Vrije Universiteit Brussel
Faculty of Sciences
Department of Computer Science

Using Semantic Descriptions for Building and Querying Virtual Environments

A thesis presented to the Vrije Universiteit Brussel
in fulfilment of the thesis requirement for the degree of
Bachelor/Master of Science in Computer Science
(Licenciaat in de Informatica)

By
Haïthem Mansouri

Promoter:
Prof. Dr. Olga De Troyer
Supervisor:
Dr. Frederic Kleinermann

June 2005

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Frederic Kleinermann, for his valuable guidance and support and for always being there for me as well as to my promoter, Prof. Dr. Olga De Troyer, for giving me the opportunity to work on this project. Furthermore, I would like to thank all my colleagues at Omron Electronics nv/sa and especially my bosses Jan Thijs and Stephen Decoussemaker for being so comprehensive and supportive of my studies. Last but not least, I would like to say a big thanks to my family for bearing with me and supporting me all these years.

Samenvatting

Tegenwoordig worden 3D-virtuele werelden meer en meer gebruikt. Voorbeelden daarvan zijn de afbeeldingen van echte steden in de vorm van interactieve 3D werelden. Aangezien deze virtuele werelden zeer groot en complex neigen te worden en vele verschillende virtuele voorwerpen opnemen, wordt het steeds belangrijker om de virtuele wereld te vragen voorwerpen precies te kunnen vinden of om snel tot aan een gewenste positie in de virtuele wereld te gaan.

In dit werk hebben wij de mogelijkheid onderzocht om zoekmotoren voor virtuele werelden efficiënter en domeingerichter te maken. Om dit te verwezelijken hebben wij een mechanisme gecreerd dat (automatisch) een groot aantal semantische informatie associeert aan een virtuele wereld volgens het domein waarvoor een virtuele toepassing is ontwikkeld. Deze informatie wordt later gebruikt door de zoekmotor om de gebruiker toe te laten de virtuele wereld efficiënt te vragen naar voorwerpen.
Vragen is flexibel en domeingericht. Dankzij deze benadering is het mogelijk om onduidelijke vragen zoals ïk zoek een lange boomöf ïk zoek een oude auto met een paardenkracht van meer dan 100 öp te lossen, zelfs met een basis vragenoplosser (query solver).

De metagegevens met betrekking tot een virtuele wereld worden opgeslagen in een open en gestandaardiseerd formaat, zodat informatie tussen toepassingen herbruikt en gedeeld kan worden. Verder kunnen deze gegevens gebruikt worden om informatie omtrent de verschillende voorwerpen van de virtuele wereld te gebruiken als een soort begeleiding.

Abstract

Nowadays, 3D virtual worlds are being more and more often used. An example of that can be seen with digital cities on the Web being interactive 3D representations of the real cities. As these virtual worlds tend to become very large and incorporate many different virtual objects, it is becoming very important to be able to query the virtual world in order to precisely find the objects needed or to get quickly to a desired location in the virtual world.

We have investigated the possibility and provided a model for making search engines for virtual environments more efficient and domain oriented. For this, we have created a mechanism to (automatically) attach a vast amount of semantic information to virtual worlds according to the domain for which a virtual application has been developed. This information is later used by the search engine to allow the user to efficiently query the virtual world for objects populating it.
Querying is flexible and domain oriented. Thanks to this approach, it is possible to solve queries expressing vagueness such as ”I am looking for a tall tree”or ”I am looking for an old car with a horse power superior to 100”, even with a basic keyword-matching query solver.

The meta-data relative to a virtual world are stored in an open and standardised format. This allows reusability and information sharing between applications. Furthermore, this data can also be used in order to provide information about the different objects populating a virtual world as a sort of user guidance.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Nowadays, virtual applications are being used in various domains including the military, medical, entertainment and architectural domains [10]. They are also becoming more frequently available across the web thanks to languages such as VRML [15], X3D [15] and the efforts of the recent Web3D consortium group, which aims at standardising such languages.
As computers get ever more powerful, it becomes easier to generate more complex virtual applications having an important number of virtual objects. An example of this can be seen in some architectural and gaming applications [10]. Nevertheless, these complex virtual applications will only succeed in attracting more users if they are easy to interact with. For instance, it is a well-known fact that low usability is a limiting factor for the proliferation of virtual environments on the Web [9]. This fact is especially true for novice users or when the virtual environments become too complex to be explored without appropriate assistance. Furthermore, a number of studies [8] have shown that poor usability leads users to become rapidly frustrated, which results in a risk of having them leave the world, fail to recognise or visit interesting part of the world, or simply complete their visit with a feeling of not having adequately explored it. For this reason, a number of research groups have been developing ways to assist the user when using a virtual application by providing assistance through the availability of maps and guided tours, for instance, and more recently also the possibility to query the virtual environments [1].

1.1.1 Virtual worlds navigation aids

Navigation aids like maps, help visitors orient themselves in the virtual environment. However they are sometimes hard to interpret especially by novice users and might lack the level of details required [1]. Guided tours, on the other hand, allow the users to assume a passive role while they are lead through the virtual environment and given explanations about the objects and places populating it. While this method allows users to get information about the places being visited and not miss important parts of the world, it might not be adequate for all types of virtual environments. For instance, in a large virtual city it is unthinkable to explore every part of the world through a guided tour as users will inevitably lose interest after some time or would want to visit only specific parts of the world. This problem can be solved by querying.

1.1.2 Navigation by querying

Recently, the principle of search engines (with the same idea as the ones currently used on the Internet such as "Google"[23] and "Lycos"[24]) have been developed for virtual reality in order to help users to quickly find what they want in large and complex worlds. Search engines allow the users to query virtual worlds in order to rapidly find places and objects without having to perform the time-consuming task of exploring the entire, or even a portion of the world to find those objects and places. Furthermore, they prevent users from getting lost or disoriented in the world by allowing them to easily return to known or previously visited places in the virtual environment when needed. This principle is similar to the hyperlink principle in HTML.

Until now, research developments for search engines in VR have been focusing more on how to make smart queries (Boolean query [8] , Fuzzy query [1]) and how to navigate with the user. Unfortunately, these search engines are still too general and not enough domain related, while most of the time, a virtual application is developed for a particular domain. Hence, there is a need to be able to easily customise search engines according to the virtual application they are applied to. It becomes also necessary to be able to easily attach a correct semantic information according to the domain for which the virtual application has been developed. Most of the time, the only semantic information used in virtual applications are the attributes of an object, i.e. its size, name and so on, which is somehow restrictive when it comes to allowing flexible querying to be performed.

1.2 What this thesis provides

The efficiency of any search engine for virtual worlds depends heavily on the annotation of the worlds. This thesis introduces an innovative mechanism to easily and intuitively create domain oriented annotations for virtual worlds in an open and standardised format. Our model allows flexible and context oriented querying of such worlds even with basic query solvers. Furthermore, a new approach for providing information and guidance to users exploring virtual environments is proposed.

We start by introducing the mechanism to annotate virtual environments with context oriented semantic information and show how this information can be used to query such environments in a flexible way. Later, we explain how this annotation can be re-used in order to provide assistance and information about objects to visitors who choose to explore the world by browsing (in contrast to exploring it by querying).

1.3 Structure of this document

This document is structured as follows: in the next chapter, we remind the reader of the main models and languages used today to represent and create virtual worlds and explain our choice. In chapter three, the structure of a search engine for virtual worlds is presented. This chapter also discusses the research efforts that have been conducted in the corresponding fields. Chapter four and five describe our querying model. In chapter four, the reader is introduced to the OntoWorld tool for the creation of virtual worlds. We also show in this chapter, how this tool has been adapted to support the annotation of the virtual worlds during their creation. Chapter five deals with the search engine itself. In chapter six, we discuss the use of the query results to navigate in a virtual world and show further uses of annotations, namely to provide the user with information about objects populating the virtual world.
As a proof of concept, we have created a virtual shop application and annotated it with our tool. In chapter seven, we review this application and discuss if our search engine behaves as expected. In the last two chapters, we point out open issues for further research and finish by providing our conclusions about the work that has been conducted.
Finally, appendix A shows how to dynamically create viewpoints for the different objects in virtual worlds.

Chapter 2

Representing virtual worlds

Figure 2.1 is a virtual representation of the Babylon palace at the time of Alexander the Great[1]. The proliferation of virtual worlds on the web and outside of it would not have been possible without a set of standardised programming languages and tools to create and view these worlds. In this chapter, we review the main models used for representing virtual worlds. These can be divided in two groups: 3D programming languages (or toolkits) and 3D file formats. Whether from the first or from the second group, most of these languages and file formats are based on the notion of a *scene graph*.

Figure 2.1: A virtual view of the Babylon Palace (from [25])

We start this chapter by explaining the concept of the scene graph. Later, we remind the reader of some of the main 3D programming languages and 3D file formats currently used. Finally, this chapter ends by explaining our choice for the representation of the virtual worlds used in this work.

[1]Image source: [25]

2.1 The scene graph

Most of the languages used to create virtual environments are based on the concept of the *scene graph.* A scene graph is a hierarchical approach to describing objects and their relationship to each other in a virtual world. Hierarchical scene graphs are created by adding nodes as children of group nodes. A node is an object that represents a 3D shape, property or group. Shape nodes represent 3D geometric objects. Property nodes, are nodes used to express characteristics of objects such as appearance, etc. Finally, group nodes can be seen as containers of other nodes. The top node of the scene graph is called the root node. Grouping nodes has the advantage of simplifying object manipulation in 3D worlds. For example, making a logical connection between a hand and an arm would allow the hand to be moved together with the arm as this latter moves. This can be easily achieved as the hand would be positioned according to the local coordinates of the arm. The description of information relative to parent objects is called a local coordinate system, and is the heart of the scene graph approach to virtual worlds. More on the scene graph can be found in [28].

2.2 3D programming languages

2.2.1 Open Inventor

Produced by Silicon Graphics Inc. (SGI), Open Inventor is an object-oriented 3D graphics toolkit for the C and C++ programming languages. Open Inventor is based on the OpenGL graphics library. It represents a programming model based on a 3D scene database that simplifies graphics programming.
Open Includes a rich set of objects such as cubes, polygons, text, materials, cameras, lights, track balls, handle boxes, 3D viewers and editors. Its graphics capabilities are extensive.

2.2.2 Java 3D

Born from a joint collaboration between Intel, Silicon Graphics, Apple, and Sun, Java 3D is a toolkit that extends the Java programming language with 3D capabilities. The Java 3D API is part of the Java Media suite of APIs. It provides a set of object-oriented interfaces that support a simple, high-level programming model that can used to build, render, and control the behaviour of 3D objects and virtual environments. Java 3D tries to meet the various requirements of current 3D programs such as object modelling and rendering, real-time interactive navigation through virtual worlds or support for specialised input devices or stereoscopic vision.

Java 3D contains two main packages: javax.vecmath and javax.media.j3d. The first package contains all the classes concerned with manipulating vectors and matrices while the second contains all the rest of the classes. Contrary to VR modelling languages such as VRML (see further), Java 3D does not define a file or network format of its own. It is designed to provide support for markets that require higher levels of performance than VRML can provide; specifically real-time games, sophisticated mechanical CAD applications, and real-time simulation markets. In this sense, Java 3D provides a lower-level, underlying platform API on top of which some VRML implementations could be layered [19].

2.3 3D file formats

2.3.1 VRML

VRML stands for Virtual Reality Modelling Language. As its name suggests, VRML is not a programming language like Java or C++, but rather a modelling language. It is used to describe 3D scenes by using the scene graph paradigm as explained above. VRML is expressed as text. It has been especially designed to be used on the Internet. In fact, it can be seen as a 3-D visual extension of the World Wide Web [11]. Users can navigate through VRML 3D scene with the help of a VRML browser. Just like with World Wide Web, they can navigate through 3D space and click on objects representing URLs (which could represent gateways to other VRML words).
VRML has evolved in two phases. In the spring of 1995, the first version of VRML, known as VRML 1.0, was adopted. This version was based on SGI's Open Inventor format but provided a basic file format that only allowed for the creation of static 3D worlds. One year later, the second version of VRML (VRML 2.0) was released as an ISO standard called VRML97. In comparison to VRML 1.0, the second version of VRML provided dynamism and interactivity to virtual worlds by adding support for Java and Javascript together with sound and animation.

Basic concepts of VRML

We outline in this section basic concepts of 3D graphics and VRML as this language is used throughout this thesis. For more on VRML, please visit [28].

- In 3D graphics, an object is defined by its edges (points) in a three dimensional space (x-y-z). Linking these points together produces a *wireframe* rendering of the object. Later on, a *surface* or skin is applied to the object. Finally, in order to be visible, an object has to be lit by a light source or be itself a light source. In case the object is lit by a light source, it must be shaded to recreate the correct visual effect.

- The position of an object in space is given in x-y-z coordinates. Defining the position of an object in VRML is call *a transformation.* VRML uses meters as the measure of unit of the world coordinate system. One unit is therefore, equivalent to one meter. This makes the sharing of objects between worlds easier as they will always be correctly represented. Care must be taken however, if scaling Transform nodes are used.

- VRML items are defined as *nodes.* A node can be the representation of a 3D geometry, a sound, an image, etc. Each node has a set of fields.

- *Grouping nodes* are nodes used to create hierarchical transformation graphs. Each grouping node defines a coordinate space for its children. Transformations accumulate down the scene graph hierarchy.

- Sensor nodes generate events based on the passage of time (TimeSensor) or user actions such as a mouse click or navigating close to a particular object. This last category of sensors is called *geometric sensors*. They include the PromimitySensor, VisibilitySensor, TouchSensor, etc.

- In order to send events between different node fields, VRML defines a Route mechanism. This mechanism has a source field, which is the source of the event and a target field defining the target of the event. It is thus possible to define an *eventOut* event from a field of a node A and route it to an *eventIn* field of a node B.

- With Prototypes, it is possible to parametrise and encapsulate VRML objects. A prototype defines a new node type, which can later instantiated to form a standard node.

Category	Unit
Linear distance	Meters
Angles	Radians
Time	Seconds
Colour space	RGB([0.0 to 1.0],[0.0 to 1.0],[0.0 to 1.0])

Table 2.1: Standard units in VRML

2.3.2 X3D

The Extensible 3D or X3D, is the evolution of the VRML modelling language [15]. Although, X3D is set to replace VRML, it still provides compatibility with existing content and browsers of the old standard. X3D is based on the XML file format. This makes it easier to manage, control, validate and exchange information as a vast range XML-based tools for transformation, translation and processing exist and are continuously being developed. For instance, thanks to the adoption of the XML format, it is now possible to develop well-formed and validated scene graphs, which was not the case with VRML. Moreover and contrary to VRML, the X3D standard forces compatibility with the different players and browsers as well as simple and consistent 3D scenes authoring. A concerted effort has been made during the development of the standard in order to provide adequate specification of X3D behaviour so that scenes and environments can inter-operate between browsers. The SAI (Scene Authoring Interface) on the other hand, specifies a unified set of abstract services that can then be mapped to any programming or scripting language (e.g. Java, C++, Javascript, etc.) so that the virtual environments can play independently of the language used.

X3D profiles

In contrast with the monolithic nature of VRML, X3D has a component-based architecture that supports the creation of different *profiles*. These profiles can be individually supported by different applications and can be tailored to particular

market segments such as CAD, the medical or the visualisation market segments [15]. One example of such a profile is *core* profile, which supports simple non-interactive animation. Another example is the *base* VRML compatible profile providing compatibility with most of the VRML standard.
These profiles can be individually extended or modified through the addition of new *levels*. In the same way, new profiles can be created to introduce new features such as streaming. This architecture allows the advancements of the specifications to move quickly since development in one area does not slow down the specifications standard as a whole.

Security and compression

Currently, a compressed binary encoding [2] for X3D is under development. This will allow encryption for security and high compression ratio's for X3D environments. X3D encoding will provide a compact transmission format minimising delivery size and maximising parsing speed. The size, parsing and security capabilities of this encoding will be superior to the ones provided by the gzip compression currently used in VRML [15].

2.3.3 MPEG-4

MPEG-4 is an ISO/IEC standard developed by the Moving Picture Experts Group (MPEG), which is also responsible for the production of the widely acclaimed MPEG-1, MPEG-2 and MP3 standards. MPEG-4 builds on the proven success of digital television, interactive 2D/3D graphics applications and interactive multimedia systems such as the World Wide Web [16]. It is targeted at delivering both static and interactive multimedia content to any platform (including PCs, PDAs and advanced cell phones) over any network. Potential applications include distance learning, entertainment, product marketing, etc [18].

Coded representation of media objects

MPEG-4 audiovisual scenes are composed of several media objects, organised in a hierarchical fashion. At the leaves of the hierarchy, we find primitive media objects, still images (e.g. as a fixed background), video objects (e.g. a talking person without the background) and audio objects (e.g. the voice associated with that person, background music) [17].
MPEG-4 standardises a number of such primitive media objects, capable of representing both natural and synthetic content types, which can be either 2- or 3-dimensional. In addition to the media objects mentioned above, MPEG-4 defines the coded representation of objects such as text and graphics, talking synthetic heads and associated text used to synthesise the speech and animate the head, animated bodies to go with the faces and synthetic sound.
A media object in its coded form consists of descriptive elements that allow handling the object in an audiovisual scene as well as of associated streaming data, if needed. It is important to note that in its coded form, each media object can be represented independent of its surroundings or background [17].

[2]http://www.web3d.org/x3d/specifications/ISO-IEC-19776-3-CD-X3DEncodings-CompressedBinary/

Binary Format for Scene descriptions

In addition to providing support for coding individual objects, MPEG-4 also provides facilities to compose a set of such objects into a scene. The necessary composition information forms the scene description, which is coded and transmitted together with the media objects. Starting from VRML (the Virtual reality Modelling Language), MPEG has developed a binary language for scene description called BIFS. BIFS stands for BInary Format for Scenes.
Just like in VRML, an MPEG-4 scene follows a hierarchical structure, which can be represented as a directed acyclic graph. Each node of the graph is a media object. This tree structure is not static. Node attributes can be changed while nodes can be added, replaced, or removed.

BIFS is essentially a superset of VRML. While VRML lacks some features required by the applications targeted by MPEG-4 such as natural audio and video, the MPEG-4 scene description builds on the strengths of VRML while providing answers for the above mentioned applications. The latest version of BIFS (called advanced BIFS) include some new features not supported by the first version such as PROTOs and EXTERNPROTOs, associating interactive commands to media nodes, inclusion of hierarchical 3D meshes to BIFS scenes, advanced sound environment modelling in interactive virtual scenes, etc. [17].

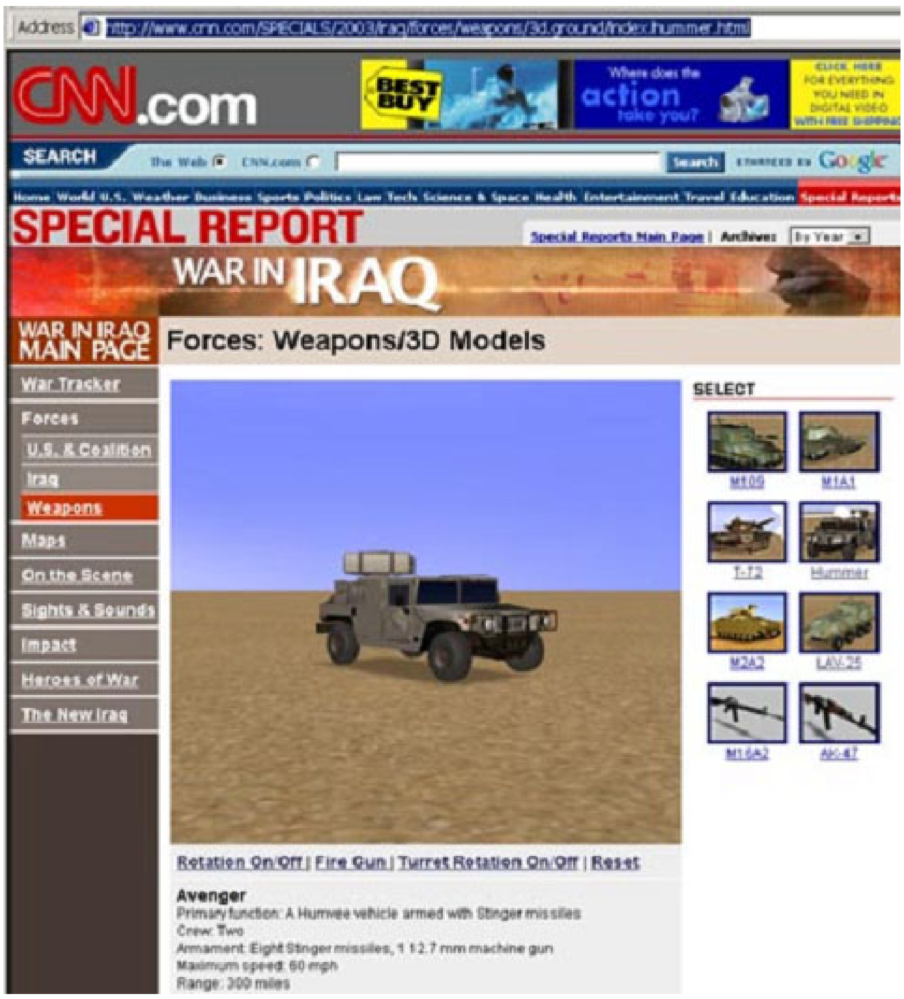
Media description

Recently, support for X3D has been incorporated into the MPEG-4 standard [22]. This should enable the delivery of X3D-based 3D graphics in rich-media broadcast and network applications to a wide range of platforms, for applications covering entertainment, product and data visualisation, electronic commerce as well as distance learning and others.

In combination with MPEG-4, the XML-based *Multimedia Content Description Interface* (MPEG-7) can be used to describe multimedia content, and by extension, X3D-based content as well. For instance, MPEG-7 can be used to describe MPEG-4 and X3D contents in ways that are not possible with these standards alone. It is expected to form the basis of a new generation of multimedia applications such as media search and retrieval systems, digital libraries, multimedia editing systems, etc.

2.4 Discussion

Next to the semantic web, it is recognised that the next evolution of the Internet is the proliferation of 3D content, or the so-called WEB3D [15]. According to the Web3D Consortium, 3D will eventually be the next true media revolution where visualisation and immersion will become the ultimate user interface enabling applications ranging from e-commerce to collaborative engineering [15]. Furthermore, thanks to 3D media, it is possible to avoid the transmission of full audio or video content. Instead, it will be possible to exchange 3D models and their behaviours as well as structured music or textual representation of the human speech [29]. This has the great advantage of considerably decreasing the required bandwidth to transmit audio-visual content as the amount of data to be exchanged is greatly reduced. An example of this can be seen on the CNN website [30] where 3D representations of military vehicles and weapons used in current conflicts can be seen and interacted with.



Avenger
Primary function: A Humvee vehicle armed with Stinger missiles
Crew: Two
Armament: Eight Stinger missiles, 1 12.7 mm machine gun
Maximum speed: 60 mph
Range: 300 miles

Figure 2.2: The 3D representation of a US military vehicle (from [30])

The combination of the VRML and X3D based VR streaming capabilities of MPEG-4 together with its associated multimedia content description standard, MPEG-7, could pave the way for a real 3D revolution on the Internet. Therefore, domain oriented querying of virtual worlds on the web will become even more important. Based on this, we decided to orient our research towards enhancing querying for virtual reality for the Web. For this, we have used Web oriented VR modelling languages such as VRML and X3D.
Naturally, our approach is not specific for virtual worlds on the web. In fact, it can be used with other types of virtual worlds as well since it does not depend on the language in which a virtual world has been programmed nor on the platform

on which it runs.

Chapter 3

Querying virtual worlds

This section will remind the reader about the main stages that constitute a search engine for virtual reality and highlight the related work, which has been conducted for each stage.

3.1 Annotation

The first stage in the creation of a 3D search engine is the annotation process. This process consists of associating meta-data to objects and scenes in the virtual world. Adding meta-data to virtual environments takes them one step higher in the interaction with users or other virtual environments. For instance, meta-data allows search engines to explore virtual environments and return precise results to the user. It also makes the exchange of information between virtual worlds or agent migration from one virtual world to another possible.

The type of meta-data and the way it is associated are also important as the accuracy of the search engine depends on them. For instance for Cavazza [5], it is necessary to simultaneously access both concrete and abstract information. Concrete information reflects the direct visual representations of the scenes, while the abstract information is needed to master the complexity of the scenes and their dynamic behaviours.

We note however, that most of the work conducted in annotation so far, are limited to add meta-data, which are often just object attributes such as an object's size, name, etc. In other words, the meta-data associated to an object in a virtual world, represents a small description of the object or place related to its location and size inside the virtual world. Nonetheless, there are few works as described in [1] and [9] oriented towards story telling, which provide some sort of context to the object being queried for or looked at in the virtual world. Furthermore, there is no real standard to model the semantics of virtual worlds [1]. This is often application dependent, which strongly limits its re-usability [1].

3.2 Querying

This is the second stage in the process of creating a search engine. It allows the user to enter a query and receive answers in return. The querying process depends mainly on the meta-data entered during the annotation process. While the field of data retrieval and analysis has witnessed a lot of research efforts these last few years, these efforts have mainly been focused on text-based searching methods such as the ones for searching the Internet, and shape-based ones for searching 3D models [1], for instance. Content-based searching of virtual environment, on the other hand, was the subject of very few researches.

An example can be seen in Van Ballegooij & Eliens where they constructed an online community as a test-bed for experiments in the context of their RIF project (Retrieval of Information in virtual worlds using Feature detectors) [8]. In this project, the world was annotated with only object names. Querying was limited to simple Boolean keyword matching. Although this method is fast, it has the main disadvantage that querying for an object type in a large virtual world could return a significant number of results. For instance, querying for a pool table in a large store having dozen of pool tables would return all of them as result.

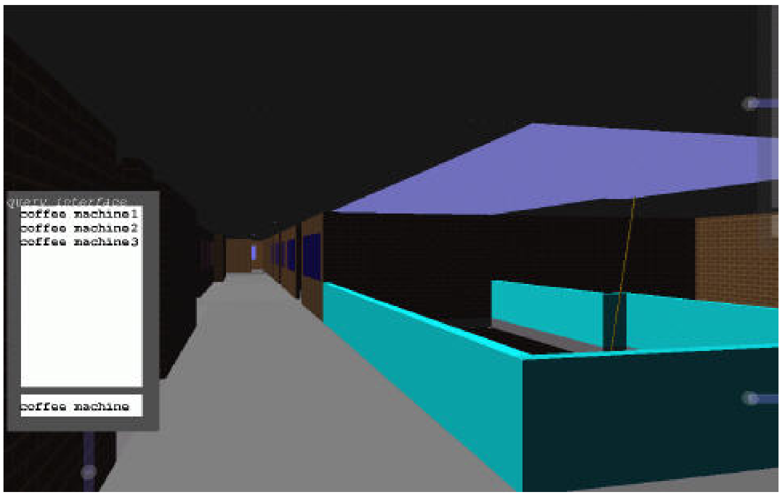


Figure 3.1: Querying for a coffee machine in the RIF project (from [8])

In order to avoid querying on simply the name, others have tried to give other kinds of meta-data like the size and the position. This has given the possibility to make smarter queries. An example of that can be found in [1]. Here, Ibanez provides a querying model that allows users to find objects and scenes in virtual environments based on their size as well as their associated meta-information. This model is based on fuzzy logic and is even able to solve queries expressing vagueness. For instance, it is possible to solve queries such as "I am looking for a tall tree"or "find a park bench near approximately five tall trees". Compared to what had been done before, this approach was very innovative as more complex queries could be treated by search assistants. Nonetheless, the amount of meta-data used in this work was still limited to some object properties such as name, width, height, location, and so on. This still limits the efficiency of the search engines. More precisely, it is not context oriented.

3.3 Navigating

According to Jul and Furnas [7], navigating virtual environments can be subdivided into tow tasks: searching and browsing:

- Searching concerns looking for a known target.
- Browsing defines a state where the user just looks around to see what is available (with or without a specific target in mind).

For the work being conducted, we are mainly concerned by the first task, i.e. navigating by searching or querying. Navigation by querying involves bringing the user to a chosen object after a query has been entered. Furthermore, it is not enough to simple bring the user in front of an object. In fact, in order to correctly view the object, the user's field of view must be well positioned and oriented.

3.3.1 Navigation by querying

If we consider the case of search engines for the Internet like Google, we notice that they operate as follows:

1. after a query is entered by the user, a web page containing all indexes of the pages related to the query is returned,
2. the user clicks on one of the indexes (hyperlinks),
3. the web page related to the chosen index is displayed

By analogy, a search engine for 3D content should:

1. return a list of object names of the 3D content relative to the query,
2. the user then selects one object from the list
3. he is then brought before the object in question.

The last point is important in VR as there are different ways to take the user to the desired object. This can be done through one of two ways: by jumping or by path following:

- Jumping is the easiest way of navigation through the virtual environment. The user is simply taken from one location to the next. While this method can sometimes be desired for users familiar with the virtual environment they are in, it can at other times cause confusion or disorientation for novice users.

- With path following, on the other hand, the system guides the user to the specific location following a pre-defined or auto-generated route at a slow enough pace to allow the user to visualise where he is being lead to. Path following is the trickiest of all navigation options as, for instance, mechanisms must be provided to ensure that the user does not go through obstacles or walls.

In order to move in a virtual environment, it is necessary to define a set of possible paths that a visitor or a search assistant can take. The meta-data associated with these paths is called a *navigation network* and is defined by:

- A set of *accessible nodes*: each accessible node is a reachable three-dimensional point in the virtual environment.
- A set of *connectivity segments*: every connectivity segment connects two accessible nodes, indicating the possibility to move from one node to the other.

This information is used to determine the possible navigable paths from one object (node) to the other. The choice of which path to choose is in itself another field of study, but can either be hard-coded in advance by the creator of the virtual environment (such is the case with Unreal Editor [26]) or calculated at run-time by the system.

3.3.2 Orientation

Object orientation is an important notion in virtual environments: it defines how a visitor will view an object in a virtual environment. Orientation means that the "camera"of the virtual world will be correctly positioned and oriented in order to allow the user to view an object in the best possible way. Defining the position and orientation for an object is called creating a *viewpoint.* Viewpoints can either be deduced from the position of the object in the virtual world, or explicitly entered by the world creator. This is particularly interesting if an object is best viewed from a certain angle for example. Viewpoints will be used by the search assistant when taking the user to the desired object or location.

In the following three chapters, the different stages of the creation of the search engine for virtual worlds are reviewed.

Chapter 4

Stage 1: Annotation

The aim of this thesis is to build a model for flexible and domain oriented querying the virtual environments. We have mentioned that in order to make a search engine efficient, it is important to provide it with as much information as possible relative to the virtual world. This step was called: annotating the virtual world. In order to encourage users to provide a maximum of information about a virtual world, the annotation process must be as easy and intuitive as possible.

there are mainly two ways of annotating a virtual world. Namely, directly when the world is being created or after it has been generated. Very often, virtual worlds are developed by VR experts and not domain experts as most of the authoring tools used are too complex to be used by non-VR experts. However, VR experts cannot be expected to be knowledgeable of the terminology used in the domain in which the virtual world is being created. This makes the annotations limited, not domain oriented and mainly centred around object attributes such as name, shape, size, colour, etc.
Even though some advanced search techniques such as Fuzzy logic [1] and neural networks have been used in order to improve search results, the efficiency of today's search engines for virtual worlds is still limited due to the poor annotation. In fact, this is also the reason why search engines for virtual worlds are currently still too general and not enough domain oriented.
We believe that search engines for virtual worlds can become much more efficient if more information is put at their disposal. This information must not only be relative to object properties, but must also encompass the knowledge about the domain in which the virtual world is built. For this, it is necessary to find a mechanism to capture this knowledge when building virtual worlds.

Recently, a tool named "OntoWorld"has been developed at the WISE research group (Vrije Universiteit Brussel, Belgium) for creating virtual worlds. This tool aims at decreasing the gap between VR developers, domain experts and end-users. It uses ontologies as a means of generating virtual worlds more intuitively and more domain oriented by using the same terminology as the domain expert's and reasoning with high-level concepts. This implies that the virtual world is expressed in terms comprehensible by the domain expert and not in terms of VR primitives. Furthermore, it has the advantage of improving the quality of the annotation of a virtual world as the domain terminology can be used.

This chapter shows how we have extended the OntoWorld tool to allow the annotation of virtual environments. Next to the user's annotations, this method allows a vast amount of information about the objects populating the world to be automatically added to the description. This helps to reduce the load on the user as well as the time that it takes to annotate a virtual environment. Furthermore, and contrary to other approaches, the annotation process will not be limited to some typical object properties such as type, colour, height, width, depth, location, etc. The user will be free to add as much information as desired as well as indicate spatial relationships between objects and logically group objects together.

We first start by explaining how OntoWorld and ontologies work. We then introduce the MPEG-7 standard: an ISO standard for describing multimedia content. Finally, we show how we adapted OntoWorld to support annotating the virtual worlds thanks to MPEG-7.

4.1 OntoWorld

Created by the VR-WISE group, OntoWorld is a tool designed to generate virtual environments from explicit semantic descriptions such as knowledge bases or ontologies. Using this approach, it becomes unnecessary to know specific programming languages or authoring tools to design 3D-objects.

4.1.1 Ontologies

Ontologies have long been used in philosophy to give a systematic account of Existence. Tom Gruber used this term to mean *a specification of a conceptualisation* [36]. That is, an ontology is a description of concepts and relationships that can exist in a so-called Universe of Discourse. In its most basic form, an ontology can be seen as an abstraction of a computer-based lexicon, thesaurus or glossary suitably extended with knowledge about a given domain.

According to [2], the use of ontologies for building virtual worlds has the following advantages:

- **Grasping the knowledge of a domain**
  Thanks to ontologies, it is possible to capture an expert's knowledge about a specific domain. These domain ontologies can be shared and reused for multiple applications as they are of general use.

- **Expressing the virtual world much more in terms of the domain**
  Once knowledge about a specific domain has been captured in an ontology, it is possible to express the design of a virtual world by using the domain terminology. This makes it easier for the domain expert to understand and to participate in the design process. For example, it is possible to model the requirement of having a bottle of wine standing on a shelve in the virtual world as having a 'wine bottle' object on a 'shelve' object, rather

than bothering with exact representations and positions of the bottle and the shelve in the virtual world.

- **Generating the virtual world more easily**
  From the knowledge available in the domain ontology, it is possible to derive a number of properties for a specific object. For example, it is possible to describe a bed in the domain ontology in terms of its height, length and depth. Thanks to this knowledge, it might be possible for an ontology-based VR modelling tool to deduce that a suitable VR primitive to represent a bed is a box. With traditional VR authoring tools, however, the designer would have to choose the correct VR primitive and map the height, length and depth of the bed to the corresponding attributes of the VR primitive himself. This is part of the reason why virtual worlds are still being generated by VR experts instead of the domain experts. Thanks to this approach, it becomes easier to generate (semi-) automatically the required virtual world.

4.1.2 Using ontologies to build virtual worlds

The use of ontologies to create virtual environments allows us to store a great amount of information about general concepts and relationships between them for these environments. For a particular type of worlds, this information needs only to be introduced once and can be called each time a new virtual environment is created. This can be seen as a template for the creation of similar virtual environments (environments which share the same concepts). Specialisation can be obtained through instantiations and parameterisation of these concepts.

The creation of a virtual world as used in OntoWorld follows three stages: a *specification stage*, a *mapping stage* and a *generation stage*. The different stages are given in the figure 4.1 [2].

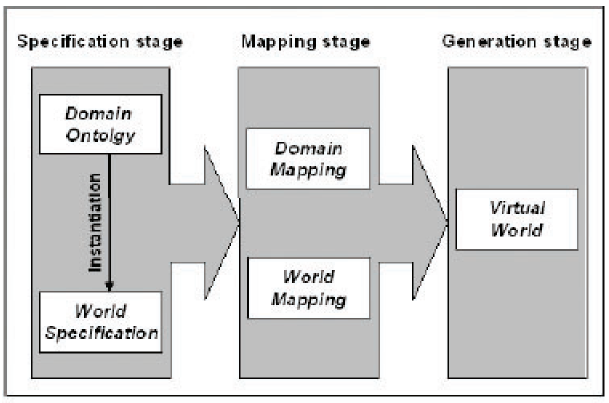


Figure 4.1: Overview of the OntoWorld architecture (from [2])

1. In the specification stage, the VR application is specified at a conceptual level using domain ontologies. First of all, the different concepts are cre-

ated and added to the ontology. Later, instances of these concepts are made, given attributes and added to the virtual world. This last step is referred to as *world specification* because these instances will be used to populate the virtual world. World specification describes how these instances are related, where they are placed in the world, how to interact with them and what behaviour they have.

2. The mapping stage is used to specify how the domain concepts and instances will be represented in the virtual world. This is done by mapping the concepts defined in the domain ontology and the instances defined in the world specification onto VR primitives (e.g. 3D spheres and boxes), constrains and so on.

3. Finally, in the generation stage, the virtual world defined by the previous two stages will be generated in a standard VR format such as X3D or VRML.

Finally, all ontologies used in OntoWorld are based on the DAML+OIL ontology language defined by the DAML Ontology. For more information, please visit [27].

4.2 MPEG-7: the multimedia description language

In order to be able to find (3D) objects in a virtual world, any tool must be able to know exactly what each object is and if it matches pre-defined search criteria. This information is encoded by the user when he is building his virtual world with the help of OntoWorld. The process of attributing specific identifiers to objects is called annotation. Furthermore, the domain ontology being used to generate a more domain-oriented virtual world will also be used during this process in order to enhance the querying capabilities of our search engine. During the generation of the virtual environment, the meta-data created during the annotation process is saved in an MPEG-7 compliant format.
We have chosen to use this format for the simple reason that it is an open standard, which is also used by MPEG-4 in order to describe multimedia content such as audio and video. As MPEG-4 incorporates a way to display 3D content with the help of X3D, MPEG-7 was the logical choice to describe 3D scenes. Furthermore, by using this format we make searching and broadcasting of 3D content via streaming easier.

4.2.1 Introduction to MPEG-7

Developed by the Motion Picture Expert Group, MPEG-7 (Multimedia Description Interface) is an emerging ISO standard for describing and annotating multimedia content. MPEG-7 introduces a set of Multimedia Description Schemes (MDSs or simply DSs) and Descriptors (Ds) for this purpose. These schemes provide a standardised way of describing in XML the important concepts related to multimedia content description and content management in order to facilitate searching, indexing, filtering and access. A description scheme is typically a combination of several descriptors or other subordinate description schemes. Descriptors are the lowest-level elements of the description. The syntactical structure of both descriptors and descriptor schemes is expressed by a Description Definition Language (DLL).

The DLL is derived from the *Extensible Markup Language* (XML) and uses the XML Schema language. This allows to specify MPEG-7 specific extensions, which are necessary to define Ds and DSs. The description itself is then instantiated and stored in an XML file. XML however, is a pure file format not directly suitable to support real-time applications such as streaming of descriptions, dynamic change of descriptor values etc. To overcome this, MPEG-7 also specifies a *Binary Format for MPEG-7 Descriptions* (BiM), which can uniquely be mapped with any XML-based descriptions. Further, BiM simplifies access to partial descriptions by specific sub-tree pointer mechanisms, which allows to process XML files piece-wise.

4.2.2 The MPEG-7 descriptors

MPEG-7 provides a number of Schema Tools that assist in creating, packaging and annotating MPEG-7 descriptions. The Descriptors provide different types of information that could mainly be divided into the following logical areas [3]:

- *Creation Information*: provides information related to the creation and classification of the content. It includes information about the title, the authors, textual annotations, date and location, etc. The classification information describes how the content is classified into categories such as genre, subject purpose, language and so forth.
- *Media Information*: provides information on the format of the content (i.e. data compression, dimension, the possibility of having various instances of the same content but with different resolutions, codec, etc.)
- *Usage Information*: provides information about the modality of usage (i.e. copyright pointers, broadcast schedule, means of payment, etc.)
- *Structural and Low Level Aspects*: provide information on the spatial, temporal or spatio-temporal components of the content (i.e. colour, texture, shape, sound timbers, melody description, scene cuts, segmentation in regions, region motion tracking, etc.). The *Segment* DS can describe a recursive or hierarchical decomposition of the content into segments that form a segment tree. The *SegmentRelation* DS describes additional spatio-temporal relationships among segments.
- *Semantic Aspects*: provide information on the conceptual information of the reality captured by the content (i.e. objects, events, interactions with objects, relations, etc.)
- *Collection Organisation*: provides information about collections of objects (i.e. colour histogram of collection clusters, degree of similarity of the collections, etc.)

- *User Interaction Aspects*: provide information about the interaction of the user with the content (i.e. user preferences, usage history, privacy, etc.)

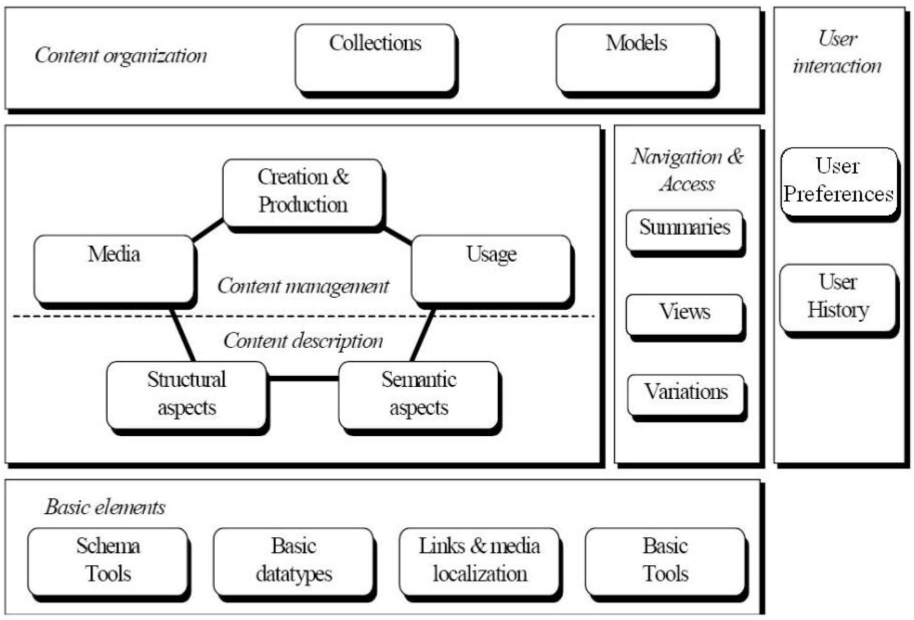


Figure 4.2: Overview of the MPEG-7 MDs (from [3])

An MPEG-7 description begins with a root element that indicates if the description is complete or partial. A partial description carries only partial information that is meant to be added to an existing description. In the case of a complete description, the MPEG-7 top-level element represents the root element. The top-level element orients the description around a specific description task, such as the description of a particular audio visual content (e.g. an image, an audio, video or multimedia segment) or a particular function related to content management such as the creation, the usage, a summary and so forth, of the content [3].

Furthermore, MPEG-7 content descriptors are divided into Audio and Visual descriptors. The group of Visual description tools covers all low-level features of visual information, and includes in addition mechanisms for visual-specific structural description, as well as higher-level information for specific visual signals such as human faces. The different categories of visual descriptors are as follows:

- Basic structures,
- Colour,
- Texture,
- Shape,
- Motion,

- Localisation, and
- Face description.

For instance, 3D shape information can be described in MPEG-7 through the 3D Shape descriptor. Likewise, an object's structure, colour, texture, motion and spatio-temporal localisation can be described through the corresponding descriptor. Unfortunately, MPEG-7 does not specify how to extract or use descriptions, or how to evaluate similarities between contents.

Textual annotation

Being an important component of many DS's, MPEG-7 provides a number of different basic constructs for textual annotation. The most flexible one is the data type for free text. Free text allows the user to create an arbitrary string of text, which can include information about the language being used.
MPEG-7 provides also more structured text annotation types by including fields corresponding the the questions: Who?, What object?, What action?, where?, When?, Why?, and How?. Moreover, it is possible to define more complex textual annotations by explicitly describing the syntactic dependency between the grammatical elements that form a sentence (e.g. specifying the relation between a verb and a subject). Finally, MPEG-7 allows also the definition of a vocabulary for a particular application or domain thanks to *classification schemes* and *controlled terms*. The classification schemes provide a language independent set of terms that form the vocabulary while the controlled terms are used in the description to reference the entries in the classification schemes [3].

4.3 Adding annotation to OntoWorld with the use of MPEG-7

Extending the OntoWorld tool with the possibility to annotate newly created virtual environments was the first step to the creation of an intelligent search engine for virtual worlds. Next to the VRML or X3D file, OntoWorld now generates an MPEG-7 compliant document representing the annotation of the virtual world.

As we will see shortly, the annotation process can be done on different levels. First, descriptions can be given at a concept level. This description is inherited by each instance of the concept and can be adapted if necessary.
Furthermore a lot of information that is entered for the creation of the virtual world can be reused in order to automatically annotate it.

We have use Joanneum Research's free MPEG-7 library for C++ (available at [20]) to work with this standard. This library is a set of C++ classes that allow the creation, manipulation, serialisation and de-serialisation (parsing) of MPEG-7 compliant multimedia content descriptions in an easy way. More on this library can be found at [20].

Throughout this chapter and the next two, we will take the example of a simple world consisting of a tree and a house. This world is shown in figure 4.5. We have in total eight objects (instances) in our virtual. The house consists

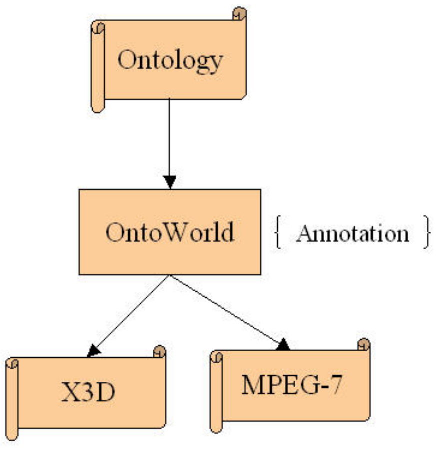


Figure 4.3: Input and outputs of the OntoWorld tool

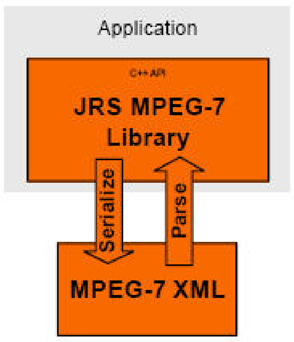


Figure 4.4: The MPEG-7 library from Joanneum Research (from [20])

of four walls (called FrontWall, BackWall, LeftWall and RightWall), a roof (RoofTop) and a chimney (SmokePipe). The tree is composed of a tree trunk (TreeTrunk) and leaves (TreeLeaves). We can distinguish between five concepts: Leaves, Trunk, Roof, Wall and Chimney.
After having created the concepts and instances, we have mapped all the instances to a geometric structure (cone, cylinder, box, etc.) and defined their colour.

4.3.1 Annotating concepts

Annotating concepts can be seen as giving default descriptions to instances on one hand, and as providing mechanisms for automatic generation of descriptions (again for the instances) on the other hand.
The creator of the virtual world can start by entering a main description for the concept (e.g. a ball or a football if all the balls in the virtual world will be used to play football). A main description provides the user with a summary of a specific object in the virtual environment. This main description will be

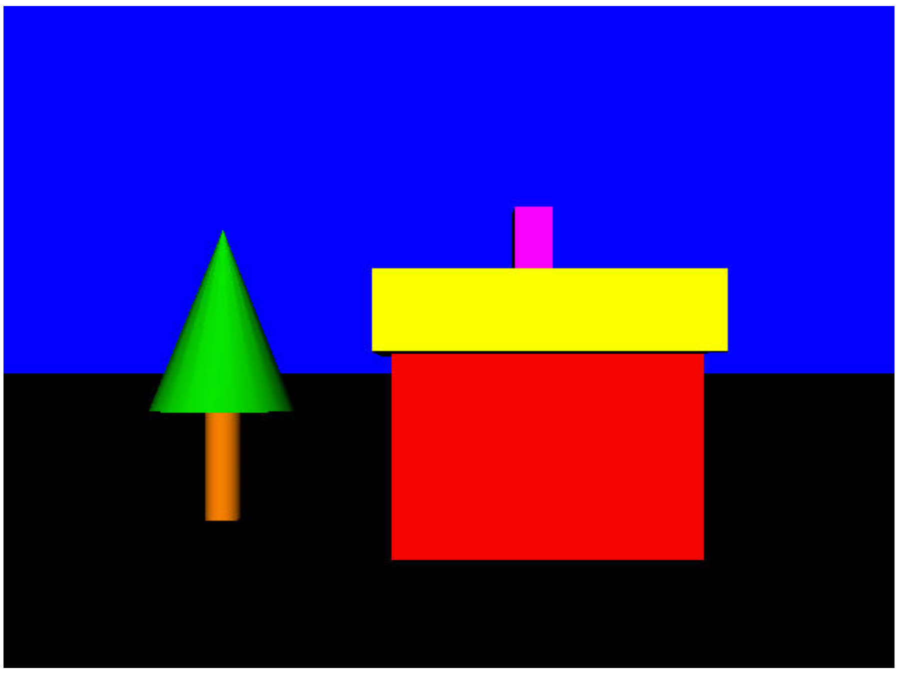


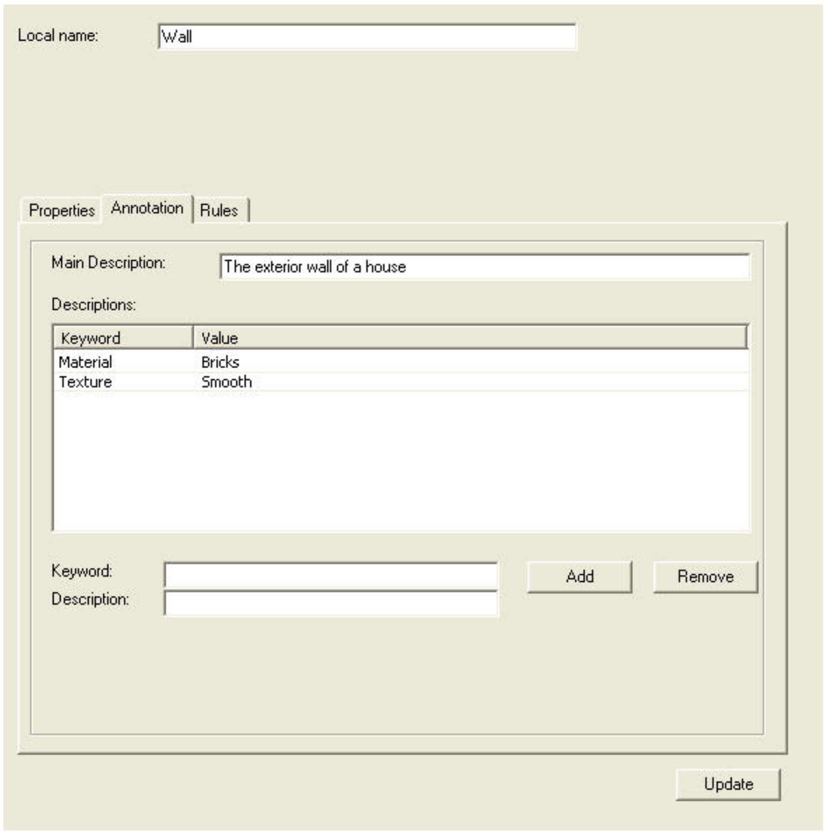
Figure 4.5: A simple virtual world

the default one for all instances of that concept. Later, the user can enter the different descriptions for the concept. Descriptions consist of a keyword and value. A keyword can be entered more than once if it has different values. The concept annotation screen of OntoWorld can be seen in figure 4.6.

In this example, we have defined that the default description for objects of type "Wall"is "The exterior wall of a house". We then defined that the material of the wall is "Bricks"and that it has a smooth texture.

The other aspect of the concept annotations is the possibility to build mechanisms for automatic generation of annotation at an instance level, as we have mentioned earlier. This helps the virtual world creators during the annotation process by automatically generating annotation of instances based on their properties. The creators are therefore, able to define what we called "rules". A rule consists of four elements: an adjective, the property which will be considered, a judgement criteria (i.e. bigger than, less then, etc.) and a judgement value. Later, when the instances will be created, a comparison (relative to the judgement criteria) will be made between the value of the instance property and the judgement value for that property if applicable. If there is a match, the adjective will be added to the instance as an annotation.

It is important to note here, that these rules must be created independently for each concept since for a specific adjective the judgement values may vary from one object to the other. For instance, we may consider a watch to be "expensive"if its price is higher that 100 euros. However, for a car, this adjective


Local name: Wall

Main Description: The exterior wall of a house

Descriptions:

Keyword	Value
Material	Bricks
Texture	Smooth

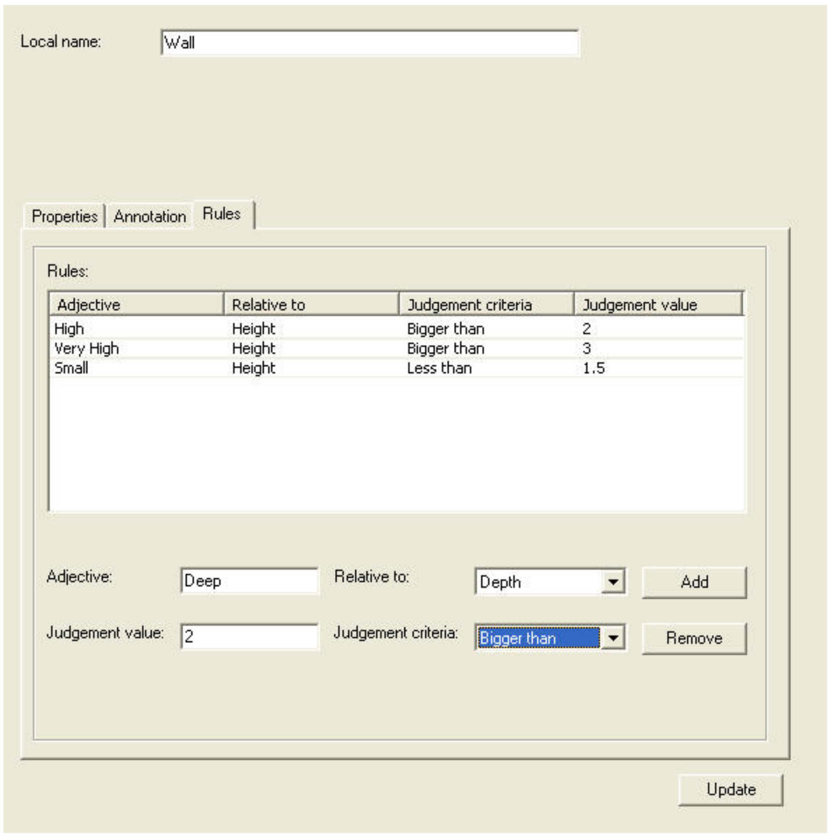
Keyword:
Description:

Figure 4.6: Annotating concepts

would only be true if the price is higher than, let's say, 20,000 euros.
Further, the creator of the virtual world must be aware that the adjectives entered may be very subjective and subject to interpretation. For instance, no one can deny that a person with a height of two meters is tall, but can we consider a three meter high tree as being tall? Some people may answer yes, while others may argue that in comparison with some trees found in the tropical forest with a height of nearly fifty meters, a three-meter high tree is small.
therefore, these differences in interpretation should be taken into account when annotating a virtual world in order to be successful.

Finally, different rules can be constructed for the same property. This helps to compensate for some fuzziness inherent to human perception of our world, especially when a natural language interface is used for the search engine. Using this approach, it becomes easier to provide more flexibility in querying without the need to implement more complex search engines such as the ones using fuzzy logic, for instance. Of course this approach has its limitations and cannot in any way, be compared to the more advanced fuzzy logic approach as used in [1], for example. Nonetheless, the power of this approach lies in its ease of implementation and usage.

Figure 4.7 shows an example of adding rules to a concept. Here, we define



Local name: Wall

Rules:

Adjective	Relative to	Judgement criteria	Judgement value
High	Height	Bigger than	2
Very High	Height	Bigger than	3
Small	Height	Less than	1.5

Adjective: Deep
Relative to: Depth
Judgement value: 2
Judgement criteria: Bigger than

Figure 4.7: Concept rules

a wall as being high if its height exceeds two meters. In the same way, it will be considered very high if this value is higher than three meters and small if it's less than 1.5 meters. It is important to note that this rule engine, in its present state of implementation, has no means of interpreting the opposite of *FALSE* as being *TRUE*, for instance, or inversely. Hence, in order to cover the different situations, the user must explicitly enter both descriptions as rules. For example, if we consider that a particular object can either be small or big, the user will enter both adjectives as rules with the correspondent judgement criteria and value.

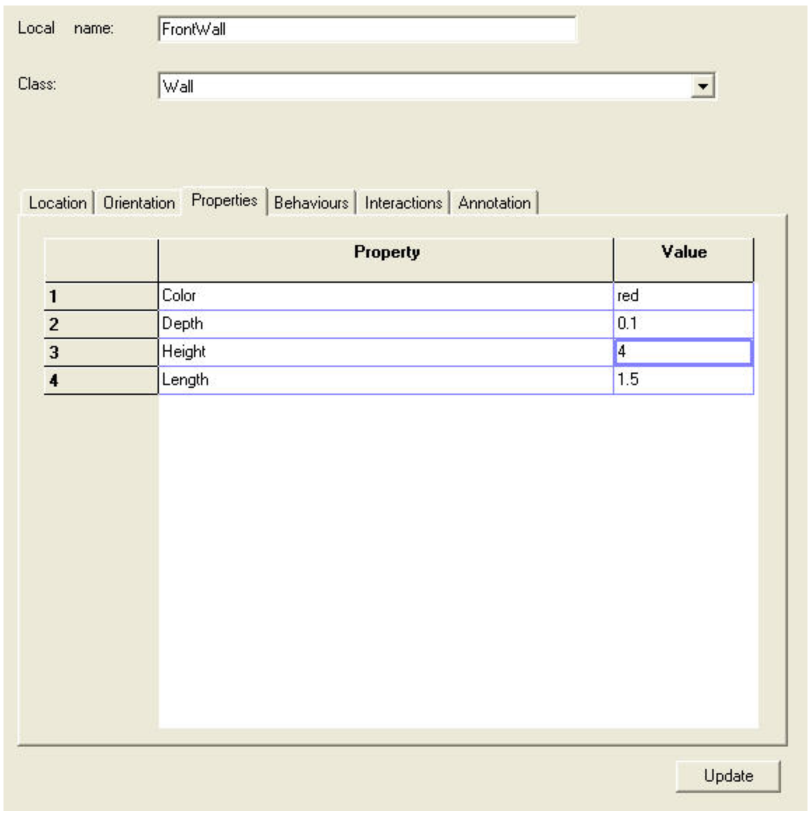
4.3.2 Complex objects

Up until now, no mechanisms exist in OntoWorld to represent complex objects. Nonetheless, the notion of complex (or container) objects is important as it allows us to group objects in a logical way. For instance, we need this mechanism to define our house as the combination of the four walls, the roof and the chimney. In this way, we would be able to introduce the concept of the house in our virtual world even though we have no "physical"object of that type.
In order to overcome this limitation in OntoWorld, we have extended this tool with the notion of "Containers". Containers are not a substitute for complex

objects. They just provide a way to logically group objects together in the absence of the concept of complex objects. One difference between containers and complex objects is the fact that containers are not aware of their children. Only instances "know"in which container they belong (if in any). Furthermore, in the current implementation, it is not possible to annotate or give a description to container objects. More on containers can be read in the following section, which deals with the annotation of instances.

4.3.3 Annotating instances

During the process of annotation of instances, a maximum of information is collected and automatically added to the MPEG-7 description. This automatically generates annotation covering the descriptions inherited from the parent concept, information relative to the properties of the instance and its behaviours as well as the annotation generated from rules that produced matches.


Local name: FrontWall

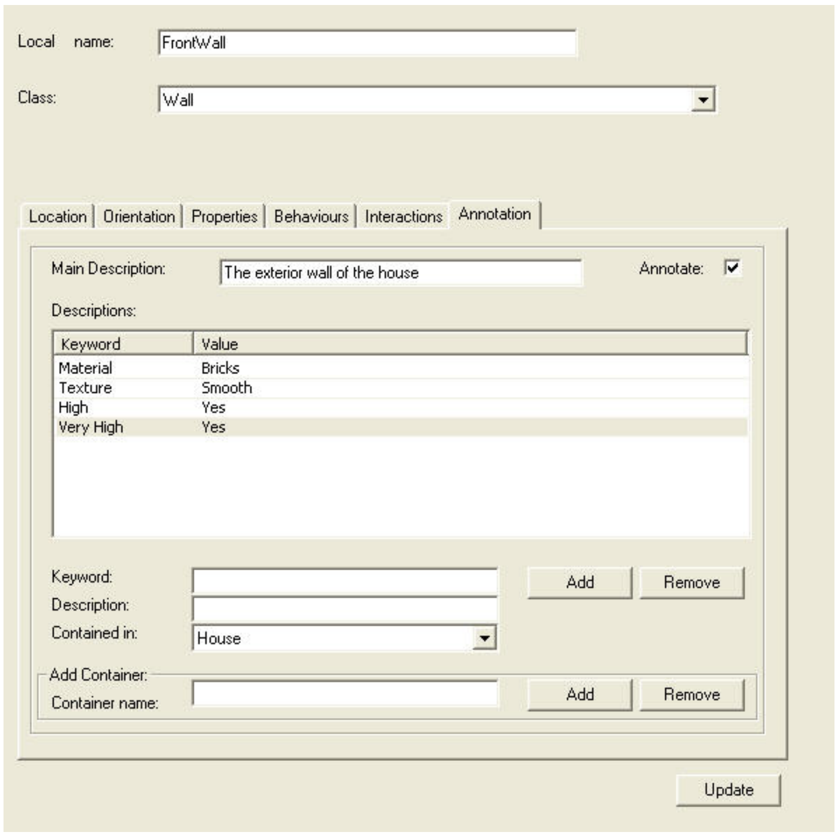
Class: Wall

	Property	Value
1	Color	red
2	Depth	0.1
3	Height	4
4	Length	1.5

Figure 4.8: The properties of the instance FrontWall

The properties of the instance "FrontWall"are shown in figure 4.8. When an instance is created, all annotation relative to the parent concept are automatically added to its description. The user is free to edit this information before saving it as well as adding or removing annotation as wished. Although not visible to the user, each property of an instance as well as its corresponding value

is automatically added to the description. This includes also the *behaviours* and *interactions* of the instance. Behaviours describe what an object is capable of doing (e.g. spinning, changing colour, etc.) while interactions describe the interactivity of the object with the user in the virtual world. Examples of interactivity include approaching, touching (clicking), etc. Often, behaviours and interactions are related as an action (interaction) from the user can trigger an event (behaviour) relative to an object.
As mentioned earlier, an other aspect of automatically generated annotation is the use of rules. In figure 4.9, we see the result of the rules input as shown in figure 4.7. Here we can see that the rules with adjective *High* and *Very High* were both added to the description. This is logical since a ”Very high”object is per definition also ”High”.


Local name: FrontWall
Class: Wall
Main Description: The exterior wall of the house
Annotate: ✓
Descriptions:

Keyword	Value
Material	Bricks
Texture	Smooth
High	Yes
Very High	Yes

Contained in: House

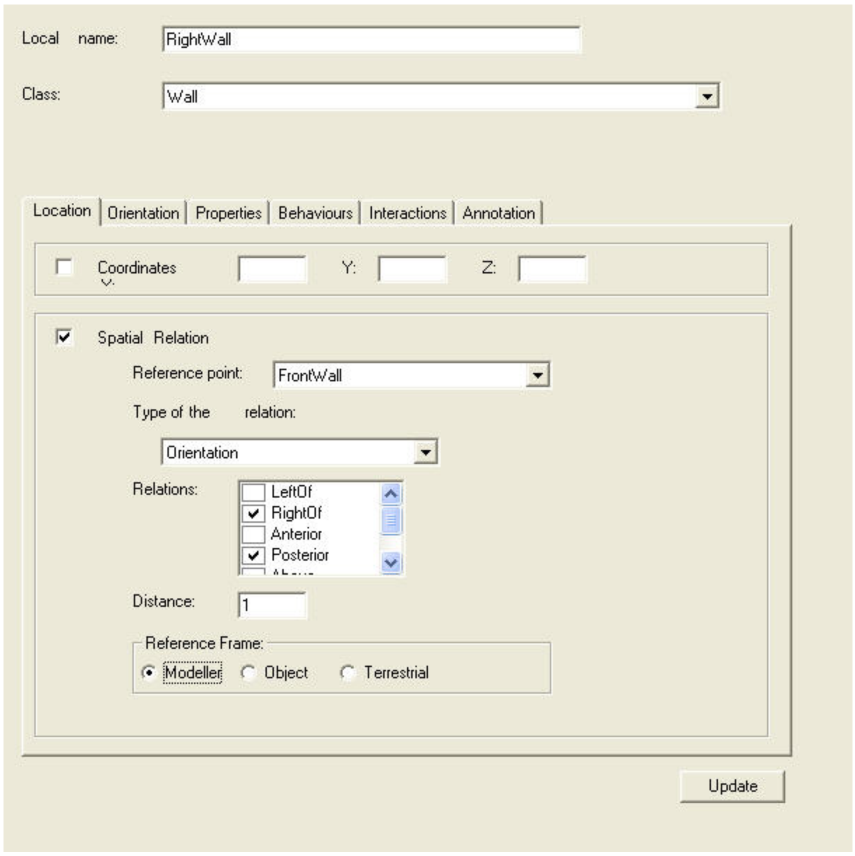
Figure 4.9: Annotating instances

We have mentioned in the previous section the notion of ”containers”. Containers are our way of logically grouping objects together to create part-whole relations since OntoWorld does not support complex objects yet. Containers are solely used in the description of the virtual world and do not have a ”physical”representation in the world. As a reminder, we defined our house as the composition of four walls, a roof and a chimney. Although, the visitor will perceive this combination as representing a house, the notion of this object is not present in the VRML or X3D file. Container objects are shared by all instances. The user needs only select the desired container from the list or simple add (or remove) new ones.

Finally, we can note that not all objects in a virtual world are worth annotating. For instance, we can create a floor or sky in the virtual world just for aesthetic reasons. These objects are in principle of no interest to the visitor. In order to allow the user to indicate which object should be added to the description, a small check-box has been added to the GUI. Per default, this check-box is not checked indicating that the object should not be added to the description. However, as soon as a description is entered for the object or if the parent object has been annotated, the check-box is automatically checked.

Spatial relations between objects

OntoWorld allows to define a position of an object in the virtual world in two ways: either via its absolute coordinates (XYZ coordinates) or relative to other objects.



Local name:	RightWall
Class:	Wall

☐ Coordinates X: Y: Z:

☑ Spatial Relation

Reference point:	FrontWall
Type of the relation:	Orientation
Relations:	☐ LeftOf ☑ RightOf ☐ Anterior ☑ Posterior
Distance:	1

Reference Frame: [✓ Modeller] Object Terrestrial

Figure 4.10: Spatial relations between instances

Defining spatial relations between objects in a virtual world can prove to be interesting for some applications. If we consider the case of a virtual museum for example, a user could ask for information relative to a painting in his field of view. The information returned could include a description of the painting being viewed together with information relative to the painting right next to it if it happens to also be in the user's field of view. Information such as "the painting in front of you was created in 1618 by Rubens while the one on your right is from Rubens' school student and dates from 1650"could be returned.
In our example, we have defined the "RightWall"of the house as being at the "RightOf"and "Posterior"to the "FrontWall". other possible adjectives are: "LeftOf", "Anterior", "Above"and "Below". In the same way, we can define these relations as being: "NorthOf", "SouthOf", "WestOf", "EastOf", "Up"or "Down". These adjectives are automatically added to the description by OntoWorld.

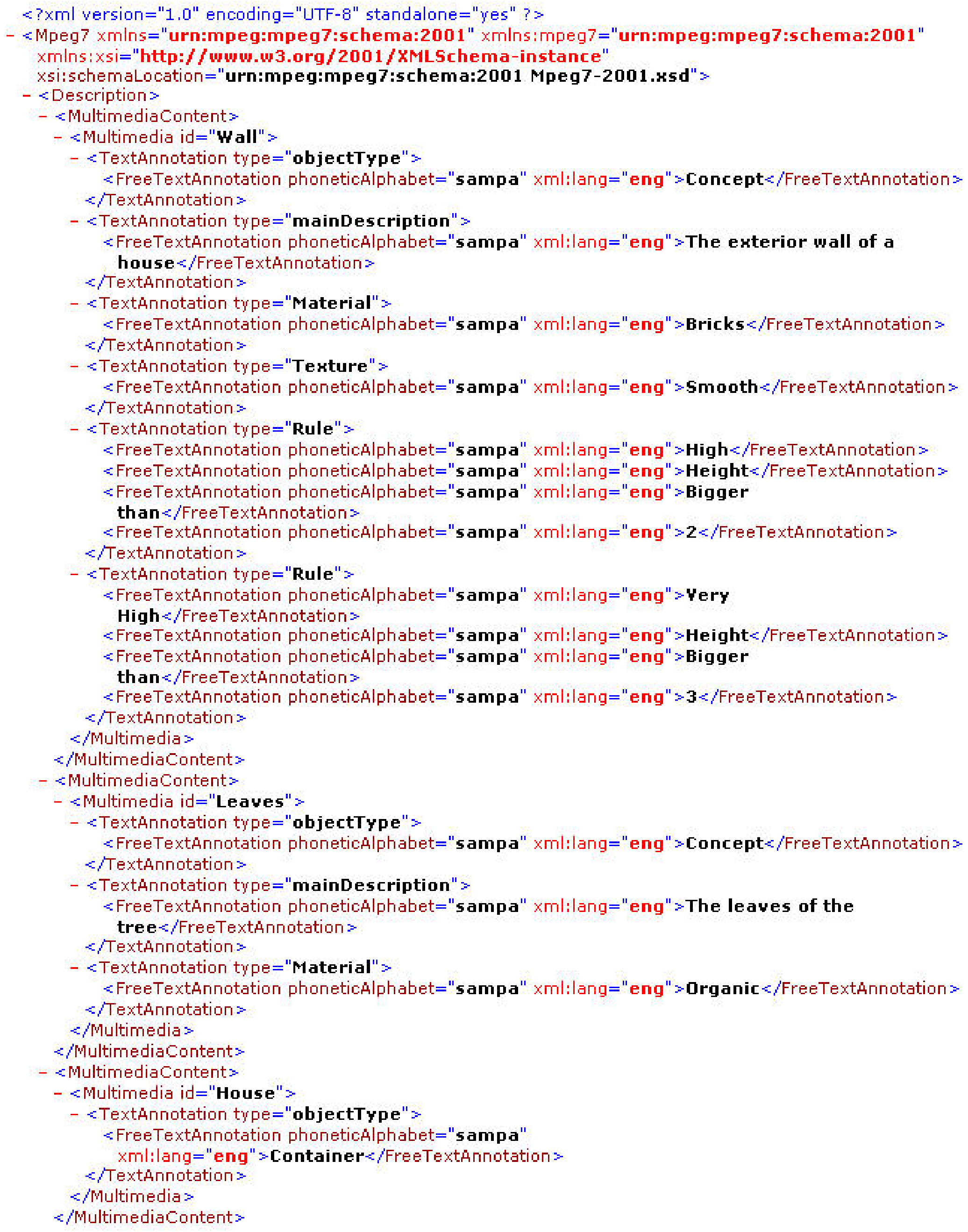
4.3.4 The resulting description

Figures 4.11, 4.12 and 4.13 show part of the resulting annotation of our virtual world. The result is an XML document compliant with the MPEG-7 format. Each annotated object is represented as a "MultimediaContent"object type with a unique identifier ("id"). This identifier corresponds to the object's VRML node name. This id will later be used by the search engine to identify the different objects in the description.

There are basically three object types in our description: Concepts, Containers, and Instances. The keywords "objectType"and "mainDescription"are reserved ones. The type of object is one of the annotations that are automatically added to the description. For concepts and container objects, the value of this keyword is always "Concept"and "Container"respectively. For instances, it corresponds to the name of the parent concept. Furthermore, instances contained in objects have an annotation with keyword "ContainedIn"and value the name of the container object [figure 4.12 and 4.13]. Finally, concept rules are indicated by the keyword "Rule". Rules have four parameters: the adjective, the property it is based on, the judgement criteria and the judgement value. These parameters are always represented in the correct order. Although these rules are not used in the search engine, we prefer to leave them in the description in order to keep it complete.

In this example we can also see that spatial information relative to other objects are also automatically added when available (in this case our "RightWall"lies "RightOf"and "Posterior"to the object "FrontWall"). Furthermore, the result of our "rules"has also been added to the description. We can see for instance, that the keywords "High"and "Very High"have been added. Moreover, the object properties such as "Colour", "Depth", "Length", and so forth have also been automatically added to the description.

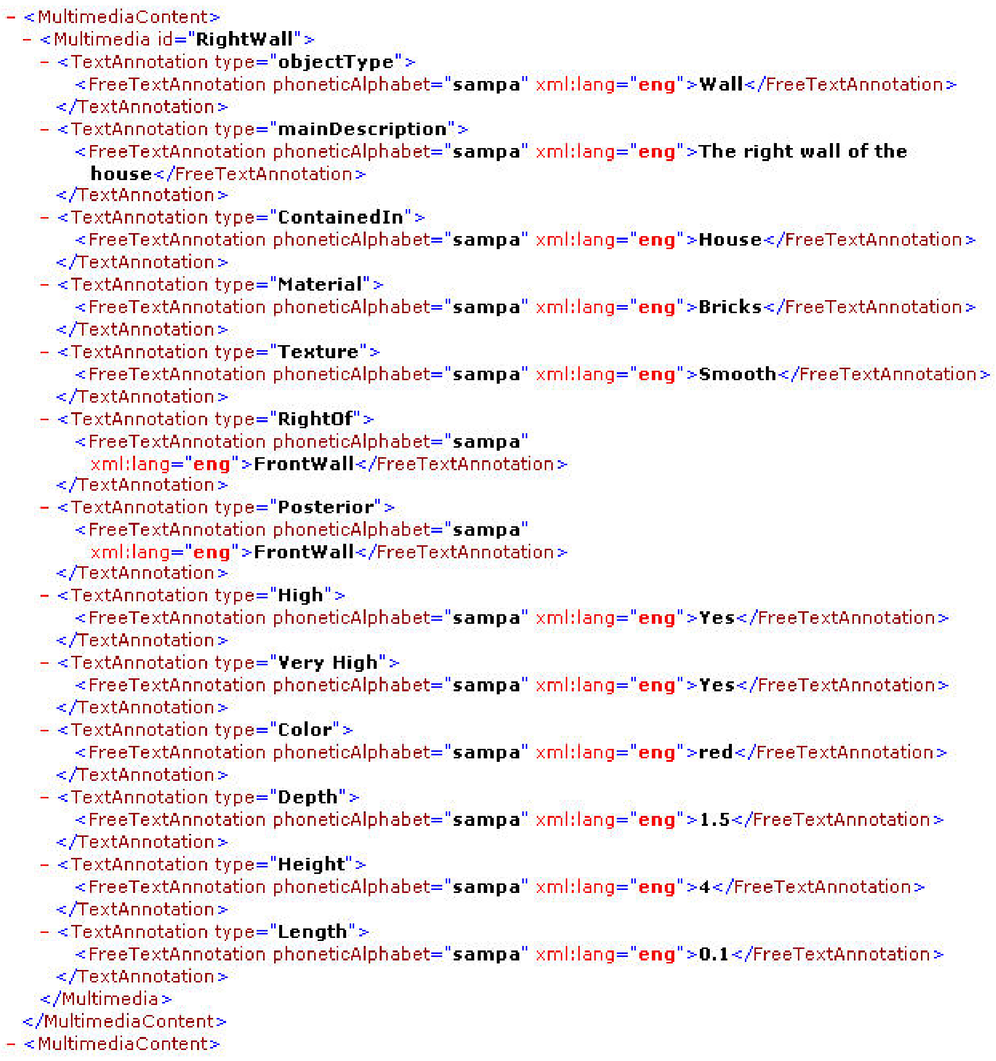
Finally, it is interesting to note that thanks to the MPEG-7 standard, such descriptions need not be confined to a single language. It is perfectly thinkable to include different descriptions for the same keyword in order to support the different languages used in an application. In this case, it suffices to change the language parameter of the "FreeTextAnnotation"tag to differentiate between the



```
  <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:mpeg:mpeg7:schema:2001 Mpeg7-2001.xsd">
  - <Description>
    - <MultimediaContent>
      - <Multimedia id="Wall">
        - <TextAnnotation type="objectType">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Concept</FreeTextAnnotation>
          </TextAnnotation>
        - <TextAnnotation type="mainDescription">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">The exterior wall of a
              house</FreeTextAnnotation>
          </TextAnnotation>
        - <TextAnnotation type="Material">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Bricks</FreeTextAnnotation>
          </TextAnnotation>
        - <TextAnnotation type="Texture">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Smooth</FreeTextAnnotation>
          </TextAnnotation>
        - <TextAnnotation type="Rule">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">High</FreeTextAnnotation>
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Height</FreeTextAnnotation>
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Bigger
              than</FreeTextAnnotation>
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">2</FreeTextAnnotation>
          </TextAnnotation>
        - <TextAnnotation type="Rule">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Very
              High</FreeTextAnnotation>
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Height</FreeTextAnnotation>
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Bigger
              than</FreeTextAnnotation>
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">3</FreeTextAnnotation>
          </TextAnnotation>
        </Multimedia>
      </MultimediaContent>
    - <MultimediaContent>
      - <Multimedia id="Leaves">
        - <TextAnnotation type="objectType">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Concept</FreeTextAnnotation>
          </TextAnnotation>
        - <TextAnnotation type="mainDescription">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">The leaves of the
              tree</FreeTextAnnotation>
          </TextAnnotation>
        - <TextAnnotation type="Material">
            <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Organic</FreeTextAnnotation>
          </TextAnnotation>
        </Multimedia>
      </MultimediaContent>
    - <MultimediaContent>
      - <Multimedia id="House">
        - <TextAnnotation type="objectType">
            <FreeTextAnnotation phoneticAlphabet="sampa"
              xml:lang="eng">Container</FreeTextAnnotation>
          </TextAnnotation>
        </Multimedia>
      </MultimediaContent>
```

Figure 4.11: Structure of the MPEG-7 document: Concepts and Containers

different languages and choose the correct one at run-time.

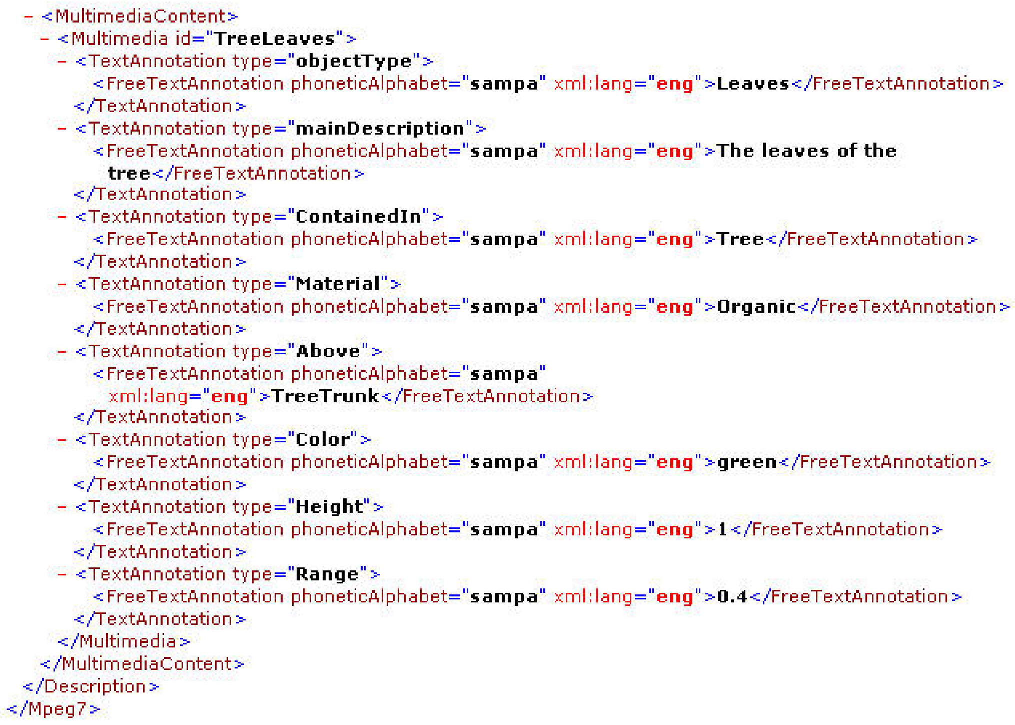


```
- <MultimediaContent>
  - <Multimedia id="RightWall">
    - <TextAnnotation type="objectType">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Wall</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="mainDescription">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">The right wall of the
          house</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="ContainedIn">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">House</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Material">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Bricks</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Texture">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Smooth</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="RightOf">
        <FreeTextAnnotation phoneticAlphabet="sampa"
          xml:lang="eng">FrontWall</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Posterior">
        <FreeTextAnnotation phoneticAlphabet="sampa"
          xml:lang="eng">FrontWall</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="High">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Yes</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Very High">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Yes</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Color">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">red</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Depth">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">1.5</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Height">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">4</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Length">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">0.1</FreeTextAnnotation>
      </TextAnnotation>
    </Multimedia>
  </MultimediaContent>
- <MultimediaContent>
```

Figure 4.12: Structure of the MPEG-7 document: Instances - 1

4.4 Discussion

We have proposed a model to create a comprehensive description about objects populating a virtual world with the help of the open standard MPEG-7. Each object in this description is uniquely identifiable thanks to its node name. Each annotation is composed of a keyword and a value, making it easy to be looked up by a query solver.
To make it easier on the world creators, we implemented mechanisms for automatic generation of annotations. These mechanisms include inheritance of default annotations from parent concepts, automatic capture of instance properties and attributes as well as automatic generation of adjectives based on user input rules.These rules allow the world creator to define for each concept (or object type) what the threshold is for each of these subjective adjectives. For



```
- <MultimediaContent>
  - <Multimedia id="TreeLeaves">
    - <TextAnnotation type="objectType">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Leaves</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="mainDescription">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">The leaves of the
          tree</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="ContainedIn">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Tree</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Material">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">Organic</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Above">
        <FreeTextAnnotation phoneticAlphabet="sampa"
          xml:lang="eng">TreeTrunk</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Color">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">green</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Height">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">1</FreeTextAnnotation>
      </TextAnnotation>
    - <TextAnnotation type="Range">
        <FreeTextAnnotation phoneticAlphabet="sampa" xml:lang="eng">0.4</FreeTextAnnotation>
      </TextAnnotation>
    </Multimedia>
  </MultimediaContent>
 </Description>
</Mpeg7>
```

Figure 4.13: Structure of the MPEG-7 document: Instances - 2

example, a car that costs 30 000 euros can be tagged as *expensive* while another one that costs 14 000 would bear the adjective *rather cheap* .

While the use of the MPEG-7 standard allowed us to create a powerful description of the different objects populating a virtual world, we recognise that this description is not complete. For instance, with the current implementation of OntoWorld, we lack the possibility to describe complete scenes as well as relations between objects other than the ones related to their physical location. Describing complete scenes is particularly useful when integrating virtual worlds into larger ones or when a virtual world is composed of different smaller ones. For example, we can build a virtual city and provide annotations for the different objects contained in it as well as a general description about the city. When incorporating this city into a virtual representation of a country or region, the scene description of the virtual city can be used to deduce a lot of information about the objects contained in it without the need to iterate through each of them to check its nature and attributes.
In this way, we would be able to indicate, for example, that the city in question is a touristic one and that it contains many hotels and archaeological sites. If the user is particularly interested in such sites, the city will be further investigated by the search engine during a query. On the other hand, we could also specify that a certain city is an industrial one and that it contains no archaeological or historical sites. This city would then be completely ignored by the search engine when querying for such sites. This would obviously, greatly increase the efficiency of the search engine.

Providing descriptions for complete scenes can be linked to the notion of complex objects. Complex objects are (logical) objects composed of different

smaller objects representing a part-whole relation. For instance, we can conceptualise a house in a virtual world as the combination of four adjoining walls and a roof. In the same way, we can define a city as combination of houses, buildings, streets, trees, etc.
Providing abstractions for complex objects in virtual worlds can be beneficial in many ways. It would allow the user to query for objects at a higher abstraction level (e.g. it would be possible to query for "houses"instead of for "four adjoining walls and a roof"). Furthermore, it would allow the search engine to query for objects by just "asking"the complex entity about the type of objects it contains. If the description of the complex entity does not fit the search parameters, the search engine moves on to the next complex entity without bothering with examining all the different objects contained in the first one. If, on the other hand, it appears that the type of object being queried for is present as a sub-entity of the complex object, the search engine can further investigate the complex object.
In this way, we could provide a description of the complete scene in a virtual world by considering it as a sort of logical complex entity containing other (complex) objects.

Unfortunately, the OntoWorld tool does not support complex objects at the time being. This is why we have tried to incorporate the notion of "container objects"in order to still be able to express part-whole relations. This approach offered us much more flexibility to describe the scene. Nonetheless, we recognise that this solution is not optimal. We therefore, strongly recommend the implementation of complex objects in OntoWorld together with the possibility to precisely describe the different entities composing them. Moreover, we suggest providing the possibility to (automatically) generate a complete description for the scene.

Chapter 5

Stage 2: The search engine

Once the virtual world has been annotated, a search engine can use this information to solve user queries. In this work, we want the search engine to be domain oriented. This means that the user must be able to use the domain terminology to query for objects in the virtual world. Furthermore, with this approach we have less chances of users making queries about objects that fall outside the domain for which a world has been built. For instance, if we have a world showing a virtual representation of an airplane and all its components, we want users to query for objects relative to this plane and not for a ”coffee cup”or a ”tennis racket”.

We have chosen to create a web-based interface to our search engine and used Java, VRML and the EAI (External Authoring Interface) to interact with our virtual world. We would have preferred to use the new X3D standard instead of VRML, unfortunately, the Scene Authoring Interface (SAI) meant to work with X3D is currently only in the specification stage. Nevertheless, X3D and VRML are compatible. This means that as soon as the SAI will be supported by the major web browsers, migration to X3D can be accomplished with a minimum of efforts. The structure of our solution is illustrated in figure 5.1.

The search engine is divided into two parts, namely the front-end and the back-end. The front-end deals with the GUI for the search engine and the plug-ins for the web page to access the virtual world. The back-end, on the other hand, represents the kernel of the search engine. It processes the user queries thanks to the annotations contained in the MPEG-7 compliant document as described in chapter four.
The following section describes the front-end of the search engine. We then take a look at the implementation of the back-end and show some examples of user queries and their results. Finally, the last section discusses the achievements as well as the limitations of this approach.

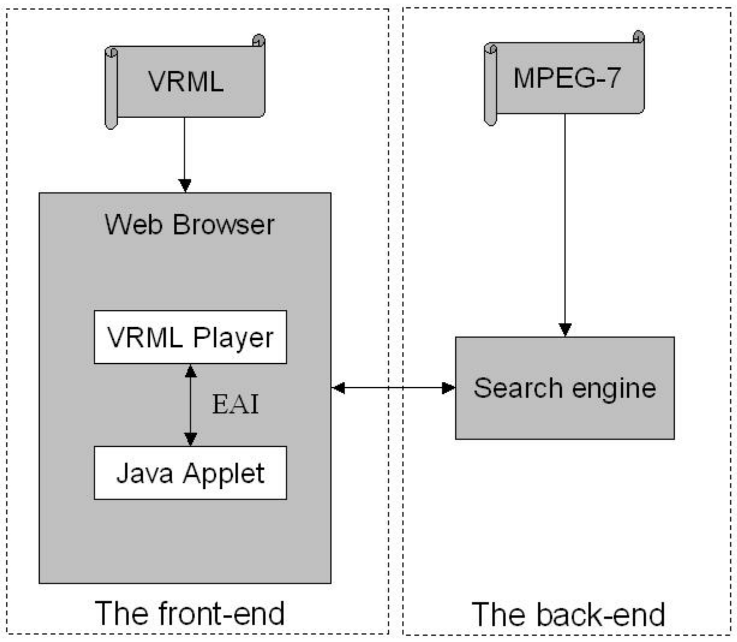


Figure 5.1: The structure of the search engine

5.1 The front end

5.1.1 The External Authoring Interface

The External Authoring Interface or EAI is a set of classes that allow Java to control a VRML world externally (for instance, through a web browser). Thanks to EAI, it is possible to send or receive events to/from VRML nodes, create new nodes and query the VRML world about its state and characteristics. In other words, EAI allows a Java program or applet to change the colour, position, size or any other property of an object in a virtual world, add new objects, etc.

Furthermore, thanks to EAI, it is possible to dynamically create and add viewpoints and sensors for objects in a scene. For example, it is possible to create a dynamic behaviour or to add some interactivity on the fly to static worlds. For more on EAI, please consult the EAI FAQ [14].

5.1.2 The interface

Initially, we have thought of implementing a natural language interface for our query solver, however this idea was soon abandoned. According to [1] such an interface is not intuitive enough for non-specialised users. Some of the problems they have encountered with this type of users were as follows [1]:

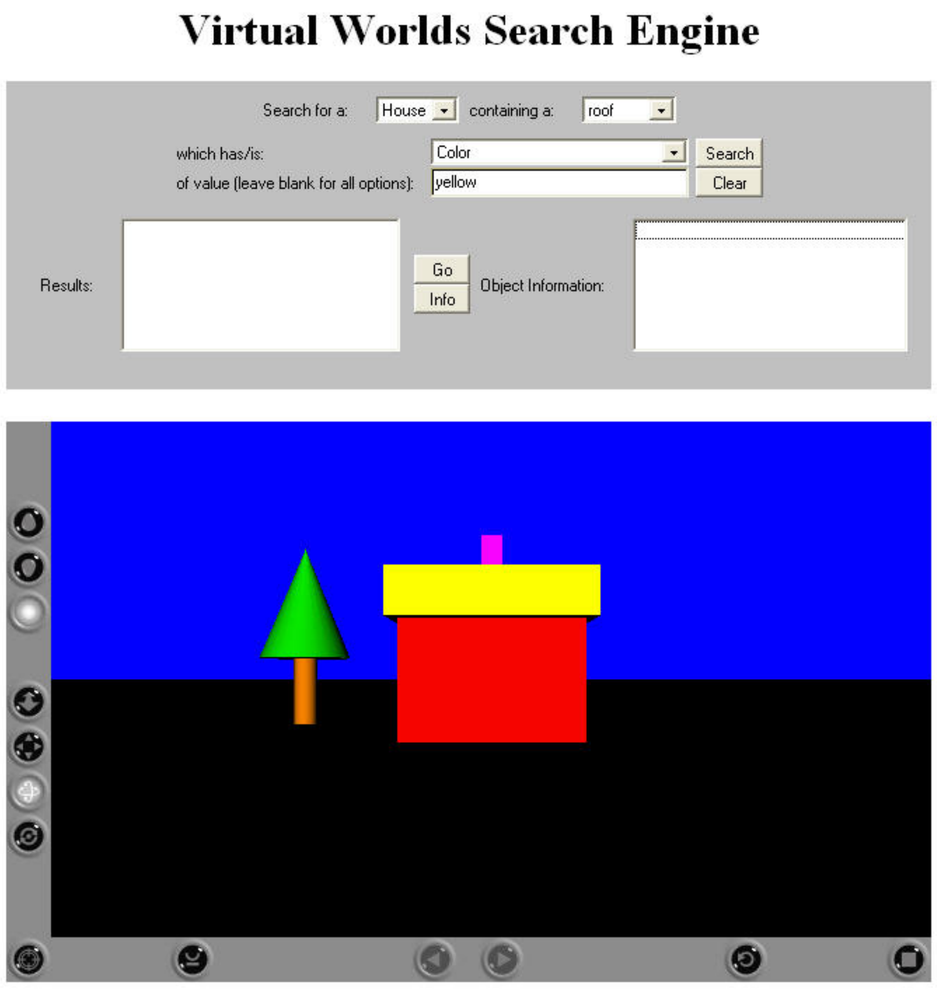
- *The interface required the users to know the type of objects contained in the world being queried.*

- *The interface required the users to know the linguistic values (e.g. short, tall, big, etc.), the linguistic hedges (e.g. very, slightly, etc.) and the quantifiers (e.g. many, few, etc.) they could use.*
- *Some linguistic values are ambiguous. For example, ”medium”refers to width, height, depth or size. therefore, if one of these linguistic values is included in the user query, the system could ask the user about the linguistic variable he is referring to.*
- *Some queries are ambiguous. For example, the query ”park with many tall trees near a tall building”can be interpreted as ”a park such that it has many tall trees and it is near a tall building ”or as ”a park which has many tall trees that are near a tall building”.*

Based on this, we have decided to use a ”Query By Example”querying model. This means that the user will be able to choose from certain values in drop-down lists containing object types and object properties and fill in the desired values in text fields. These object types and properties are retrieved from the MPEG-7 description.

Figure 5.2 depicts the user interface as used for our virtual world. Users are able to define what they are looking for in terms of the context, type of object as well as object properties.
When the applet is started, the MPEG-7 description is scanned and the different drop-down lists filled. Selecting a value in one of the drop-down lists results in the content of the following lists to be altered. For example, if the field ”House”is selected from the first drop-down list, only the types of objects contained in the ”House”are shown in the second one. In the same way, only the object properties of the objects contained in the second drop-down list are shown in the third list.
In this example, the user is searching for a house with a yellow roof. The ”Search”button is used to start the query. The results are listed in the ”Results”field. When the results are returned, the user is able to select one item from the list and click on the ”Go”button. This action results in the user being transported in front of the selected object in the virtual world.
Finally, the object information box is used to give information about specific objects in the virtual world, while the ”Clear”button is used to clear the different fields in order to reset the search engine.

We have inserted the ”Any”keyword in every drop-down list in order to help to user when he is uncertain or ignorant of some object property or type. This keyword allows the return of all objects or properties in the specified category. In this way, the chance of returning null results is greatly reduced as queries can only be made on objects or properties present in the virtual world. This, in turn, greatly reduces the user frustrations and further encourages his to explore the world.


Virtual Worlds Search Engine

Search for a: House containing a: roof

which has/is: Color

of value (leave blank for all options): yellow

Figure 5.2: The search engine graphical user interface

5.2 The back end

The class diagram of the query solver is given in figure 5.3. The class Mpeg7Parser is used to parse the MPEG-7 document while the QuerySolver class is used to solve the queries received from the main class: Vese. The class Converter is used to determine the orientation of the camera for the definition of viewpoints.

5.2.1 The Converter class

The Converter class is used to determine the orientation of the camera when viewing a specific object. This class is called only if a viewpoint has to be dynamically created. That is, only if no viewpoint has been defined in the VRML file for the specific object [1]. The algorithm used is based on Stephen

[1]For more on viewpoints please refer to the next chapter.

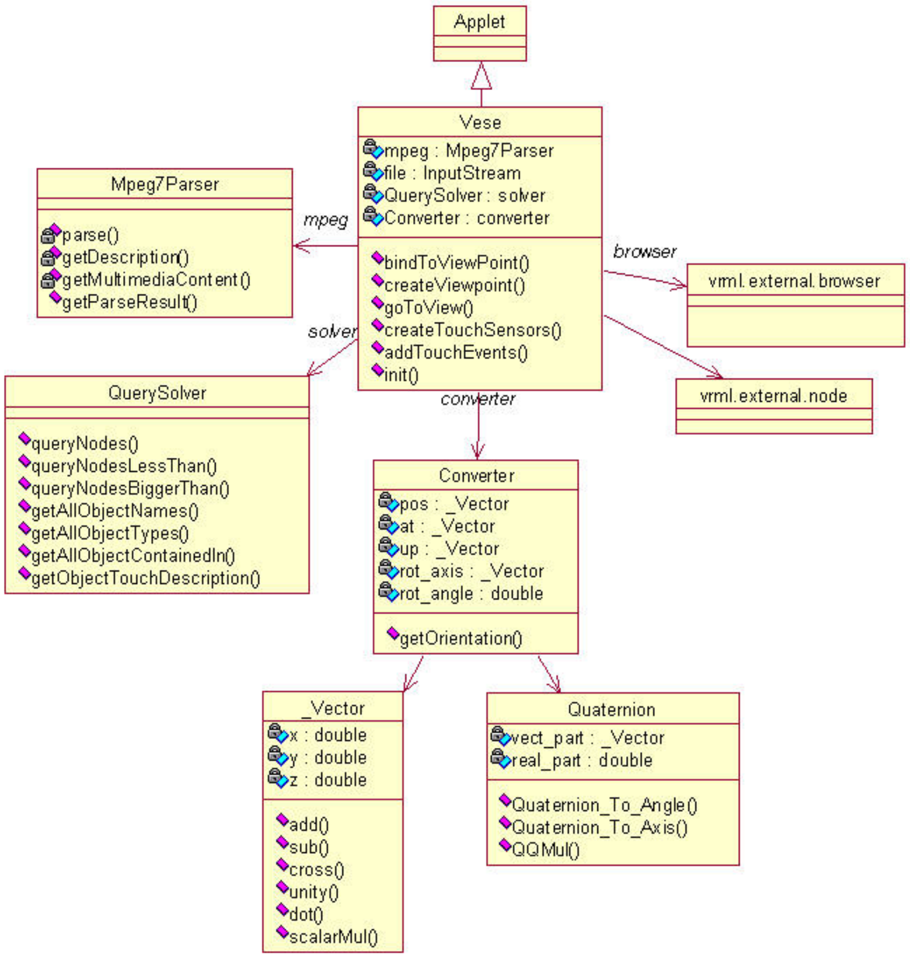


Figure 5.3: Class diagram of the Virtual Environments Search Engine

Chenney's "Orient"and translated from C to JAVA. To know how this class works, we refer the reader to appendix A.

5.2.2 The Mpeg7Parser class

When the applet is first started, the MPEG-7 description file is parsed and the different object descriptions are stored in a list of hashtables. Each hashtable represents an object description. The annotation keywords are reused to store the annotations in the hashtable.

We chose to implement our own parser to parse the XML document rather than using and XML parser like DOM or SAX. The decision was mainly due to the fact that our MPEG-7 documents have a limited number of different XML tags. Furthermore, the order of occurrence of these tags can easily be predicted. It was therefore, much simpler to build our own parser rather than using DOM or SAX parsers.

5.2.3 The QuerySolver class

The query solver takes as input a list (Vector) containing all the object annotations found in the MPEG-7 document. Each object annotation is represented by a hashtable.
When a query is entered, the query solver checks the list of objects in search of the specified object type. Once an object type corresponding to the desired one is found, the query solver checks the occurrence of the keyword in the object's keyword list. If no keyword matching the one entered is found, the query solver moves on to the next object in the list. If, on the other hand, the keyword is found, its value is then checked against the one entered. Our search engine is able to differentiate between numerical and alphanumerical values. For alphanumerical values, our query solver checks the occurrence of the specified string in the annotation text. The comparison is not case sensitive. On the other hand, if a value is preceded by a comparison sign such as `'<'` or `'>'`, this value is treated as a numerical one and a numerical comparison takes place.
If the object matches the query, it is added to a list of correct results. The search engine continues until no more objects are left in the initial list. Once this is done, a list of all correct results is returned.

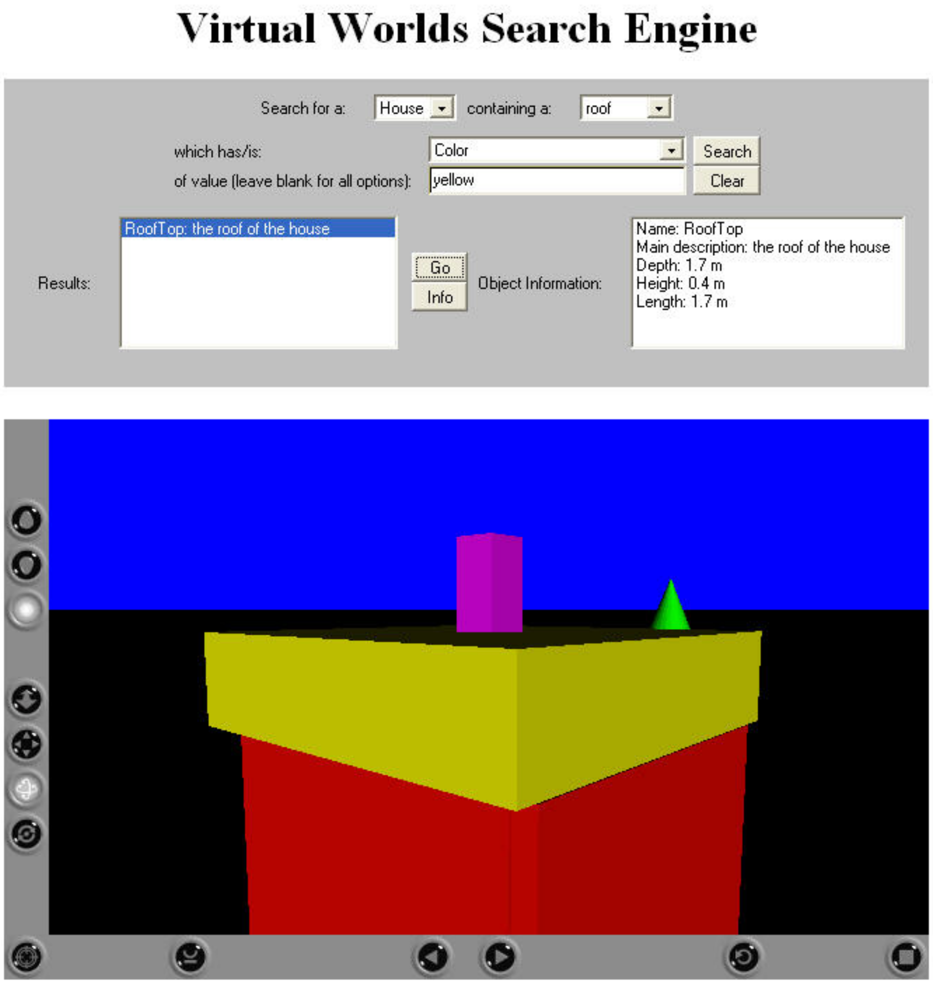
5.2.4 The main class

The main class "Vese"is the implementation of the front-end of the search engine. It creates the user interface and forwards queries to the query solver. It is also concerned with displaying the search results and interactions with the virtual world.

5.3 Results

The result of the query for the house with a yellow roof is shown in figure 5.4. Of course, since no object of type "house"exists in our virtual world (since it is just a logical concept), the object "RoofTop"is returned instead. This is due to the fact that the search has been conducted on this particular item. By selecting this object from the list and clicking on the "Go"button, the user is transported in front of the roof of the house. Since only one house exists in our tiny world, the query returned only one object.
Figure 5.5 on the other hand, shows the result of querying for a "Very high"wall. In this case, four results were returned since all four walls of the house were tagged as being very tall. Alternatively, the user can search for walls with heights equal or superior to a certain value.

Finally, the information field provides information concerning the selected object. In this case, the object name, its main description as well as some of its attributes like height, width and so forth are displayed. It can be noted, that the choice of what information to display in the information field is left to the world creators. This information should reflect the attributes that are the most important to the visitor. For example, in a virtual music shop, it is


Virtual Worlds Search Engine

Search for a: House containing a: roof

which has/is: Color

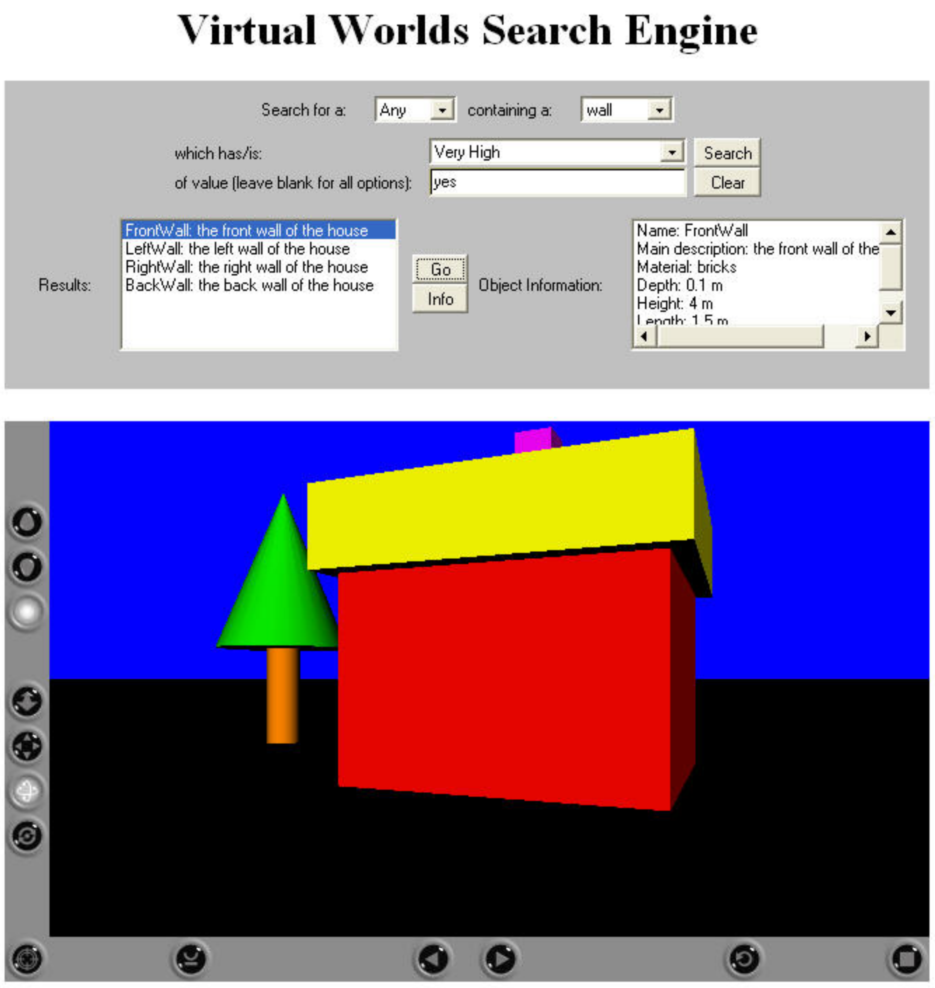
of value (leave blank for all options): yellow

Results: RoofTop: the roof of the house

Object Information:
Name: RoofTop
Main description: the roof of the house
Depth: 1.7 m
Height: 0.4 m
Length: 1.7 m

Figure 5.4: The query result for a house with a yellow roof

useless to give information about the size and shape of a compact disk since the user would be more interested in information about the record itself, such as the name of the author, the title of the CD, the type of music, etc.
On the other hand, it could also be possible to provide a sort of information filtering in function of the user's preferences. This would allow the user to define what he thinks is important to view.



FrontWall: the front wall of the house
LeftWall: the left wall of the house
RightWall: the right wall of the house
BackWall: the back wall of the house

Name: FrontWall
Main description: the front wall of the
Material: bricks
Depth: 0.1 m
Height: 4 m
Length: 1.5 m

Figure 5.5: The query result for a very high wall

5.4 Discussion

Thanks to the domain oriented description of the virtual worlds, we have been able to create a search engine, which in turn, is also domain oriented. Users are able to query the world with the terminology of the domain expert. This fact will become more apparent when we present our proof of concept in chapter seven.
Even though our search engine is basically just a keyword matching query solver and is not equipped with advanced artificial intelligence reasoning algorithm such as Fuzzy logic or neural networks, we are still able to solve queries expressing vagueness. In fact, queries such as ”find a table owned by xyz”or ”find an expensive car”or even ”find an expensive car that is not old”can perfectly be solved by our search engine. Nonetheless, the efficiency of the search engine, is deeply affected by the quality of the annotation. That is why it is essential to create a good description for the virtual world being queried.

From our small example, it is clear that searching for complex objects in virtual worlds while no such objects are explicitly defined is not the best solution. However, given the current status of implementation of the OntoWold tool, it was the best we could do in the time given. Thanks to our implementation of the concept of containers, users are still able to query (or have the impression to query) for complex entities (such as the house in our example) even though, this entity has no physical representation in the virtual world. In chapter seven, we will see how information about container objects can be used in a different way.

We have avoided the use of a natural language interface for the query solver as this approach proved to be not efficient [1]. Instead, we chose to implement a "Query By Example"interface. This interface had the main advantage of allowing users to easily find what they are looking for even if they have little or no knowledge of the content or the terminology used in the virtual world.
Nonetheless, we acknowledge that this approach might not be the best one especially with very large virtual worlds. For instance, it would not be efficient in this case to make the user choose from a drop-down list of thousands of objects and even more properties. To resolve this problem, a combination of both approaches could be used. That means, a user could start by typing the kind of object he is looking for (e.g. "table") and submit the query. The search engine then searches for all possible occurrence of the object being queried and presents the user with a menu where he can further fine-tune his search by using a Query By Example interface for example.

Chapter 6

Stage 3: Navigation

We have mentioned at the beginning of this document that providing assistance to visitors of virtual worlds was crucial in order to keep users (and especially novice users) interested in the world. When visitors cannot find what they are looking for, they can become rapidly frustrated and leave the world [8]. If such a world was a virtual shop for instance, it would have failed miserably in attracting a maximum number of visitors, but most importantly in making them buy goods. It is therefore, in the best interest of the virtual world creators to include navigation aids in their creation.

During the years, a number of navigation aids have been invented for virtual worlds. Navigation aids like maps, help visitors orient themselves in the virtual environment, however they are sometimes hard to interpret especially by novice users. Furthermore, they might lack the level of details required by some applications [1]. Guided tours on the other hand, allow the users to assume a passive role while they are lead through a virtual world and given explanations about the objects and places populating the world. While this method allows users to get information about the places being visited and not miss important parts of the world, it might not be adequate for all types of virtual environments. For instance, in a large virtual city, it is unthinkable to explore every part of the world through a guided tour as users will inevitably lose interest after some time or would want to visit only specific parts of the world. This problem can be solved by querying.

As explained earlier, navigating a virtual world can be done in two different ways: by browsing or by searching. Browsing defines a state where the user just looks around to see what is available (with or without a specific target in mind) while searching concerns looking for a known target.
It could be noted that using one way of navigation does not exclude the use of the other. For instance, a user could query the world for a specific target and ask to be transported to that object. He could then use browsing to explore the environment in the immediate vicinity of the object having been queried for.

6.1 Browsing and providing information about objects in a virtual world

When browsing is concerned, the user is free to explore the virtual world as he wishes. Therefore, it becomes important to be able to provide him with information about the objects he comes across.

6.1.1 Providing information in virtual worlds

For some applications like virtual museums and virtual training sites, providing information to the users is crucial as it is a fundamental part of their experience in these worlds [8]. Yet, providing information about the different objects populating a virtual world is far from being trivial. This is especially true for large virtual worlds having hundreds or thousands of objects.
Typically, instructions to users are given by introductory pages, which give a description of the world being visited, the different objects populating it and how a user can interact with them [8]. Most of the visitors to a virtual world will be too eager to explore the world that they will skip these pages or a good deal of them. This may lead them to fail to explore the world properly or not be aware of some objects or places in it and how they can interact with them [8].

Guided tours are another approach used with larger virtual worlds to provide information about specific objects and places. Guided tours usually exploit humanoid animated characters (the so-called H-Anim characters) to guide the user through pre-defined routes. Guided tours can be seen as both a navigational aid (since they help users find places and objects) and information aids (since they give information about the objects and places visited) [8].
While guided tours provide an excellent way of informing users about the different objects and places populating a virtual world, they do not provide a solution for users who want to explore the world on their own. Such users must be able to move freely in the virtual world, but still be able to inquire about the objects they come across.

6.1.2 Using annotation to provide information in virtual worlds

While search engines have their usefulness as we have seen, a user needs not necessarily to use them. One can choose to explore the virtual world by wandering around with or without a specific goal in mind. Such visitors must be able to obtain information about a particular object they find interesting by simply pointing at it or clicking on it. While it is possible to include such scenario's in a VRML or X3D file (i.e. displaying an object's properties when the user points to it), such an information generally needs to be hard-coded, which is quite time consuming and hardly flexible.
A better approach would be to simply use the object descriptions already stored in the MPEG-7 file for this purpose. Once the user clicks on a particular object (or a nearby icon bearing a question mark, for example) the information relative to that object is fetched from the MPEG-7 description and displayed on the screen.

At this point, the object of discussion would be what information to display and in what amount. It is clear that displaying all information stored in the MPEG-7 description for a particular object poses no technical difficulty. Nonetheless, it can be argued that a visitor to a virtual world would not be interested at first to all the information available for a particular object, which could prove to be overwhelming. Most of the time, providing information about the kind of object (e.g. football, portrait, table, etc.) and its purpose in the virtual world should suffice. The user may then decide to continue exploring the virtual world without further caring about that particular object. On the other hand, the possibility exists that the user would be more interested in the object once this preliminary information is provided. At this point, it would be adequate to allow the user to further inquire about that particular object. For instance, the user may want to know the material of a table or how much horse power contains a particular car. But what could be even more interesting would be the information about the different ways the user can interact with that particular object. Indeed, finding out how a user can manipulate a specific object may not be obvious at first, especially for novice users. For instance, without guidance, a user can find it difficult to know that clicking on a book would make it open on a particular page, or that standing in front of a door would make it open. This is the kind of information that can easily be retrieved from the MPEG-7 document and displayed as wished without big efforts.

This approach greatly reduces the time and efforts it takes to include interactivity with the user in a virtual world, while at the same time providing discrete but efficient guidance. For instance, some users may not be very keen on asking for help or having a virtual guide telling them everything about a particular world. These users would usually prefer to explore the world on their own and by doing so, may miss important aspects of the world if they are not guided. Providing information on objects as well as on the way the user can interact with them in the way described above, could help to overcome this problem without giving the user the impression that he is being helped.

We use this technique in order to provide information about the different objects in our virtual world. When the program starts, a list of the objects annotated is retrieved from the MPEG-7 document. Once this is done, a ”touch sensor”node is added to each of these objects in our virtual world. Touch sensors allow for an event to be generated when the object concerned is clicked. Later on, when the user clicks on one of these objects the information relative to the correct node is displayed on the screen.

6.2 Navigation by querying

Querying is another option for navigation through a virtual world. The user is able to enter a query about a specific object or object type. Once the list of object is returned, he can choose the one that best fits his needs and ask to be transported to that object. Since the querying process has been discussed in the previous chapter, we will focus in this section about transporting the user from his current location to the desired object. This transportation can occur in one of two ways:

- Jumping is the easiest way of navigation through the virtual environment. The user is simply taken from one location to the next. While this method can sometimes be desired for users familiar with the virtual environment they are in, it can at other times cause confusion or disorientation for novice users.
- With path following, on the other hand, the system guides the user to the specific location following a pre-defined or auto-generated route at a slow enough pace to allow the user to visualise where he is being lead to. Path following is the trickiest of all navigation options as, for instance, mechanisms must be provided to ensure that the user does not go through obstacles or walls.

Since the OntoWorld tool does not support the creation of navigation networks yet, and implementing it would have taken too much time, we have used jumping as means of transportation. Once the user clicks on a specific object from the list returned by the search engine, he is immediately transported in front of that object.
As we will see in the next chapter, our test world is small enough not to cause disorientation when a user is transported by jumping. In a larger virtual world, this might not have been the case. However, in both cases, a user's view must be correctly oriented in order to correctly view the desired object. This topic is discussed in the following section.

6.2.1 Creating viewpoints

An important aspect to consider when transporting a user through the virtual world is to correctly position the object in his field of view. In other words, the camera must be correctly positioned and oriented to allow the user to perfectly view the object. Positioning the camera is called creating *viewpoints* in virtual worlds.

Viewpoints can either be defined when the virtual world is being created or dynamically created at run-time. We have chosen this last option for our implementation. This approach has the advantage of always creating the correct viewpoint even if objects are moved in the virtual world. Appendix A explains Stephen Chenney's algorithm to calculate the correct orientation of the camera when creating viewpoints dynamically. Since our world is static, we have chosen to create all the viewpoints when the program is loaded so that we do not need to recalculate the viewpoint each time the user want to go to the specified object.

6.3 Discussion

We have seen that querying facilitates navigation and exploration of virtual worlds. This is especially true if the virtual world is a large one having tens or hundreds of objects. Querying allows users to easily find object and places that would otherwise be hard to find without proper knowledge of the layout of the world. We were able to provide visitors to virtual worlds with a powerful search

engine in order to allow them to easily find what they are looking for and be transported to those objects with a simple click of a button.
Due to time limitations, however, we were only able to implement jumping as means of transportation. This mechanism has the clear disadvantage of possibly causing disorientation for the user and especially for first-time visitors to a virtual world or novice users. This fact becomes even more important as the virtual world gets larger.
Nonetheless, there exist other means of transportation such as navigation networks that can be implemented in future works. A number of recent studies such as [9], have even shown that these networks can be dynamically created at run-time.

There are other ways of exploring a virtual world, of course. One of such ways is browsing. Browsing defines a state where the user wanders around the virtual world with or without a specific target in mind. For users preferring this type of exploration, it is still essential to provide them with assistance. We have been able to implement a mechanism by which a user can click on any object (or on a nearby tag) in the virtual world and immediately receive information concerning that object. This information can also include the different ways in which a user can interact with the specific object.
As mentioned, this approach greatly reduces the time and efforts needed to provide efficient guidance in virtual worlds as the use of virtual guides for example, could be avoided to some extent.

Chapter 7

Proof of concept

In order to prove our approach and show that it can be applied to other types of virtual worlds other than our tiny world example, we have used the example of an E-shopping furniture store like the well-known IKEA. This E-shop contains furniture such as beds, wardrobes, desks and so on. We have constructed this world using OntoWorld and added a semantic description for each piece of furniture.

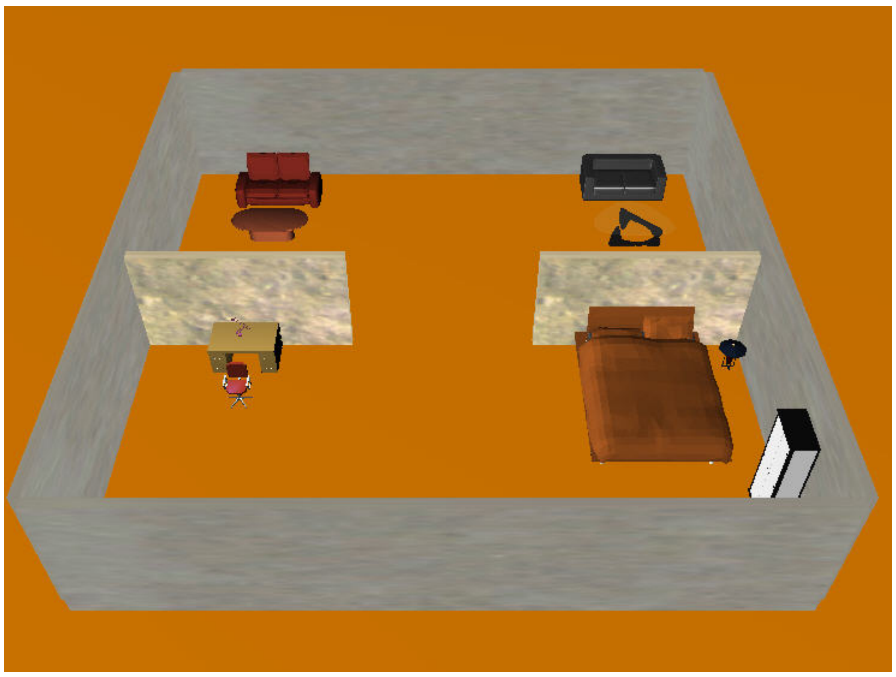


Figure 7.1: The virtual shop

Our virtual shop is shown in figure 7.1. It is divided in three logical areas: a study room area, a bedroom area and a living room area. The objects contained in each area are as follows:

- In the study room area:
  - a desk,
  - a rolling chair and
  - a study lamp.
- In the bedroom area:
  - a double bed,
  - a bedside table and
  - a glass cupboard.
- In the living room area:
  - a two-seat couch,
  - a three-seat couch,
  - a leather couch,
  - a glass table and
  - an oval table.

7.1 Annotation

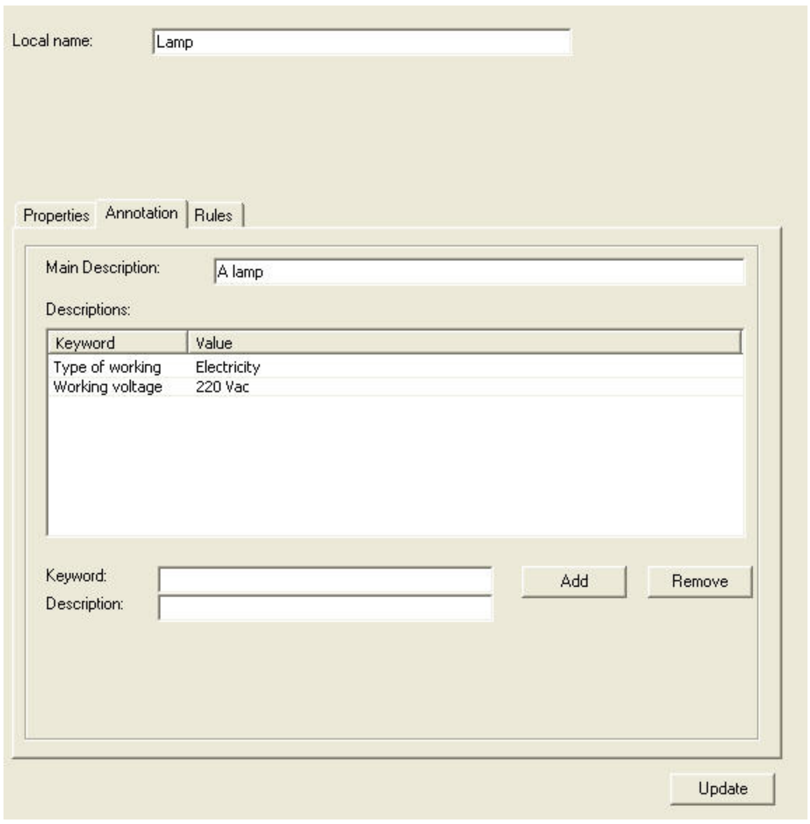
7.1.1 Annotation of concepts

We have started by annotating our virtual shop at a conceptual level. For each object type, we have entered a main description as well as different annotations. For example, objects of type ”Lamp”have ”A lamp”as their main description (which is the default description for all instances of ”Lamp”). We have also defined that all lamps represented in our virtual shop work with electricity[1] and that the voltage to be applied is equal to 220 Vac. This annotation is shown in figure 7.2. This description is later inherited by all instances of this object type.

Later, some rules were added in order to automatically add annotations to instances. This is shown in figure 7.3. Our rules comprised the adjectives: *Small*, *Tall*, *Expensive* and *Cheap*. These rules will later be used to add annotations to instances if there is a match. We have defined our lamps to be tall if their height is superior to 49 cm. In the same way, these items will be tagged as small if their height does not exceed 30 cm. Finally, all lamps costing more than 30 euros will be considered as expensive while those with a price inferior to fifteen euros will be considered as cheap.

The properties of the concept ”Lamp”that are interesting for our annotations are ”Height”, ”Weight”, ”Price”and ”Packed_Size”. Each of these properties have

[1]Indicating that a lamp works with electricity might seem trivial at first. However, if this shop was an antiques furniture shop for example, it could have contained different types of lamps such as oil lamps, etc. By indicating the type of working, we can differentiate between the different type of lamps.
Furthermore, it is possible that a user is specifically searching for an antique oil lamp, for example. By indicating that all the lamps contained in the our virtual shop work with electricity, we avoid returning non-pertinent results to the user.


Local name: Lamp

Main Description: A lamp

Descriptions:

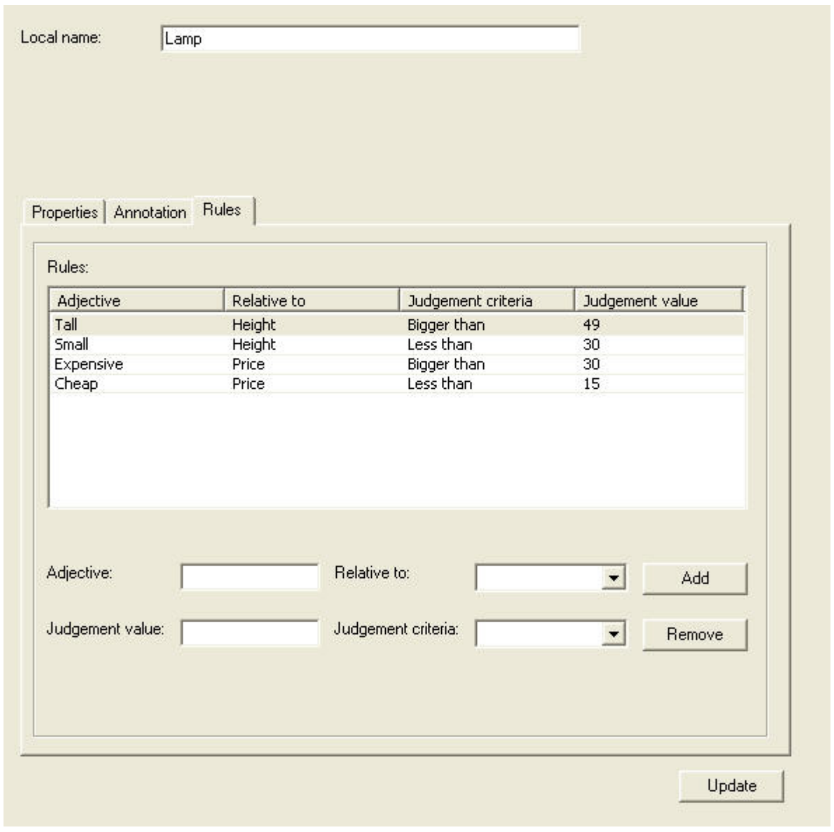
Keyword	Value
Type of working	Electricity
Working voltage	220 Vac

Keyword:

Description:

Figure 7.2: Annotating a Lamp concept

been given default values. These default values will be inherited by all instances of this concept. Naturally, these values can be adapted at the instance level.


Local name: Lamp

Rules:

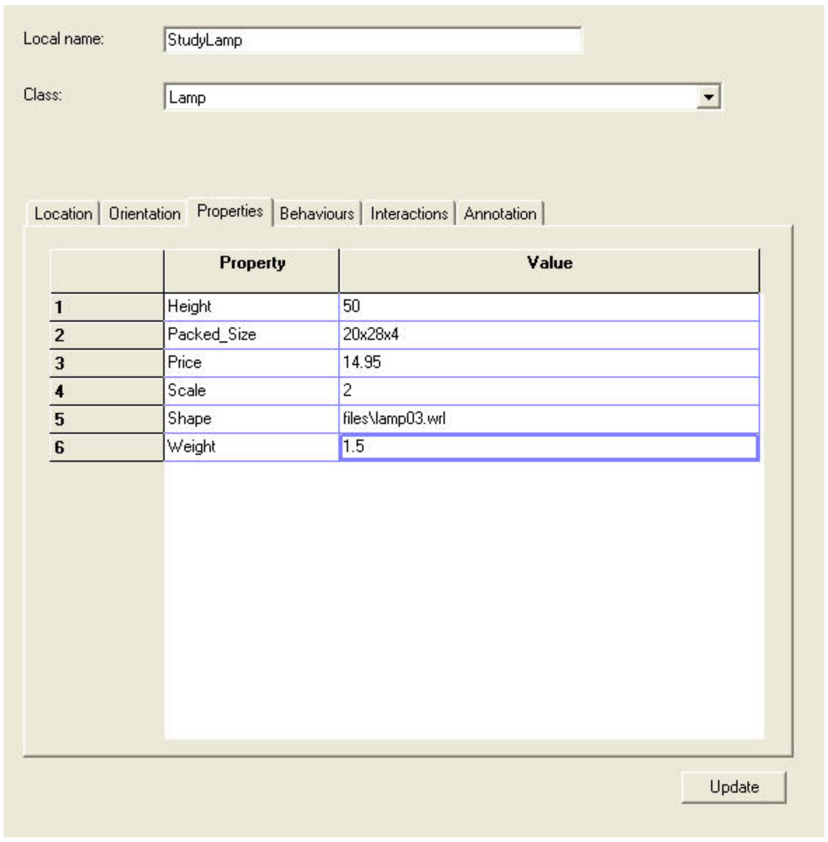
Adjective	Relative to	Judgement criteria	Judgement value
Tall	Height	Bigger than	49
Small	Height	Less than	30
Expensive	Price	Bigger than	30
Cheap	Price	Less than	15

Figure 7.3: Adding rules to a Lamp concept

7.1.2 Annotation of instances

After the concepts, the annotation is performed at the instance level. Thanks to the annotation of the parent concept and the different rules entered, a number of annotations are automatically added to the instance description. Figure 7.4 shows the properties of the instance ”StudyLamp”(an instance of the concept ”Lamp”). From this figure, we can seen that our study lamp has a height of 50 cm, a price of 14.95 euros, and that its weight is equal to 1.5 kilos. The other properties listed, with the exception of the packed size, are used to represent the object in the VRML file. They are, therefore, of no interest for the annotation. Figure 7.5, shows the annotations that have been added to this instance.

In this example, we see that the annotations inherited from the parent concept (i.e. ”Type of working”and ”Working voltage”) have been automatically added to the list. In the same way, we have inherited a main description that we adapted from ”A lamp”to ”A study lamp”to indicate that this lamp is in fact a study lamp. Next to the inheritance of annotations, we can see that the adjectives *Tall* and *Cheap* have been added as well. These adjectives are the results of rules entered at the concept level. There, it was specified, among other things, that a lamp should receive the adjective *Tall* if its height was superior


Local name: StudyLamp

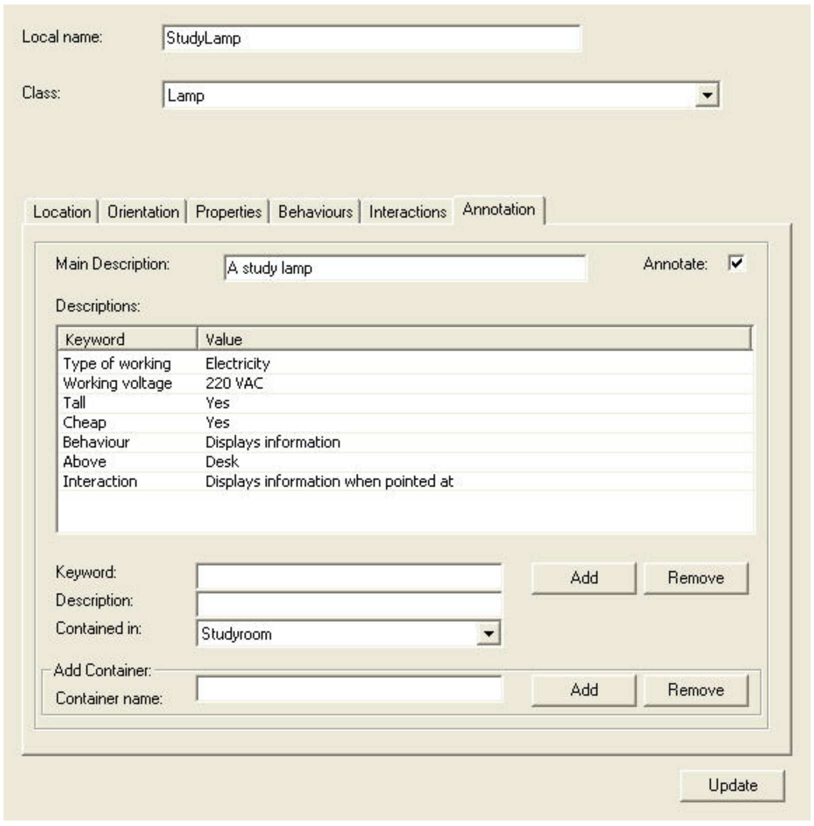
Class: Lamp

	Property	Value
1	Height	50
2	Packed_Size	20x28x4
3	Price	14.95
4	Scale	2
5	Shape	files\lamp03.wrl
6	Weight	1.5

Figure 7.4: The properties of the StudyLamp instance

to 49 cm. In the same way, every lamp with a price inferior to 15 euros should be tagged as *Cheap* . In this case both rules produced a match.
As a last step, the spatial relation as well as the interactions and the behaviours of the instance were added. In this case, the study lamp is placed ”Above”the object ”Desk”. The behaviour associated with this object is ”displaying information”. Finally, the interaction of this object with the user concerns displaying information when the user points at it with the mouse.

At this point we are free to add or remove any annotation to/from the instance. As a last step, we have indicated that this lamp belongs to the studyroom. To do this, we simply added this room to the list of containers. Later on, when a user will query for objects belonging to a studyroom, our study lamp will be one of them. Annotations of the other objects is done in the same way.


Local name: StudyLamp

Class: Lamp

Main Description: A study lamp

Annotate: ✓

Descriptions:

Keyword	Value
Type of working	Electricity
Working voltage	220 VAC
Tall	Yes
Cheap	Yes
Behaviour	Displays information
Above	Desk
Interaction	Displays information when pointed at

Keyword:

Description:

Contained in: Studyroom

Add Container:

Container name:

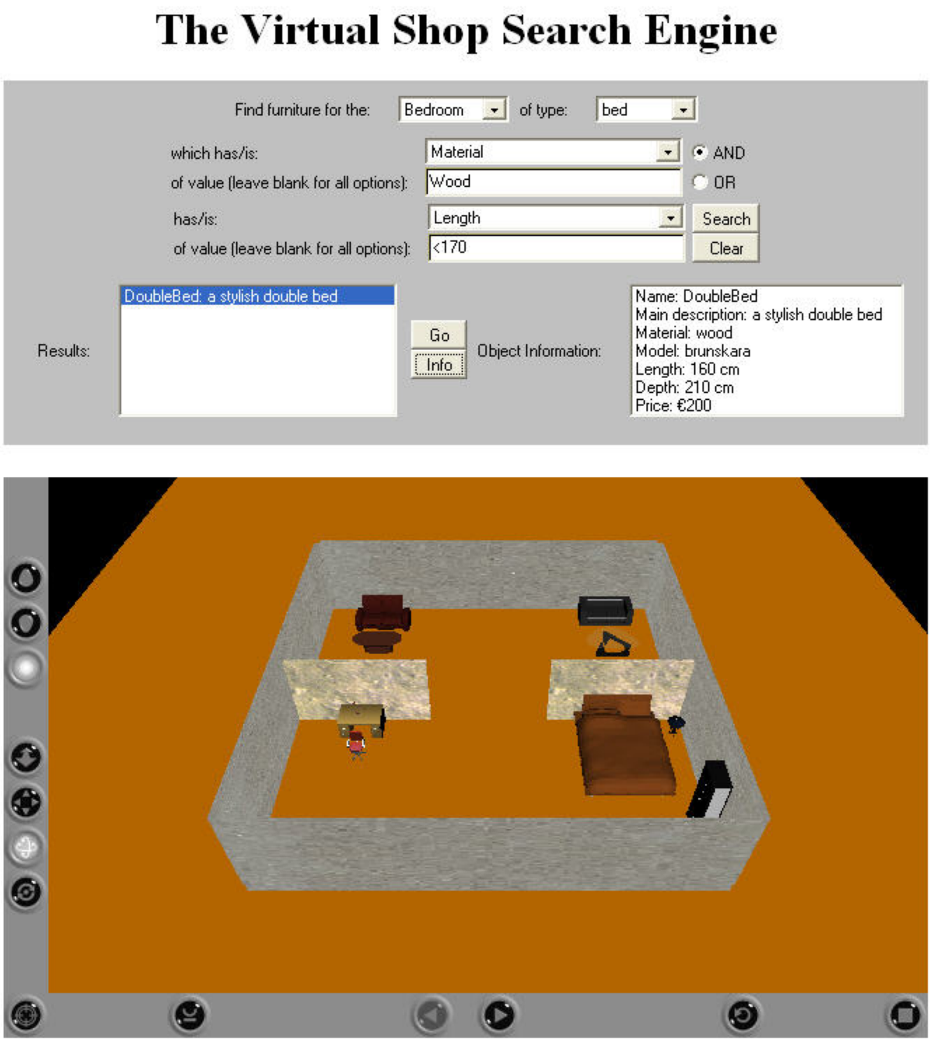
Figure 7.5: Results of the automatic annotation of the instance StudyLamp

7.2 Querying and navigation

Our virtual shop is shown in figure 7.5. We have developed an applet with the help of Java and EAI for the search engine. The virtual world is embedded in the web page thanks to the Cosmo player plug-in for Internet Explorer. Unfortunately, EAI currently only works in combination with Microsoft's Java Virtual Machine, which supports only version 1.1 of Java. We were therefore, not able to take advantage of the latest Java technologies.

Users are able to wander around the virtual shop and examine the different furniture or alternatively query the system for a specific piece of furniture. Each piece of furniture has a name, a type, a room where it belongs (e.g. bedroom, livingroom, etc.) as well as other properties such as price, weight, etc. further, we have tagged the furniture with subjective adjectives such as *expensive*, *cheap*, etc.

When the applet is started, viewpoints and touch sensors are created for every annotated object in the scene and added to the scene graph. Normally, objects that were not annotated such as the walls and the floor, are of no in-


The Virtual Shop Search Engine

Find furniture for the: Bedroom of type: bed

which has/is: Material [✓ AND] OR

of value (leave blank for all options): Wood

has/is: Length

of value (leave blank for all options): <170

Results: DoubleBed: a stylish double bed

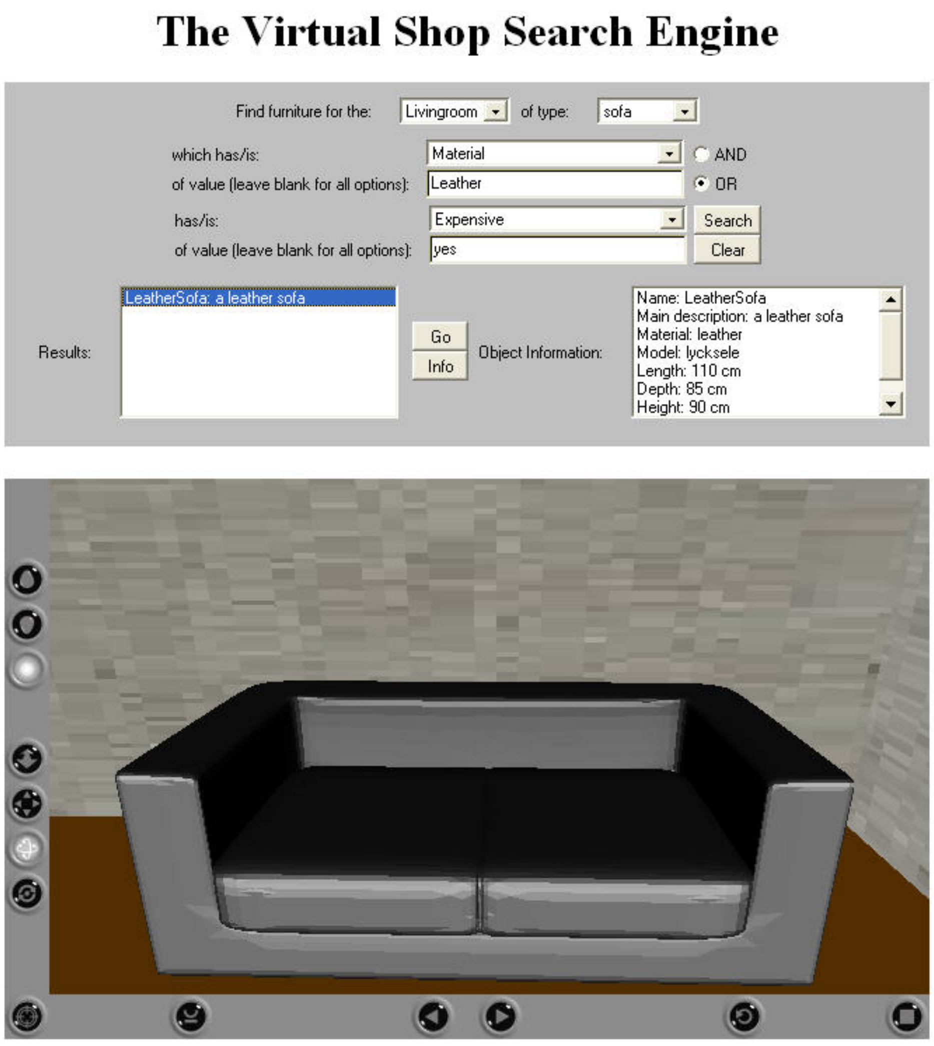
Object Information:

Name: DoubleBed
Main description: a stylish double bed
Material: wood
Model: brunskara
Length: 160 cm
Depth: 210 cm
Price: €200

Figure 7.6: The virtual shop applet

terest for the user. Therefore, there is no need to attach viewpoints or touch sensors to them.
If the world creators prefer to set a specific viewpoint for an object (e.g. for objects that are best viewed from a certain angle), they can do so by creating it in advance and storing it in the VRML file. The tool is created so that if a viewpoint for a specific object already exists, no new viewpoint will be created and the one existing will be used.

Users can choose to search for a specific type of furniture. For this they can start by choosing where the furniture belongs to (i.e. bedroom, studyroom or livingroom). Once they have done this, the list of all the type of furniture available in that room is displayed. For example, choosing furniture that belongs to the bedroom would yield the following result: ”Bed”, ”Table”and ”Cupboard”. Let’s suppose the user is searching for a bed. By choosing this article type and


The Virtual Shop Search Engine

Find furniture for the: Livingroom of type: sofa

which has/is: Material

of value (leave blank for all options): Leather

AND [✓ OR]

has/is: Expensive

of value (leave blank for all options): yes

Search Clear

Results: LeatherSofa: a leather sofa

Go Info

Object Information:

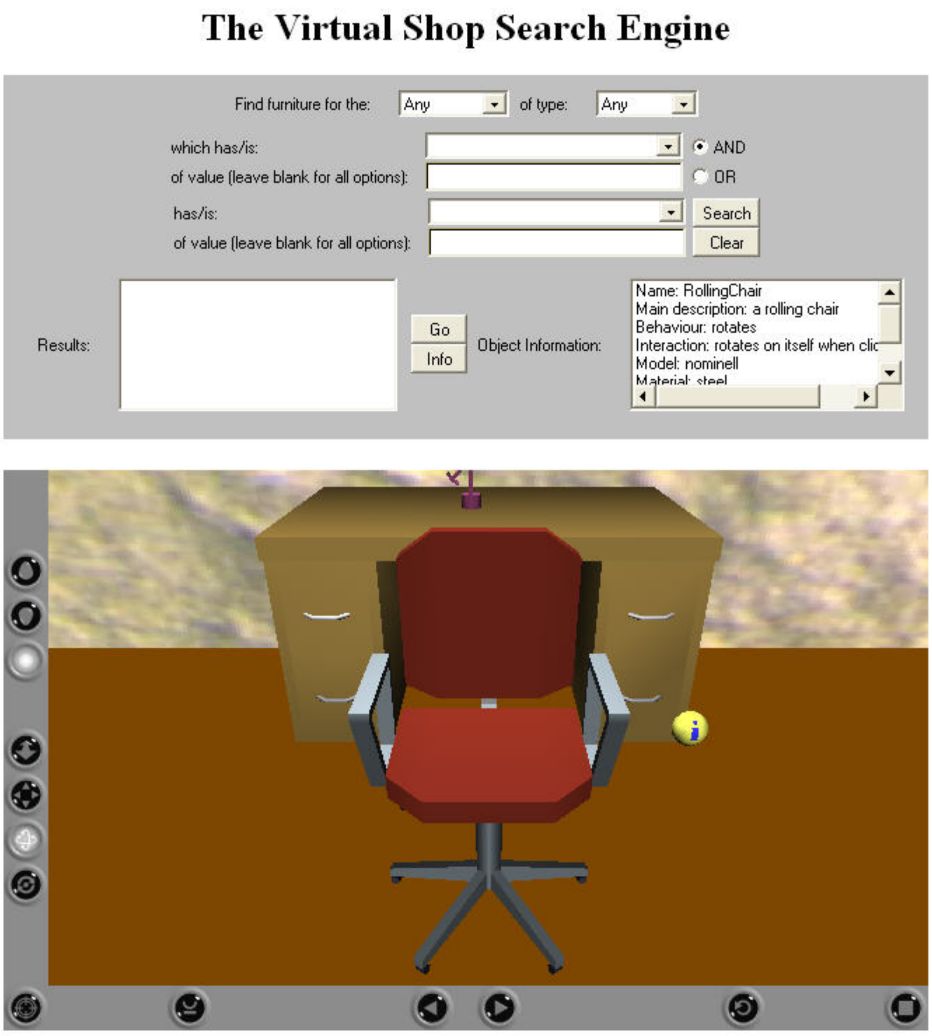
Name: LeatherSofa
Main description: a leather sofa
Material: leather
Model: lycksele
Length: 110 cm
Depth: 85 cm
Height: 90 cm

Figure 7.7: Transportation to a selected object

pressing the "Search"button, a list is displayed of all the different beds contained in the shop (in this case, however, there is only one). Alternatively, the user may choose to refine his search. In this case, he can choose one of the properties related to the bed and enter a search criteria (e.g. "Bed with Price < 200 euros). Once again, hitting the "Search"button will return all the objects that match these criteria. The user can then click on an object from the list and press the "Info"button to display the different information relative to that object. The information displayed are the name of the object, its main description, its model, the main material it is made of, its price and its (real) dimensions, if available. This information allows the user to compare between different items in order to find what suits his best. Of course, there are other ways in which the user can query this world. He can choose to search for all objects made of wood for example or for all tables that are not expensive, etc.
Finally, by clicking on the "Go"button, the user can be transported in front of

the selected object. This button also displays the information relative to the chosen item. This information is also displayed if the user clicks on an item in the virtual world. This allows him to explore the world by browsing with or without the use of the search agent.

Figure 7.7 shows the result of querying for a ”an expensive sofa or a leather sofa”. The object ”LeatherSofa”is returned. By clicking on the ”Go”button, the user is transported in front of the object. We can notice that the user’s field of view is correctly positioned to view the item. At the same time, information about this furniture is displayed in the information field. It can seen that this sofa is indeed made of leather and that its model is called ”Lycksele”and that it has a length of 110 cm, etc.


The Virtual Shop Search Engine
Find furniture for the: Any
of type: Any
which has/is:
of value (leave blank for all options):
AND
OR
has/is:
of value (leave blank for all options):
Search
Clear
Results:
Go
Info
Object Information:

Name: RollingChair
Main description: a rolling chair
Behaviour: rotates
Interaction: rotates on itself when clic
Model: nominell
Material: steel

Figure 7.8: Displaying information about objects when browsing

Finally, for users that prefer to explore the virtual shop by browsing, we have provided means of retrieving information about the furniture. By clicking on any object, the information relative to that object is displayed in the information field. For objects that can be interacted with, we have preferred to add a small information icon to the scene next to the object in question. The purpose of this icon is to provide the information relative to the displayed object when clicked. In other words, the information relative to a dynamic object will not be displayed if that object is clicked. Instead, it will be displayed when the information icon associated with that object is clicked. This has the advantage of not interfering with the object's interaction. Furthermore, such a strategy makes it possible to easily differentiate between static and dynamic objects. From figure 7.8, it is possible to see that the information displayed also indicate the means to interact with the selected object. In this particular case, the chair can rotate on itself if it is clicked.

This last point is particularly useful for virtual objects having different ways of interacting with the user. Indeed, in the virtual shop, it could be possible, for example, to view each piece of furniture from different angles, change its colour or even put different ones together in order to view how a room would look like, etc. These are different types of interactions, each triggered by a specific user action. It is therefore, important to let the user know how he can trigger each of these actions. The information provided about each object is a means of achieving this.

7.3 A short case study

In order to test our approach with real users (i.e. non VR experts), we have asked a small number of people to use our virtual shop and its querying possibilities. Users were simply asked to use the interface to query for furniture. The results showed that users could understand the concept of the search engine quickly. They were able to easily query the virtual world and be aware of the different objects contained in it. They have appreciated the fact that the terminology used to query for furniture is the same one they would usually use during a visit to a real furniture shop. Furthermore, the information provided about each piece of furniture proved to be pertinent. indeed, information such as "height", "length", "price", etc. were particularly appreciated as these are some of the main information on which a buying decision is based.

What appears from this short case study is also the fact that users were glad to learn about the different ways in which they could interact with some of the objects contained in the shop. The information icon present next to each dynamic object made it possible for them to easily see that they could interact with those objects. Furthermore, by clicking on these icons they could see all the ways they could interact with these objects. Some of the users even admitted that they would not have noticed that interaction was possible with some objects if the information about it was not provided.

These tests proved that this approach was indeed intuitive enough to be used by the majority of people. It requires no special skills or knowledge of the virtual world, the objects populating it or the terminology used to describe it. We note, however, that a simple explanation for first-time users about the purpose of the tool and how it works is advisable in order to reduce the time it takes them to get familiar with it.

Chapter 8

Further research

A number of continuations for this research can be considered. In the following sections we discuss the cases of making the search engine more powerful and more audience-driven as well as adapting it to support dynamic worlds.

8.1 More powerful querying

In this work, we allowed the user to make queries using adjectives such as *tall*, *cheap*, *small*, etc. To make this possible, we have added these adjectives to the world description through annotations. However, more advanced reasoning algorithms such as Fuzzy logic and neural networks could be used in the query solver in order to deduce these adjectives and others, also from objects that have not been (completely) annotated. Furthermore, these techniques would also allow to perform more vague queries based on the world's domain.

On the other hand, the annotation process could also be improved in order to maximise the use of the MPEG-7 descriptions possibilities. For instance, we could include a complete description of the scene as well as the relation of all objects in the world relative to each other. In the same way, it could be possible to connect the virtual world to different (existing) ontologies in order to maximise the knowledge about a specific domain.
The combination of a powerful query solver and a detailed world description could prove to be the best approach for querying virtual worlds.

8.2 Audience-drive search engines

We have created this search engine without consideration for user profiles or preferences. For a specific query, the first result returned is simply the first result found. We could consider to include a user's profile or preferences when returning search results. In this way, the results that would best fit the user's profile would be returned first. For example, let's consider the case of a user passionate by sport looking for a cloth shop in a virtual city. Knowing the user's profile would inspire the search engine to return first shops that sell sportswear. Furthermore, if the user is a male character, shops selling womenswear could be disregarded unless explicitly specified by the user.

Another possibility would be to allow users to annotate or re-annotate a specific virtual world according to what they find most interesting. This would further tune the search engine to be even more efficient and audience-driven.

8.3 Support for dynamic worlds

In the current thesis we have only considered querying static worlds. In order to be able to correctly query dynamic worlds (i.e. worlds where objects can be moved or changed) it is crucial to be able to adapt the object descriptions as the world is altered.
If we suppose for instance, that prices can be changed or new items are added or retrieved from our virtual shop example, it must be possible for the world creator to be able to (automatically) adapt the world description accordingly.

Further, path following could be used as means of transportation instead of jumping. In this case, it must be possible to dynamically create new navigation networks as objects are moved around inside the virtual world.

Chapter 9

Conclusions

9.1 Annotation

No matter how advanced a query solver for virtual worlds can be, its efficiency still heavily depends on the content of the meta-data associated with the world. Bad annotation seriously limits the capabilities of such search engines. However, up until now, annotation of virtual worlds has been, in general, limited to basic object attributes such as object names and properties (e.g. size, height, colour, etc.). A number of works have tried to associate a context to this meta-data. Nonetheless, the general description was still poor.
The approach described in this thesis allows domain experts to create their own virtual worlds and annotate them with an unrestricted and unlimited number of meta-data using their own terminology. This allows the annotation to be extremely relevant and context oriented, which in turns, greatly enhances the efficiency of VR search engines.

Even though only a portion of the possibilities offered by the MPEG-7 standard have been used in this work, this model has proved to be particularly suitable for our needs. We are able to create descriptions for objects identifiable with the help of a unique identifier corresponding to their VRML and DAML+OIL ontology node name. This allows a direct one-to-one correspondence between the different file formats.
Annotations are clearly marked with a keyword allowing an easy keyword look-up by the query solver. Furthermore, we have also been able to include spatial associations between objects in order to enhance the power of the search engine. In the same way, the automatic annotation possibilities greatly reduce the load on the virtual world creators with respect to the annotation process by capturing a maximum of properties of the objects populating the virtual world.
Moreover, we have made it possible to (automatically) add object adjectives (e.g. tall, small, old, new, relatively high, etc.) to the description. Thanks to these adjectives, even a simple keyword matching query solver is able to solve queries expressive vagueness, up to a certain level.
Nonetheless, and in order to make the query solver even more powerful, future research in this field should try to incorporated a complete description of the scene together with more advanced annotation of relationships between objects

in the world's description.

Being an open standard, MPEG-7 offers the possibility to standardise annotation of virtual worlds regardless of the language in which a virtual world has been written or the platform on which it runs. It is a standard fully supported by the industry. Furthermore, it can easily integrate in existing applications since it uses the widely recognised XML file format.
Standardising the annotation of virtual worlds makes it possible to share data and knowledge between applications. In this way, MPEG-7 can even be used on the Internet to search for appropriate virtual environments for a specific application or user request.
Finally, the possibility of MPEG-7 to define a language parameter makes it possible to include different languages in the same description file and choose the desired one at run-time.

9.2 Querying

As previously indicated, the search engine created for this work is intuitive enough to be used by the majority of people as it requires no special skills or specific knowledge of the virtual world. While simple to implement, this search engine has proved to be quite powerful to solve queries having also subjective or vague attributes such as "find an expensive car that is not old", for example. Thanks to this approach it is possible to create more domain oriented search engines as the annotation used reflects the context of the virtual world.

Although, the user interface might need to be slightly adapted in function of the virtual world being quarried, our query solver is generic enough to be used for any virtual world. Different techniques from artificial intelligence, such as Fuzzy logic and neural networks, could be used in order to make search engines for virtual worlds even more powerful and also allow to query for objects that have been barely annotated.

9.3 Navigation

The purpose of querying a virtual world is to easily find object and places populating it that would otherwise be hard to find without proper knowledge of the layout of the world. In other words, querying facilitates navigation and exploration of the virtual world. We were able to provide visitors to virtual worlds with a powerful search engine in order to allow them to easily find what they are looking for and be transported to those objects with a simple click of a button. Due to time limitations, however, we were not able to implement navigation networks as means of transportation and had to settle for jumping instead.

Although initially not a specific goal in itself, annotating the virtual world allowed us to effortlessly provide information about the objects populating it. Information on how to interact with a specific object is particularly interesting as it allows the virtual world creators to guide users through the virtual world without bothering them with introductory pages to explain the role of each object type in the world or implementing more complicated help mechanisms such as the use of virtual guides, etc.

Appendix A

Dynamic creation of viewpoints in 3D worlds

In order to create a viewpoint dynamically, we need to position the camera close to the object being viewed. Next we need to calculate the orientation of the camera. That is, in which direction and at what angle should the camera be positioned. The algorithm used is based on Stephen Chenney's "Orient"and translated from C to JAVA. It takes six arguments as input: the three coordinates of the position of the camera and those of the object to look at. The output of this algorithm is String representing the orientation of the camera.

We first start by defining a vector and its operations like addiction, multiplication, etc. Vectors are used to define axis.

```
/*
**  Adapted from: orient.c, Code to convert a from/at/up camera
**  model to the orientation model used by VRML.
**  Copyright (C) 1995  Stephen Chenney (schenney@cs.berkeley.edu)
*/

class Converter{

    /* Define a vector.  */
 class _Vector {
    public _Vector(double xx, double yy, double zz){
        x = xx;
        y = yy;
        z = zz;
    }
       public _Vector(){
        x = 0;
        y = 0;
        z = 0;
    }
    public _Vector add(_Vector v2){
        double rx = x + v2.x;
        double ry = y + v2.y;
        double rz = z + v2.z;
        _Vector r = new _Vector(rx, ry, rz);
        return r;
    }
     public _Vector sub(_Vector v2){
        double rx = x - v2.x;
        double ry = y - v2.y;
        double rz = z - v2.z;
        _Vector r = new _Vector(rx, ry, rz);
        return r;
     }
     public _Vector cross(_Vector v2){
        double rx = y * v2.z - z * v2.y;
        double ry = z * v2.x - x * v2.z;
        double rz = x * v2.y - y * v2.x;
        _Vector r = new _Vector(rx, ry, rz);
        return r;
     }
     /* Takes the modulus of v */
     public  double mod() {
        return (Math.sqrt(x * x + y * y + z * z));
     }

     public _Vector scalarMul(double d){
        double rx = x * d;
        double ry = y * d;
        double rz = z * d;
        _Vector r = new _Vector(rx, ry, rz);
        return r;
     }
     public _Vector unit(){
        double t = 1 / mod();
        return scalarMul(t);
     }
     /* Returns the dot product of v1 & v2 */
    public double dot(_Vector v2) {
        return (x * v2.x + y * v2.y + z * v2.z);
    }

    public double x;
    public double y;
    public double z;
}//end of _Vector class
```

We then define a Quaternion class. Quaternions are a mathematical concept invented by William Rowan Hamilton. By analogy with the complex numbers being representable as a sum of real and imaginary parts, a quaternion can be expressed as a linear combination of one real part and three imaginary parts:

$$H = a.1 + bi + cj + dk.$$

Quaternions can also be expressed as a scalar and a vector by defining a as: $a = a_1 + a_2 i + a_3 j + a_4 k = (a_1, a)$, where a $\equiv [a_2 a_3 a_4]$. A quaternion can therefore be noted: $q = (s,\mathbf{v})$ where s denotes the scalar part, and $\mathbf{v}$ the vector part. In this notation, quaternion multiplication has the particularly simple form[1]:

$$q_1 q_2 = (s_1 \mathbf{v_1}) \cdot (s_2 \mathbf{v_2}) = (s_1 s_2 - \mathbf{v_1} \cdot \mathbf{v_2}, s_1 \mathbf{v_2} + s_2 \mathbf{v_1} + \mathbf{v_1} \times \mathbf{v_2})$$

```
class Quaternion {

    public Quaternion(_Vector axis, double cos_angle){
        double  sin_half_angle;
        double  cos_half_angle;
        double  angle;
        /* The quaternion requires half angles. */
        if ( cos_angle > 1.0 ) cos_angle = 1.0;
        if ( cos_angle < -1.0 ) cos_angle = -1.0;
        angle = Math.acos(cos_angle);
        sin_half_angle = Math.sin(angle / 2);
        cos_half_angle = Math.cos(angle / 2);

        vect_part = axis.scalarMul(sin_half_angle);
        real_part = cos_half_angle;
    }

    public Quaternion(){
        vect_part = new _Vector();
        real_part = 0;
    }
    //Convert the quaternion to an angle
    public  double Quaternion_To_Angle(){
            double half_angle = Math.acos(real_part);
            double sin_half_angle = Math.sin(half_angle);
            return half_angle * 2;
    }
    //Convert the quaternion to an axis (vector).
    public  _Vector Quaternion_To_Axis(){

            double half_angle = Math.acos(real_part);
            double sin_half_angle = Math.sin(half_angle);
```

[1]For more on quaternions, please visit: http://mathworld.wolfram.com/Quaternion.html

```
            if ((sin_half_angle < 1e-8) && (sin_half_angle > -1e-8))
                return (new _Vector(1, 0, 0));
            else{
                sin_half_angle = 1 / sin_half_angle;
                _Vector axis = vect_part.scalarMul(sin_half_angle);
                return axis;
            }
    }

    //Multiplication of two quaternions
    public Quaternion QQMul(Quaternion q2){
        Quaternion      res = new Quaternion();
        _Vector         temp_v = new _Vector();

        res.real_part = real_part * q2.real_part -
                                vect_part.dot(q2.vect_part);
        res.vect_part = vect_part.cross(q2.vect_part);
        temp_v = vect_part.scalarMul(q2.real_part);
        res.vect_part = (res.vect_part).add(temp_v);

        temp_v = (q2.vect_part).scalarMul(real_part);
        res.vect_part = (res.vect_part).add(temp_v);
        return res;
    }
        public _Vector  vect_part;
        public double   real_part;
} //End of Quaternion class
```

Finally, this is the main class. Converter takes six arguments: the three coordinates (X, Y and Z) of the position of the camera and those of the object to look at. We use quaternions to define the axis of rotation and the angle of orientation of the camera. The method getOrientation() returns the positioning of the camera in the form of a String.

```
    private _Vector         pos, at, up, rot_axis;
    private double          rot_angle;

    public Converter(double posX, double posY, double posZ,
                     double atX, double atY, double atZ){
            at = new _Vector(atX, atY, atZ);
            pos = new _Vector(posX, posY, posZ);
            up = new _Vector(0,1,0);
    }

public String getOrientation(){

    _Vector         n = new _Vector(), v = new _Vector();
    _Vector         new_y = new _Vector(), temp_v = new _Vector();

    Quaternion      rot_quat = new Quaternion();
    Quaternion      norm_quat= new Quaternion();
    Quaternion      inv_norm_quat= new Quaternion();
    Quaternion      y_quat= new Quaternion(),
    Quaternion      new_y_quat= new Quaternion();
    Quaternion      rot_y_quat= new Quaternion();

    double          temp_d=0;

    /* n = (norm)(pos - at) */
    n = at.sub(pos);
    n = n.unit();

    /* v = (norm)(view_up - (view_up.n)n) */
    up = up.unit();
    temp_d = up.dot(n);
    temp_v = n.scalarMul(temp_d);

    v = up.sub(temp_v);
    v = v.unit();

    _Vector norm_axis = new _Vector(n.y, -n.x, 0);

    if(norm_axis.dot(norm_axis) < 1e-8){
            /* Already aligned, or maybe inverted. */
            if ( n.z > 0.0 ){
                    norm_quat.real_part = 0.0;
                    norm_quat.vect_part =
                            new _Vector(0, 1, 0);
            }
            else{
                    norm_quat.real_part = 1.0;
                    norm_quat.vect_part =
                            new _Vector(0, 0, 0);
            }
    }
    else{
            norm_axis = norm_axis.unit();
            norm_quat = new Quaternion(norm_axis, -n.z);
    }

    /* norm_quat now holds the rotation needed
    ** to line up the view directions.
    ** We need to find the rotation to
    ** align the up vectors also.
    */

    /* We need to rotate the world y vector to see
    ** where it ends up.
    ** Find the inverse rotation. */

    inv_norm_quat.real_part = norm_quat.real_part;
    inv_norm_quat.vect_part =
            (norm_quat.vect_part).scalarMul(-1);

    /* Rotate the y. */
    y_quat.real_part = 0.0;
    y_quat.vect_part = new _Vector(0,1,0);

    new_y_quat = norm_quat.QQMul(y_quat);
    new_y_quat = new_y_quat.QQMul(inv_norm_quat);
    new_y = new_y_quat.vect_part;

    /* Now we need to find out how much to
    ** rotate about n to line up y. */

    temp_v = new_y.cross(v);
    if (temp_v.dot(temp_v) < 1.e-8 ){
            /* The old and new may be pointing in
            ** the same or opposite directions. We need
            ** to generate a vector perpendicular
            ** to the old or new y.
            */
            temp_v = new _Vector(0, -v.z, v.y);
            if ( temp_v.dot(temp_v) < 1.e-8 )
            temp_v  = new _Vector(v.z, 0, -v.x);
    }
    temp_v = temp_v.unit();
```

```
        rot_y_quat = new Quaternion(temp_v, new_y.dot(v));

        /* rot_y_quat holds the rotation about the
        ** initial camera direction needed
        ** to align the up vectors in the final position.
        */

        /* Put the 2 rotations together. */
        rot_quat = rot_y_quat.QQMul(norm_quat);

        /* Extract the axis and angle from the quaternion. */

        rot_axis = rot_quat.Quaternion_To_Axis();
        rot_angle = rot_quat.Quaternion_To_Angle();

        /* Return the result in a String format. */
        String ret = ("orientation  " + rot_axis.x + " " +
                        rot_axis.y + " " + rot_axis.z + " " +rot_angle);
        return ret;
    }
}
```

Bibliography

[1] Jesus Ibanez, *An intelligent guide for virtual environments able to answer fuzzy queries and tell stories from her own perspective*, Phd thesis, University of Murcia, Murcia (Spain), 2004

[2] Wesley Bille, Olga De Troyer, Frederic Kleinermann, Bram Pellens and Raul Romero, *Using ontologies to build virtual words for the web*, The International Conference on WWW/Internet 2004, October 6-9, 2004, Madrid (Spain), pages: 20-28

[3] Philippe Salembier and John R. Smith, *MPEG-7 Multimedia Description Schemes*, IEEE transactions on circuits and systems for video technology, Vol. 11, No. 6, June 2001, pages: 748-759

[4] Jens-Rainer Ohm, *Multimedia Communication Technology*, Springer, 2004, ISBN: 3-540-01249-4

[5] Marc Cavazza, *High-level interpretation in dynamic virtual environments*, Working Notes: Intelligent Virtual Environments, Workshop at the 13th Biennial European Conference on Artificial Intelligence (ECAI-98), Brighton (UK), 1998, pages: 125-144

[6] Pavel Halabala, *Semantic metadata creation*, The 7th Central European Seminar on Computer Graphics, Vienna (Austria), 2003

[7] Susanne Jul and George W. Furnas, *Navigating in electronic worlds: a CHI 97 workshop*, ACM SIGCHI Bulletin, Volume 29, Issue 4 (October 1997), ACM Press, pages: 44 - 49

[8] Alex Ballegooij and Anton Eliens, *Navigation by query in virtual worlds*, Virtual Reality Modeling Language Symposium, Proceedings of the sixth international conference on 3D Web technology, Paderbon (Germany), 2001, ACM Press, pages: 77 - 83

[9] Luca C. Hittaro, Roberto Ranon and Lucio Leronutti, *Guiding visitors of Web3D worlds through automatically generated tours*, 3D technologies for the World Wide Web, Proceeding of the eighth international conference on 3D Web technology, Saint Malo (France), 2003, ACM Press, Pages: 27 - 38

[10] Grigore C. Burdea, Philippe Coiffet, *Virtual Reality Technology*, Second Edition, Wiley Publishing, 2003

[11] Daniel K. Schneider and Sylvere Martin-Michiellot, *VRML Primer and Tutorial*, Faculte de Psychologie et des sciences de l'education, University of Geneva, 1998

[12] Jed Hartman, Josie Wernecke, *The VRML 2.0 Handbook*, Addisson-Wesley Publishing, 1998, ISBN: 0-201-47944-3

[13] Andrea L. Ames, David R. Nadeau and John L. Moreland, *The VRML 2.0 Sourcebook*, Wiley Publishing, 1997, ISBN: 0-471-16507-7

[14] —, *The EAI FAQ*, available at: http://www.frontiernet.net/ imaging/eaifaq.html, accessed on February 13, 2005

[15] —, *The Web3D Consortium website*, at: http://www.web3d.org/, accessed on January 20, 2005

[16] —, *The MPEG-4 industry forum*, available at: http://www.m4if.org/mpeg4/, accessed on April 21, 2005

[17] —, *The MPEG-4 homepage*, available at: http://www.chiariglione.org/mpeg/index.htm, accessed on April 21, 2005

[18] Gabriel Taubin, *3D mesh coding in MPEG-4*, 2000, available at: http://mesh.caltech.edu/taubin/personal/taubin-research.pdf

[19] —, *The JAVA 3D FAQ*, available at: http://java.sun.com/products/java-media/3D/forDevelopers/java3dfaq.html, accessed on April 16, 2005

[20] —, *Johanneum Reasearch website*, at: http://iis.joanneum.at/MPEG-7/, accessed on January 12, 2005

[21] Guiliana Ucelli, Raffaele De Amicis, Guiseppe Conti, Gino Brunetti and Andre Stork, *Shape semantics and content management for industrial design and virtual styling*, in proceedings of the Workshop towards Semantic Virtual Environments (SVE 2005), 16-18 March, Villars (Switzerland), 2005

[22] —, *Digit Online news*, available at: http://www.digitmag.co.uk/news, accessed on May 2, 2005

[23] —, *Google*: http://www.google.com, accessed on January 15, 2005

[24] —, *Lycos*: http://www.lycos.com, accessed on January 15, 2005

[25] —, *Alexander The Great movie site*: http://alexanderthemovie.warnerbros.com, accessed on April 26, 2005

[26] —, *The Unreal Editor*, available at: http://www.planetunreal.com/, accessed on May 16, 2005

[27] —, *The DAML+OIL ontology*, available at : http://www.daml.org/, accessed on December 1, 2004

[28] —, *VR Modelling*, available at http://www.cs.bath.ac.uk/˜pjw/NOTES/110-VR/ch2-vr-modelling.pdf, accessed on May 13, 2005

[29] Martin Sperka, *3D web virtual reality and Internet*, Europrix summer school and scouting workshop, Salzburg, July 4-13, 2003

[30] —, *The CNN website*: http://www.cnn.com, accessed on May 20, 2005

[31] Kevin Laurent, *Kevin Laurent's Viewpoint calculation*, available at: http://vrmlworks.crispen.org/orient.html, accessed on April 12, 2005

[32] —, *Virtual Reality Modeling Language*, available at: http://docs.rinet.ru:8083/CGI3.2/ch28.htm, accessed on January 25, 2005

[33] —, *The MPEG-7 DLL Home Page*: http://archive.dstc.edu.au/mpeg7-ddl/, accessed on January 12, 2005

[34] —, *The XML Home Page*: http://XML.com, accessed on January 12, 2005

[35] —, *IBM Research - VideoAnnEx Annotation Tool*, available at: http://www.research.ibm.com/VideoAnnEx/, accessed on January 12, 2005

[36] Thomas R. Gruber, *Toward principles for the design of ontologies used for knowledge sharing*, 1993, available at: http://sherry.ifi.unizh.ch/gruber93toward.html