Vrije **U**niversiteit **B**russel
Faculteit Wetenschappen - Departement Informatica
Academiejaar 1999 - 2000

Het modelleren van een Virtuele Omgeving op het Internet

De Baene Christophe
chdbaene@vub.ac.be

Proefschrift ingediend met het oog op het behalen van de graad van Licentiaat in de Informatica

Promotor: Prof. Dr. Olga De Troyer

-

Vrije **U**niversiteit **B**russel
Faculteit Wetenschappen - Departement Informatica
Academiejaar 1999 - 2000

Het modelleren van een Virtuele Omgeving op het Internet

De Baene Christophe
chdbaene@vub.ac.be

Proefschrift ingediend met het oog op het behalen van de graad van Licentiaat in de Informatica

Promotor: Prof. Dr. Olga De Troyer

Samenvatting

Virtuele omgevingen vormen vandaag het onderwerp van academisch onderzoek, bijvoorbeeld voor educatieve doeleinden, ontspanning, e.d. Bij virtuele omgevingen komen heel wat nieuwe ideeën en toepassingen kijken, vooral wanneer het gaat om gedistribueerde virtuele omgevingen op het Internet.
Omwille van de uitbreiding van het Internet, het feit dat bandbreedte toeneemt in capaciteit en dat computers vandaag krachtiger worden, is er een toenemende vraag naar een methodologie voor het modelleren van virtuele omgevingen. Wat dit laatste betreft, staat de technologie nog niet helemaal op punt of ten volle ontwikkeld. Toen ik de nodige research uitvoerde, stelde ik inderdaad vast dat er nog maar weinig literatuur bestaat over het modelleren van een virtuele omgeving.
Het hoofddoel van mijn thesis bestaat erin een mogelijke methodologie voor te stellen voor het modelleren van een virtuele omgeving. Ik heb me in het bijzonder geconcentreerd op interactieve omgevingen en niet zozeer op statische omgevingen waar het slechts om een visuele voorstelling gaat van de informatie.
Hoewel er nog enkele valkuilen dienen overwonnen te worden, kan deze thesis beschouwd worden als een eerste stap in de richting van het creëren van een methodologie voor het modelleren van interactieve virtuele omgevingen.
In het eerste hoofdstuk geef ik kort een definite van de term virtuele realiteit en de toepassingen ervan in de dagdagelijkse praktijk. Het tweede hoofdstuk beschrijft de bouwstenen voor het construeren van een virtuele omgeving op het Internet. En tenslotte stel ik in hoofdstuk drie een methodologie voor gaande van de eisen van de gebruiker tot implementatie design.

-

Vrije **U**niversiteit **B**russel
Faculty of Science - Department of Computer Science
Academic Year 1999 - 2000

Modeling Virtual Environments on the Internet

De Baene Christophe
chdbaene@vub.ac.be

Dissertation submitted in view of obtaining the degree of Licentiate in Computer Science

Promoter: Prof. Dr. Olga De Troyer

Abstract

Virtual environments are currently the subject of academic research, e.g. for educational purposes, entertainment, etc. Virtual environments certainly engage new ideas and applications, especially when talking about distributed virtual environments on the Internet.
Because of the growth of the Internet, the fact that bandwidth is extending in capacity and that computers have nowadays more computational power, there is a greater need for a methodology for the modeling of virtual environments. Regarding the modeling of a virtual environment, not all the necessary technologies are fully mature and fully understood. And when I carried out researches, I found that there is little literature on modeling virtual environments.
The main objective of this paper is to provide a possible methodology for modeling a virtual environment. I focused in particular on interactive environments, and not so much on static environments which only provide a visualization of the information (visual representation). Although there still are some pitfalls to overcome, this paper constitutes a first step towards the setting-up of a methodology for the modeling of interactive virtual environments.
In the first chapter I briefly discuss the definition of virtual reality and its applications in everyday life. The second chapter describes the building blocks for constructing a virtual environment on the Internet. And, finally, in chapter three I propose a methodology from user requirements to implementation design.

Acknowledgements

I would like to express my gratitude to my promoter, Prof. Dr. Olga De Troyer, for her guidance and for having had so much faith in me during the accomplishment of this paper.
I also would like to thank my girlfriend for her unrelenting support and encouragements. I could not have survived the last four years without her enthusiasm and energy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the last couple of decades we have become very familiar with a phenomenon that is prevailing in the computer world: the Internet. Nowadays we simply cannot talk about computer science without mentioning the Internet. It is growing at a very fast pace, and the enabling technologies with it. We do not only find ourselves in the midst of a rapid high technology evolution, but also the designing of interactive interfaces and applications undergoes constant mutation. One such phenomenon, which is being fervently developed today, are (distributed) Virtual Environments (VEs) on the Internet. VEs provides an ideal human/computer interface for a wide variety of applications, ranging from educational benefits to ways of expressing our visions and insights even more vividly. With VE we are laying the foundations for new ways of seeing and experiencing the world and more importantly, it will change the way in which man interacts with computers.
In this chapter we define 'virtual environment', which is a trying matter, and illustrate a number of application domains in which VEs are being realized, in particular for the Internet.

1.1 What is Virtual Reality?

A succinct historical note ... Virtual Reality (VR) is not altogether new, as the popular media would like us to believe today. In fact, it all started with interactive 3D computer graphics, which are being developed ever since the computer was invented. After 30 years of development, 3D computer systems can eventually generate sufficiently complex images with reasonable speed in order to produce a broad spectrum of interactive applications. At one point computer graphics traded off interactivity in order to become more photo-realistic. Scientists tried to understand the physics and invented algorithms to simulate the physics (e.g. to simulate walking, running, movement of living beings) and they coded these methods into a computer language. Finally after some years of research, the combination of two technologies, interactivity and 'realistic' rendering techniques, brought to life the early VR systems.

There are many tentative definitions of the term Virtual Reality (VR) and they all have different interpretations. There are probably as many definitions as there are application domains for Virtual Reality. I have come across a plausible definition phrased by Steve Bryson (NASA Ames):

'*Virtual Reality is the use of computer technology to create the effect of an interactive three-dimensional world in which the objects have a sense of spatial presence.*

Many of the definitions of VR, however, are defined in terms of a collection of hardware devices (Head Mounted Display, Glove Input Device, etc.). For example:

> '*Virtual Reality is electronic simulations of environments experienced via head mounted eye goggles and wired clothing enabling the end user to interact in realistic three-dimensional situations* [1]'.

A device-driven definition of virtual reality is, however, unacceptable: it fails to provide any insight into the processes or effects of using these systems. Moreover, one of the things that this definition implies, is that, if you do not have the mentioned hardware, it is classified as non-VR. This is not an acceptable definition if we talk about VR on the Internet, because the majority of Internet users do not possess the required hardware. Most of the users do only have the commonly accessible keyboards and mouse to interact with the computer.

[1] Program from Invisible Site-a virtual sho, a multimedia performance work presented by George Coates Performance Works, San Francisco, CA, March, 1992

Therefore a definition on the effects of VR is more suitable than looking at it from a mere technological viewpoint. A more suitable definition would be:

> '*Virtual Reality is a way for humans to visualize, manipulate and interact with computers and extremely complex data* [2]'.

Virtual Environments (VEs) are computer-generated worlds with which the user can interact. Users can move around, look at, and manipulate graphical objects using input devices ranging from keyboards and mouse to HMDs (**H**ead **M**ounted **D**iplay) and cybergloves.
One of the main research issues of virtual reality, is the idea of **distributed VR**. This means that the simulated world runs on several computers, and not only on one computer system. This means that several computers can interact in real time with the same virtual world connected over the internet. In literature they also talk about multi-user VR/VE or collaborative virtual environments.

1.2 Types of VR Systems

1.2.1 Immersive VR

In Immersive VR the participant gets an illusion of the world through stimuli such as vision, hearing and touch. This is done through special hardware devices, such as head mounted displays (HMD), gloves, etc. So, most of the time in Immersive VR one is completely isolated from vision and sound from the real world.
It is clear that this type of VR will not be used for the average user on the WWW, because this approach of VR often requires a high performance graphics workstation. The user will merely use the Non-Immserive VR (see section below).
This does not mean that Immersive VR cannot be applied over the WWW. It can be useful, for example, in the medical world for tele-operation.

1.2.2 Non-Immersive VR

Another form of VR consists in generating vision and audio on a desktop monitor and speakers. So, you use the monitor to display the virtual world. Non-Immersive VR is also called *Desktop VR* or *Window on a World* **(WoW)**, because you experience the world as if you were looking through a window.

[2] From 'The Silicon Mirage: The Art and Science of Virtual Reality' by Steve Aukstakalnis & David Blatner

This approach is widely used, certainly on the WWW, because of the fact that you do not need expensive hardware, nor as much computating power as with Immersive VR.

1.3 Applications

The applications being developed for VR run a wide spectrum, from games to architectural planning. Many applications are worlds that are very similar to our own, like CAD or architectural modeling. Some applications provide ways of viewing from an advantageous perspective not possible with the real world, like scientific simulators, air traffic control systems.

The whole field of scientific visualization e.g. has changed significantly due to the new technologies in virtual reality. Many applications of science are situated for presenting complex multidimensional data sets. In the beginning visual representations were either simple animated sequences or complex static representations of some event. With the growth of technologies we are capable of displaying a very large set of data in a dynamic way so that the user can interact.

Virtual environments have now emerged where the user can actually become immersed in the multidimensional data set. A domain in which visual representation is used, is chemistry. In [10] they use VRML worlds (which will be discussed in chaper 2) for visualizing complex chemistry models.

Also in medicine there are many useful applications that deal with VR. But in medical VR we merely talk about telepresence. The difference between VR and telepresence is that with telepresence the interaction is with real objects through real images, whereas VR deals with virtual images.
One good example of a medical application that can be realized over the Internet is so-called Remote Operation.

In the educational field: virtual science laboratories, virtual planetariums, etc. In the field of entertainment: immersive games, etc.

VR is clearly more than just interacting with 3D worlds. By offering presence simulation to users as an interface metaphor, it allows operators to perform tasks on remote real worlds, computer generated worlds, etc. The simulated world does not necessarily have to obey natural laws of behavior. This evidently means that nearly every area of human activity is a candidate for VR application and it shows convincingly that virtual environments will find their way into our lives.

Chapter 2

Issues and technologies for NVEs on the Internet

In this chapter we describe first of all the different kinds of network topologies that are available and the problems associated with them, and secondly the different protocols and technologies that are available for building a networked virtual environment (NVE) on the Internet.

2.1 Networked Virtual Environments

One of the great things about distributed VR is that multiple users can navigate and communicate simultaneously. Especially when we are building large-scale VEs, there are a number of factors which have to be considered. There are several aspects of network communication applicable to VEs, namely: bandwidth, distribution schemes, latency and reliability, which are discussed in the following paragraphs [8].

2.1.1 Bandwidth

Bandwidth is an important network aspect for constructing a virtual environment on the Internet, because it will influence the richness of the virtual environment, in particular the dynamic aspects of the virtual environment.
When the available bandwidth is defined, it will have an impact on the number of simultaneously participating users using the shared virtual world, because the available bandwidth is proportional to the number of users (hosts).
In Local Area Networks (LANs), bandwidth is not a major issue, because technologies, such as standard Ethernet (10 Mbps), are cheap and the number of users in a LAN for a shared virtual environment is limited. Whereas the bandwidth of Wide Area Networks (WANs) is generally limited to T1 (1.5 Mpbs) and the number of users is much larger through the internet.

2.1.2 Distribution

There are essentially three ways to transmit from a source to multiple recipients on the Internet, namely unicast, multicast and broadcast.

- **Unicast** is a point-to-point transmission mode where the source has to send an individual copy of a message to each requester. This approach is very inefficient for a large number of requesters (= the number of users participating in the shared virtual environment), because it requires the sending computer to retransmit a copy of the information to each recipient. This means that the network transmits duplicates of the information, which results in losing precious bandwidth.
  Virtually all Internet content (such as text, graphics, audio and video) is delivered as a unicast stream. This means that, for example, a user accessing a webpage receives an individual data stream directed to his PC.
- **Broadcast** is a one-to-all transmission procedure where the source sends one copy of the message to all systems. This implies that all systems receive it whether they

wish to receive it or not.
This means that all hosts must examine a packet, even if the information is not intended for all of them, thus they incur a major performance penalty because they must interrupt operations in order to perform this task at the operating system level.

- **Multicast** is a standard procedure which provides a solution to the problems of unicasting and broadcasting. With IP multicast, applications send one copy of the information to a group address, thus reaching whomever desires to receive it. So the information is sent to a set of hosts.
  For example, when we have a distributed VR application, the host sends his current position in the virtual world to every host in the VR world to update his local copy of the world.

  MBone stands for Multicast Backbone and is used for distributing live audio, video and other packets on a global scale. The MBone is a virtual network layered on top of the physical Internet to support IP multicast where it uses routers (mrouters) that support multicast.

2.1.3 Latency

Network latency is a synonym for delay, and is an expression of how much time it takes for a packet of data to go from one point to the other. It seems that data should be transmitted directly between two points, without any delay, although there are some factors that contribute to network latency. For example, the medium (like fiber optic cable) introduces some delay and also a router can cause some delay, because it has to examine packets.

2.1.4 Reliability

Reliability means that a system can logically assume that sent data is always received correctly, which avoids to re-send the information. If we want to guarantee delivery, the underlying network architecture must use acknowledgment and error recovery, which unfortunately introduces some delay (= latency).
The most known transport protocols are TCP (**T**ransport **C**ontrol **P**rotocol) and UDP (**U**ser **D**atagram **P**rotocol). Essentially, TCP is a connection-oriented which ensures a reliable delivery, while UDP is connectionless and offers no guarantees, but which causes less overhead.

2.2 Network topologies

When we have a distributed virtual environment, each user is represented by an entity (avatar) in the shared virtual world. The main idea is that, when an entity changes (the entity has moved in the world) or when there is an entity that modifies the world, an update must be done to all users who are participating in the shared virtual world [4].

Several network topologies are possible, depending on the requirements of your distributed virtual world. There are mainly 2 types of network architectures, namely *peer-to-peer* and *client/server* architecture. A peer-to-peer network is a network in which each computer has equivalent capabilities and responsibilities. Each party can initiate a communication session. Whereas in a client/server architecture a computer is either a client or a server. A program, a client issues a service request from another program; the server (also called a daemon [1]), fulfills the request. So, essentially the server is activated and awaits client requests.

Peer-to-Peer Topologies
In a peer-to-peer system design, there are a set of computers (users) connected over a network (internet) in which each peer can send messages directly to any other peer. If you have a network that only supports unicast, each peer sends a message to any other peer. If we have N computers, we have to send a message to each $N-1$ computer, when an entity changes. This means $O(N^2)$ updates messages during every simulation step. If you have a network that supports multicast, a peer just has to send a single multicast message to all the other peers. With this approach, we have a total of $O(N)$ messages, because every computer must still process every update message from the other computers.

Client/Server Topologies
Communication between client workstations is managed by message servers. Clients do not send messages directly to other clients, but instead send them to servers which forward them to other clients and servers participating in the same distributed virtual environment. An advantage of the client/server approach is that servers can examine messages before propagating them to other clients. For instance, a server may determine that particular update message is relevant only to a small subset of clients and then propagate the message only to those clients or their servers.

[1] Is a program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive.

2.2.1 Dead Reckoning

Dead reckoning is an algorithm used to reduce the number of messages sent between the hosts in order to reduce the bandwidth traffic. Certainly with peer-to-peer topologies, where we have to send many messages, this algortihm can be used.
To illustrate the dead reckoning algorithm, we will give an example of a distributed "dogfight" game with n players, where each player can control his own space ship. When a player X moves his own ship, the player sends a message to the other $N-1$ players to tell that he has moved his ship. Similarly, if Player Y moves his ship, he will send a message to the other $N-1$ players. When all players have moved once, this gives a total of $N*(N-1)$ messages to be sent, like the peer-to-peer topology with a unicast system. Now to reduce the number of messages, player X sends the ship's position and velocity (direction and speed). If we move the ship again and we move it along the same path as the previously established velocity, player X does not need to tell the other $N-1$ player to give his new position. The other $N-1$ players know where the ship of player X is, because they have the ship's old position and current velocity. This calculation for establishing the ship's position based on the last-known velocity, is called *dead-reckoning*.

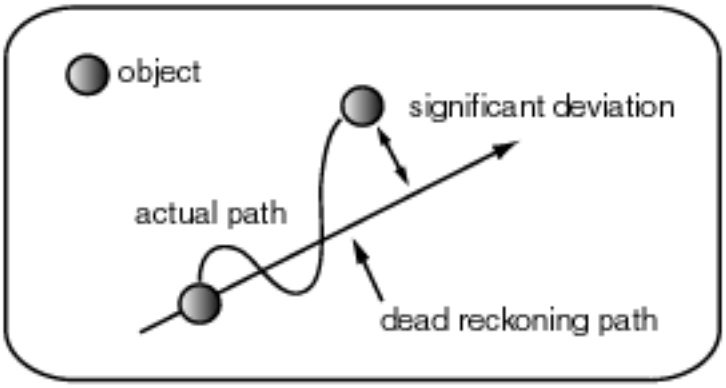


Figure 2.1: Dead reckoning: path and deviation

We have 2 kinds of objects: the one where we have direct control over the object, is called the "live" object, and the other objects are called "ghost" objects [2]. To accomplish the *dead-reckoning* algorithm, each player maintains a database with applies the *dead-reckoning* algorithm to the "live" and "ghost" objects. Also from the "live" object, we compare and determine when the object has changed significantly in value. If the difference is significantly, the program sends a message to the other players to update their "ghost" objects with the new position and velocity (see figure 2.1).

[2] called a "ghost" object because the object is controlled by a different process

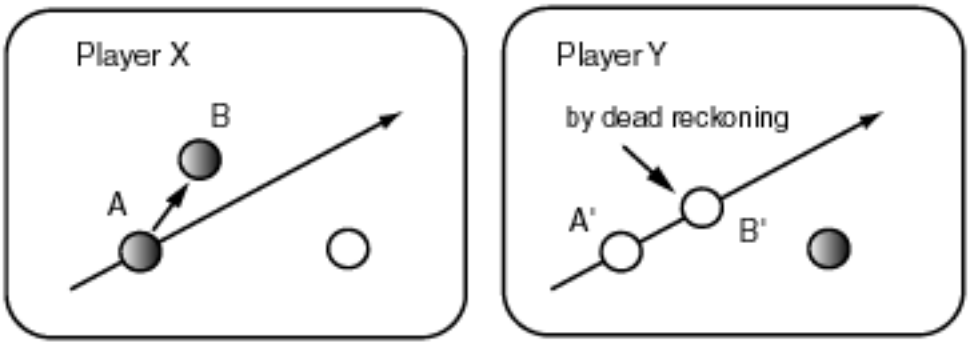


Figure 2.2: Dead reckoning: no significant deviation

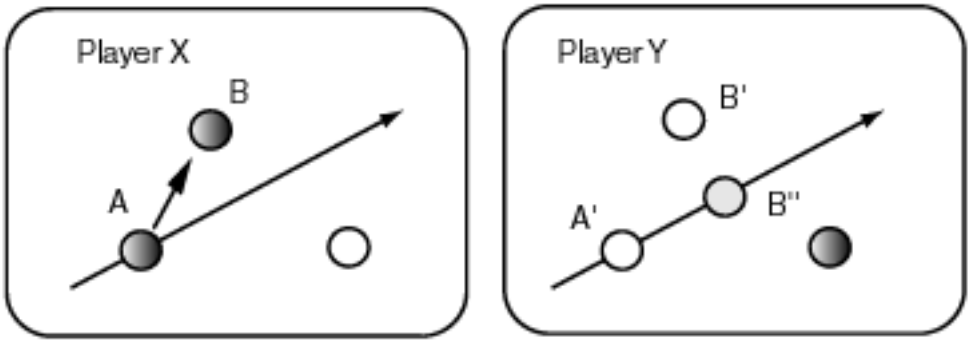


Figure 2.3: Dead reckoning: significant deviation

Figure 2.2 gives an example with 2 players, player X and Y. When player X moves from A to B, and the movement exceeds not a certain value from the dead-reckoning, no message is sent. So as you can see from figure 2.2, there is no update to player Y, it stays on the dead reckoning path. Note that in the figure a full circle means a "live" object whereas the other circle is the "ghost" object. When player X moves (live object) then this is a ghost object for player Y.

When we deviate significantly from the *dead-reckoning* path, player X will send a message to all other players, in this case player Y, to update their "ghost" objects (ships) with the new correct ship position and velocity. So as you can see from figure 2.3, the position of the ghost object is B' and not B", which is the dead reckoning path.

2.3 VRML

The choice of 3D scene description language has again been influenced by the fact that our primary network target is the Internet and the WWW hypermedia system it supports. The predominant language used for text documents in the WWW is the HyperText Markup Language (HTML). A significant effort has been underway to produce an equivalent 3D description language, namely VRML. VRML is an acronym for "**V**irtual **R**eality **M**odeling **L**anguage" and it is an International standard file format for describing interactive 3D multimedia on the Internet.

The VRML standardisation began with a small working group defining a set of requirements for a scene description language suitable for the WWW. The first release of VRML was version 1.0 created by Silicon Graphics, Inc. Mainly in this version the worlds are static.
VRML version 1.0 is a simple graphics language based on a model of a scene made up from a series of transformation nodes, i.e. spatial position markers. Each node can consist of a number of sub-nodes forming a tree. A set of geometry nodes, e.g. cube, cone, etc., and they have a set of properties including color and material. VRML, like HTML, supports the notions of linking and embedding, and allows scene authors to embed other media types, or to link to other 2D or 3D documents.
In the next release of VRML, namely 2.0, there are more features, like sound, animated objects, etc. and especially what interactivity means to the visitor. For example, the ability to have sensors, scripts (Java or JavaScript) for behaviors, event routing, etc.
VRML 2.0 was reviewed by the VRML moderated e-mail discussion group and later adopted and endorsed by many companies and individuals. In December 1997, VRML97 replaced VRML 2.0 and was formally released as International Standard ISO/IEC 14772.

Motion capture systems capture either magnetically or optically the motion of a person and translate that into computer data which coordinates to specific placements on a 3D model. Therefore, the movement of a virtual human in a virtual environment is exactly the same as the actual human movement. WHD (**W**eb **H**uman **Director**) is a simple motion capture system, allowing HANIM humanoids to be animated using VRML and Java [3]
VRML and Java can communicate with each other through the ***E**xternal **A**uthoring **I**nterface*. This means that you can control the VRML world embedded in a web page from a Java applet on the same page.

[3] Humanoid Animation Group
http://ece.uwaterloo.ca/ h-anim/

With VRML we can build large-scale virtual environments using the Internet and the World Wide Web. Until now, VRML uses the HTTP (**H**ypertext **T**ransfer **P**rotocol) protocol and this is insufficient for large-scale environments. Capabilities, such as peer-to-peer communications plus network monitoring, need to be combined with the client/server capabilities of http. Therefore VRTP (**V**irtual **R**eality **T**ransfer **P**rotocol) is designed to support interlinked VRML worlds. So, it will be used with 3D browsers to navigate and join large interactive distributed VRML worlds. More informatio can be found at *http://www.stl.nps.navy.mil/dis-java-vrml/vrtp/*

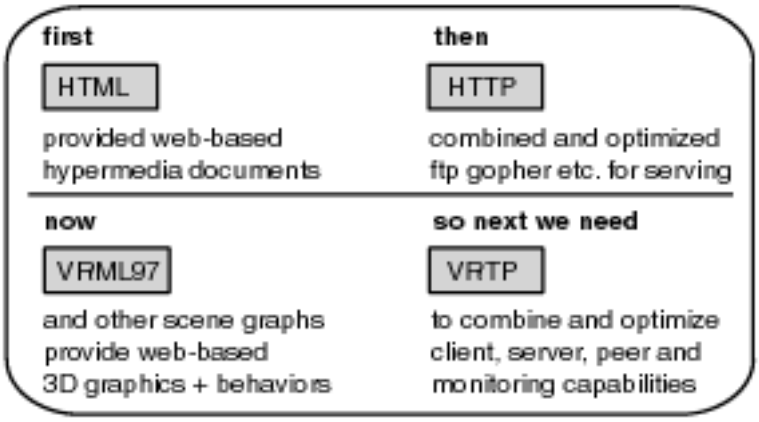


Figure 2.4: VRML and VRTP

2.4 X3D

X3D is an acronym for **E**xtensible **3D** and is an extensible 3D graphics specification that extends the capabilities of VRML97. The name X3D has been chosen above VRML-NG (VRML Next Generation) to indicate the integration with XML (**E**xtensible **M**arkup **L**anguage)

> XML is a text-based markup language for interchanging data on the WWW, intranets, and elsewhere. As with HTML you identify data using tags [4], but in XML the tags tell you what the data means, in contrast to how to display it like in HTML. For example, the following XML data could be used for a message application:

[4] identifiers enclosed in angle brackets, ¡ ... ¿

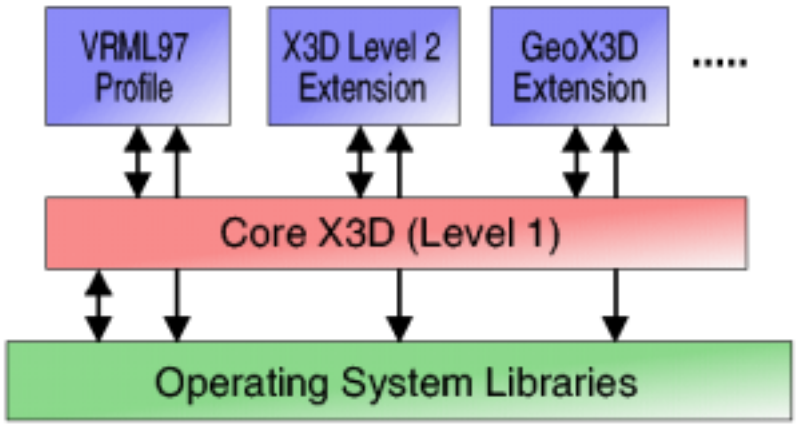
```
<message>
    <to>to@domain</to>
    <from>from@domain</from>
    <subject>Example of XML</subject>
    <text>
       Here comes some text of the message
    </text>
</message
```

As mentioned above, XML specifies how to identify data, not how to display it. XSL (Extensible Stylesheet Language) is essentially a translation mechanism that lets you specify how to convert an XML tag so that it can be displayed (for example in HTML). This allows you to specify several different XSL formats that can be used to display the same data in different ways.

X3D is being developed by the X3D Task Group, which is an official task group of the Web3D Consortium [5]. And X3D has the following goals to accomplish: 1) *Integration with XML:* this means that you can express the VRML97 capabilities using XML 2) *Backward compatibility with VRML97:* this means that you can still view VRML content using X3D technologies 3) *Componentization:* enables a lightweight core that can be easily extended to provide new functionality 4) *Extensibility:* use components to add functionality, new nodes and corresponding implementation code to the core.
We mean by componentization that we identify the most crucial functionality of VRML and encapsulate this into a small extensible, lightweight Core which all applications of the standard must support. On top of the Core X3D, we can have extensions and profiles. An extension is a single unit and extends the Core's functionality in some specific domain.

The core can be extended using profiles which in turn enable more complex or application-specific capabilities. Like the figure below, we can build a full VRML97 extension on top of the X3D Core, H-Anim, EAI or GeoX3D extensions or other extensions, like GeoX3D. This approach (componentization) gives the ability to extend X3D be adding new functionality.

[5] Web3D Consortium: http://www.web3d.org



VRML is just an encoding for representing 3D where XML is essentially another syntax. So, below is an example of a possible mapping:

VRML97: `DEF MyView viewpoint { position 0 0 10 }`
XML: `<viewpoint id='MyView' position='0 0 10'>`

Or if you want to use PROTOs [6] using XML, this would look as follows:

```
<Proto type="myBox">
  <Field id="mySize" type="vec">
  <Box size="mySize">
</Proto>

...

<ProtoUse type="myBox" DEF="aTwoThreeTenBox">
  mySize="2 3 10"
</ProtoUse>

...

<ProtoUse USE="aTwoThreeTenBox"/>
```

[6] Using PROTOs enables you to make new node types for use in the VRML world.

2.5 DIS

DIS is an acronym for **D**istributed **I**nteractive **S**imulation and is an IEEE stanndard protocol for logical communication among entities in distributed simulations. Although initial development was driven by the needs of military users, the protocol specifies the communication of physical interactions by any type of physical entity and is adaptable for general use. Information is exchanged via PDUs (**P**rotocol **D**ata **U**nits), which are defined for a large number of interaction types.
DIS uses the the technique "dead reckoning" where it produces impressive results. Since updates are only sent when needed, the amount of traffic is greatly reduced. The system works perfectly for military simulation (where it's designed for). Since every entity is either a vehicle or projectile. More information can be at `ftp://taurus.cs.nps.navy.mil/pub/mosaic/nps_mosaic.html`.

Chapter 3

Methodology for Modeling VEs

Today there is a growth in the use of virtual environments on the Internet, e.g. for educational purposes. Hence, there is a greater demand for methodology and modeling techniques in order to describe interactive virtual environments. In this chapter I put forward a methodology for modeling a virtual environment, going from user requirements to implementation design.

3.1 Introduction

Owing to the expansion of the Internet there is also an increasing demand for virtual environments on the Internet as the number of users on the net is growing every day. The fact that bandwidth is extending in capacity and that computers have more computational power, accounts for virtual environments becoming more feasible. Moreover, users want to interact in a different way, i.e. in a virtual environment.
When a client wishes to build a virtual environment, you need to analyze mainly 2 questions, namely: *what* does the client want to do with a virtual environment, i.e. what are the possible interactions and goals to be realized, and *how* the client would like the environment to look like, i.e. the visual representation. So, you mainly need to describe what the possible *interactions* are as well as the *visualization* of the environment.
These issues are dealt with in the paragraphs below, which in fact represent a possible methodology for describing the requirements which are needed to create a virtual environment. These requirements will particularly have an impact on the implementation design, which is also covered in this chapter.

3.2 Methodology

In this section we describe the methodology for modeling a virtual environment. The methodology is a user-centered design and I chose to concentrate on the interactions between the avatar and other entities (defined further) in the environment, and between the entities themselves. So, not solely static virtual environments, which only provide a representation of the information.
The methodology essentially encompasses 3 steps: the requirements, the conceptual design and the implementation design, as indicated in figure 3.1. Each one of the three steps is discussed further.
The methodology provides as a first step an outline of the functional requirements, i.e. which interactions the environment must realize and the requirements relating to the visual representation. Of course, there are also non-functional requirements which are also dealt with in the methodology discussed further.

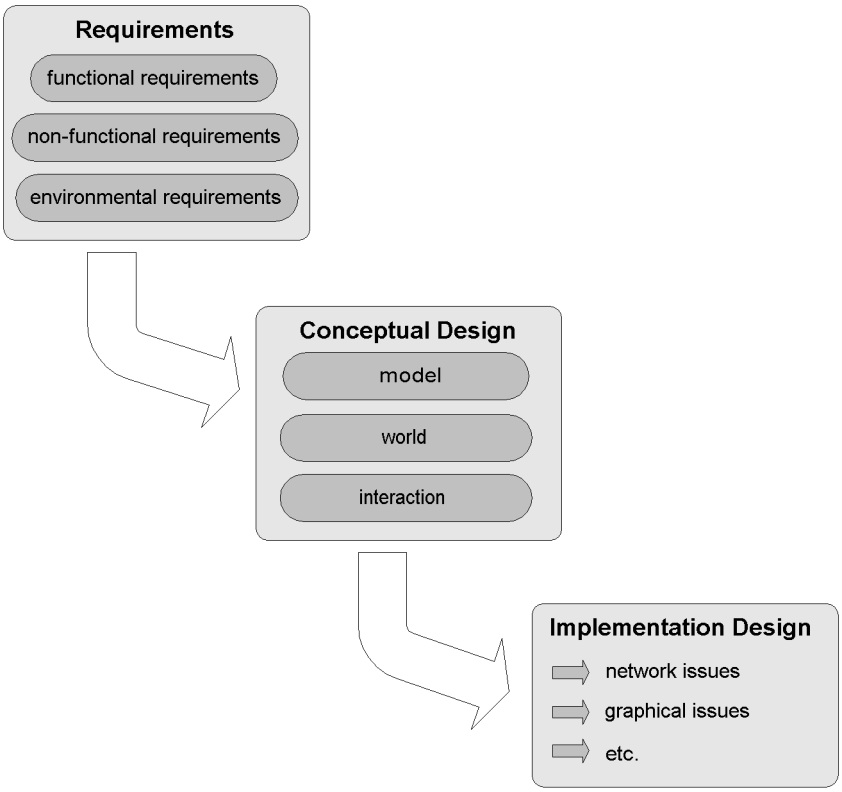


Figure 3.1: The methodology used for modeling a virtual environment

3.2.1 Requirements

At this stage, we try to gather all kinds of requirements that the client wants for his virtual environment. I have grouped the requirements into 3 categories: functional, non-functional and environmental requirements.

3.2.1.1 Functional Requirements

In this step we give a description of which kind of application we are talking about for our virtual environment, and of the functional requirements that the environment must fulfill. So we first provide a good characterizing of the application and then we look further into the functional requirements for the virtual environment.
There is a good characterizing of the type of application for virtual environments, proposed within three dimensions:

1. The *number of participants* and entities simultaneously involved in the same world.
2. The *complexity of the objects and their behaviours*, ranging from static data to those responding to participant interaction, and on to dynamically changing objects without user intervention.
3. The *degree of interaction among participants*, ranging from low (users can see each other), through medium (users can be involved in complex activities), to high (synchronized activities to achieve a common task).

It is very important to give an idea of the characterizing as indicated above, because it will have a great impact on the implementation design. The three dimensions given above are typical for a virtual environment and give an idea of the complexity of the application. Hence decisions have to be made in the implementation stage. For example, a possible characterizing would be:

> *It is a multi-user virtual environment with a maximum of 10 simultaneously interacting participants, where each user can interact with other users to communicate. There must also be the ability to interact with objects of the virtual world, e.g.: using an elevator, picking objects, etc.*

Now we are going to have a closer look at the functional requirements of the virtual environment. At this stage we describe the interactions the environment must fulfill.
It is not always a simple task to get an idea of the functional requirements for your environment. For example, if you want to build a virtual shopping center, it is not

immediately clear what the possible interactions and requirements are for the environment. To get an idea of the several functional requirements we write a narrative [17]. Before writing a narrative, we develop the characters (or the potential system users) and choose a theme, primary goal, task or activity. The more concretely the users, domain and world features are specified, the more applicable the story will be to the solution. Of course we can write more than one story to focus on different requirements and interactions.
After having written the narrative, we analyze the narrative and highlight all actions that can constitute a requirement. Remark, that even a line in a story gives already a set of requirements. For example, the sentence *"He turned and looked up at him angry."* would imply the following requirements:

- The ability to turn on at least two axes
- Allowing that the users (avatars) have different heights
- Allowing for a facial expression, e.g.: looking angry

After having edited and analyzed the stories, one has a good idea of the functional requirements in order to build the virtual environment.

3.2.1.2 Non-Functional Requirements

Non-functional requirements are the system constraints affecting development and design. These may include: performance, portability, operating environment, reliability, etc. Further on we give a brief discussion of the common non-functional requirements.
Following is a description of what is meant by human-computer interaction and we give a brief overview of the possible input/output devices. There are several hardware devices for VR systems available, dependent on to which degree you want to create the illusion of immersion.

Human-computer interaction is a bidirectional communication between the user and the system. It is a circular process, where the user gives some *input* and the computer responds to it by updating it to *output*. As you can see from the figure on the next page, the user observes the state of the system and accordingly plans an action and subsequently executes the task at the interface via the input devices. These actions are then transferred as input into the system which processes them, and which sends the output to the interface for presentation to the output devices.

Following is a brief description of the input/output devices available for virtual environments.

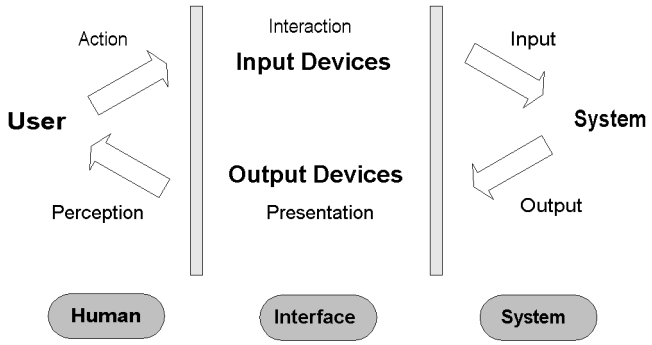


Figure 3.2: Human-computer interaction

- **Input** devices are those with which the user emits information and interacts with the environment. These include kinaesthetics (body movement of the user), acoustics (voice input) and/or graphics (video capturing with possible gesture recognition, etc.).
- **Output** devices are those which give the user information from the world. So, they stimulate our senses. The main senses used in a VR system are: vision, hearing, touch and force perception. Smell and taste are secondary.
  In any VR system vision is the essential part of perception. This can be achieved by a simple display, a monitor or with HMD (**H**ead-**M**ounted **D**isplay), etc.

Most of the time, especially users of the internet in a (distributed) VR, use desktop-VR. This consists of the keyboard and/or mouse as input channels and the monitor and/or speakers as output channels. One of the main advantages of this approach is that it does not require a powerful graphical workstation and it is cheap.

Other non-functional requirements are, for example, that the VR system should be cross-platform (the capability of software or hardware to run identically on different platforms), whether the environment should reside in a browser (browser-based), such as VRML, etc. Another requirement could be that the virtual environment must be implemented using one of the technologies discussed in the previous chapter. For example, one can opt for using DIS and VRML for building the virtual environment. Another aspect that needs to be mentioned here, is that the VR system can use Multicast Backbone (discussed in the previous chapter), because it also influences the implementation design.

3.2.1.3 Environmental Requirements

In this section we describe the requirements which are inherent to the environment itself. They are not related to technological requirements, but merely to the 'look and feel' of the virtual environment. We discuss below some issues that are important for describing the environmental requirements.

- **Viewpoint**
  Virtual environments may be experienced from two kinds of viewpoints, i.e. *egocentric* and *exocentric* viewpoint. In an egocentric viewpoint, the sensory environment is constructed from the viewpoint actually assumed by the users (figure 1). Whereas in an exocentric viewpoint, the environment is viewed from a position other than that where users are represented to be (figure 2). In this case, they can literally see a representation of themselves. In game design we use the term *first-person POV* for the egocentric viewpoint and for the exocentric viewpoint we use the term *third-person POV*.

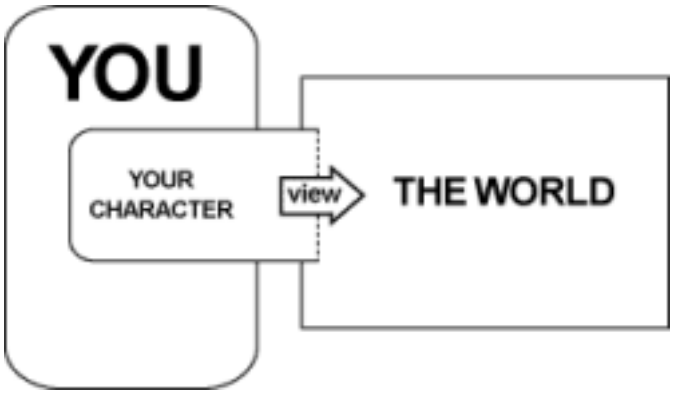


Figure 3.3: First-Person POV [6]

  Choosing a viewpoint may entail a number of consequences. First, as far as the implementation is concerned, it is easier to implement a first-person POV in a 3D world. Because, for example, issues as depth perception are easier to accomplish, i.e. you simply draw straight lines from the POV to anything they see: if it's big, it's close; if it's small, it's far way [6]. Secondly, in the first-person POV you have a higher degree of immersion than in third-person POV.

- **Appearance**
  Here we describe how the world should look like. This is essential because, if we do not specify this, the world could be literally anything, certainly with virtual

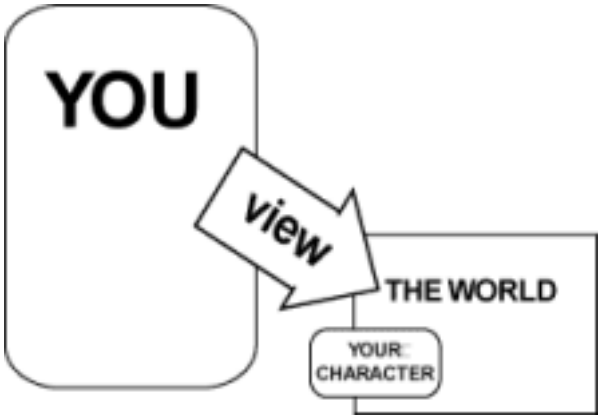
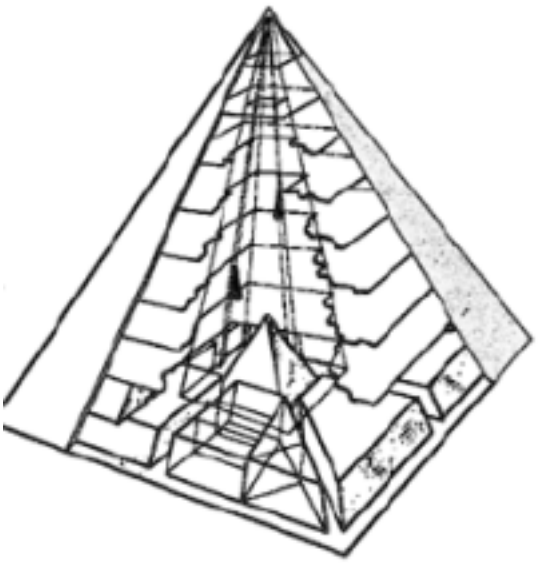


Figure 3.4: Third-Person POV [6]

environments where we have the liberty to be creative and use our imagination to the fullest.
It is obvious that we immediately create an atmosphere for the virtual environment. For example, we could phrase the following in order to get an idea of the environment that we would like to create:

> *The whole environment should take place in an authentic castle of the 17th century. There we create the illusion that from the outside, the castle looks authentic, whereas the inside of the castle looks modern and displays our information in a modern-looking way.*

Of course, there are also other methods to illustrate which kind of environment you would like to create. To which extent we go into detail depends on the kind of virtual environment we are building. If, for example, you want a dedicated virtual labyrinth, it is useful to get into detail as to how the labyrinth is constructed. For example, with a sketch like the figure below, we can illustrate a draft of the virtual environment. It shows that everything is situated in a pyramid with several floors where possibly an elevator in the middle of the pyramid goes to the different floors.



- **Visual Representation**
  Here we describe properties, such as: should the virtual environment be photo-realistic or not, how should the avatars look like, maybe for some reason you only want a set of colors that may be used in the environment, etc. Note that when you have such requirements, it is important to point them, because often these requirements have an impact on the implementation design.

3.2.2 Conceptual Design

At this stage, we describe which elements of the virtual environment the user wants to interact with, the interaction itself between the elements of the environment and the visual representation of each one of the elements of the environment.
We can categorize elements of a virtual environment into so called entities (which will be described further). We will use the concept of entities as a basis for setting up and modeling our virtual world. After that we will apply the concepts of object-orientation to the entities, which will enable us to use UML [1] for describing our virtual world. Below, we give a description of what an entity is and which kind of entities appear in a virtual environment (VE).

[1] UML is an acronym for **U**nified **M**odeling **L**anguage and is a modeling technique for OO (object-oriented) modeling

3.2.2.1 Dividing a VE into entities

A virtual environment comprises a collection of entities. We discuss here what an entity is and how we can categorize the entities that describe our world.
An entity has two parts: an attribute and a behavior. A typical collection of attributes are: location, orientation, shape, etc. The entity has a behavior, this means that it may change the state of all the attributes of the entity. For example, a robot certainly has an attribute 'location'. The way in which the location changes over time is called the behavior of the robot. According to [12] we can divide an entity into four categories.

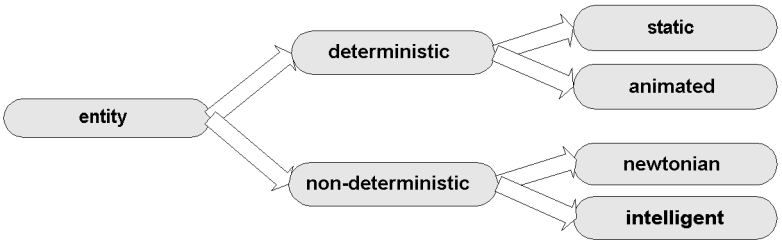


Figure 3.5: The different categories of an entity

As you can see from the figure above, we can divide entities into two basic types, i.e. with a deterministic behavior or a non-deterministic behavior. Each basic type can be further divided. An entity with deterministic behavior can be static or animated, whereas an entity with non-deterministic behavior can be newtonian or intelligent.

Deterministic behavior
Saying that an entity has a deterministic behavior means that the state is only a function of time. This means that we always know the state of the entity at any given time in the environment. For example, we could say that we want to see what the state is at time $t = 650s$, and we can do that at any simulation tick. Following is a description of the two kinds of entities that have a deterministic behavior.

- **Static entity**
  With a static entity we always know the state at any given time because the state never changes (it is static). So these kinds of entities could be streets, mountains, houses, etc., i.e. structures that cannot change in any way.
  If you want, for example, a tree in your environment and you want that the leaves fall from the tree, this is not considered as a static entity, because the state has changed. So, if you define a tree in your environment as a static entity, the tree will look the same during the entire simulation time, even when you have a notion of seasons in your environment.

- **Animated entity**
  Here the entity does change its state over time. Remark that an animated entity is only a function of time and nothing else.
  Animated entities in a virtual environment are, for example, the hands of a clock, an animated texture map for the flames of an open fireplace, the ambient light level of the environment (day/night cycle), etc. For all of these examples, you can examine the state at any simulation tick.

Non-Deterministic behavior
Entities that have a non-deterministic behavior are unpredictable. We cannot say, as for a deterministic behavior, which state a given non-deterministic entity will have at a certain time. It could be anything. For example, a human being has a non-deterministic behaviour. We make decisions for several (internal) reasons and it is impossible to predict what a human being will do at a given time. Also robots in a virtual environment that have artificial intelligence, show non-deterministic behaviors. Here follows a categorization of entities with a non-deterministic behavior.

- **Newtonian entities**
  Newtonian entities are entities that respond to changes in their environment. These entities have no will, no goal, nor any sense of intelligence. We can manipulate these entities, we can pick them up, throw them, bump them against each other, etc. and they also react to laws of virtual physics (we mean by virtual physics that we can for example simulate gravity, the laws of action and reaction, etc. in a virtual environment). Note that:

  > *Any entity that responds to interaction (direct or indirect) with a non-deterministic entity is itself non-deterministic, because we cannot know in advance the state resulting from unpredictable actions.*

  For example, the switch of an elevator is a newtonian entity. It responds to the action of the avatar when he toggles the on/off switch of the elevator. Another example is a virtual billiard-table, the cue and the ball are both newtonian entities, because the avatar interacts with the cue, whereas the cue interacts with the ball.

- **Intelligent entities**
  These entities have a free will, have a goal and are intelligent. Remark that these entities have the ability to interact by themselves with another entity, whereas a newtonian entity is just waiting there until it undergoes some action/event.
  Human beings are intelligent entities, but also, for example, robots and animals that are simulated with artificial intelligence are intelligent entities.

3.2.2.2 Applying OO concepts

The entities that we have discussed so far, have also attributes and behavior as in object-orientation (OO). But the aspects of object-orientation which are not limited solely to attributes and behavior, can also be extended to the entities discussed above. Following is an overview of the several object-oriented concepts with a short explanation, and we will see that these concepts will be useful for describing our virtual environment.

- **Abstraction**
  Abstraction means that you filter out the attributes and behavior that are necessary. So this means that you only consider the attributes and behavior that are necessary to have in your virtual world, of course this is dependent on the type and requirements of the environment you will build.

- **Inheritance**
  An entity (= class in OO) is a category of objects. An object is an instance of an entity. There are many real-world examples where knowledge of a more general entity is also applicable to a more specific entity, this is called *inheritance*. If we have an entity A and we have defined an extension of this entity called B, then will entity B (child entity) inherit the behavior of entity A (parent entity). An example of the use of inheritance will be given further in the practice section.

- **Polymorphism**
  Often an operation executed in different entities has the same name. For example, you can open a window, a door, a wallet, etc. In each of these cases you execute a different operation. This means that each entity knows how the operation must be executed. Therefore we can, for example, talk about behaviours like *open* and *close* for several types of entities, without having to use special terminology for each kind of entity so as to obtain unique names.

- **Encapsulation**
  Here we use the idea that classes can be used as abstract data types. An abstract data type can be viewed as having two faces. From the outside, the user sees only a collection of operations that define the behavior of the abstraction. On the other side we see the variables used to maintain the internal state of the object.
  For example when you have a newtonian entity like *fireplace*, and you have the operation *lightFire* then you do not care about how this is implemented to simulate fire in a virtual environment. It could be that it is simply implemented with an animated texture, or implemented with the use of particles.

- **Message passing**
  Objects can communicate with each other. They do that through messages: one object sends a message to another object. For example, in case of two newtonian entities: *RemoteControl* and *Television*. When you push the on-button on the remote control, the entity *RemoteControl* will send a message to the entity *Television* to put the television on.

Now that we have extended the entities with concepts of object-orientation we can use UML for modeling our virtual environment. UML is an industry standard adopted by the OMG (**O**bject **M**anagement **G**roup) [2] and is a general-purpose OO modeling language, so it is used for representing OO designs. UML is a language and a process neutral notation, this means that you can reason in the object-orientation paradigm and that you can implement it in any programming language.
UML provides the ability to analysts to present their vision understandably in a standardized way. This implies that the client can understand what the development-team is going to do. It gives also the opportunity to report changes if the development-team did not fully understood the requirements of the client or if the client has changed his mind about the requirements.
In UML there are 8 different diagram types. Diagrams 2, 7 and 8 give a static view of the world, whereas 1, 3, 4, 5 and 6 give a dynamic view of the system.

1. Use-Case Diagrams
2. Class Diagrams
3. Sequence Diagrams
4. Collaboration Diagrams
5. Activity Diagrams
6. State Diagrams
7. Component Diagrams
8. Deployment Diagrams

At this point, we will use diagrams 2, 3, 4, 5 and 6 to model our virtual environment. I will now give a brief description of the different diagrams. More details about the notation of UML can be found in the Appendix.

[2] http://www.omg.org/

Class diagrams show the entities (= classes) of the system and their relationships, including inheritance, aggregation and associations. An association is a structural relationship between two classes. For example, if an avatar interacts with a switch of an elevator, we say in object-oriented terms that the avatar is in association with the switch, whereas the switch elevator is in association with the elevator. Aggregation is a special kind of association which indicates that one or more entities are part of another entity.

Sequence Diagrams give the interaction between an object and other objects. The idea is, that the interactions between objects happen in a certain sequence and this sequence takes an amount of time to go from begin till end.

Just as the sequence diagram, the *Collaboration Diagrams* also show the interactions between the objects. Both sequence and collaboration diagrams, are semantic equivalent. This means that you can translate a sequence into a collaboration diagram and vice versa. The sequence diagram emphasizes the sequence of the interactions in time, whereas the collaboration diagram emphasizes the context and the global organization of the interacting objects.

The *State diagram* displays the states in which the object can appear during its lifetime. And transitions between the states are also displayed. A transition is typical for changing one state to another, but can also take on the same state. There are two special states: an initial state and a final state.

The *Activity diagram* is a variant to the state diagram. The differences are that, on the one hand, only activity-states occur in an activity diagram and which therefore has automatic transitions, and that,on the other hand, an activity diagram describes states of more than one object.

3.2.2.3 Model-World-Interaction model

Designing an environment is not a straightforward step, it demands some analysis. Therefore it is useful to divide an environment and then to design each divided part. This gives us a better understanding of the environment about which we are talking.
Therefore we divide an environment into a Model-World-Interaction model. A description of each one of the three components is given below.

- **Model**
  The model contains all the entities that are meaningful for the different interactions that can take place in a virtual environment. In other words, the model contains by default the avatar (intelligent entity) and the entities the avatar wants to interact with. Note that we not only mention the entities which have direct interaction

with the avatar, but also indirect interactions among the entities (for example, the cue and the ball). To model this part we use the class diagram.

The model only contains non-deterministic entities, because in the model we describe the direct and indirect interactions.

- **World**
  The world is the visual representation which is described in the virtual environment, i.e. what the user will see when navigating through the environment. For example, it describes the different rooms and passages, the decoration of the walls and floors, etc, so the filling-up of the environment. Remark, that the world also contains the visual representation of the entities of the model.

  The world contains the visual representation of the deterministic entities and the non-deterministic entities (= model).

- **Interaction**
  Here we describe the different interactions and/or events that are possible in and with the environment. Here we will only describe the interactions of the entities in the model. The interactions do not only describe the interactions between the avatar and the non-deterministic entities, but also all the interactions between the non-deterministic entities. For example, the interaction between two newtonian entities. To model this part we can use sequence, collaboration, activity or/and state diagrams.

  The interaction describes the interactions between the entities of the model that are required for the virtual environment.

Now that we have divided our virtual environment into three parts for modeling, we have still to fulfill the requirements of the previous step of the methodology. We can say that the model and interaction part must fulfill the functional requirements and that the world must fulfill the environmental requirements.

Note that with the model and the interaction part, we can build several virtual worlds that have the same interactive possibilities. In other words, for the same model and interactions, we can build multiple worlds. This gives us, for example, the opportunity to represent different kinds of worlds depending on the same interactive possibilities. This might be useful for a shared virtual world where there are different representations of worlds depending on the kind of users.

3.2.2.4 In practice

Here we illustrate the model-world-interaction model with two examples. One example is a virtual billiard-table where the avatar can play billiard. The second example is an elevator in a virtual environment which can transport us from one floor to the other. Also, we are going to look more closely into modeling the avatar. Now we examine which entities we have in the model.

Model

One of the entities that the model certainly has is the `Avatar` entity, the representation of the user in the virtual environment. An entity is represented in a divided rectangle where you can put the name of the entity, the attributes and the operations (see Appendix for further details for representing a class in UML). Below you find an example of an `Avatar` entity.

Figure 3.6: Avatar entity for first-person POV in a single VE

Here we say that the avatar can move in so-called 6DOF (**D**egrees **O**f **F**reedom), which means that you can track movement up/down, forward/backward, left/right or you track rotation based on pitch, roll and yaw (the terms pitch, roll and yaw are explained in the Appendix).

Remark that pitch and roll are not always necessary for navigating through a virtual environment. The behavior of the *AvatarFP* entity is most suitable for a single-user environment with a first-person POV. The avatar can move in 6DOF but there is no motion of the arms and legs as in the way we walk, for example. Now the avatar entity moves like a particle through the environment, that is why this is suitable for a single-user and first-person POV environment, because you do not see yourself and there are no other avatars.

If we have a more complex virtual environment, especially a multi-user environment, we can have a more complex model of the avatar. I use the word *bot* to represent the avatars in the virtual environment, but also intelligent entities like robots, virtual animals, etc. that have the same characteristics as a human (avatar). As the class diagram below indicates, it would consist of an environment where we have different types of bots. For example, there are robots that can move like a differential drive ($BotDifferential$) and robots that can walk ($BotWalk$) like a human or even bots that have facial expressions and gestures ($BotExpression$) but cannot walk. Next to robots, we have also our avatar which inherited the behavior of $BotWalk$ and $BotExpression$. So the avatar can move through the virtual environment and has features such as facial expressions and gestures.

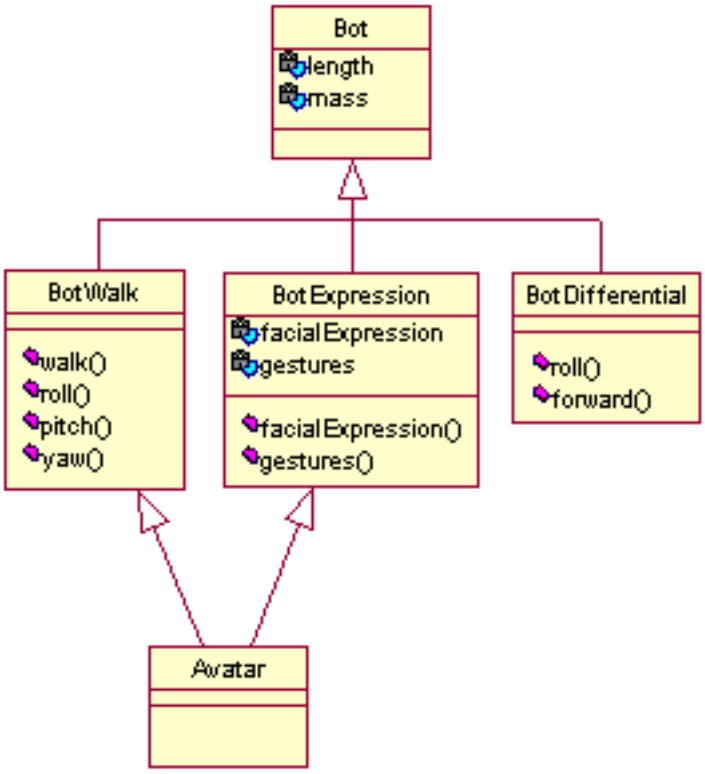


Figure 3.7: Class diagram of a bot

As in the example above, a bot can have a set of attributes, for example, $facialExpression$ and *gestures*. We will see that we can use state diagrams to illustrate which kind of facial expressions we can have, and how they can go from one state of facial expression to another state of facial expression.

For example, if we want to model a virtual billiard-table, we would have the following non-deterministic entities (remark that we only consider non-deterministic entities in our model), namely the trivial intelligent entity *Avatar* and the following newtonian entities: *Cue*, *Ball*, *BilliardTable*, *Wall* and *Hole*.

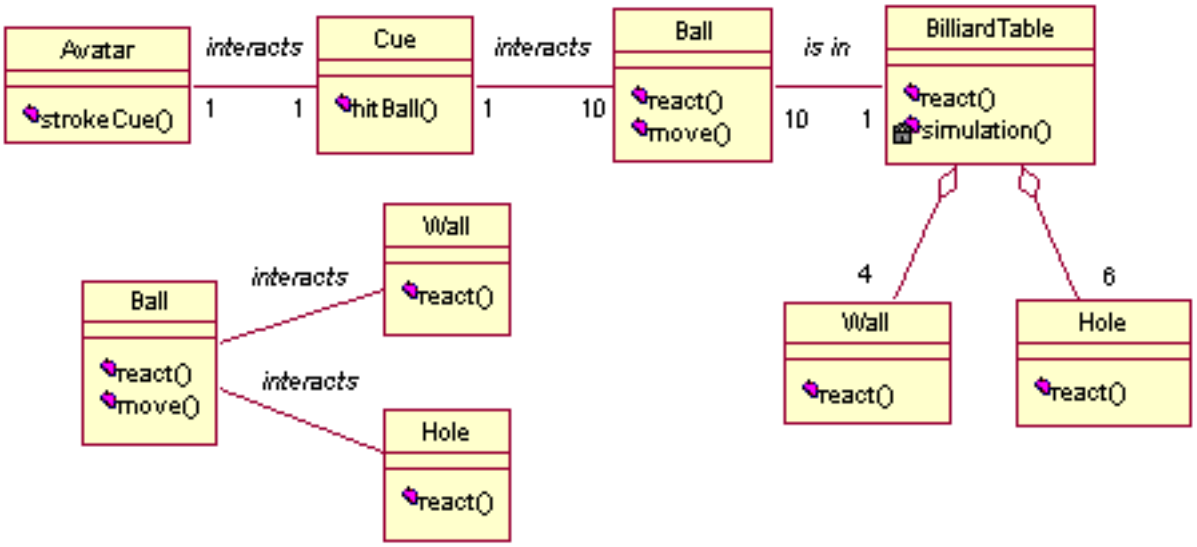


Figure 3.8: Class diagram for a billiard-table

The diagram above shows that we have used associations and aggregation. The *Avatar* entity interacts with the entity *Cue* which in turn interacts with the *Ball*. The aggregation shows that the table consists of four walls and six holes. There is also an association of the *Ball* with *Wall* and *Hole* in the sense that the ball rebounds against the wall and that the ball disappears from the table when it is in the hole. We will see that this is necessary for the collaboration diagram which we will use to describe the interactions in the interaction part.
Note that we have a method *simulation*, this essentially indicates that, the moment the ball is hit by the cue, there is a whole process which simulates the billiard-table. The ball moves over the table, it collides with other balls, it rebounds against a wall, etc.

In the example of the elevator we would obtain a diagram as shown below. We have the following newtonian entities: *Avatar*, *ElevatorDoor*, *Elevator* and *Button*.
The diagram shows that *ElevatorDoor*, which are the doors of the elevator on each floor, consists of two buttons, namely up and high. In contrast to the elevator, which has minimum two to n (number of floors) buttons (so a button for each floor).
The *ElevatorDoor* does operations, such as opening and closing the doors, and will be responsible for calling the elevator. Whereas the *Elevator* entity is mainly responsible for going from one floor to the other and requesting to open and close the doors.

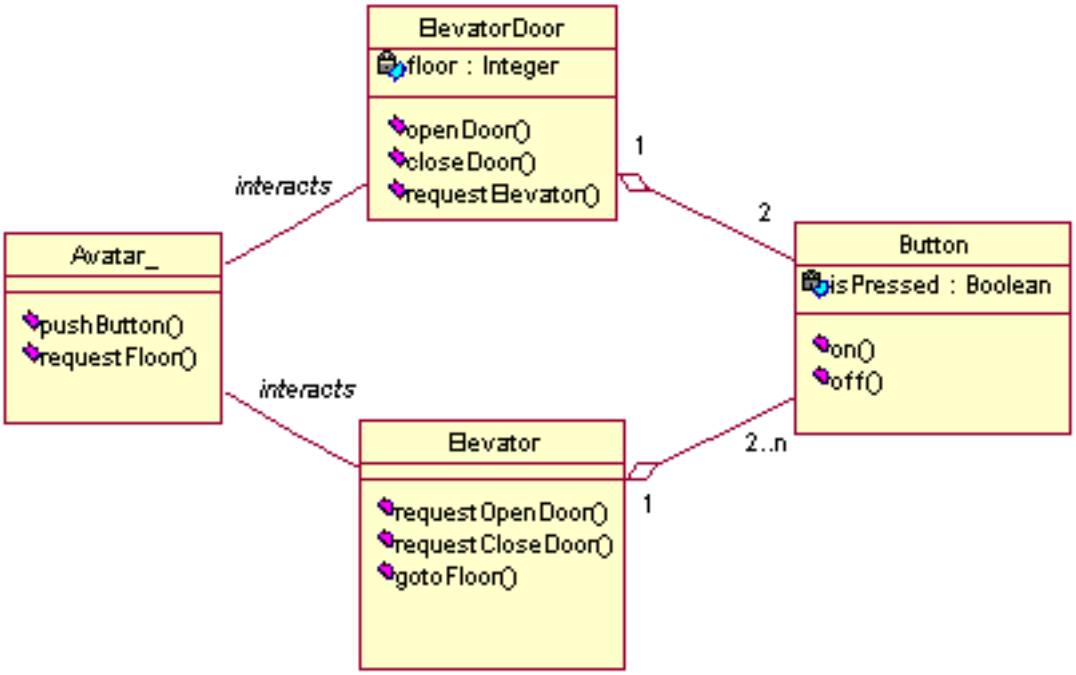


Figure 3.9: Class diagram for an elevator

Interaction

Now we have look at the interaction part. Here we used the collaboration diagram to illustrate the interactions for the billiard-table, then we used the sequence diagram for the elevator example, and finally we used the state-diagram to show the possible transitions between the facial expressions of the avatar.

In the case of the billiard-table we used a collaboration diagram to illustrate the interactions. In a collaboration diagram the numbers left to the label give the sequence of the interaction, and the conditions are placed between square brackets. The avatar strokes the cue, whereas the cue hits a ball from the billiard-table. The third step is that the ball moves and there are 3 things that can happen: the ball collides with another ball (3.1), the ball collides with the wall (3.2) or the ball falls into the hole (3.3).

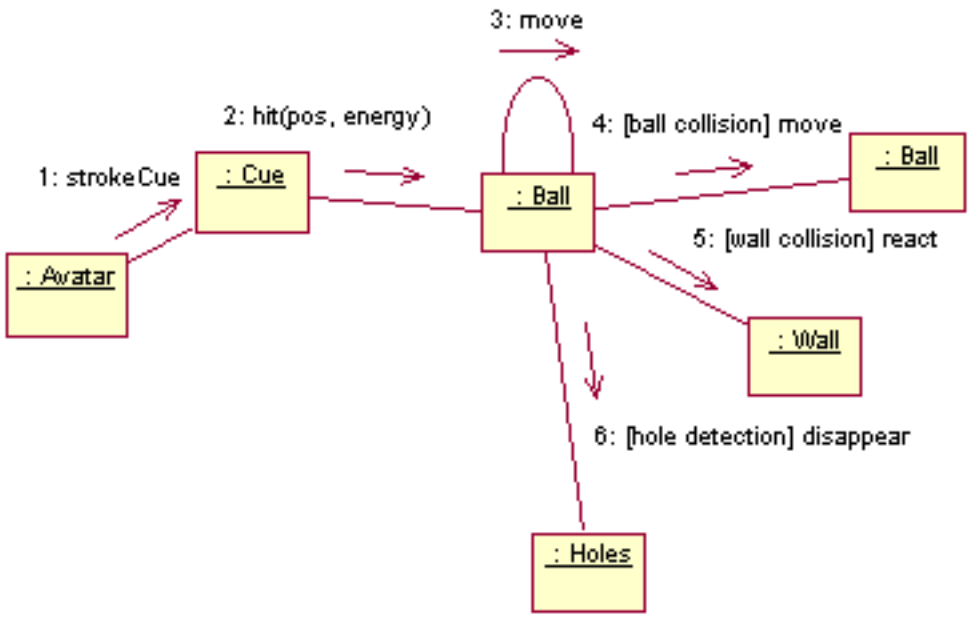


Figure 3.10: Collaboration diagram for the billiard-table

The model shows that the avatar can produce facial expressions. The state-diagram shows what kind of facial expressions exist: *normal*,*happy*,*bored* and *angry*. And it shows the possible transitions between the different facial expressions. For example, the avatar can go from a normal facial expression to happy and vice versa. The state-diagram illustrates also that we cannot go from a happy to a bored state.

In the case of the elevator, we have the following sequence diagram (which can be found on the last page of this chapter). We describe the interactions between the objects, *Avatar*, *ElevatorDoor* and *Elevator*. It shows what interactions are happening when the avatar requests an elevator.

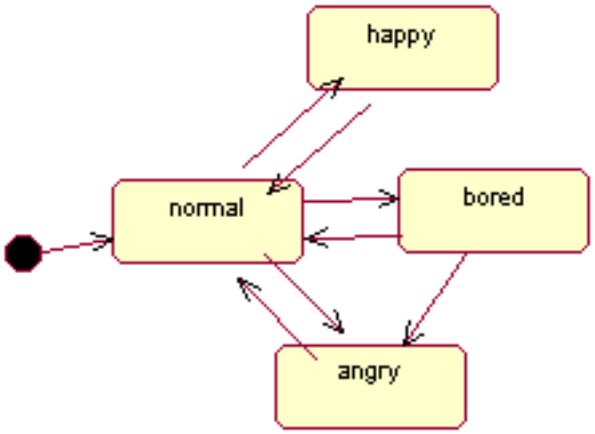


Figure 3.11: State diagram for facial expressions

World

The world thus describes the geometry of the entities in the model and the other entities that fill up the world, such as rooms, floors, decoration, etc. So, for example, we could say that the cue must have such and such dimensions with such and such wood-texture, etc.
You can describe it as above, or you can describe your environment or how you would like shape your virtual billiard-table by means of a screenshot. This will give you an idea of the atmosphere to be created for your virtual environment. Note that we can reuse the same model for different kinds of worlds. For example, when we have a shared virtual environment on the Internet. We will use the billiard game as an example. When one particular user is participating in the billiard game, he may be situated on the moon, and another user participating in the environment may reside in an ordinary room. The virtual environment shows the same behavior (the same interactions) but is situated in a different world (e.g.: moon and room).

One of the things you must consider before creating the virtual environment is usability. Much has been written on usability of the user interface of interactive applications (especially of GUI), but there is little research on usability in VEs. Even the methods applied to interactive applications are not suitable and rather limited for VEs [5].

There has been some research done on wayfinding in large virtual worlds. This is important, because badly designed (large) virtual environments create problems as to wayfinding in virtual worlds. For example, one of the problems can be that they have difficulty to relocate places where they have been before, they have not an idea of the global structure of the world, etc. Therefore [2] presents some design principles of wayfinding augmentations of virtual environments. The basic principles are:

- Divide the large-scale world into distinct parts, preserving a sense of "place".
- Organize the small parts under a single organizational principle.
- Provide frequent directional cues.

Another very good navigation aid is the use of a map. This gives us the ability to quickly extract spatial knowledge directly from a map. Obviously we want to get some more spatial knowledge with the use of a map, therefore it is important to give the right and necessary information on the map. For this purpose, there are also some design principles:

- Show all organizational elements (paths, landmarks, districts, etc.) and organizational principles.
- Always show the observer's position.
- Orient the map with respect to the observer in such a way that the forward-up equivalence principle [3]is accommodated.

The usability concerns are of course applicable to every virtual environment, but we can also have virtual reality on the WWW with VRML, for example. This creates some new usability concerns due to the fact that you can have a 2D/3D combination.
For example, the standard web pages are designed to interact with standard 2D input devices, such as the mouse. The mouse has 2DOF, which means you can move left/right top/down and you can click the mouse, in contrast to 3D on the web where we must map the 6DOF onto a 2D input device [16].
Therefore 3D browsers (for example the CosmoPlayer for viewing VRML worlds) provide an interface to handle the new possibilities of navigation. For example:

[3]states that the upward direction on a map must always show what is in front of the viewer

Figure 3.12: The interface of the CosmoPlayer VRML Browser

3.2.3 Implementation Design

At this stage we examine which design we will use to implement the virtual environment. Here we mainly decide which type of protocol, network characteristics, etc. we will use. Especially when there is a need of a distributed virtual environment (which is described in the functional requirements), this will have an great impact on the design decisions of the network. Issues which are described in chapter 2.

Also the type of applications that we have described in the functional requirements, will have a great impact on the implementation design decisions. Certainly the number of users simultaneously using the shared virtual world, will be a issue when designing the protocols, etc.

The main idea behind distributed virtual environments is when there has been locally a change of the entity's state. This change must be distributed among the participants that are participating in the same environment. Otherwise, we would have inconsistent worlds between the participants. Note that only entities that change the state must be updated and hence be distributed to the other hosts (participants).
In particular, the non-deterministic entities change the environment. So, when a non-deterministic entity is changed, it must be distributed among the others. This means that, when we have modeled our virtual environment into three parts, namely model-world-interaction, the model describes the set of entities that must be updated among the participants.

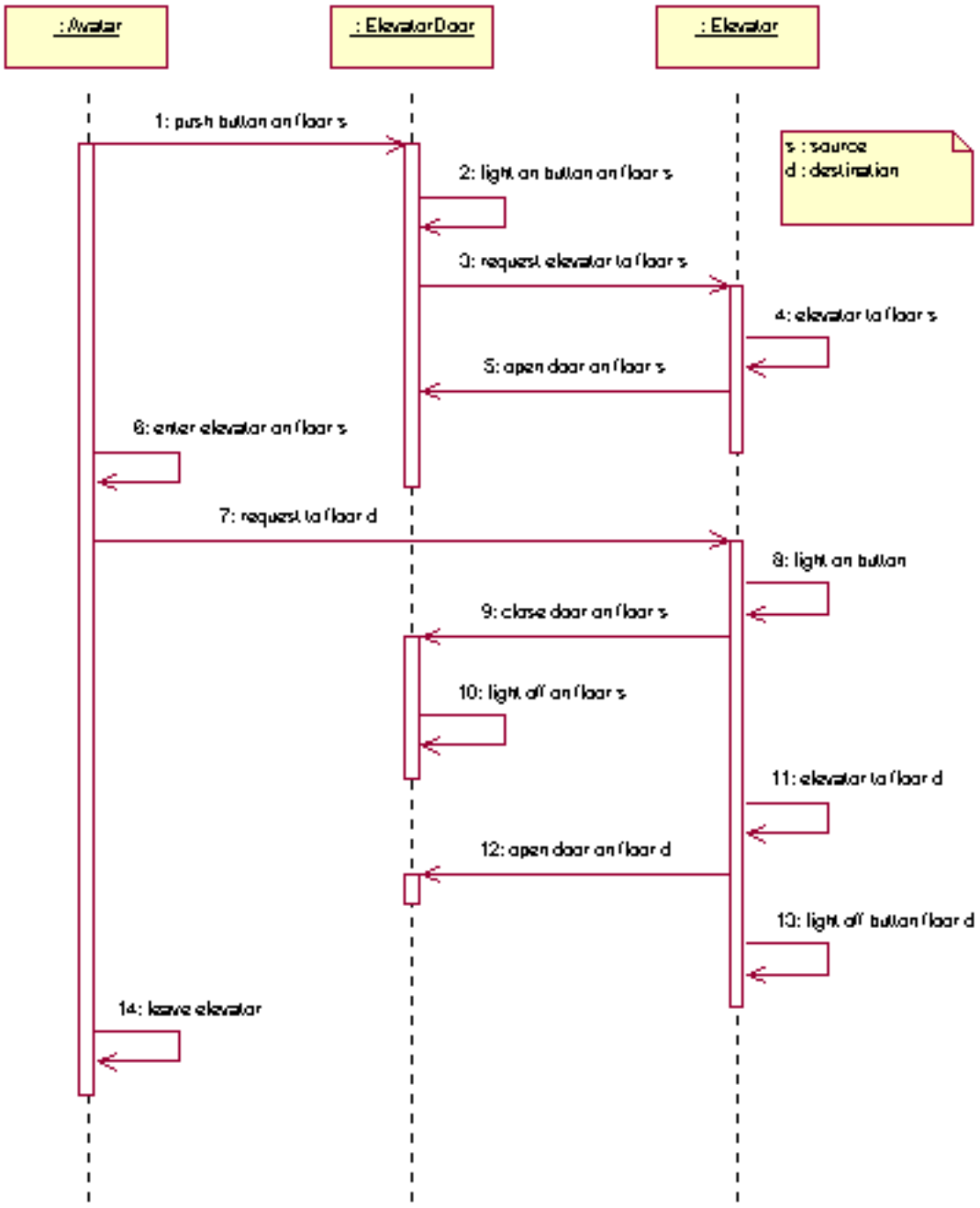


Figure 3.13: Sequence diagram for the elevator

Chapter 4

Conclusions

Applications involving virtual environments are multiple and entail many specific domains. Therefore it was not easy to set up a general modeling procedure for describing all the variants of virtual environments.
Nevertheless, the methodology was a first attempt at modeling a basic general interactive virtual environment. The issue illustrated is that it is very important to have a methodology for modeling a virtual environment, because it provides us insights into what exactly the environment must fulfill, especially the interactions and the visual representation, which have a direct impact on the implementation design (e.g. the number of users).
The splitting-up in the model, i.e. world and interaction, enabled us to get a better insight and build up a reasoning about the requirements for an interactive environment and the dynamics involved therein. The splitting-up also proved to be beneficial in the implementation design, especially when we are talking about distributed virtual environments.

Appendix A

Human Body

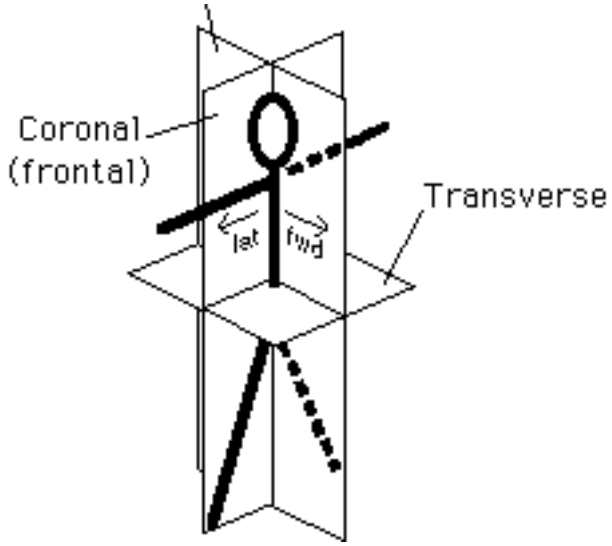


Figure A.1: Reference planes of the human body

Roll: Body rotation in the coronal plane.
Pitch: Body rotation in the sagittal plane.
Yaw: Body rotation in the transverse plane.

Appendix B

UML Notation

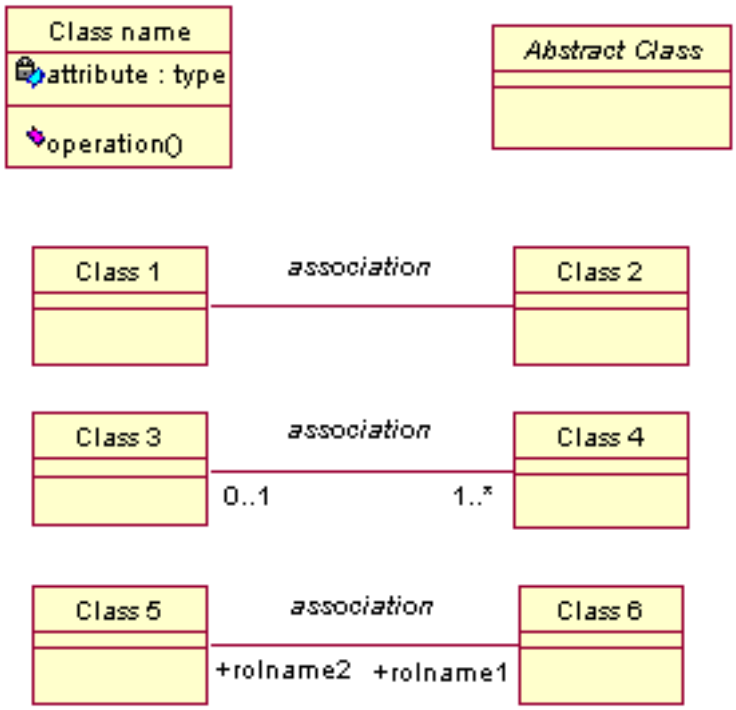


Figure B.1: Classes

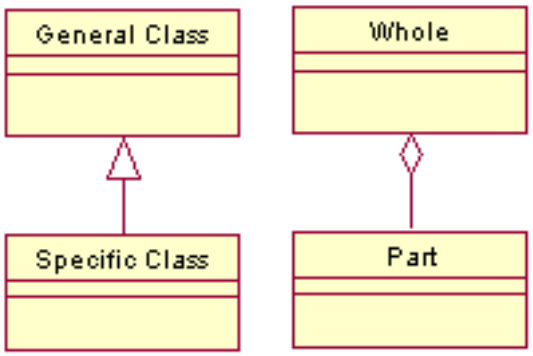


Figure B.2: Generalisation and Aggregation

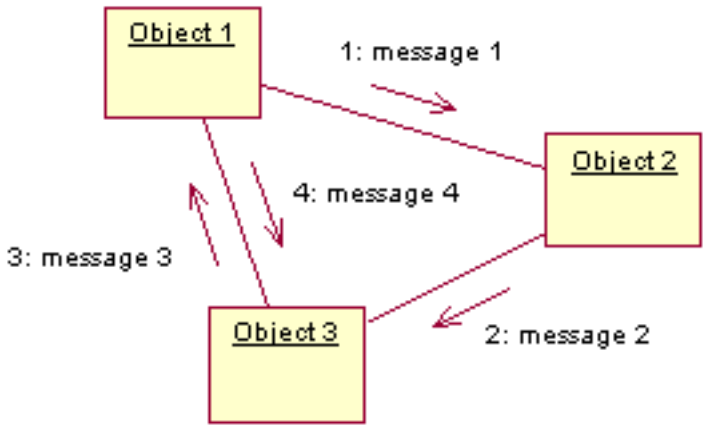


Figure B.3: Collaboration diagram

Bibliography

[1] M.K.D. Coomans and H.J.P. Timmermans. Towards a taxonomy of virtual reality user interfaces. *Proceedings of the International Conference on Information Visualisation*, August 27-29 1997.

[2] Rudolph P. Darken and John L. Sibert. Wayfinding strategies and behaviors in large virtual worlds. 1996.

[3] Michael R. Macedonia Donald P. Brutzman and Michael J. Zyda. Internetwork infrastructure requirements for virtual environments. *Proceedings of the VRML Symposium, San Diego Supercomputer Center (SDSC), San Diego, CA*, December 13-15 1995.

[4] Thomas A. Funkhouser. Network topologies for scalable multi-user virtual environments. *IEEE VRAIS '96, San Jose, CA*, April 1996.

[5] Joseph L. Gabbard and Deborah Hix. A taxonomy of usability characteristics in virtual environments. November 1997.

[6] Clark Dodsworth Jr. *Digital Illusion (Entertaining the Future with High Technology.* Addison-Wesley, 1998.

[7] Marcus Kaar. *A Multi-user Environment for the World Wide Web - The Virtual Shopping Mall.* PhD thesis, Vienna University of Technology - The Institute of Computer Graphics, Oktober 1998.

[8] Michael R. Macedonia and Michael J.Zyda. A taxonomy for networked virtual environments. *IEEE Multimedia*, 4(1):48–56, January - March 1997.

[9] Peggy Miles. *Internet World Guide to Webcasting.* John Wiley Sons, Inc., 1998.

[10] Christopher S. Page Omer Casher and Henry S.Rzepa. Virtual reality modelling language (vrml) in chemistry. *HITL Technical Report.*

[11] Rich Gossweiler Randy Pausch Robert J. Laferriere, Michael L. Keller. An introductory tutorial for developing multi-user virtual environments.

[12] Bernie Roehl. Some thoughts on behavior in vr systems. August 1995.

[13] Joseph Schmuller. *Sams Teah Yourself UML in 24 hours*. SAMS Publishing, 1999.

[14] Jonathan Steuer. Internet unicasting vsersus multicasting.

[15] Jonathan Steuer. Defining virtual reality: Dimensions determining telepresence. *Journal of Communication, 42(4) (Autumn, 1992), 73-93*, 1992.

[16] Gareth Griffiths Steven Kerr and Victor Bayon. 3d-web page usability issues; present and future. *VIRART, University of Nottingham*.

[17] Shihming Yen Li-Sheng Shen Tom Furness Susan Tanney, Paul Schwartz. A design method for virtual environments using narrative and pattern languages. *HITL Technical Report*.