VRML Generation tools for visualization of database content in three dimensions

by

Cyril Morcrette

Diplôme d'ingénieur of

Ecole Pour l'Informatique et les Techniques Avancées (EPITA) [1995]

Submitted to the Department of Civil and Environmental

Engineering in partial fulfillment of the requirement

For the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1999

Author.....

Civil and Environmental Engineering

January 15, 1999

Certified by.....

Steven R. Lerman

Professor of Civil and Environmental Engineering

Thesis Supervisor

Accepted by.....

Andrew J. Whittle

Chairman, Departmental Committee on Graduate Studies

VRML Generation tools for visualization of database content in three dimensions

by

Cyril Morcrette

Submitted to the Department of Civil and Environmental
Engineering on January 15, 1999 in partial fulfillment of the requirement
For the degree of Master of Science

Abstract:

New technologies for the three-dimensional scene description, such as VRML97, open new possibilities to visualize large amounts of information in a new way. These technologies used through the Internet should make the access to the information independent of the platform and of the location.
To reach these goals, we explore a fusion of the VRML, Java, and Database technologies to design applications for the visualization of information stored in databases in three-dimension. The research is focused on design tools that allow a multimodal display of any kind of information in a three-dimension scene described in VRML97. The architecture associated with this design should allow interaction and dynamic refresh of these worlds. The technology is essentially client-server based with ambition to make the tools as modular as possible to have the ability to apply them visualization on different kind of databases, and to use different representations on the same database. The design should also abstract as far as possible the data from the different components of the project and store the maximum in the database to make easier to create of new applications.

Thesis supervisor: Steven R. Lerman
Title: Professor of Civil and Environmental Engineering

Acknowledgement

I would first like to thanks my advisor Steve Lerman for his support and the confidence he put in me and in my work during my time in the Center of Education Computing Initiatives, and for giving me the chance to find my way in MIT. I am especially grateful for his advice, guidance and flexibility during the writing of this thesis.

I would like to thanks Philip Bailey for the advice he gave to me and for his continuous support and help. I would like to extend this thanks to Michael Privat, for sharing his infinite Java knowledge on this project, being a work night mate and for his friendship.

I would also thanks all staff and RAs in CECI for making the lab a very nice place to stay and study.

And all my friends, thanks for making life possible at MIT and making this period of my life enjoyable. Among them Gwenaelle, Magali, Ioana, Gael, Olivier, Huy, Fred, Richard, Thomas and all the European Club.

This research was supported by FRANCE TELECOM.

To my wonderful family which grows every day.
To Dad, Mom, Dolphin, and Corinne.

Table of Contents

List of Figures

1 – Introduction

1.1 – Overview

This thesis explores a fusion of Java, database and VRML technologies to design applications for the three-dimensional visualization of information stored in generic databases.

This chapter describes the context and the different goals of the research. To fully understand the next chapters of the thesis, we introduce the different technologies that have been used in the research project. We give an explanation of the major features and advantages of each of them, but also try to provide an overview of the important vocabulary.

1.2 - Objectives of the research

The technology on the World Wide Wed (WWW) is evolving every day. VRML, Virtual Reality Modeling Language, is a scene description language which describes 3D environments over the Net. The first version of VRML was not powerful enough to allow a good immersion in Virtual Reality worlds. VRML2, now called VRML97, brings a large set of new features that give more realistic aspects to 3D worlds and allows better interactivity. VRML97 allows the VRML browser to communicate with external applications, especially Java programs, giving the possibility of dynamic changes in a scene. At the same time, the 3D technology is evolving at high speed. 3D accelerator boards are becoming a common hardware piece in PCs, and more applications are using

3D acceleration technology. It reasonable to think that virtual 3D worlds will be a fast growing part of the WWW content in near future.

This thesis explores what may be the good choices in the future for 3D interactive worlds over the network and how to use 3D and multimode[1] interfaces to improve what two-dimensional representation cannot display efficiently. Through a fusion of the Java, VRML and Database technologies, the major goal is to be able to design an architecture to allow the creation of dynamic worlds from any kind of database on the Internet where a user can easily and intuitively find the information he needs. The architecture is essentially client/server based with ambition to abstract as far as possible the data from the different components and store the maximum in a formal database.

1.3– VRML

VRML (Virtual Reality Modeling Language) is a text based file format for describing interactive 3D objects and worlds. VRML is designed to be used either locally or over the Internet. VRML is also intended to be a universal interchange format for integrated 3D graphics and multimedia. VRML is capable of representing static and animated dynamic 3D and multimedia objects with hyperlinks to other media such as text, sounds, movies, and images. VRML is in its second version, called VRML 2.0 or VRML97, and became an ISO standard in 1997. The major improvement of VRML 2.0 upon VRML 1.0, which was only designed for static worlds, are the interactions (sensors) and scalability (script nodes)

[1] In this context, multimode stands for the possibility to display information in various ways, HTML or VRML for instance, or with different kinds of VRML objects.

1.3.1 – Tree and Nodes

A VRML file is a collection of objects, called nodes, with specific characteristics and arranged in a particular order. The node is the fundamental building block of a VRML file, each of them defining a part of the scene. Some nodes can be physical objects, such as a box, a cone, a spotlight, or 3DText; others are containers or provide special features. Containers can group nodes together to apply an effect on them or for later reuse. Their order is important as it creates a hierarchical tree, as shown in figure 1.

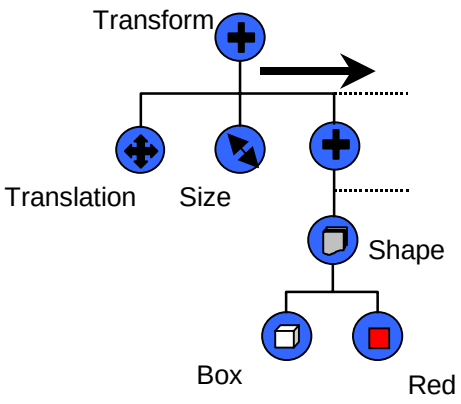

Figure 1 - An example of VRML tree node

The special feature nodes can do a translation, change the size, instantiate a program or provide hyperlinking. Each node contains fields which hold the data for the node.

The VRML browser reads the tree from the root node and from left to right. Each transformation node (translation, rotation, etc...) changes the states for all the following nodes on the same and lower levels of the tree.

A node, or a group of nodes, can be named and be reused later as instances. The new instances are pure copies of the original, but can be attached to others nodes to change

their position, size or color. VRML is also fully internet-oriented, so external information, such as textures[2] or external files[3] fields are designated by URLs.

VRML also provides fully interactive and dynamic 3D scenes. A class of nodes called sensors that react to user events in some prescribed way provides the interaction. For instance a touch sensor can be attached to an object and generate an event when the user clicks on the specific object. Often, a sensor is wired to a *Script node* (a node which contains program code) that runs a little program, usually in Java or JavaScript, when the sensor sends it a message.

In addition to their fields, most nodes contain events. An event is an indication that something changed (e.g. a field value changed, user is close to a proximity sensor, time has elapsed). There are two kinds of events: incoming event (*eventIn*) and outgoing event (*enventOut*). The wiring that connects the eventOut of one node to the eventIn of another node is called a *route*. When a field sends an event, the event's value is set to equal the value of the field that is sending the event.

[2] A texture is an image mapped on an object, such as a wallpaper, to improve the realism of a simple object.
[3] Some objects can act like hyperlinks, and jump to a HTML or VRML page when the user clicks on them.

1.3.2 – Example

Here a simple VRML example, to illustrate how a VRML scene is written.

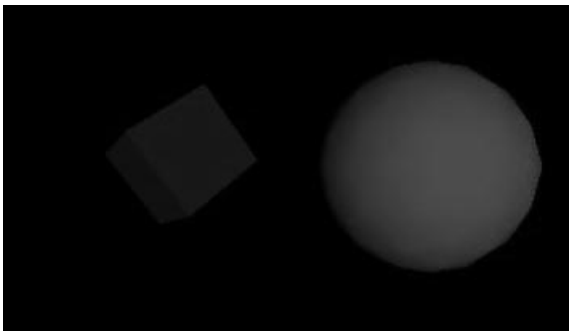

Figure 2 - Simple VRML scene

```
#VRML V2.0 utf8
Transform {               # Container node
  children [
    Transform {           # First child – a red Sphere
      translation 3 0 1   # translation from the origin
      children [
        Shape {
          geometry Sphere { radius 2.3 }
          appearance Appearance {
            material Material { diffuseColor 1 0 0 }   # Red
         }
        }
      ]
    }

    Transform {               # Second child - a blue box
      translation -2.4 .2 1   # translation the origin
      rotation     0 1 1 .9
      children [
        Shape {
          geometry Box {}
          appearance Appearance {
            material Material { diffuseColor 0 0 1 }  # Blue
         }
        }
      ]
    }
  ] # end of children for world
}
```

This code creates a simple scene with a blue rotated cube and a red sphere. Each Transform node is a container for the branches of the VRML hierarchical tree. The first

half of the code defines the red sphere, which is translated 3 units in the x direction and 1 in the z direction. The Shape node is also a container that contains all the object shapes. In the sphere case, we define a Sphere node with a radius field of 2.3 units and a color (material) red.

The same is done for the blue cube except that a rotation node is added.

1.4 – Java

Java was introduced by Sun Microsystems in 1991 when their programmers tried to develop a distributed system for the consumer electronic market. They found quickly that the object-oriented language C++ has inherent historical flaws, inherited from C and unneeded complexity from its evolution. They decide to design a simple, refined language that was not hindered by backward compatibility and would still be familiar to most programmers.

The Java language is directly derived from C++ and was designed with many goals. Its designer wanted a language which was familiar, simple, object-oriented, platform independent, portable, interpreted, high-performance, threaded, robust, secure and strategic.

Java tries to keep everything simple, and removes many of confusing techniques of C and C++ such as pointers, multiple inheritance, operator overloading and explicit management of memory. It's also a true, object oriented language, unlike C++, as everything is an object and a descendant of a root object. Java also contains a complete environment with class hierarchies for networking, input/output, graphical user interface and numerous utilities as hashtables, data structure management and data compression.

Java is designed to thrive in a heterogeneous, networked environment. The Java Virtual Machine, the Java interpreter, exists on almost every architecture and operating system. The portability of Java is designed to be complete; a program written on a Solaris workstation should work in the same way on a PC or a Macintosh without any modification. This portability extends to the graphical user interface and the different library components (such as the network classes).

Java has built-in support for threads, and allows easy creation of applications that require performing multiple tasks simultaneously, such as checking the network communication at the same time as executing the other parts of the program.

Java is a robust language with many security features to ensure safe code. First, there is no Java preprocessor or operator overloading, so the program will be executed as it is written. Second, Java has strict compile time and runtime checking for the proper use of the object type system. Third, Java manages the memory with garbage collection[4], which relieves the programmer of explicitly deallocating memory. And finally, as there are no address pointers in Java, array overflow and memory overwrite are impossible.

One major concept in Java is the applet in the World Wide Web. The portable nature of Java, combined with the web browsers, allows Java programs to be transported over Internet and be executed on the client browser.

[4] Garbage collection is a language technique that eliminates the need for memory management by periodically reclaiming unused memory.

1.5 - Database Technology

1.5.1 – Database

Database technology is now well developed. A database management system (DBMS) is one or several programs that provide functionality for users to develop, use, and maintain a database. Thus, a DBMS is a general software system for defining, populating (creating), and manipulating databases for different types of applications. DBMS's usually use Structured Query Language (SQL) to operate on information from the database.

Database technology has tremendous advantages that can be combined with 3D technology: data abstraction and data independence, robustness from crash recovery and the transaction model, concurrency control (possibility to handle several simultaneous transactions so they will not influence each other), and, of course, efficient access to large data sets.

1.5.2 – ODBC and JDBC

ODBC is the abbreviation for Open DataBase Connectivity, a standard database access method developed by Microsoft Corporation. The goal of ODBC is to make it possible to access any data from any application, regardless of which database management system (DBMS) is handling the data. ODBC does this by inserting a middle layer, called a database driver, between an application and the DBMS. The purpose of this layer is to translate the application's data queries into commands that the DBMS understands. For this to work, both the application and the DBMS must be ODBC-compliant - that is, the

application must be capable of issuing ODBC commands and the DBMS must be capable of responding to them.

JDBC stands for Java Database Connectivity. It's a Java API that enables Java programs to execute SQL statements and allows Java programs to interact with any ODBC-compliant database, or any DBMS supporting a JDBC driver. Since nearly all DBMS support ODBC, and because Java itself runs on most platforms, JDBC makes it possible to write a single database application that can run on different platforms and interact with different DBMSs.

JDBC is similar to ODBC, but is designed specifically for Java programs, whereas ODBC is language-independent.

2 - Related Work

2.1 – Overview

The growing complexity of data information both on the web and in databases often push the limits of two dimensional representation.

This chapter describes the current state of the art in three-dimensional representation of complex data, both in commercial applications and in research laboratories such as Xerox PARC. Several proposals had been made on the convergence of databases and 3D representation. These proposals explore possible ways to use the third dimension and the capabilities of new 3D standards to make interactive representations of elaborate structured data. Two of the most advanced proposals have been made by the VRML Consortium Database Working Group and by the Infomaniacs company with SQL3D. But as these projects usually require 3D information to be included in a database; they usually not provide a complete abstraction of a database or web content to the user. We focused our research on the database visualization tools in 3D.

2.2 – Current Research

The Xerox PARC User Interface Research Group has conducted extensive research about information visualization. Several projects in this group are related to 3D visualization of generic information. Information Visualizer and the Butterfly browser are two leading projects of Xerox on new interfaces that can be applied to the database visualization in 3D.

2.2.1 - The Information Visualizer

The Information Visualizer is an experimental system to develop a new user interface paradigm for information retrieval, oriented toward the amplification of information-based work. It is based on the Xerox analysis of several aspects of information use that have led them to simplify the information retrieval problem as an information workspace [11]. The research has led to the evolution of the computer desktop metaphor toward (1) 3D/Rooms (to manage information storage cost hierarchies), (2) the Cognitive Co-processor interaction architecture (to support highly-coupled iterative interaction with multiple agents), and (3) information visualization (to increase the level of information abstraction to the user). Xerox had some projects based on the Information Visualizer, specifically the Cone Tree and the perspective wall, which used 3D concepts to represent complex patterns of information in simplest way.

2.2.1.1 - Cone Tree

The Cone Tree [13] is one of the Information Visualization techniques which is used for visualizing hierarchical information structures. The hierarchy is presented in 3D to maximize effective use of available screen space and enable visualization of the whole structure. Interactive animation is used to shift some of the user's cognitive load to the human perceptual system.

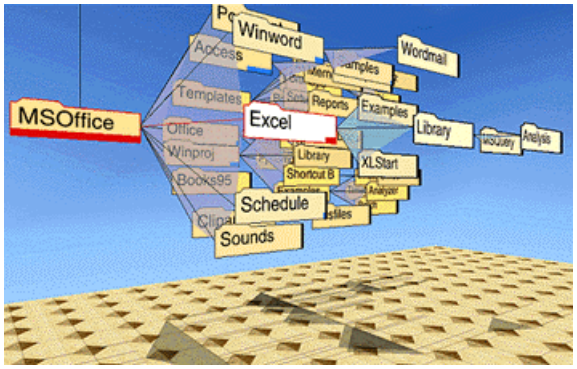

Figure 3 - The Cone Tree visualization on a file system hierarchy

An example application of the Cone Tree is an organizational structure browser. Search is done in a database of facts about each person (e.g., title or office location) and a database of autobiographies. Users can search for other people with biographies similar to a selected person's biography.

2.2.1.2 – Perspective Wall

The technique called the Perspective Wall is used for visualizing linear information by smoothly integrating detailed and contextual views [12]. It uses 3D interactive animation to fold wide 2D layouts into intuitive 3D visualizations that have a center panel for detail and two perspective panels for context. The resulting visualization supports efficient use of space and time.

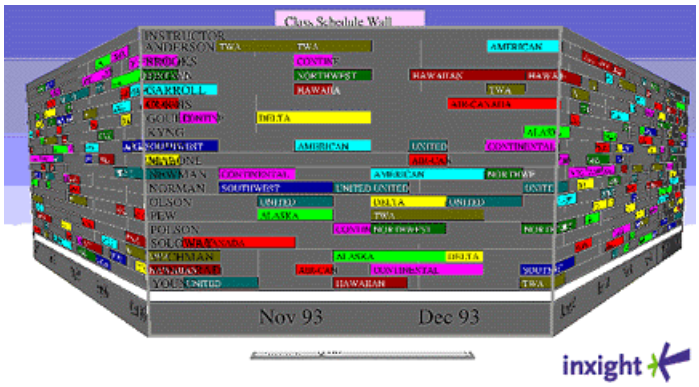

Figure 4 - The Perspective Wall display

The Perspective Wall works with any 2D layout that has been described as a list of 2D vectors and 2D positioned text. The placement of the 2D layout on the panels is determined by a single parameter that specifies what part of the layout should be in the center of the detail panel. The wall scrolls when this parameter is set to a new value, for example, when a card is selected. The example of the figure 2 shows an implementation of pilot class scheduling that depends on the teacher, the airplane company and the date. Only the cards are moving along the wall (when the navigation tools are used or if a card is selected), while the background layout stays fixed.

2.2.2 – Butterfly browser

Butterfly, developed by Inxight, is an application for accessing DIALOG's[5] Science Citation databases over the Internet [10].

[5] DIALOG is a service providing access to millions of document sources on very large set of fields. It is complex, on-line database document based, and can also contain multimedia documents.

Network information often involves slow access that conflicts with the use of highly-interactive information visualization. Butterfly addresses this problem, integrating search, browsing, and access management via four techniques:

- Visualization supports the assimilation of retrieved information and integrates search and browsing activity.
- Automatically-created ``link-generating" queries assemble bibliographic records that contain reference information.
- Asynchronous query processes explore the resulting graphs for the user.
- Process controllers allow the user to manage these processes.

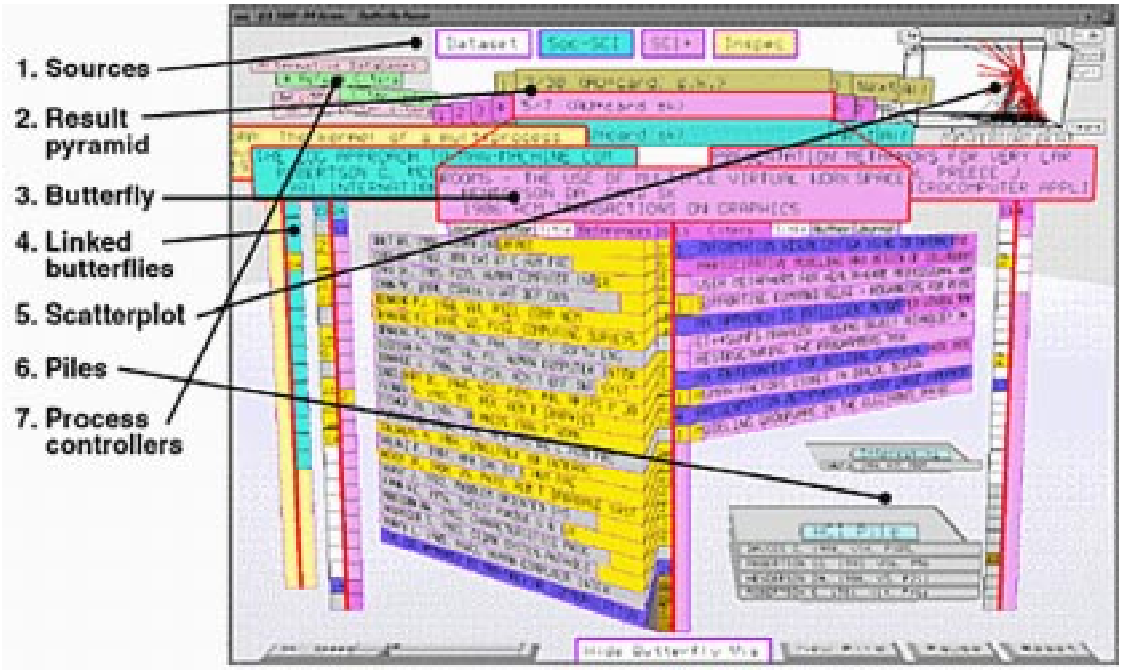

Figure 5 - The Butterfly Browser

The 3D interface allows a comprehensive and rapid browsing through the DIALOG database by displaying all information about the subject and all related contents in just one screen. But the main issue is this interface is limited to one specific kind of database.

2.3 – Database and VRML

2.3.1 – SQL Database Access by the VRML Consortium Database Working Group

The VRML consortium defined in August 1997 the recommended practices for SQL Database Access [14]. The Database Working Group's charter defines two areas of interest: the Database Extensions, to extend VRML with common database connectivity features, such as embedded SQL; and the Database API, to add enterprise services such as scalability, security, and persistence to VRML.

Actually, only embedded SQL has been deeply defined by the group, with the SQL Scripting node specification. A SQL Scripting node is a special Script node that is used to execute an arbitrary SQL command. It contains several predefined fields used to specify the SQL statement to be executed and the events generated as a result of executing the SQL. It employs a Java class with specific defined behavior based on the predefined fields.

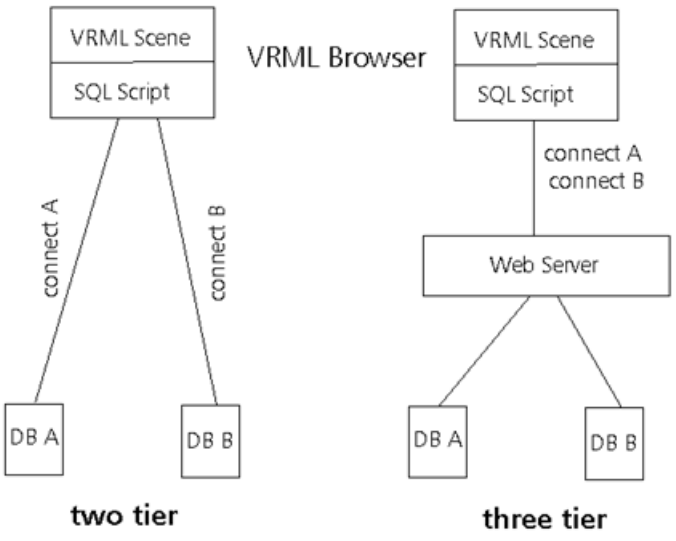

Figure 6 - The two possible implementations of SQL scripting

Two possible architectures for implementing SQL Scripting have been suggested: two-tier and three-tier.

In a two-tier implementation, a VRML Browser makes direct connections to a back-end database. This is a traditional client/server configuration. This architecture is most appropriate for an intranet, where security issues such as firewalls are not a consideration.

In a three-tier implementation, a VRML Browser makes a connection to a middle tier server, which manages connections to the database.

This architecture is most appropriate for the Internet, where the middle tier can be located on the same side of the firewall as the database system. The middle tier can therefore manage security issues, as well as improve performance through techniques such as connection pooling and reuse.

At the VRML level, a SQL Scripting node is a special Script node that is used to execute an arbitrary SQL command. It contains several predefined fields used to specify the SQL statement to be executed and the events generated as a result of executing the SQL. It employs a Java class with specific defined behavior based on the predefined fields, to execute all the database access.

Example:

```
Script {
    eventIn SFBool execute
    eventOut MFString ename
    field SFString sql "FOR ALL SELECT ENAME INTO :ename FROM EMP"
    url "execsql.class"
    }
```

When this script is triggered, a query is made to the connected database. The field ename is filled, and with a simple route to an external field (a title for instance), the complete VRML object can be updated depending of the query result.

2.3.2 – SQL3D

SQL3D is Infomaniacs' technical architecture [15] that builds on and extends industry standards to enable applications using 3D data to interoperate with each other, notably with CORBA[6]. It supports multiple users sharing 3D data without conflict, seamlessly integrates 3D, multimedia, temporal, and alphanumeric data, and exploit the strengths of distributed data and object management.

SQL3D integrates real-time 3D content into traditional data management systems and application platforms and provides such content with persistence, multi-user concurrency, consistency, durability, security, and scalability.

SQL3D uses mixed techniques from CORBA, object oriented databases and 3D to deliver to the user a multi-user application, where all users can operate concurrently on the same data through 3D interfaces with complete platform or application independence.

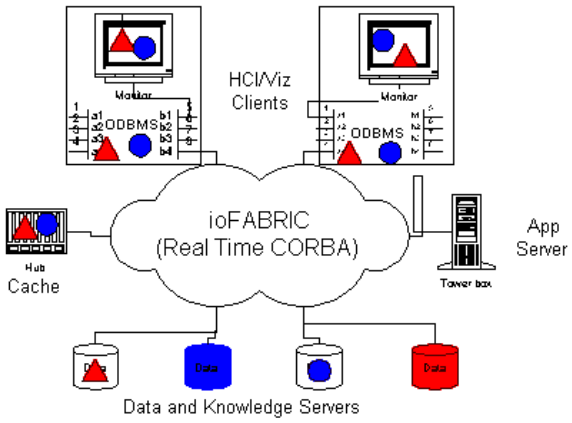

Figure 7 - SQL3D Conceptual Architecture

[6] Short for Common Object Request Broker Architecture, an architecture that enables pieces of programs, called objects, to communicate with one another regardless of what programming language they were written in or what operating system they're running on.

2.4 – 3D-XML

XML stands for eXtensible Markup Language, a new specification being developed by the W3C[7]. XML is a pared-down version of SGML[8], designed especially for Web documents. It enables designers to create their own customized tags to provide functionality not available with HTML. For example, XML supports links that point to multiple documents, as opposed to HTML links, which can reference just one destination each.

3DXML [16] is a way of describing websites and other structured information spaces in XML and publishing them in VRML. The image in the Figure 8 is what a site, a page/node and links designed with the current VRML prototype implementation look like. 3DXML has not been implemented yet, and is currently being studied by the VRML Consortium.

Figure 8 - An example of 3DXML site map

[7] World Wide Web Consortium, an international consortium of companies involved with the Internet and the Web. The organization's purpose is to develop open standards so that the Web evolves in a single direction rather than being splintered among competing factions.

[8] Standard Generalized Markup Language is a meta-language for organizing and tagging elements of a document. SGML itself does not specify any particular formatting; rather, it specifies the rules for tagging elements. These tags can then be interpreted to format elements in different ways.

For each XML flag, a new VRML node is placed in the site space. Usually, it is a VMRL prototype instantiation, which basically means that the hard part of doing the VRML has already been done at the XML level.

For instance:

A VRML node for the example shown in figure 6 might be:

```
NodeHolder {
   ID "n0"                              # Id of the node
   position 0 0 0                       # the node is in the center
   vptitle "welcome"                    # The title of the node/page
   titleText ["Welcome to Intervista"] # The node text on screen
   nodeUrl ["n0.wrl"]                   # The link to the next VRML page
}
```

So the corresponding XML code should be:

```
<node ID="n0" position="0 0 0" vptitle="welcome" titleText="Welcome to
Intervista" nodeUrl="n0.wrl" >
```

The nodeUrl field in the VRML or in the XML determines what is loaded when the user navigates (either via a link or via free navigation) into the node's space.

Because of the flexibility of XML, any kind of representation can be modeled in VMRL, including multimedia information or complex data structures. But there still needs some interface to be able to interact with databases, such as CGI or a Java server-side application.

2.5 – What is missing ?

We saw in the proceeding examples that extensive research has been done on new interfaces for complex data representation. But most of them have not been implemented yet and are still just proposals, and have not been tested, or confronted the limitations of the current technology. Furthermore some concepts, such as representation of generic data coming from any kind of database, have not been really explored yet. Also, it would be very useful to have tools modular enough to be able to be use several kind of representations on the same set of data or use the same representation on different data.

Our idea of the Database Visualization Tools in 3D is close to the proposal made by the VRML Consortium Database Working Group, but without the need to implement a new script node and to be able to generate any kind of representation, not only text in 3D. We also aim to have something completely platform independent and which does not need complex application installation.

Our solution is to use a web interface with a complete server application that is used as an intermediary between the user and the database, and is smart enough to be able to generate 3D representations from generic data received from a database.

In the next chapter, we explain the complete architecture which implements these goals.

Chapter 3 – 3D-World Generation Tools Architecture

3.1 - Overview

The use of data about three-dimensional models that comes from any kind of database raises several issues. These issues are at different levels such as engine architecture level, 3D-code generation techniques, and how to present the “entities” stored in the database in a three-dimensional representation. A completely generic engine (i.e., one that automatically reconfigures to different data) requires a total data abstraction through its modules, which is impossible without a minimum of information about the data itself. However, if we can decompose the different tasks of the 3D-code generation well enough, it is possible to come closer to this paradigm. Therefore, each module just needs a small amount of information to map the input data to an output result, which must lead to a complete data representation in 3D. For instance, if we want to make a presentation of a movie database, we first have to know how to represent information about movies, but also how to link information from the database to this visual representation.

In this chapter, we describe the different components designed to reach this goal. All components are specialized to be able to manage database interaction, VRML generation, user interface generation, and user events with a high degree of data abstraction. The architecture is designed to be as modular as possible to let the engine to adapt to any kind of database with minimal modifications. In the last part of this chapter, a complete example illustrates all the concepts.

3.2 - Client/Server Architecture

Our objective is to provide a set of tools that automatically generates a visual representation of a database's content according to user queries. Users should be able to ask for information from the database, which is converted to an SQL query and mapped to a 3D representation. This research is based on a customizable 3D representation, which depends on several factors such as network speed and the level of detail required. To reach this goal, several issues must be addressed. These issues include generation of VRML content "on the fly", communication between a VRML browser and the client external application (event propagation, dynamic creation of VRML nodes), multimode presentation, networking efficiency and 3D representation of the data stored in a database.

The architecture is composed of three major components.

- Client – responsible for displaying and forwarding information from the user
- Server / ADO (Application Domain Object) – responsible for data communication from the network and between the modules and event management.
- ToolSet – responsible for the 3D code generation.

The client is composed of a viewer, in our case a VRML browser, and an applet used as a bridge between the viewer and the client side application. The client has a plugable engine that manages the representation of the 3D information coming from the server. The engine receives instructions from the server to display the generated world. This

client engine is also in charge of managing and propagating events from the viewer to the network interface.

Numerous messages are exchanged between the client and the server to create the user interface. This interface is usually generated on the server and displayed through the user interface (designated by *DB3D Interface* in the following architecture figures). Therefore centralized management of messages is a central piece of the client. The same centralized management design is also used on server side, which provides a generic way of sending messages through the network and between the different modules of each component.

The server is composed of three major parts: the interfaces, the Application Domain Object (ADO) and the ToolSet.

The network interface (*Main Engine* and *ObjectInputThread* in the architectural figures) manages everything related to networking and messaging. It is divided into several parts, each of them usually implemented in a separate thread[9] to guarantee robustness and the ability to handle asynchronous, non-blocking messages.

The ADO, which is a database-dependent module, handles the database connection, user preferences and user queries. It is also the module where the messages are handled. The ADO is completely modular, and could easily be replaced by another one dedicated to another database.

The ToolSet, which is the display mode-dependent module, manages all object creation in the mode that it represents. Only the ToolSet knows what kind of visual objects are

[9] A thread is a part of a program that can execute independently of other parts. Multithreading makes it possible to design programs whose threaded parts can execute concurrently.

being used, and how to map the information that is coming from the database to the objects by applying the Rules. The Rules are information given by the user, or by the application, to describe how “concepts” from the database should be visually represented. The choice of the ToolSet allows the user not only to switch among representation mode (HTML, VRML, etc...) but also to customize each mode, such as the choice of the level of detail.

3.3 - Server: global architecture

Below is a complete overview of the project architecture, as well as the interactions between the different modules and the way they communicate.

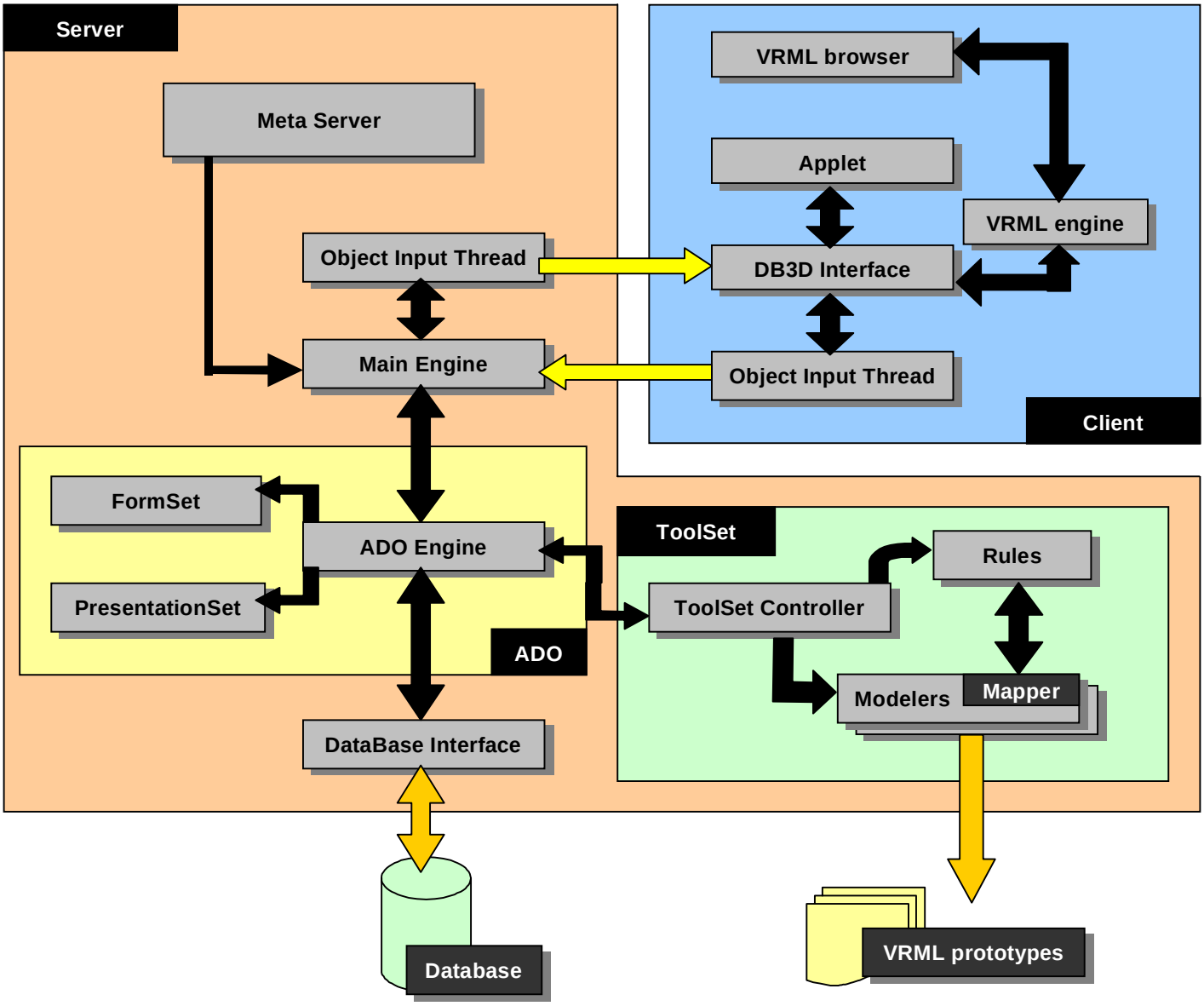

Figure 9 - Database Visualization tools in 3D Architecture Overview

Component	Part of	Function
VRML Browser	Client	Displays on screen the 3D model generated
VRML Engine	Client	Create the VRML model from the data structures created by the server
Applet	Client	Make the connection between the application, the web browser and the VRML browser
DB3D Interface	Client	Handles and dispatches all event from the VRML browser of the server. Generate the user interface from the data structure created by the server
Object Input Thread	Client Server	Handles network communication
Meta Server	Server	Receives news connections
Main Engine	Server	Foundation of each client-dedicated server, received all communication from the Object Input Thread
ADO Engine	ADO	Handles all events and all module communication. Manages the ToolSet, FormSet and PresentationSet.
FormSet	ADO	Generation of query user interface, interpretation of these forms to SQL queries.
PresentationSet	ADO	Generation of Rules user interface
Database Interface	Server	Interface to ODBC compliant databases.
ToolSet Controller	ToolSet	ToolSet interface, Modelers and Rules manager
Rules	ToolSet	Gives mapping information
Modeler	ToolSet	Generates VRML code from prototypes

3.3.1 – The Main Engine

The first block in the server architecture is the Meta Server. This module sits waiting for connections from clients. For each connection, the Meta Server creates a client-dedicated thread which contains the complete server. Having a complete server dedicated to each client has several advantages. First, it guarantees robustness, as a server crash does not affect the Meta Server. Another advantage is to be able to ignore several multi-user issues, such as access concurrency, as each server only has one client. Also, the server responsibility can be distributed across several machines, the Meta Server giving the new connection to the least loaded machine.

For each client-dedicated server, a new Main Engine is instantiated in a thread. The main role of the thread is to handle asynchronous non-blocking network communications. The Main Engine begins communication with the new client and, after receiving the choice of the Database, creates the ADO (Application Domain Object), which is a database dependent object.

The Main Engine, during the application life, checks the integrity of the connection and redirects most of the messages to the ADO for handling.

3.3.2 – ADO

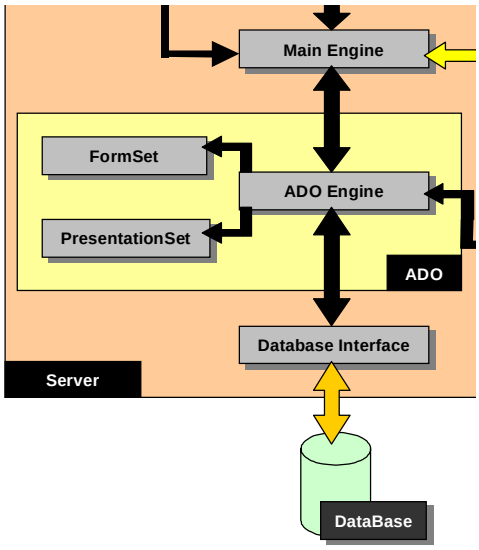

Figure 10 - The ADO architecture

The ADO is where all incoming messages are handled. The ADO has a relation with all the components of the server-side application. Therefore, it is used as a switch with the ability to choose which component is most likely to be able to solve the problem posed by the client. For instance, a message from the client containing a query will be redirected to the database, and a message containing an event triggered by the user will be sent to the ToolSet, which is the only part of the server that knows what is rendered on the server-side (VRML, HTML, etc…).

The ADO is also the class where every peripheral component is instantiated: FormSet, PresentationSet, Database Interface and ToolSet. The first action of the ADO is to create a new Database Interface (DBI) and connect to the chosen Database.

To create a new ToolSet, the ADO must have received user preferences from the client interface. With these preferences, the ADO can create the adapted ToolSetController. Other components are created at the user's request.

3.3.2.1 – FormSet

The FSclasses are the implementation of the FormSet interface. Each class represents a query (by time, by category, etc...). Each class is in charge of creating the dynamic User Interface adapted to the particular query. The description of the user interface is sent to the client to be displayed with the generator class name imbedded. Therefore, when the user feedback comes back from the client, the ADO redirects it to the appropriate FSClass without knowing to which class[10] it is sending the information.

The user information from the dialog is processed directly by the FSclass itself, because only the class knows how to interpret results to generate the adapted SQL query. So, to add a new query, the only requirement is to add the appropriate FSclass.

3.3.2.2 - PresentationSet

The PresentationSet class is in charge of generating the Rules dialog. The ADO makes a query to the database to have every different value of a specific field. For instance, if we want to highlight results from a specific attribute, we need a rule on this attribute to describe how and which value to highlight. The result is sent to the PresentationSet class that creates the dialog in which the user can associate a color, texture or any kind of property to each value. The ADO created all the rules at the application initialization, and all the forms are sent in a single message to the client. When rules information is returned, the ADO redirects it to the Rulemapper to add the new rules.

---

[10] In Java, you can instantiate new classes dynamically with only the name of a class stored in a variable.

3.3.3 – Toolset

3.3.3.1 – VRML modeler

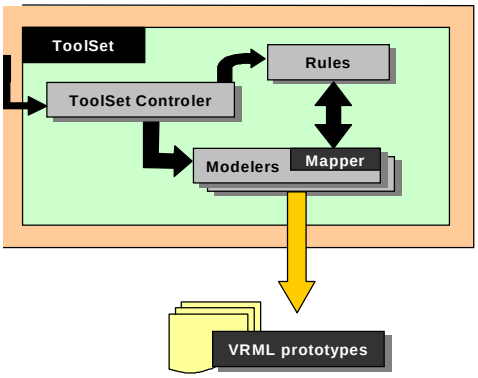

Figure 11 - ToolSet Architecture

The ToolSet is responsible for creating and managing the multi-mode user interface. This module is the only one that knows what mode is currently being used to render the database content on the client side (VRML, HTML, etc…).

The ToolSet can construct itself, using a configuration file. This file contains information about what should be displayed depending on what kind of data has to be rendered. During its construction, the ToolSet instantiates a list of Modelers responsible for the VRML code generation. Each modeler is a Java class dedicated to only one unique VRML prototype, and each VRML prototype is associated to one "concept"[11] present in the database. A VRML prototype is an unfinished object that simply needs more information (width, color, height, etc…) in order to be able to render an instance. For an application that generating a building, the same modeler could be used to generate all the buildings that have the same shape but not the same size or the same texture.

[11] We define concept here as a kind of information present in the database. For instance, in a Calendar database, each concept could be a type of social event: concert, dance, movie, lecture, etc...

To change the display mode, only the ToolSet classes need to be changed. With this kind of division, it is quite easy to generate HTML instead of VRML for instance.

The real place of VRML code generation is in the Modeler. The ToolSetController receives the result of a user query to the database from the ADO, and for each tuple it sends the data to the correct Modeler. Each Modeler is in charge of mapping these data to the parameters of the VRML prototype. For this task, the Modeler uses the Java class associated with the prototype[12], Map file[13], and User Rules. Once the mapping is done, the modeler creates an Instance, which contains a PresentNode and eventually a PresentProto.

The PresentProto is the code of the VRML prototype; it is just sent once, as it is cached by the client for later uses.

The PresentNode is a hashtable with all information for the generation of VRML nodes (i.e., the value for each parameter). Both PresentProto and PresentNode are optimized to be as small as possible and be able to compress themselves when they are sent through the network interface, and uncompress automatically at the point in time when the client needs them.

3.3.3.2 – ToolSet configuration file

The core of a ToolSet is in fact a simple text file that associates a value with a name of a prototype. The value is an identification of a "concept" stored in the database (a kind of

[12] See 4.2.1

event, type of concert, etc...) and is associated to a prototype to be represented. For instance, in an application in which we want to represent different categories of events (music, movie, etc...), we associate a specific prototype to each type of event. Each type has an ID, a value which design the type of each event. A music event can have the Id 1 for instance and be associated to the kiosk prototype. Having one file defining a ToolSet makes the creation of new ToolSets easier, and allows having different kinds of prototypes depending of the ToolSet chosen. Each ToolSet is created according to the preferences of the user. In the current engine, there are three kinds of preferences: Network speed, Level of Detail (LOD) and World Customization. Each time the user changes his or her preference, the ADO creates a new ToolSet by reading the associated file.

3.3.3.3 – VRML Prototypes

Each prototype may have a file with the same name[14] but with the extension `.map`. The main use of this file is to create the mapping information between the parameter names of the VRML prototype and the names used in the rules and in the database.

The file is just an association of names:

```
[Database/Rule Name] [Mapped Name]
```

For instance, the color parameter in the VRML prototype can be linked with a field in the database called SponsorColor. Therefore, in the map file we have:

```
SponsorColor color
```

---

[13] See 3.3.3.3

This means that the values returned by the database for the field SponsorColor will be applied directly to the color parameter of the VRML prototype.

The same applies to the Rules, except the information from the database must be handled in the RuleMapper instead of being directly mapped. Therefore, instead of a VRML prototype parameter, we use the keyword `#Rule` in the map file to indicate that the real value will be returned by the RuleMapper. This module contains all the associations between values[15] chosen by the user. The value returned by the RuleMapper is labeled with the PresentationSet name; therefore this name needs to be mapped to the associated VRML prototype argument.

Example:

```
Sponsors #Rule
PSColor color
```

This means that the values of the field Sponsors go to the RuleMapper. In this specific example, the user may have assigned a color for each value of Sponsors (ex: Coca-Cola ⇨ blue and Pepsi ⇨ red). This value is returned by RuleMapper under the name PSColor. In the Map file, PSColor is associated to color, so the color of the prototype will be blue if the Sponsors field equals Coca-Cola or Red for Pepsi.

---

[14] For instance: building.wrl for the VMRL prototype and building.map for the mapping information

[15] For each different value of a specific field, the user can associate a color, texture or URL. Therefore, users can customize the way data are represented.

3.3.4 - Database Interface

The database Interface makes the link between any relational database and the ADO. Most of the database dependent components are abstracted there, and the interface allows making complex queries both through direct SQL Query or prepared statements. The prepared statement is a way to pre-compile a general category of SQL queries by giving an incomplete query to the database. With this information, the database knows which fields will be used and what operations are needed, and therefore can dramatically speed up the query process.

The interface is written in JDBC (Java DataBase Connectivity), that guarantees compatibility with any OBDC compliant DBMS[16].

[16] Database Management System.

3.4 - Client architecture:

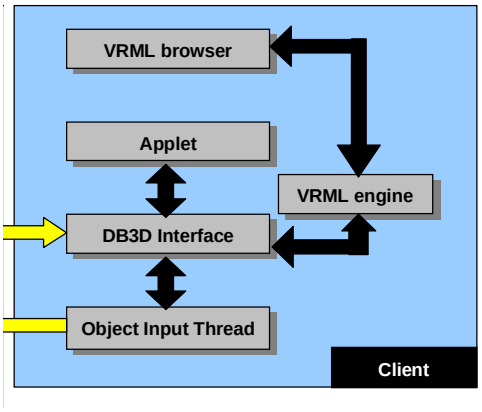

Figure 12 – The Client Architecture

The client has been designed to be lightweight regarding its size, but heavyweight regarding its features. The client must abstract the database as much as possible, as the user can ask any kind of database to be displayed in various ways. Therefore, the client architecture must be able to adapt to the server needs and possibilities. We reach this goal by moving all user interfaces generation and all VRML generation to the server. The client is just displaying forms about which it has no information, registering VRML nodes and waiting for user events.

3.4.1 – DB3D Interface / User Interface Engine

The user interface engine is in charge of displaying all the forms and menus to enable the user to communicate with the application. All forms (query forms, preference forms) are generated on the server and sent to the client. Therefore, the interface is independent of the client and can be adapted to any kind of application. The User Interface Engine is only used to render dynamic forms and forward the results.

3.4.2 – VRML engine

The VRML engine is responsible for all the three-dimensional representation. It receives the Instances nodes, composed of the PresentProto and PresentNode classes, from the server. Every new incoming VRML prototype is cached, so it will be sent only once. Another reason why we need to cache the VRML prototypes is the External Application Interface[17] (EAI) does not keep track of prototype[18], and they are needed for future generation of nodes. The VRML engine then generates and registers the new nodes from the PresentNode class (the description class for all generated VRML nodes). By registering the node, the VRML engine is able to received user events from it, triggered by the VRML sensors (touch sensor, proximity sensor, visual sensor, etc...). But to be able to identify which node is responsible for the event, we need to keep a copy of every created node on the client.

Once an event is received, the VRML engine looks in its cache and labels the event with the node name and some information about the node. The labeled event is sent to the server where it is handled.

The VRML engine is designed as an independent module. The reason for this is to make it possible to reuse the code out of the context of Java applets[19] and to be able to change mode easily (for instance to switch to a HTML engine or another kind of 3D representation language).

---

[17] The EAI is an API used for accessing dynamically VRML nodes from a Java program
[18] See 4.2.1 about the constraints of the actual VRML browsers.
[19] The actual implementation of the External Application Interface of the Cosmo player works only inside an applet to be able to connect to the VRML browser through the web browser.

3.4.2 – Event Management

The client uses an ObjectHandler class in charge of trapping all incoming messages (either from the network or from another client module) and distributing them to the correct recipients. This system allows easy management of messages for complex applications.

The entire messaging system is based on the CommandRequest class. The CommandRequest has a label that describes both the addressee module and what is carried. The data is actually a vector of Objects. This makes it possible for it to carry several similar data at the same time (in just one CommandRequest, you can carry all the VRML nodes which need to be displayed) . All CommandRequest types are described in the next section.

3.5 – Communication Protocol

The amount of information exchanged between the client and the server is big. Therefore it was necessary to create a communication protocol as well as a data structure to carry the data. The transport is supplied by the CommandRequest class that carries messages and data at the same time. This information wrapping in the transport class allows an easy, generic and centralized management of incoming messages.

The following table shows the different exchanges between the server and the client. The communication is completely asynchronous; however, messages sent by the client usually generate specific replies from the server. Therefore, the messages are grouped in the next table as message/replies. For each message raised by the client, a short explanation of what occurs in the server is provided.

Client	Server
AD_INIT ➔ At the initialization or when the user wants to change the database, the client sends this message to choose which ADO must be instantiated by the main server.	**← UI_PSCREATE** The newly created ADO, after internal initialization, queries the database for the fields that defined the Rules. For each rule, the ADO creates a form and sends the complete set to the client. The client stores them until the user wants to change a specific rule. **← UI_FSLIST** The ADO checks which queries are available and send the list to the client.
AD_PREFCHOICE ➔ This message is sent to the server to specify the preferences (Network speed, Level of Detail, User customization, ...) to the server to be able to create the appropriate Toolset. At the initialization, a default preference choice is sent.	**← VR_INITWORLD** With the preference information, the ADO creates the associated ToolSet and generates the initial world (the "background" world), if it exists, by a set of Instances, that contains all information needed to generate VRML.
AD_FSCHOICE ➔ This message is generated when the user chooses one the queries available.	**← UI_FSCREATE** As soon as the ADO has a query, it can instantiate the specific class that generates the form. This form is sent to the client

AD_PSREPLY → This message is sent when the user changes a rule. The client sends the complete list of associations between database values and representation values, which are stored in the current ToolSet (in the Rule module) by the ADO.	
AD_FSREPLY → When the user fills in the query form, the client sends the result to the server.	**← VR_CREATE** With this information, the server creates a SQL query and processes the result in the ToolSet. A set of Instances is generated and sent back to the client to be displayed. Each Instance corresponds to a new VRML node. **← VR_UPDATE** In the case that a node already exists only the changed values are sent to the client through this message.
AD_RESETWORLD → This message is sent when the user asks for a complete reset. The VRML world reverts to the initial world and the ADO resets the ToolSet caches.	
AD_EVOUT → This message is sent to the server when an event is received from the world. The CommandRequest contains all information about the event emitter.	**← UI_HTMLUPDATE** The server gives an URL to the client to display information from the database. Usually sent after a AD_EVOUT.

3.6 – Global Example

To illustrate the preceding descriptions of the different engines, we will use a simple example of a user making query about restaurants. The database contains all the information about all restaurants in the user's town (name, position, type, price, etc...).

Once the client is created and all the browsers are connected, the client connects to the server and requests access to the restaurant database. The main engine creates an ADO dedicated to this particular database. The new ADO connects to the database and sends the client the list of the possible queries: query by location and query by price.

In our example, we will have only one user Rule that associates a color to a type of restaurant. The ADO queries the database to have all unique types of restaurants: Chinese, French and Italian. The user interface that allows the user to make the association between the restaurant and a specific color is generated and sent to the client.

Meanwhile, the client sends the default user preference, which define a network speed, a level of detail and a default choice of visualization. With this information, the ADO reads the corresponding file, and creates a ToolSet with the Modelers. Once the ToolSet is created, the ADO builds the "background" world by either sending the URL of a VRML file in the case of a static world, or making a query to the database and generating the associated VRML code through the ADO in the case of a interactive "background" world. The client receives this world and stores it for future reuse. Depending on whether the world is dynamic or static, the client's VREngine generates VRML and registers the nodes, or sends the URL directly to the VRML browser.

The user decides to fill in the rules and assigns a color for each restaurant type: blue for French, red for Chinese, and yellow for Italian. Once this information received by the server, the ADO fills the Rule module of the ToolSet with the color-value association.

After that, the user chooses to make a query by location. The event triggers the instantiation of the associated FormSet class in the ADO. This class generates the user interface for this particular query and sends it to the client. The DB3DInterface handles the user interface message and displays it on the user's screen.

Once the user fills in the form and sends it back to the server, the information are sent to the FormSet, which generates the corresponding SQL query, to find the closest restaurants around the user's home. The DatabaseInterface processes the query and returns an array of hashtables corresponding to the cursor[20] of results. This cursor is sent to the ToolSet Modelers for VRML code generation.

The ToolSetController sends each tuple from the database query to the corresponding Modeler (according to the information stored in the ToolSet description file). Each modeler is associated with a VRML prototype, in our case a prototype of a building representing a restaurant.

The modeler takes the information from the tuple and tries to map it to the VRML prototype parameters according to the information stored in the associated .map file.

According to the map file we have:

```
RestoName title
Type  #Rule
PSColor color
```

[20] Array of results. Each row (tuple) is a set of results and each column is a different field.

The Modeler can directly fill the title field with the value of `RestoName` as it is a direct map. The `Type` needs to be processed by the RuleMapper as the keyword `#Rule` is present. The RuleMapper has stored the user Rule and returns the associated color corresponding to the value of `Type` under the name PSColor (for instance if `Type` is equal to French, `PSColor` is equal to blue). This result is mapped to the `color` parameter. The VRML generated code is stored inside a PresentNode and sent to the client. The corresponding VRML prototype is sent at the same time if it has not been sent already, in a PresentProto.

The client receives a set a PresentNode and PresentProto and gives them to the VRML browser after registration of each node. The client needs to register each node to be able to catch events from this node.

At this moment, all the restaurants in VRML are displayed on the user screen, with the correct color and position, answering the user's request.

If the user clicks on a restaurant, an event is sent to the ADO. The ADO makes another query to the database for this specific restaurant, requesting all information about it and generates a temporary HTML file with the results. The URL is sent to the client and displayed in the user's HTML browser.

The last feature could change depending of the application and the ADO. Instead of returning an URL, the user could jump to the restaurant's web page.

Chapter 4 – Application built with the tools

4.1 – Overview

The 3D visualization tool engines can be applied to several kinds of applications in which the third dimension displays more information than the equivalent display in two dimensions. However, the current VRML and database technologies impose several constraints, such as the External Application Interface implementation, database tables optimization and structure. To satisfy these constraints, we had to include new behavior both in the client and the server.

In this chapter the different constraints and how we modify the modules to be able to reach the design objectives are explained. After, we give some examples of possible applications for the 3D visualization tools and how the tools can be adapted to each of them. In the conclusion of the chapter the IAP[21] event browser we created using the current implementation of the tools is described.

[21] Independent Activity Period. During the month of January is held at MIT, several hundred of academic and non-academic activities are held everywhere on the MIT campus.

4.2 – Constraints

4.2.1 – EAI

The VRML97 specification includes a Java based interface designed to allow an external environment to access nodes in a VRML scene using the existing VRML event model.

In this model, an eventOut[22] of a given node can be routed to an eventIn[23] of another node. When the eventOut generates an event, the eventIn is notified and its node processes that event. Additionally, if a script in a Script node has a reference to a given node it can send events directly to any eventIn of that node and it can read the last value sent from any of its eventOuts.

The External Application Interface (EAI) adapts this interface by giving access to the top-level named nodes in the system to the external environment. The interface then mimics the access of the Script node with a reference to the named node. To facilitate the sending of events to the external environment, a new mechanism is made available. To receive notification when an event is sent to an eventOut, the external environment registers interest in it. This mechanism is similar in concept to the event process functionality in the VRML specification, but is effectively different due to the asynchronous nature of the external environment.

So through the EAI, new nodes can be created and registered to have the ability to receive eventOuts generated. However, the EAI had two major limitations: the session process and the eventOut identification.

---

[22] An event generated by a node (usually a sensor) when a user clicks on the sensor (TouchSensor) or if the user's avatar is in the range of the sensor (ProximitySensor) for instance.

[23] An eventIn is a message to a node, usually for updating a parameter such as the position or the size.

The session process problem is based on the fact that to send dynamically VRML code to the VRML browser, we need to use the EAI command `createVrmlFromString`, which take a set of VRML commands and returns an array of references to the created nodes. Unfortunately, the `createVrmlFromString` command is a stand-alone session, which means the command is not based on what has been created from another session. This is a problem as we use VRML prototypes to create the nodes, and all the node prototypes must be included for a session to be valid. This means for each node sent to the VRML browser, we need to include again its prototype code in the session.

The eventOut identification is also a problem as the EAI send only the internal ID of the node which generate an event, which often make the identification of the VRML object difficult.

These problems are resolved with two techniques. The first one is caching, implemented in the client VREngine. Each new prototype is cached, and each node received from the server “knows” its prototype. Therefore, the client knows which prototype needs to be included in a session and always has a local copy of them which are also labeled with the internal cosmo player ID and allows an easy indentification.

The second technique is optimization of the number of node sent. To avoid too small session, the server always tries to maximize the number the new created nodes sent to a client in one CommandRequest. This way, the client can minimize the number of retransmission of the VRML prototype to the browser.

4.2.2 – Mapping Information

The current implementation of the Cosmo Player[24] has some imperatives that application architects must follow to be able to use events. As the VRML node `PROTO` does NOT register properly through the `CreateVrmlFromString` EAI function, the only possible workaround is to create an `EXTERNPROTO` for each prototype and register it. The `EXTERNPROTO` is just a copy of a `PROTO` node excepts that the prototype body is an external file or URL. The `EXTERNPROTO` contains only the parameters, and the external file must contains both the body and the default values. With this technique, the browser will not read the prototype through `CreateVrmlFromString`, but from an external file. As it is the VRML browser that reads the file, and not the EAI engine, it is the only way to register properly the VRML prototypes.

Another constraint of the current release of the 3D visualization tools is the need to have a Java class associated with each prototype to be sure that the information coming from the database will be formatted or recalculated to match a strict VRML grammar.

The classical example is information about size and position are often split in 3 different fields in the database (one for x, y and z) to be able to query on each of them independently of the others. But this is an invalid argument for a `SFVect3f`[25] which requires something like (x y z) as one field. Also, rotations are often given in degrees in the database, but VRML works only with radians.

---

[24] The most advanced browser for VRML97, created by Silicon Graphics.
[25] The SFVect3f is a VRML type to which define a coordinate in 3D by 3 float values: x, y and z.

Therefore if the prototype does not have exact information from the database in a correct VRML grammar, an associated Java class must exist that converts information coming from the database to something understandable by the prototype.

In our application we create the interface `prepareVRMLproto` to reformat the data. The interface is also used to calculate the position of the VRML nodes corresponding to query results (where do we put the result object) as the VRML language is not designed to make calculation operations.

This use of the associated Java class is independent of the .map file used for the correspondence table between the database field name and the VRML prototype argument names. Both of them are optional: if the data are already in a VRML compliant format, there is no need for the Java class, and if the fields of the database have the same name as the VRML prototype, there is no need for a map file.

4.2.3 - Database

One of the bottlenecks of the application is the query to the database. As we making a lot of queries to the database and receive huge tables as results, we need to optimize the database structure. We also need to have the structure explicit enough to be able to represent all the “concepts” present in the database.

To keep the database in an optimal structure we have two conditions. First, we try to minimize the number of tables. Second, we keep all tables in a normalized form to avoid redundancy, update and insertion anomalies. The Boyce-Codd Normal Form (BCNF) is recommended for a best use of databases. The BCNF techniques guarantees the non-

redundancy and the consistency (all needed information is present with a minimal number of null field) of the database tables.

To represent all the “concepts” present in the database, for instance which kind of social event or the type of a shop, we need to have a field in the database that associates a number (ID) with each concept. For instance, in a calendar database, 1 could be associated the music events, 2 with the movie events, and 3 with theater events. This allows us to assign a particular VRML prototype to each concept, by an association ID<->Prototype, it also helps in choosing the models used to represent the database content. This model is indeed dependent on the structure of the database, and choosing which field is associated to each ID helps to know what is used for the main representation in 3D.

The last optimization is to make extended use of the prepare statement to shrink as much as possible to processing time of the query. As all queries are known, it is easy to make the chosen FormSet to fill the unknown values of the statement instead of making a simple SQL query.

4.3 – Possible applications

4.3.1 – Virtual Cities

France Telecom's research department began several studies on the user interfaces through the web. Two of them could be direct applications of our 3D visualization tools: the Yellow pages for shops and the Virtual Cities project.

France Telecom was the national telecom company in France until the beginning of 1998. They are in charge of the distribution of the white and yellow page directories, and are trying to distribute them in several forms, including through the web.

The web version offers more services than the others do, and is composed of several subdivisions. One of these subdivisions displays all the shops on a particular street, with several pieces of information about each and a picture of the shop's facade.

In this case, we can use the 3D visualization tools quite easily. The query is just a name of a street, and as the facade pictures are stored in the database, the final visualization is the street in 3D. Each facade is textured on a simple cube representing a building. Other components can be added depending on store type (a sign with the shop logo, a shop window with a commercial movie, could be sensitive and be a link to the shop's web page, etc...).

To browse through streets, at each end of the road, we can add a special 3D object that makes the query for a connected street.

The other France Telecom project is Virtual Cities. This project is a local agenda of social events in the user's town. The user chooses a particular event and receives a map showing the location of this event. Each location is a link to an entry in the yellow pages.

We can use the 3D visualization tools here to have a global point of view and have a complete 3D representation of the town in which each event is displayed by a particular object (a kiosk for music event, the movie poster for a movie, etc...).

In this case, we have a initial world which is the town. The user can choose the level of detail to change the size of the town model and zoom in to a particular place. The ToolSet file stores the complete association between the events and the VRML prototype.

With this interface the user can have a realistic view of the events in his neighborhood with more information than in 2D. One advantage is the ability to have two events at exactly the same place without overlapping; the two objects are just piled up in the Z direction. Another advantage is that more information can be displayed at the same time, such as whether there is still tickets available, the length of the event, etc... To display these data, we can use properties of the 3D objects as color, texture, size or sound[26].

4.3.2 – Commercial Data analysis visualization

Another kind of application is a visualization tools for data analysis. A 3D model can directly represent complex data relationships contained in a spreadsheet far more easily than in 2D. For instance, it might show profit and loss statements for a 12 months across all of a company's divisions. A large collection of 2D charts break these relationships into slices that the user must mentally reintegrate.

[26] In VRML, a special node, the proximity sensor, is able to make sounds when the user is close to an audio source.

As the 3D visualization tools creates interactive worlds, it lets the user navigate in 3D information space and make comparisons from many perspectives, rather than being restricted to one point of view.

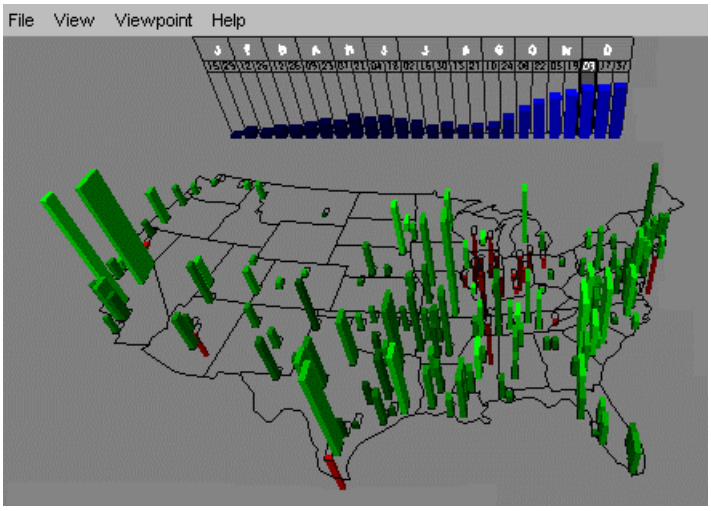

Figure 13 - Example of visualization of a federal profit/loss statement

Here, the tools are perfectly adapted to present this kind of visualization. The same representation can be used on different databases by simply changing the ToolSet file. The user can make queries in two ways, by the forms to ask results from a defined period or by clicking on interactive objects (such as the blue chart in Fig. 13). These objects generate events that trigger a new query to the database, and update the user's display.

In this case, no complex prototypes are needed (just cubes for instance) but we need still to have prototypes as each VRML object includes Script nodes in order to have the ability to be interactive (as only a Script node can send event back to the EAI).

4.4 – The IAP database example

To demonstrate the possibility of the 3D visualization tools, we created an information browser application which gives the ability to search into the MIT IAP event database. Each event has a location, a category, a time slot, a teacher, web information and a description.

The Initial world in our case is a simple model of the MIT campus. At the initialization of the world, a request is made to the table `Building`, which returns the correct ToolSet (the default one in the demonstration application), i.e. the complete set of information (prototype name, position, size, rotation, color,...) needed for the world to be constructed.

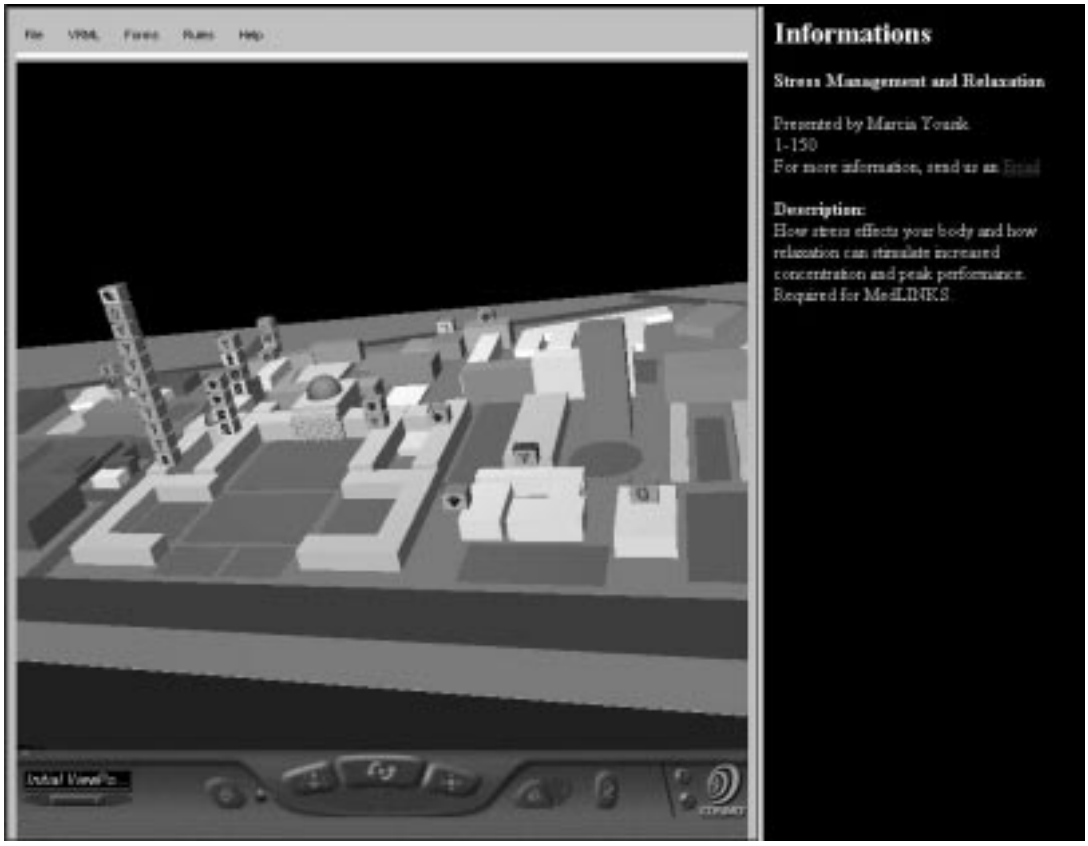

Figure 14 - Main Interface of the IAP application

The nodes are sent to the client, which creates the initial world in the VRML browser. At this moment, the user can chose the rules which apply to the result display (in which color some particular result will be) and chose the type of query he wants to send to the database.

All forms that appear on the user's screen are in fact generated on the server side and sent to the client, which only displays them. Once the user fills in the form, the result is sent back to the server and interpreted to generate a SQL query. The result is sent to the ToolSet that generates the result nodes to be displayed. All nodes in this case are cubes, the faces of which depict the requested information.

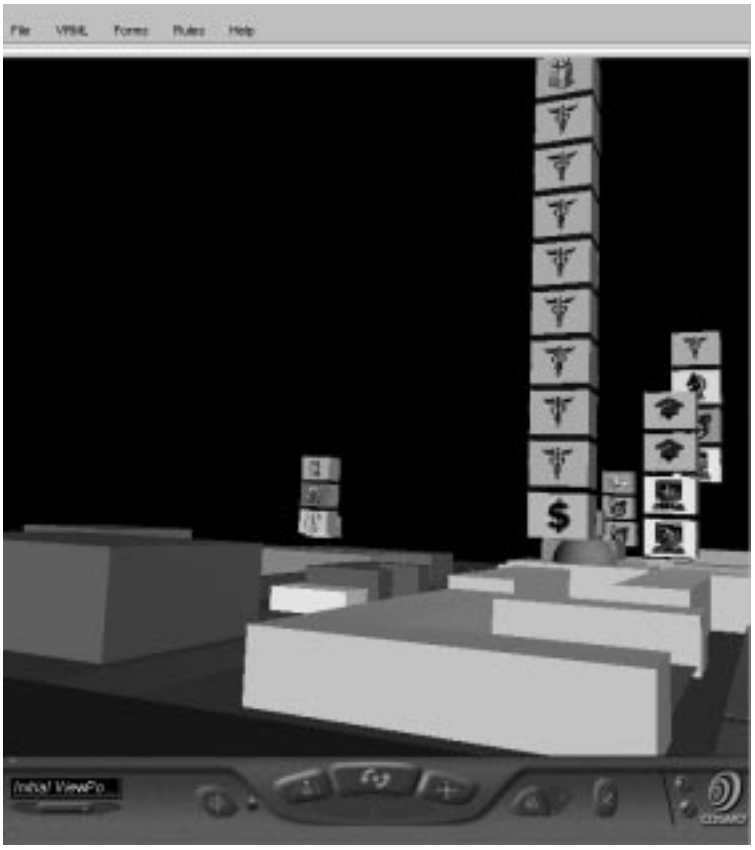

Figure 15 - Example of results display

A texture reflecting an event's category is mapped onto two opposite faces of the cube. If there is still room for new students in this activity, the cube is green, otherwise its color is red. On the others faces are mapped the rules chosen by the user.

If several activities occur at the same place, the new cubes are piled up. This allows efficient use of the third dimension to keep the localization of each event easy to visualize, which is not possible in a 2D map.

All cubes and buildings are sensitive to mouse events; if the user clicks on one of them, an event is sent to the server which returns an URL. This URL is loaded in the right frame of the screen and displays detailed information about this specific event or building.

One thing to remember with this application is that the ToolSet here is very short, just one line:

```
0 cube
```

We could have a different shape per category instead of a simple cube if the user wants to. To do that, we just need to replace this short ToolSet by a more complex one in which each category is listed (designated by its ID) and associated with a different prototype.

5 - Conclusion

5.1 – Summary

We presented the database visualization tools as a new possible three-dimensional user interface to display complex database content in intuitive way.

The current architecture of the tools is strongly client-server based, in which the client application is completely generic and all the graphic user interface and VRML generation is handled by the server. Since the client has been reduced to a Java applet and is based on an external VRML browser, the client is also platform independent.

The server and the client quickly exchange information to create the background world along with the list of queries supported by the application and the list of possible rules. A complete protocol has been designed to be able to carry this information between the client and the server. Particular attention has been paid to keeping the bandwidth required as low as possible, so only minimal information is sent across the network and all classes sent have auto-compression capabilities. Also, to keep the client independent of the database and its contents, the user interface dialogs used to make the query, are generated in the server and sent to the client which just displays them. Once the query done, the VRML is generated according to the results extracted from the database. To generate the three-dimensional visualization, several mapping files are needed to make the bridge between the “concepts” stored in the database and the VMRL prototype, and between the attributes names and the prototype argument names.

The resulting VRML is sent to the client where it is cached and displayed by the VRML browser. The client caches the incoming nodes to compensate for the problems with the

EAI, especially the fact that it is difficult to know which node generates any specific event. The cache helps to send to the client all information about the node we cannot have from the EAI.

We also introduced the different technologies used in the project, and introduced possible applications for the tools and 3D representation in general. The current working example is the IAP browser in which the user can visualize the results of his query on Independent Activity Period on a 3D MIT-campus model.

5.2 – Tools for new interface studies

As discussed in chapter 4, this three-dimensional tool has many applications. It also can be used to undertake research on new interfaces in 3D, and to find when it is advantageous to use the 3D to browse through databases instead of the current 2D/table representation.

5.2.1 – Benefits and drawbacks 3D vs 2D

The three-dimensional representation has some benefits over the two-dimensional representation. The obvious one is the possibility to use the third dimension and to be able to display more information in one screen. Representing multiple information at the same point of a map is nearly impossible. For instance in a map, as there is no space to display everything and graphics are overlapping and unreadable. This improvement in information grouping works also for just one object as the difference faces of the representation metaphor can convey a different information, and depending on the user's angle of view, give different feedback.

Another obvious benefit is the ability to recreate a natural environment for the user in which he can retrieve information more intuitively. For instance, a representation could be the user's office, in which the result of his query could be books in his library or objects on his desk.

Associated with this idea, virtual reality languages can create high interactive worlds. For instance, an audio file could be played only at a specific place in the virtual world and let the user know more information about his environment, or give an oral description of the object near him.

But of course, three-dimensional display has drawbacks. First of all, the creation of a 3D world is an expensive process and is still not good enough to have the capability to immerse[27] the user in a completely realistic environment. VRML actually requires keeping the world quite simple to be able to sustain a high frame rate.

Another drawback is that the conceptualization of a 3D world is a difficult task for users. If the user does not wear immersion hardware, such as a VR helmet, and the world is poorly designed, is very easy for the user to become "lost". For this reason, the worlds cannot be too abstracted, and should in some way reflect some real world logic. Creating a straightforward VRML generation from a query can be very confusing for the user. A confusing metaphor, which is not close of the idea it should represent, or a huge set of metaphor without order or without background, as an long line of kiosk for each musical event find in the database, is a bad use of the three-dimension representation. It does not really let the user transpose in a real environment what he his currently visualizing, so

[27] A 3D concept in which the user has the complete illusion of being in the virtual world. This means a perfect animation and capability to render in real time complex 3D models.

does not let him really understand the result of the query. That is the reason why the 3D visualization tools integrate the concept of “initialization world”, to create a background on which the user can have some landmarks. But all of this means a significantly higher cost to create a 3D world than a 2D page.

5.2.2 – Possibilities of modeling/visualization

As explained in the preceding section, the visualization of a 3D space should be either quite simple or be close to a real world environment depending on the kind of representation needed for the database.

The 3D representation is very well adapted to geographic representation, such as a landscape, a town or even an office. This should be used as an anchor to the real world for the user to let him visualize information intuitively. Information could then be added to the scene as metaphor, either by displaying an object close to the idea needed to be represented (a shop represented with a sign and multimedia add-on in the case of a yellow pages application) or by a simple object (cube, cylinder) with comprehensible textures (logo or icon) or color codes (such as green or blue for ok, red for not ok).

However, the creator of a 3D application should consider whether three dimensional representation really gives an enhanced view over a two dimensional one. Therefore, in cases the 3D is really used, for instance in the IAP example of the chapter 4, the cubes are piled up to represent a large number of activities in the same place without having overlapping.

5.3 - Future work

The current design of the database visualization tools in 3D as described in chapter 3 has been successfully implemented in Java in an application name DB3D.

However, the current implementation has several limitations that should be resolved. The three possible improvements are in the creation of the application, the rule system, and the integration of the embedded SQL to remove the need for application-specific Java classes associated with the VRML templates.

Actually, creating an application in the current implementation is a long process as all links between the Database and the VRML have to be described in files, and these files have to be created manually. This means difficulties creating large scale applications or dealing with complex databases. There are two solutions for this problem, the first solution is to create requirements for designing an application (constraints on the names of the objects especially), both for the database and for the VMRL templates. This allows an easy mapping between the database structure and the VRML code as the application designer does have the freedom to choose any kind of name but must follow the constraints (for instance, a color in the VRML prototype must be name `color`). Unfortunately, this could be too restrictive, especially in the case of reusing old databases which we do not want to change the structure of. The second solution is to create a parallel application with an intuitive graphic user interface in charge of generating the mapping information between a database and the library of VRML prototypes. This information could then be included directly in the database for easier recovery and management.

The second future work area is the improvement of the rule system to support any kind of complex rules and having a real statement system, such as the C-style ? : system[28]. This should extend dramatically the possibilities for user rules, and have the possibility to make rules on ranges of values instead of a simple association database value/rule value. This will be useful ,for instance, to make a rule and query on "zone", such as changing the color depending on the distance to a specific point.

The last major improvement to the current implementation of the tools should be the integration of the embedded SQL of the VRML Database Working Group described in the section 2.3.1. Having direct access to the VRML object field should make the generation of VRML smarter and should remove the need for the Java classes associated with the prototypes. The need for these classes is just to reformat values from the database to be VRML compliant, but with the embedded SQL this could be done directly in the VRML code. For instance: a VRML type SFvec3f[29] requires a value formatted as x y z, but these values are usually stored in separate attributes in the database, so the Java Class is needed to reformat the x, y, and z in x y z. But with something like

```
Script {
    eventIn SFBool execute
    eventOut SFvect3f x y z
    field SFString sql "FOR ALL SELECT X Y Z INTO :x :y :z FROM MYTABL"
    url "execsql.class"
    }
```

can reformat the x, y, z value in one eventOut value that can be routed to the actual field used by the object.

---

[28] ? : is a C shortcut to make a statement. The grammar is `test ? if_true : if_`false example: n = (a != 0 ? b/a : 0), means n = b/a if a not equals to 0, else n = 0.

[29] A type which take threes coordinate designate a position in a 3D space.

Appendix A – Example of mapping description files.

In this appendix, we present all the actual files needed to describe the complete application to make the visualization in 3D that builds on the database described in the section 4.4 example.

Building.wrl

```
#VRML V2.0 utf8

EXTERNPROTO Building [
   ExposedField SFVec3f     position
   ExposedField SFRotation  rotation
   field        SFVec3f     dimension

   exposedField SFColor     color
   exposedField MFString    texture1

   # standard fields
   eventOut     SFBool      isActive
   exposedField SFString    id
]
"http://www-ceci.mit.edu/db3d/proto/Building_Impl.wrl"
```

building.wrl has all the information about the header of the prototype, but must not have the default values, which are in the implementation of the prototype.

Building_Impl.wrl

```
#VRML V2.0 utf8

PROTO Building [
      ExposedField SFVec3f     position      0.5 0.5 0.5
      ExposedField SFRotation  rotation      0 1 0 0
      field        SFVec3f     dimension     1 1 1

      exposedField SFColor     color         0.4 0.4 0.4
      exposedField MFString    texture1      []

      # standard fields
      eventOut     SFBool      isActive
      exposedField SFString    id            ""
]
```

```
{
   Transform {
      translation IS position
      rotation IS rotation
      children [
        DEF TS TouchSensor {}
        Shape {
          appearance Appearance {
            texture ImageTexture { url IS texture1 }
            material Material { diffuseColor IS color }
          }
          geometry Box { size IS dimension }
        }
      ]
    }

    DEF S Script {
      eventIn SFBool activated
      eventOut SFBool isActive IS isActive
      url ["vrmlscript:
            function activated(value) { isActive = value; }
            "]
    }
    ROUTE TS.isActive TO S.activated
}
```

Building_Impl.wrl has all the code to create the VRML objects. It must contain default values for each parameter. This prototype has a small script that returns isActive [TRUE/FALSE] each time the touch sensor is pressed.

The following Java code receives the fields width/height/depth for dimension and x/y/z for position. It must create dimension and position (keywords from the prototype) from these three values. It also adds the field cubeNb to keep track of how many cubes have been already created for this prototype. Each cube has an unique Id with the format: BuildingName-NumberOfTheCube (example: W20-2)

Building.java

```
public class Building implements PrepareVRMLProto {
  protected double x, y, z, w, h;
  protected double[] offset;
  protected Double r;
  protected Hashtable properties;

  public Building() {}

  /**
   * Corrects the given Map to generate a new Map compliant
   */
  public Map translate(Map prop) {

    w = ((Double)prop.get("width")).doubleValue();
    h = ((Double)prop.get("height")).doubleValue();
    d = ((Double)prop.get("depth")).doubleValue();
    x = ((Double)prop.get("x")).doubleValue();
    y = ((Double)prop.get("y")).doubleValue();
    z = ((Double)prop.get("z")).doubleValue();
    prop.put("position",
             new String((x+offset[0])+" "+y+" "+(z+offset[1])));
    prop.put("dimension", new String(w+" "+d+" "+h));
    prop.put("x", new Double(x));
    prop.put("y", new Double(y));
    prop.put("z", new Double(z));

    prop.put("cubeNb", new Integer(0));

    return prop;
  }

 /**
  * Returns the position where to put a marker (flag)
  * on this proto and the id name
  */
  public Vector getMarkPosition(String buildingName) {
    Integer offset;
    Hashtable loc_prop;
    Vector result = new Vector();

    loc_prop = (Hashtable)properties.get(buildingName);

    offset = (Integer)loc_prop.get("cubeNb");
    loc_prop.put("cubeNb", new Integer(offset.intValue()+1));
    x = ((Double)loc_prop.get("x")).doubleValue();
    y = ((Double)loc_prop.get("y")).doubleValue();
    z = ((Double)loc_prop.get("z")).doubleValue();

    result.addElement(new double[]
                        {x, y+d+3+(offset.intValue()*3.5), z});
    result.addElement(new String(buildingName+'-'+offset.toString()));
    return result;
      }
```

Building.map

```
activity #Rule
PS_color color
TextureActiv texture1
```

The `activity` property must go to the RuleMapper to be able to be mapped. This means that the user has customized this particular field, apparently by a color. `PS_color` is coming back from the rule, and is mapped to color field of the prototype.

The `TextureActiv` property is directly mapped to the texture1 field.

Sp0LODhighC1

```
0 Building
```

This is a very simple Toolset in which all entities from the database with an Id of 0 will be represented with the Building prototype.

Reference

[1] David Flanagan *Java in a Nutshell*, O'Reilley 1997

[2] JavaSoft, *Java Platform 1.1 Core API Specification,* Available at http://www.javasoft.com/products/jdk/1.1/docs/api/packages.html

[3] Bernie Roehl, Justin Couch, Cindy Reed-Ballreich, *Late Night – VRML 2.0 with Java,* Ziff-David Press 1997

[4] ISO/IEC 14772-1:1997 *The Virtual Reality Modeling Language 97 specifications,* Available at http://www.vrml.org/Specifications/VRML97/

[5] Graham Hamilton, Rick Cattel, *JDBC Data Access with Java, Second Edition,* Addison Wesley 1998

[6] David M. Geary, Alan L. Mc Clellan, *Graphic Java*, SunSoft Press 1997

[7] Patrick Chan, Roseanna Lee, *Java Class Libraries, Second Edition,* Addison Wesley 1998

[8] Jed Hartman, Josie Wernecke, *The VRML 2.0 Handbook,* Addison Wesley Developers Press 1996

[9] Philip H. Bailey, Cyril Morcrette, Michael Privat, *3D world generation tools for database visualization,* White Paper CECI, September 1998.

[10] Mackinlay, J., R. Rao, and S. Card. (1995). *An organic user interface for searching citation links*. ACM Conference on Human Factors in Software (CHI '95), Denver, Colorado, ACM: 67-73.

[11] Card, S. K., G. G. Robertson, and J. D. Mackinlay. (1991). *The Information Visualizer: An information workspace*. ACM Conference on Human Factors in Computing Systems (CHI '91), ACM: 181-188.

[12] Mackinlay, J. D., G. G. Robertson, and S. K. Card. (1991). *The perspective wall: Detail and context smoothly integrated*. ACM Conference on Human Factors in Computing Systems (CHI '91), ACM: 173-179.

[13] Robertson, G. G., J. D. Mackinlay, and S. K. Card. (1991). *Cone trees: Animated 3D visualizations of hierarchical information*. ACM Conference on Human Factors in Computing Systems (CHI '91), ACM: 189-194.

[14] Daniel Lipkin, Oracle Corporation (1997). *Proposal for a VRML Informative Annex Recommended Practices for SQL Database Access*.
Available at http://www.vrml.org/WorkingGroups/dbwork/dbspec.html

[15] Erick Von Schweber, Infomaniacs. *Escape from VRML Island*, White Paper Informaniacs, August 5th 1998.

[16] Dan Ancona, *3DXML Authoring, Publishing and Viewing Structured Information.* White paper, VRML Consortium.