a darker shade of its original color, clearly showing that it has been selected

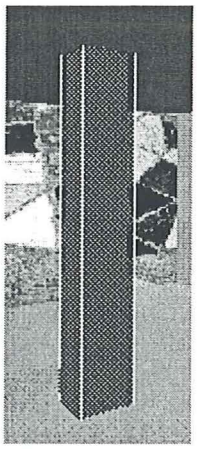


Figure 25 - Support showing the select bounding box

8.2.10 OBJECT PROPERTIES WINDOW

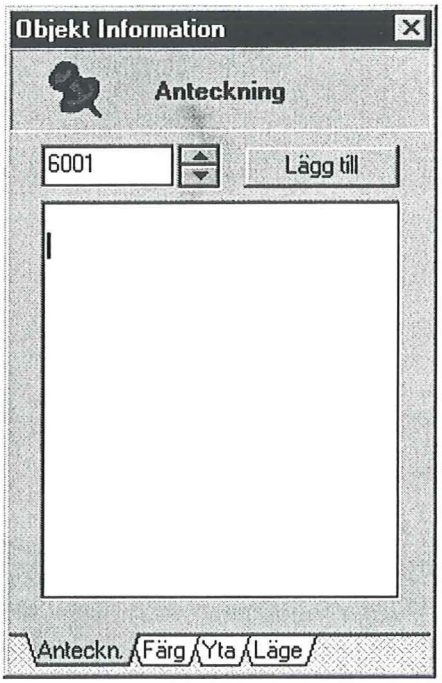


Figure 26 - The object properties window

The object properties window is shown whenever an object is selected, see figure 26. The window contains all the changeable properties of an object, for example color, opacity, specular, and position. To make the properties window smaller and simpler to understand tabs where added at the bottom of the window. The tabs let you switch between the different object properties. The tabs are described in table 5

Tab	Description
Comment	Handles the attaching and displaying of comments.
Color	Color properties. The red, green and blue components of the objects color can be changed and applied to the object.
Surface	Surface properties. The appearance of the surface can be changed in detail with these properties
Position	In this tab the object can be moved a certain amount of units in any direction

Table 5 - Object properties tabs

8.2.11 CHAT WINDOW

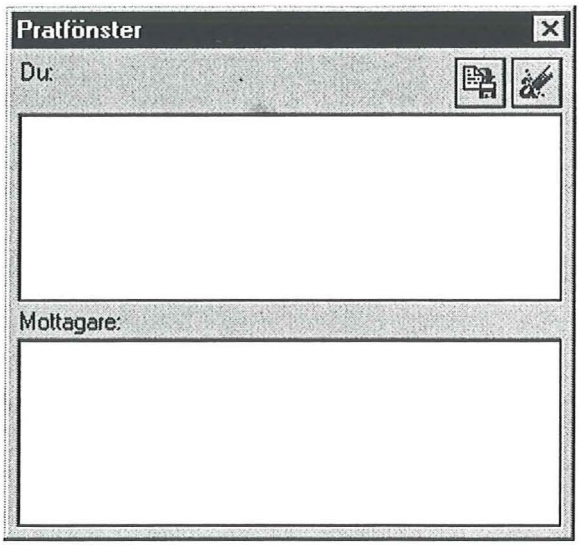


Figure 27 - The chat window

The chat window, see figure 27 provides communication between the users of the system. The window is divided in two parts. The upper part is used to enter messages. The lower part displays any incoming messages from the other system.

Future enhancements of this window will probably be support for audio and video.

 ***Save current conversation***

This buttons opens a standard file dialogbox and prompts the user to name a text file in which the current session should be saved

 ***Erase current session***

Erases the chat window and the current session.

8.2.12 VIEW MANAGEMENT

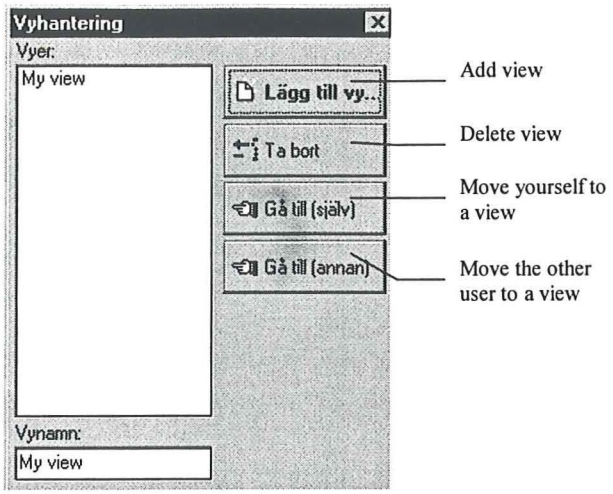


Figure 28 - The view manager window

The view management window, see figure 28 maintains a list of views of the 3D model. It also enables a user of the system to show the other user a view, by moving him. This makes the system easier to use for inexperienced users having difficulties using the movement controls.

8.3 User model

The potential users of the system are architects, engineers and project leaders. The construction process can be described as a teamwork, where the parties are working together. Exchanging information. To design a product of a good quality the different parties have to be able to easily exchange ideas and questions without time consuming travels. The following situations describes the users and the interaction needed in the user model:

User	Scenario
Architect	The architect has designed a model of a office building. He is not sure if it the large spaces in the bottom floors are feasible in an engineering standpoint. He want's the engineer to look over his design.
Engineer	The beams of the bottom floor has to be made higher to withstand extended loads due to a ventilation system on the upper levels. The engineer wants to check with other consultants if the pipes in the bottom floor could be moved in a simple manner.
Project Leader	The project leader of the new office building wants the plans to be as correct as possible when the construction is started in the spring. To do this he wants the different parties to work together as closely as possible, always exchanging information when a change in the plans has been done. He also wants to show the progress to the client/clients in a visual appealing way, so changes can discussed directly and without use of complicated drawings.

Table 6 - The user model

9 DATA MODEL

Figure 28 describes in general terms the data model used in the LCD system. Details of the system are discussed in the following chapters.

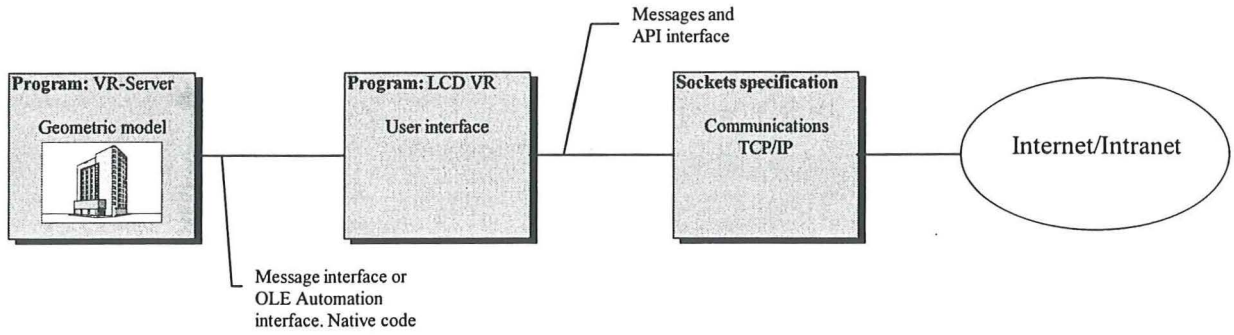


Figure 29 - Data model

9.1 VR-Server

The VR-Server is a specialized application for managing the 3D system. The server contains functions for displaying 3D models, moving the view, selection, and other VR-related functions. Figure 30 shows the VR-Server architecture.

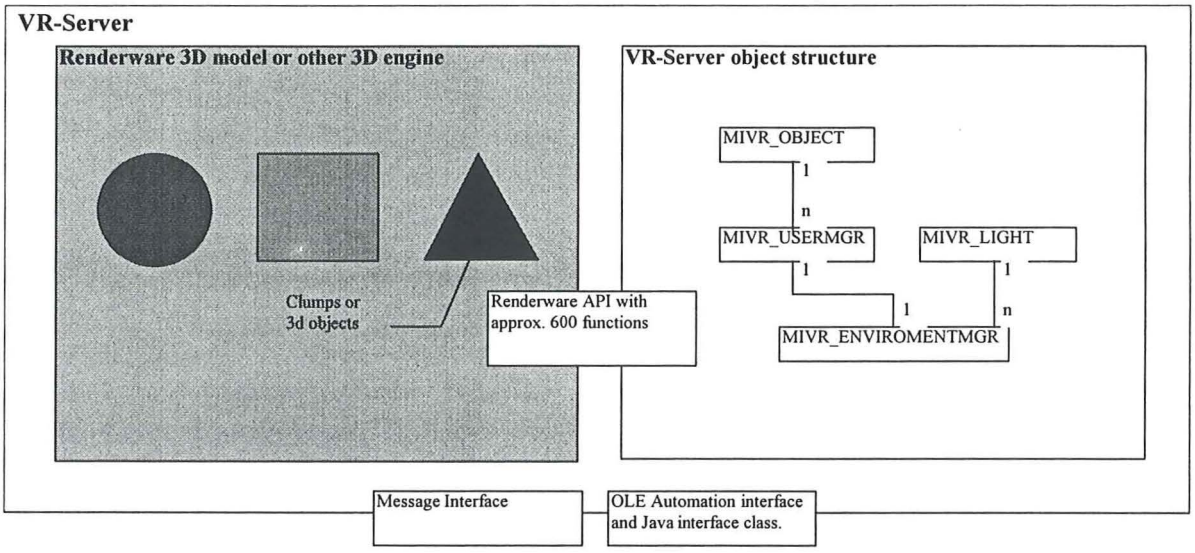


Figure 30 - VR Server architecture

The core of the system is the renderware engine. This is a API for rendering and handling of 3D models. It consists of roughly 600 API functions. To make the system less dependent on the rendering engine the API functions are encapsulated in an object structure. This also makes it easier to program.

The functional component of the renderware 3D system is the "clump". This entity contains an entire 3D geometry, surface descriptions needed to render it onto the screen. The "clump"

is encapsulated into the C++ object *MIVR_OBJECT* class. This class contains all the basic API functions for manipulating a "clump". The following class description describes the available object methods:

```
/*****************************************************************
 *
 * class MIVR_OBJECT - Clump management object
 *
 *****************************************************************/

class MIVR_OBJECT {
    public:
        MIVR_OBJECT(void);
        MIVR_OBJECT(char *name);
        MIVR_OBJECT(char *name, float p1, float p2, float p3);
        MIVR_OBJECT(RwClump *pclump);
        virtual void Create(RwScene *scene);
        virtual void SetName(char *name);
        virtual void GetHandle(RwInt32 &p1);
        virtual void SetHandle(RwInt32 p1);
        virtual void SetHints(RwClumpHints p1);
        virtual void SetColor(float p1, float p2, float p3);
        virtual void SetAmbient(float p1);
        virtual void SetDiffuse(float p1);
        virtual void SetSpecular(float p1);
        virtual void SetOpacity(float p1);
        virtual void MoveTo(float p1, float p2, float p3);
        virtual void MoveRel(float p1, float p2, float p3);
        virtual void Move(float p1);
        virtual void Rotate(int axis, int p1);
        virtual void Scale(float p1);
        virtual void Reset();
        virtual void Select();
        virtual void Deselect();
        virtual void CreateTags(RwInt32 handle, RwScene *scene);
        virtual void Destroy();
        virtual RwClump* GetClump();
        virtual void SetClump(RwClump *pclump, RwScene *scene);


    private:
        void CreateHighlight(RwScene *scene);
        void SizeHighlightToFitClump();
        void UpdateHighlight();
        char objname[13];
        RwReal   x,y,z;
        RwClump *clump;
        RwClump *highlight;
        RwClump *tag;
        RwScene *last_scene;
};
```

The VR-Server also handles the representation of users in the VR-environment. This is done through the *MIVR_USERMGR* class which encapsulates this behavior. Each user created is given a unique identifier tag. The tag is used when the server is called from the user interface. The object description is shown in the following code:

```
/*****************************************************************
 *
 * class MIVR_USERMGR - Manages users in the system
 *
```

```
 ****************************************************************/

class MIVR_USERMGR {
    public:
        MIVR_USERMGR();
        ~MIVR_USERMGR();
        void UserCreate(RwInt32 p1);
        void UserDestroy(RwInt32 p1);
        void UserMoveTo(RwInt32 p1, float x, float y, float z);
        void UserMove(RwInt32 p1, float v, int w);
        void UserReset(RwInt32 p1);
        void SetScene(RwScene *p1);
        void SetForward(RwInt32 handle, float v);
        void SetTurn(RwInt32 handle, float v);
        void SetElevate(RwInt32 handle, float v);
        void Update();
        BOOL Movement();

    private:
        int nbr_of_users;
        MIVR_OBJECT *user[MIVR_MAXUSERS];
        RwScene *scene;
        float user_forward[MIVR_MAXUSERS];
        float user_elevate[MIVR_MAXUSERS];
        int user_turn[MIVR_MAXUSERS];
};
```

The VR-environment is handled by the *MIVR_ENVIROMENTMGR* class which controls the manipulation of objects. All objects in the system are given unique tags which are used as they are manipulated.

The environment manager is also used to control the appearance of the environment. The class contains methods for creating lights, and a basic environment see chapter 8.2.6. The class is described in the following code:

```
/****************************************************************
 * class MIVR_ENVIROMENTMGR - Manages the VR environment
****************************************************************/


class MIVR_ENVIROMENTMGR {
    public:
        MIVR_ENVIROMENTMGR();
        ~MIVR_ENVIROMENTMGR();
        void SetScene(RwScene *p1);
        void SetModel(RwClump *p1);

        RwClump *FindClump(RwInt32 handle);


        void ObjectMove(RwInt32 handle, float dx, float dy, float
                        dz);
        void ObjectCreateTag(RwInt32 handle);
        void ObjectRotate(RwInt32 handle, float rx, float ry, float
                          rz);
        void ObjectColor(RwInt32 handle, float red, float green,
                         float blue);
        void ObjectAmbient(RwInt32 handle, float p1);
        void ObjectDiffuse(RwInt32 handle, float p1);
        void ObjectSpecular(RwInt32 handle, float p1);
        void ObjectOpacity(RwInt32 handle, float p1);
        void ObjectDelete(RwInt32 handle);
```

```
        void ObjectSelect(RwInt32 handle);
        void ObjectUnSelect(RwInt32 handle);

        void LightAdd(MIVR_LIGHT *p1);
        void LightDestroy(RwInt32 handle);
        void LightMoveTo(RwInt32 handle, float p1, float p2, float
                         p3);
        void LightOn(RwInt32 handle);
        void LightOff(RwInt32 handle);
        MIVR_LIGHT *LightGet(RwInt32 handle);
        void CreateGround();
        void RemoveGround();
        void CreateBasicLighting();

    private:
        RwScene      *scene;
        RwClump      *found_clump;
        RwClump      *model;
        RwClump      *ground;
        MIVR_OBJECT *object;
        MIVR_LIGHT   *light[MIVR_MAXLIGHTS];
        RwInt32      last_handle;


};
```

The server is controlled through a message interface, see table 7. The client starts the server as a separate program and sends messages it to control rendering and movement. A future enhancement would be to make the server an OLE (ActiveX) Automation object which should have the messages implemented as object methods. This also makes it possible to use the automation object with other applications. The server could then be used in conjunction with a user interface and network functions written in the Java language.

Message types	Message
General	WM_VR_INIT WM_VR_RENDER
Camera movement	WM_VR_UP WM_VR_MOVE ...
Connection	WM_VR_CONNECT WM_VR_DISCONNECT
Data exchange	WM_VR_SENDATOM WM_VR_CAMERA_INFO ...
User management	WM_VR_USER_CREATE WM_VR_USER_MOVETO ...
Object handling	WM_VR_OBJECT_SELECT WM_VR_OBJECT_MOVE ...
Environment messages	WM_VR_LOADMODEL WM_VR_ENV_GROUND WM_VR_NOTE

Table 7 - The message interface

9.2 User interface

Each part of the user interface is packaged in Delphi-units. The units contains the necessary object classes for implementation of the interface part. Table 8 shows the different units and their functions in the LCD-VR system.

Unit	Function
VRDesk	Contains the main window, toolbar and menu system. VRDesk creates other windows from other units when needed.
Settings	Handles the settings window and the .INI files
Script2	Contains classes and functions for writing and reading initialization files in a special script file format
Control	This unit contains the controlpanel window from which the entire system is managed when a model has been entered. The TControlPanel window class also owns then Dwinsock components client and server, which are used for connection to the internet via the winsock 1.1 library. The control panel also uses the VRSys unit which interfaces with the VR-Server. This unit contains classes for managing the VR environment.
VRSys	Interface unit for the VR-Server
Protocol	Implements the LCD VR-system protocol. Handles messages between the systems.

Table 8 - LCD-VR system units

The following figure shows the systems general architecture:

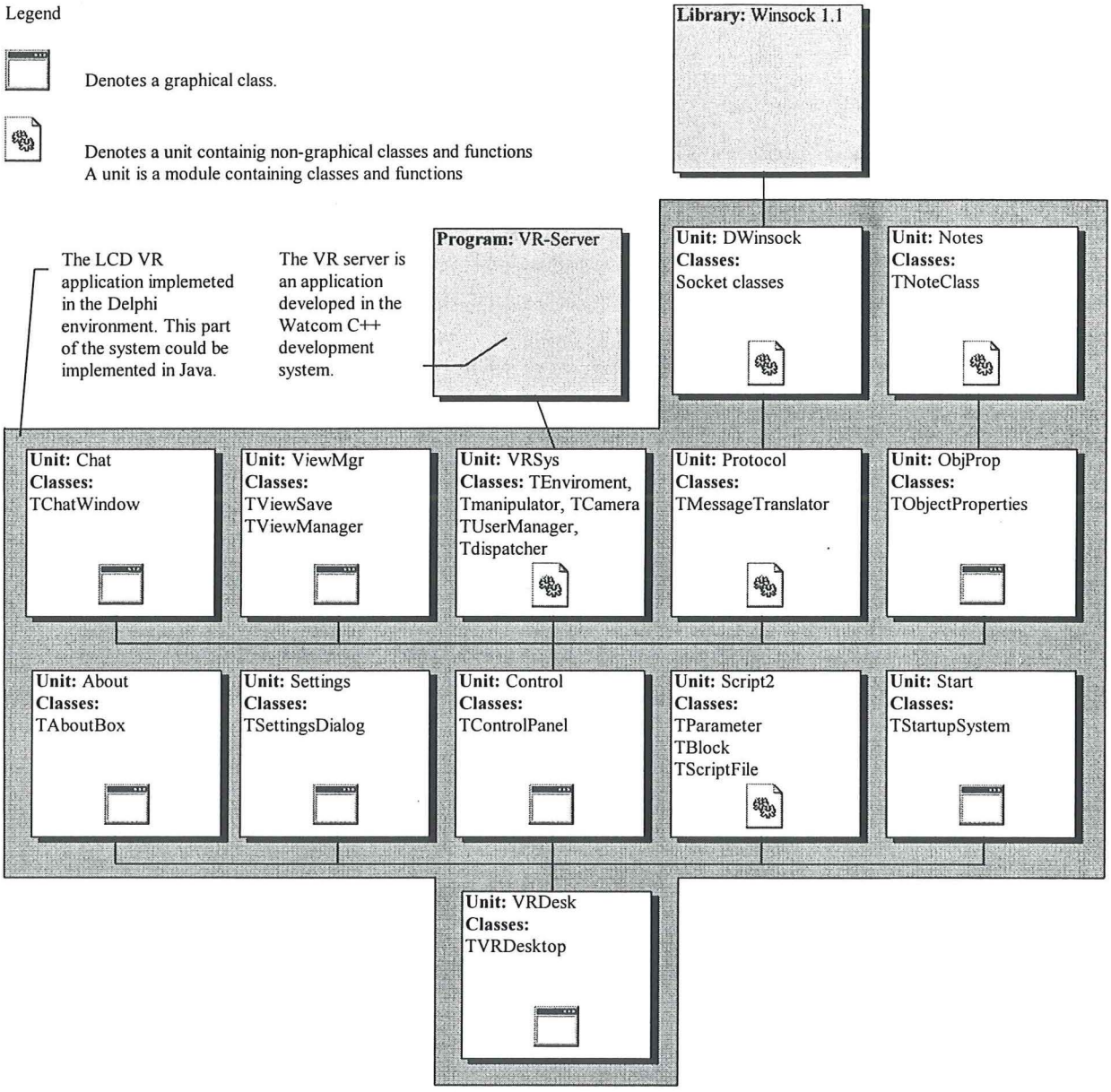
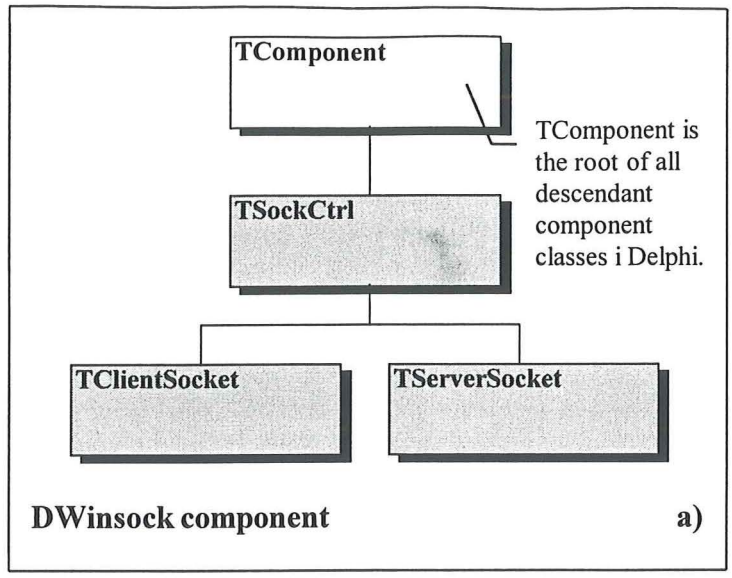


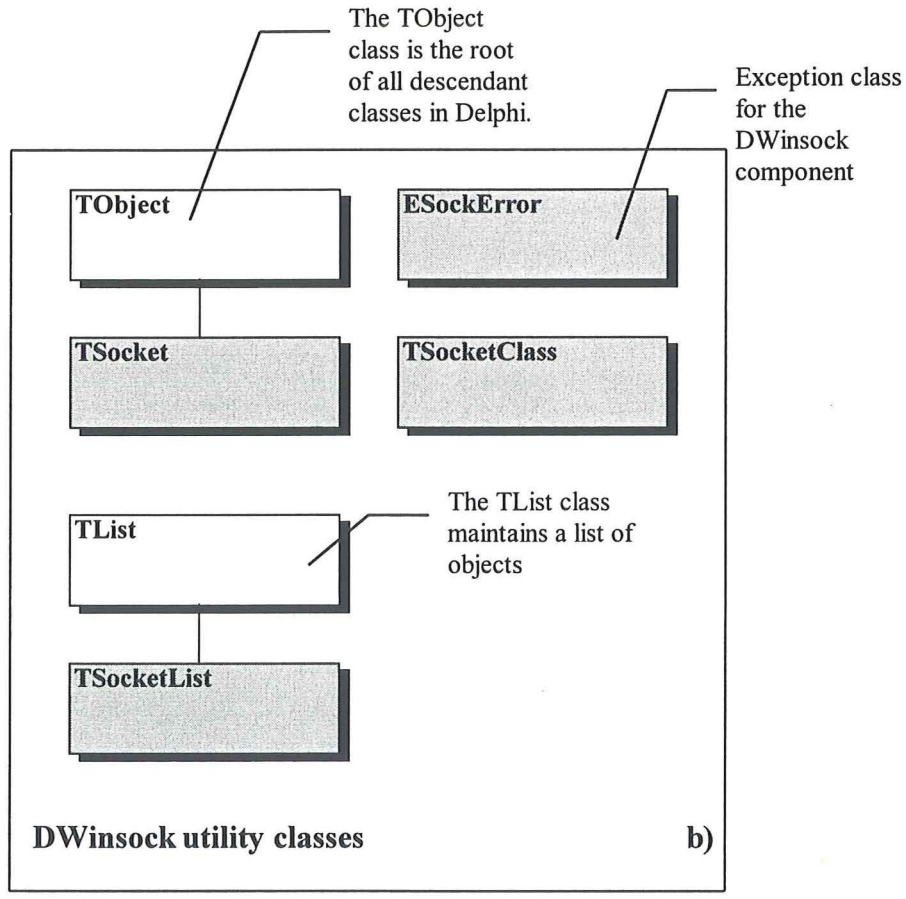
Figure 31 - System architecture

9.3 Communications

The communications block of the system is divided into three parts. The highest part is the Protocol unit in the Delphi application. This unit implements the protocol used between two LCD-VR systems. The middle part is the DWinsock component (Söderberg et.al., 1996) which encapsulates the Winsock 1.1 DLL. The lowest part is the Winsock DLL. This library is never called directly from the system, instead the DWinsock component classes are used.

The DWinsock component contains a number of classes defining socket communications. Figure 32a and b shows the class hierarchy in the DWinsock component.





White boxes indicate original VCL classes.

Gray boxes indicate derivesd classes in the DWinsock component.

Figure 32 a) and b)- DWinsock class hierarchy

The following object pascal code shows how a client connection is opened with the DWinsock component

```
procedure TControlPanel.SBConnectClick(Sender: TObject);
begin

  { Button : Connect to server }

  CLSocket.Open(TSocket);
end;
```

The *CLSocket* is an instance of the class *TClientSocket*

When a connection is made the DWinsock component can respond to a number of different events. The following table describes the main event methods responded to by the *TClientSocket* and *TServerSocket*.

Event method	Description
OnConnect	This event occurs whenever a TClientSocket has either connected or failed to connect to the remote host specified in Address or Host following a call to Open.
OnDisconnect	This event only occurs when the connection has been closed by the peer (the other end).
OnInfo	The OnInfo event is called when the state of the socket component changes. This is useful if you wish to let the user know what is happening on the socket by way of a Status Bar.
OnRead	This event occurs when there is data ready for reading on the socket specified in the 'Socket' parameter. In your handler you should call Recv or read the Text property to read the data.
OnWrite	This event is called when the socket specified by 'Socket' is ready for writing.
OnTimeout	When the number of seconds specified in the TimeOut property have elapsed after a call to Open, this event is invoked.

Table 9 - Event methods in the DWinsock component

The following delphi code handles an OnConnect event.

```
procedure  TControlPanel.CLSocketConnect(Sender:  TObject;  Socket:
TSocket);
var Msg : TVRStringMessage;
begin

  { Send connect message to server }

          Msg:=MsgTranslator.CreateStringMsg(VR_CONNECT,0,'Användare
ansluten');
  Socket.Send(Msg,SizeOf(Msg));
  CurrentSocket:=Socket;
  ProgState:=prg_Client;
  Camera.ResetPos;
  Camera.MoveTo(0,1.6,0);

  { Enable Controls }

  IMGConnected.Visible:=true;
  IMGDisconnected.Visible:=false;
```

```
    SBListen.Enabled:=false;
    SBConnect.Enabled:=false;
    SBOpenModel.Enabled:=false;
    SBNewModel.Enabled:=false;

    { Tell the other windows that we have a socket }

    ChatWindow.SetSocket(CurrentSocket);
    ViewManager.SetSocket(CurrentSocket);
    ObjectProperties.SetSocket(CurrentSocket);

end;
```

When the *OnConnect* event method is called we are given the connected socket in the input arguments.

```
(Sender: TObject; Socket: TSocket);
```

The first thing this procedure does is to create a message which should be sent to the connected server. This is done with the *TMessageTranslator* class described later in this chapter. When the message is created it is sent with the following statements:

```
Socket.Send(Msg,SizeOf(Msg));
CurrentSocket:=Socket;
```

The connected socket is stored in the variable *CurrentSocket* to tell the system which socket to use during communication. At the end of the procedure the other windows are also notified about which socket to use.

```
ChatWindow.SetSocket(CurrentSocket);
ViewManager.SetSocket(CurrentSocket);
ObjectProperties.SetSocket(CurrentSocket);
```

The *OnDisconnect* method is handled in the same manner. The *CurrentSocket* variable is set to *nil* and the other windows are also notified about this.

The *Protocol* unit is used to create and translate messages between the systems. The principle used when designing the message structure was to eliminate unnecessary information in each sent message. A notify message only contains a standard message header and a single integer data item and that is the only information sent over the network. Messages with larger information contents share the same message header, but has a different data item type. A *TVRVectorMessage* has the following record type. Se Table 9 for details.

```
TVector    = record
               X : real;
               Y : real;
               Z : real;
             end;

PVRVectorMessage = ^TVRVectorMessage;
TVRVectorMessage = record
                     Id       : integer;
                     Handle   : integer;
                     DataType : integer;
                     Data     : TVector;
                   end;
```

Field	Description
Id	Identifies the message. VR_MOVE, VR_MOVETO for example
Handle	Contains a handle to a specific object in the geometric model.
DataType	Contains information about what kind of datatype the field data contains
Data	Message data

Table 10 - Structure of a message

Five data types are used in the message structure Integer, String, Vector, Movement and View. These data types are the foundation on which all messages in the system are derived.

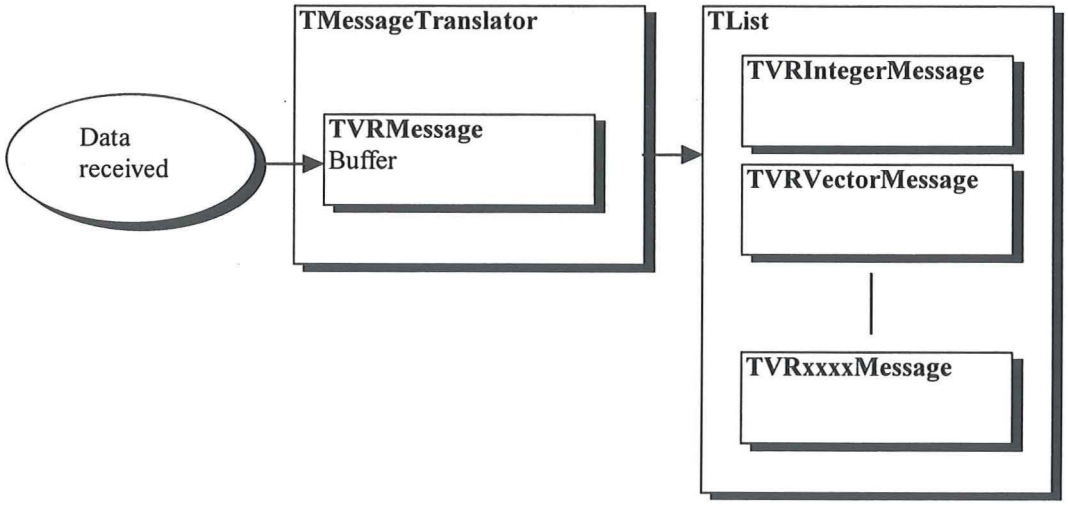


Figure 33 - Message translation

When a socket receives information, a *OnRead* event occurs and the message is received in an input buffer of the type *TVRMessage*. The message buffer is then given to the *TMessageTranslator.Translate* method. In the *Translate* method the message buffer is

processed and the messages are cut into correct sizes and returned as a list of the class *TList*. The list contains message objects derived from the *TObject* class. Each object contains a private variable of the correct message type for example *TVRIntegerMessage*. Figure 32 visualizes the message translation process.

The *TMessageTranslator* class is also used as messages are sent. The following statements show how this is implemented.

```
VRMsg:=MsgTranslator.CreateIntegerMsg(VR_SELECT,100,msg.lParam);
if CurrentSocket<>nil then
  CurrentSocket.Send(VRMsg,SizeOf(VRMsg));
```

In this example the method *CreateIntegerMsg* is used to create a notify message. The method returns the variable *VRMsg* which is of the *TVRIntegerMsg* type. The message is then sent with the *TSocket.Send* method.

The protocol is divided into the following categories, table 4.

Message types	Messages
Movement messages	VR_MOVE VR_ELEVATE
Vector messages	VR_MOVETO VR_ROTATE VR_USER_MOVETO VR_CREATE_NOTE VR_OBJECT_MOVE VR_OBJECT_COLOR VR_OBJECT_AMBIENT VR_OBJECT_DIFFUSE VR_OBJECT_SPECULAR VR_OBJECT_OPACITY
String messages	VR_TEXT VR_CONNECT VR_MEMO_UPDATE
Integer messages	VR_SELECT VR_UNSELECT VR_MEMO_CLEAR VR_CHAT_OPEN VR_CHAT VR_CHAT_CLOSE VR_CHAT_CLEAR VR_GROUND_SHOW VR_GROUND_HIDE
View messages	VR_SET_VIEW

Table 11 - Protocol message categories

Messages are processed in the *ProcessVRMessages* method from the *OnRead* methods in *TClientSocket* and *TServerSocket*. The following code shows a part of the *ProcessVRMessages* method.

```
procedure TControlPanel.ProcessVRMessages(Socket : TSocket);
.
.
.
begin

  { Receive socket buffer }

  NbrOfBytes:=Socket.Recv(VRMessage^,SizeOf(VRMessage^));

  { Translate the messagebuffer }

  MsgList:=MsgTranslator.Translate(VRMessage,NbrOfBytes);

  { Determine messagetype and translate it }

  for i:=0 to MsgList.Count - 1 do
  begin
    MyObject:=MsgList.Items[i];
    .
    .
    .

    { Process vector messages }

    if MyObject.ClassName='TVRVector' then
    begin
      VRVector:=MsgList.Items[i];
      VRVector.Get(VRVectorMsg);
      case VRVectorMsg.Id of
           VR_USER_MOVETO : begin
                              with VRVectorMsg do
                                Camera.MoveTo(Data.x,Data.y,Data.Z);
                            end;
           VR_MOVETO      : begin
                              with VRVectorMsg do
                                UserMgr.UserMoveTo
                                        (101,Data.x,Data.y,Data.z);
                            end;
           VR_CREATE_NOTE : begin
                              with VRVectorMsg do
                              begin
                                ObjCounter:=VRVectorMsg.handle;
                                Manipulator.ObjectCreate
                                            (ObjCounter,'note.rwx');
                                Manipulator.ObjectSelect
                                            (ObjCounter,false);
                                Manipulator.ObjectMove
                                            (Data.x,Data.y,Data.z);
                              end;
                            end;
.
.
.
```

A future enhancement in the architecture will be to handle the entire communications from a thread, in a multitasking environment. This will make the handling of messages more smooth and easy.

10 IMPLEMENTATION MODEL

The implementation of the system was done on a standard PC. The IBM OS/2 Warp 3.0 and the beta version of Windows 95 operating systems where chosen. In .the later part of the development process Windows 95 was released and used during the completion of the project. Though not as stable as OS/2 the Windows 95 environment was chosen because it supported the Watcom compiler better, also the graphic drivers for OS/2 was not of a good quality. This will perhaps change with later releases of the OS/2 operating system. Because of the systems object oriented design it will not be an impossible task to move it to another platform.

IT-tools	Product	Process
Operating System	Microsoft Windows 95 (IBM OS/2 Warp 3.0)	System services
Render engine	Criterion, Renderware 1.4	Real-time 3D graphics.
Communications	Dwinsock 1.4 Freeware Delphi component	Connecting and exchanging information from a TCP/IP network.
VR-Server development	Watcom C/C++ 10.0	Compilation and linking the graphics library to the VR-Server application.
User interface development	Borland Delphi 1.02 (16 bit)	Integration of VR-Server network functions and user interface, to a single application.

Table 12 - Implementation IT-tools

10.1 Rendering engine

A survey of available rendering engines for the PC platform was done before the development started. Table 13 shows the tested engines and their properties.

3D Engine	Environment	Positive	Negative
Autodesk Cyberspace development kit 2.0	Intel, C++	Object oriented programming library Integrates well with Autodesk products.	Slow performance on non accelerated PC systems.Requires a high end PC and specialized graphics.
AVRIL	Intel, C++	Shareware, good programming API.	Poor rendering capabilities. Requires device driver development.
Superscape	Intel, C++	Completely integrated enviroment with 3D editor and development system	Expensive.
Intel 3DR	Intel, C++	Freeware. Good rendering capabilites. Optimized for the Intel Pentium series of processors	Basic API only. Handling of 3D objects must be done by own program.
Renderware	Multiplatform, C++	Very extensive API. Good rendering capabilities. Portable. Student version.	Expensive runtime licences.

Table 13 - Evaluation of rendering engines

Renderware was chosen as the systems rendering engine, because of it's speed, portability and good programming API. The expensive runtime license is just a startup cost and will probably pay off when a sufficient volume is reached.

10.2 Communications

TCP/IP Communication in the system was implemented using a Delphi component suit named Delphi Winsock Components Version 1.42 developed by Ulf Söderberg, Marc Palmer and Keith Hawes. This is a freeware component freely available on the Internet.

The first version received and tried wasn't very stable and was allmost discarded until the 1.42 release was available. This version proved to be very reliable and all earlier problems disappeared.

There was no need to use a commercial library because this component functioned perfectly in the Delphi environment.

10.3 VR-Server development

The Watcom compiler was chosen because it was a relatively cheap development system compatible with the renderware library. It also had the ability to create Pentium optimized 32-bit Windows executables which can be executed without the need for the WIN32S subsystem.

The downside with the 10.0 version of the Watcom compiler is the lack of visual development tools. The 10.6 version now includes a visual development environment.

10.4 User interface development

The choice of visual development environment was the Borland Delphi 1.02 system. Delphi is a visual development system which combines Visual Basic's ease of use and the speed of a native code compiler.

Delphi uses an extension of Pascal called Object Pascal. This is a complete object oriented development system, which also provide the ability to create reusable components in the same way Visual Basic can use VBX, OCX components created in C/C++.

Delphi also encapsulates the Windows API in the Visual Components Library (VCL), which makes it possible to create applications that can be used on the entire range of Microsoft's operating systems. It also shields the developer from the API.

Speed is needed when the system has to handle networking and control of the VR-server.

11 SYSTEM PROPERTIES

The system- and hardware properties are listed in table 14 and 15

Property	Capacity
Geometric model	Depending on the speed of the system there is no limitation of the size of the model.
Shading model	Flat, Gouraud and texture mapping
Light model	Directional, Spot, point and omni source
Geometric model fileformats	DXF, 3DS and RWX (Future enhancement VRML)
Users	2
Networking	TCP/IP Socket communications
Database	Custom. Can be adapted to use any commercially available relational database, Access, Paradox
Multiuser functions	Chat window, comment attachment, and note creation. Object modification. File transfers.

Table 14 - Software properties

Property	Minimum	Recommended
System	486DX50	Pentium
Memory	8 Mb, 10 Mb available on disk	16 Mb, 10 Mb available on disk
Input devices	Mouse	Mouse
Output devices	VGA compatible graphics card.	Accelerated 32 bit graphics system with 256 or more colors

Table 15 - Hardware properties

12 DISCUSSION

12.1 Possibilities

By designing a system that connects two or more users together independent of distances new ways to communicate in the design process is created. The geometric model can be shared, viewed and edited in realtime avoiding mistakes that can arise from misunderstandings between the members in the design team.

The system would also make it possible to invite new participants, from all over the world which could work more close in the design almost as if they were present in the same office.

The low cost system brings distributed VR to the desktop without a big price tag. This means that more persons can use VR as a tool in the design process.

12.2 Market

What market is there for this system? I think that the market share for distributed VR has not been fully explored because of the belief that VR is very expensive and demanding a lot of hardware. This system shows that is quite possible to design a system that can be easily and effective on a normal personal computer. Possible markets are architects, engineers, and designers.

It is also my belief that because of the systems ease of use it is possible to use the system with all interested parties in a project including executives and other non-engineering staff.

It should though be emphasized that deeper collaboration demands retentive underlying application models and concept definitions.

13 THE FUTURE

The ultimate goal is to make a system that can be completely portable and usable over a range of platforms. One way to implement this is to convert the system to a Netscape plugin. This integrates the system with the familiar Netscape browser and uses the same interface as the browser, making it easy to use. Another way is to make the application into a Java applet, that can be used on all platforms. The problem with this approach is that the Java interpreters today are not very fast, making it impossible to write fast VR-applications. A solution to this problem is to implement the VR Server as native code and create a Java class that can communicate with the server. The user interface can then be implemented in Java code.

The most widespread operating system for personal computers today is the Microsoft Windows 3.x/95/NT. The next development steps in this environment is to make the entire system a 32-bit application. The application would then be able to handle communications in a separate thread. The performance of the rendering engine will increase using DirectX drawing routines in the Windows 95/NT operating systems.

To make it easier to collaborate in the VR-Model the chat window could be enhanced to support audio and video channels over the internet.

A promising area in this environment is OLE Automation (ActiveX automation). This is a technology where components and applications publish their objects and methods in the system, making it possible for other applications to use methods from objects inside other applications or components. The components in the VR-system could be published and used for example by an internet browser, or directly inside a word processor.

14 DICTIONARY

32-bit application	An application that executes 32 bit wide instructions in the computer processor
API	Application Programming Interface. A set of functions and procedures to communicate with a library or service.
C++	Object oriented Programming language. Extends standard C with object oriented syntax.
Class	Defines a template for an object.
Component	Compiled module that extends the Delphi environment.
DLL	Dynamic Link Library. Library containing functions and procedures, that are linked at runtime. If not used the library is unloaded.
DXF-file	Drawing Exchange file. An AutoCAD file format for importing and exporting CAD-drawings between different systems.
Event method	A special procedure or function in the Delphi enviromnent that responds to messages from the Windows operating system.
Events	An event occurs for example when a user clicks on the mouse button. An event triggers a method in an object
Goraud shading	A special rendering technique to make the faces on an object appear smooth.
Handle	A variable/pointer used to identify an object or a Windows object.
Library	A collection of functions and procedures.
Multitasking	The ability of the operating system to execute to programs simultaneaously on one processor
OCX	OLE Custom Control. An extension component for Microsoft Visual Basic 4.x and higher development system.
OLE	Object Linking and Embedding. Microsoft standard for sharing applications and data.
OLE Automation	A standard enabling programs to access procedures and functions from each other.

Operating system	The software used to administrate the system resources in the computer.
TCP/IP	Transmission Control Protocol / Internet Protocol
Texture mapping	The ability to map pictures on to objects in the 3D system
Thread	A separate flow of execution in an application.
Unit	A compiled collection of classes and functions used in the Delphi development system
VBX	Visual Basic Extension. An extension component for the Microsoft Visual Basic 3.x development system.
VCL	Visual Component Library. The Delphi object model. Encapsulates the Windows API.
WIN32S subsystem	A special extension to the Windows 3.xx environment that makes it possible for 16 bit Windows to execute 32 bit programs.
Windows sockets	An API to encapsulate the socket specification when communicating with the TCP/IP protocol.
VR	Virtual reality.
VRML	Virtual Reality Modelling Language. An internet standard implementing 3D models.

15 REFERENCES

Borland Interantional, Inc. (1995c), "Delphi Object Pascal Language Guide"

Borland International, Inc. (1995a), "Delphi Component Writer's Guide"

Borland International, Inc. (1995b), "Delphi User's Guide"

Christiansson P. (1995), "Knowledge communication in the global network". Position paper for the july 16-20 1995 Workshop on Research Directions in Architectural Computing. Published as a Chapter in a book from KLUWER in June 1996

Criterion Software (1995), "RenderWare V1.4 API Reference"

Mathiassen, Lars and Munk-Madsen, Andreas and Nielsen, Peter Axel och Stage, Jan (1993), "Objektorienteret Analyse", Forlaget Marko ApS, Aalborg

Microsoft Corporation (1992), "Windows 3.1 API Reference" (Help file)

Olsson, Ingmar (1992), "En bok om C++", Almqvist & Wiksell Förlag AB

Preece, Jenny (1994), "Human-Computer Interaction", Addison-Wesley Publishing Company

Pärletun, Lars Göran and Hansson, Pål, and Karlsson, Göran (1989) "Datorgrafik och CAD-teknik", Studentlitteratur

Silicon Graphics, Inc(1996), "InPerson 2.0" , http://www-europe.sgi.com/Products/software/InPerson/

Swedish Institute of Computer Science (1996), "The Distributed Interactive Virtual Environment (Dive)", http://www.sics.se/dce/dive/dive.html

Ulf Söderberg, Mark Palmer and Keith Hawes (1996), "Delphi Winsock Components" (Help file)

Watcom International Corp. (1995), (Help files)