

**Vrije Universiteit Brussel**

**Faculty of Science**

**Department of Computer Science**

**Academic Year 2000-2001**



# **Generating Web Query Interfaces Based on Conceptual Schemas**

**Jo De Greef**

**Graduation Thesis Submitted to Obtain a  
License Degree In Applied Computer Science**

**Promotor: Prof. O. De Troyer**

# Abstract

Databases contain vast amounts of data. Retrieving specific information from a large and complex database is not always trivial, especially for novice and non-technical users. When information from databases is made available on the Internet, typically, users will be even more unfamiliar with the database structure. Therefore, it is very important to create a user interface that 'guides' the user through the database structure. It is important this query interface remains powerful while being easy to use at the same time. More experienced users should not be limited in their possibilities by the user interface.

Most of the time, a web query interface is a form where the user can fill in some text boxes and give some selection criteria. It is clear that this is a very limiting method of performing queries. Other query interfaces show the inner structure of the database and are in essence nothing more than a database front-end, too difficult to use by non-technical people.

In this thesis, I will propose a method to create a user-friendly interface to be used on web sites especially suited for people not familiar with the structure or the content of the database.

Exploring the database is split up in two distinctive steps. Typically, a user is interested in a specific type of information. Since a database usually contains more information than the user is interested in, it is important that he can quickly focus on that part of the database that matters to him. Visualisation maps will be applied to achieve this.

Actually performing queries on the database will be done by means of conceptual queries. Once again, a user-friendly interface is very important in order to allow novice users to create these conceptual queries.

# Acknowledgements

I would like to thank my thesis advisor Prof. O. De Troyer, who's idea it was to do research on the topic of this thesis, for guiding me, and for her suggestions, corrections and help in bringing this thesis to completion.

# Table of Contents

Abstract .....	1
Acknowledgements .....	2
Table of Contents .....	3
List of figures .....	6
1. Introduction .....	8
1.1 Problem description.....	8
1.2 Goal .....	8
2. Conceptual Schemas.....	11
2.1 Information levels.....	11
2.2 Conceptual schemas .....	13
2.3 Modeling methods.....	13
2.3.1 ER.....	13
2.3.2 ORM.....	14
What is ORM?.....	14
ORM notation.....	15
TensiNet ORM diagram .....	17
Shortcomings of ORM .....	20
2.4 Multi-layered Conceptual Schemas.....	20
2.5 Conclusion.....	24
3. Visualisation Maps .....	25
3.1 Criteria for visualisation.....	26
3.1.1 Introduction .....	26
3.1.2 Focus+context .....	26
3.1.3 Metaphors.....	27
3.1.4 Navigation .....	27
3.2 Standard visualisation techniques .....	28
3.2.1 Folders .....	28

3.2.2 2-D trees .....	28
3.2.3 Conclusion.....	29
3.3 Advanced visualisation techniques .....	30
3.3.1 Cone-trees.....	30
3.3.2 Hyperbolic trees .....	33
3.3.3 3D Landscapes or cityscapes.....	36
3.4 Conclusion.....	37
4. Chapter 4 .....	38
Visualisation Map.....	38
Implementation.....	38
4.1 XML representation .....	39
4.1.1 XML .....	39
4.1.2 Document validation .....	39
4.1.3 XML processors .....	40
4.1.4 Format .....	41
4.1.3 TensiNet XML document.....	43
4.2 VRML .....	44
4.2.1 VRML basics.....	44
4.2.2 VRML advanced .....	45
4.2.3 Sensors, events and routes.....	46
4.2.4 VRML shortcomings.....	47
4.2.5 Cone-tree design.....	48
4.2.6 XML to VRML mapping .....	48
Textbox.....	49
Connection line.....	51
Touch sensor / rotation.....	51
Cone-tree .....	53
4.3 Conclusion.....	56
5. Database Query Methods .....	57
5.1 Query methods .....	58
5.1.1 Form-based queries/Query-by-Example (QBE).....	58
5.1.2 Structured Query Language (SQL) .....	59
Employees table .....	59
5.1.3 Fifth Generation Query Languages .....	60
5.1.4 Conceptual queries .....	61
RIDL.....	61
ConQuer .....	62
5.2 ConQuer interface improvements .....	64

5.3 Conclusion.....	65
6. The Complete Process .....	66
7. Conclusion.....	69
Bibliography .....	71
Appendices .....	74
Appendix A – DrawTextbox .....	74
Appendix B – Cone tree example.....	76
Appendix C – TensiNet VRML file .....	78

# List of figures

<a href="#"><u>Figure 1: Creation of a user-interface based on a conceptual database schema</u></a> .....	9
<a href="#"><u>Figure 2: ER diagram</u></a> .....	14
<a href="#"><u>Figure 3: Common ORM symbols</u></a> .....	15
<a href="#"><u>Figure 4: ORM example 1</u></a> .....	16
<a href="#"><u>Figure 5: ORM example 2</u></a> .....	17
<a href="#"><u>Figure 6: TensiNet ORM diagram (1)</u></a> .....	18
<a href="#"><u>Figure 7: TensiNet ORM diagram (2)</u></a> .....	19
<a href="#"><u>Figure 8: TensiNet schema abstraction</u></a> .....	23
<a href="#"><u>Figure 9: Creation of a user-interface based on a conceptual database schema (2)</u></a> .....	24
<a href="#"><u>Figure 10: Roadmap</u></a> .....	26
<a href="#"><u>Figure 11: Folder view</u></a> .....	28
<a href="#"><u>Figure 12: 2-D tree</u></a> .....	29
<a href="#"><u>Figure 13: VRML cone-tree. The arrow shows the direction of the rotation</u></a> .....	31
<a href="#"><u>Figure 14: Hyperbolic tree</u></a> .....	33
<a href="#"><u>Figure 15: Klein model vs. Poincare model</u></a> .....	35
<a href="#"><u>Figure 16: Inxight Site Lens Studio</u></a> .....	35
<a href="#"><u>Figure 17: SGI File System Navigator</u></a> .....	36
<a href="#"><u>Figure 18: concept drawing of cone-tree</u></a> .....	48
<a href="#"><u>Figure 19: a textbox</u></a> .....	49
<a href="#"><u>Figure 20: VRML cone-tree</u></a> .....	54

<a href="#"><u>Figure 21: Creation of a user-interface based on a conceptual database schema (3)</u></a> .....	56
<a href="#"><u>Figure 22: Form-based query found on ebay.com</u></a> .....	58
<a href="#"><u>Figure 23: Employees table</u></a> .....	59
<a href="#"><u>Figure 24: Microsoft English Query</u></a> .....	60
<a href="#"><u>Figure 25: ConQuer GUI</u></a> .....	62
<a href="#"><u>Figure 26: ConQuer GUI - close-up (1)</u></a> .....	62
<a href="#"><u>Figure 27: ConQuer GUI - close-up (2)</u></a> .....	63
<a href="#"><u>Figure 28: Field selection</u></a> .....	64
<a href="#"><u>Figure 29: Right pane simplification</u></a> .....	65
<a href="#"><u>Figure 30: Creation of a user-interface based on a conceptual database schema</u></a> .....	66
<a href="#"><u>Figure 31: From visualisation map to conceptual query</u></a> .....	67
<a href="#"><u>Figure 32: Creation of a user-interface based on a conceptual database schema - detailed view</u></a> .....	68



# Chapter 1

## Introduction

### 1.1 Problem description

The Internet can be seen as an immense resource of information. Information is being made available on the Internet for everyone. In many cases companies, institutes, organizations and so on, have databases containing vast amounts of information and wish to make this data accessible through the Internet.

Presenting this information to users in a straightforward way is however not always possible. These users, although interested in the information, are not familiar with the way it is structured. Therefore, it is necessary to provide them with a user interface that allows them to easily navigate through the system and efficiently let them execute queries on the database.

### 1.2 Goal

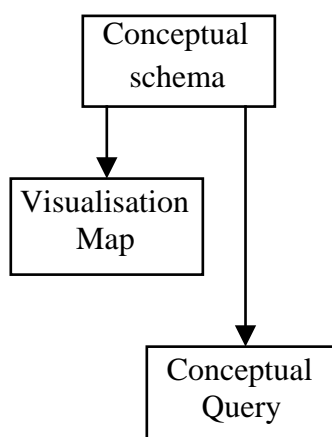
The goal of this thesis is to provide a method to build a user query interface for large information systems, specifically to be used in web applications. The user should be able to query the information system quickly and efficiently.

Conceptual schemes will serve as a critical base to obtain this goal. First, an outline of the system will be distilled from the conceptual scheme of the information system. This outline is used to show the user an

overview of the information using a visualisation map. In the context of the Internet, visualisation maps are traditionally used as site maps for large sites. I will apply the technique of visualisation maps to graphically represent an information system. This map allows the user to quickly browse through the main parts of the system.

Once the user focuses on a certain part, we will let him access the database using conceptual queries.

Diagrammatic, the process I propose roughly looks like this:



**Figure 1: Creation of a user-interface based on a conceptual database schema**

This diagram will be further detailed as I discuss the various steps later on.

In the second chapter, an introduction will be given about conceptual schemas since they play a fundamental role as building block for the creation of the user query interface. ORM<sup>1</sup>, being the modeling method I will use, will be introduced as well.

Chapter 3 will give a survey of the existing visualisation techniques. First, I will point out the disadvantages traditional techniques have, followed by a survey of advanced visualisation techniques that try to overcome these disadvantages. One of these techniques will be implemented and applied to a concrete example.

In chapter 4, I will discuss the actual implementation that generates a visualisation map based on the conceptual schema of a database.

---

<sup>1</sup> Object Role Model

The topic of conceptual queries will be approached in chapter 5.

Chapter 6 will put everything together and describes the complete system.

Throughout this thesis, the TensiNet [31] project will be used as an example to illustrate the theories presented in this thesis.

TensiNet is a European project, supervised by the VUB. One of its goals is to gather information and build a knowledge base containing data about tensile structures.

## Chapter 2

# Conceptual Schemas

As the title of the thesis states, the conceptual level of an information system will serve as a base to build our user query interface. First, a summary of existing conceptual schemas theories will be given. I will explain the reason why I use the conceptual level as a fundamental building block. Some modeling methods and mainly Object-Role Modeling (ORM), being the modeling method used in this thesis, will also be introduced.

The need for something beyond flat ORM schemas will be discussed, and several solutions to this problem will be explained and evaluated.

### 2.1 Information levels

An Information System (IS) may be viewed from four levels:

- External level
- Conceptual level
- Logical level
- Internal level

#### External level:

Describes what kind of information may be viewed and how it is displayed.

This is the level where interaction with the user takes place. An external view is created for each individual user or application. This level describes that part of the database that is relevant to a particular user. It also limits the actions a user can perform on the system.

Conceptual level:

Expresses the system in terms of natural concepts, like objects and roles. The conceptual level is used as a communication tool with the users during the design process.

Logical level:

Expresses the conceptual schema in terms of data structures and operations supported in the chosen data model (relational, hierarchic, ...).

In a relational model, the data structures would be tables and constraints are expressed using primary keys and other constructs.

Internal level:

Is closely tight to the chosen DBMS. Includes all details about the physical storage and access structures.

The internal level defines the types of stored records and indexes, how fields are represented, the various storage structures used, whether they use pointer chains or hashing, what sequence they are in, and so on.

The external level is highly dependent from the user interface while the logical and internal level depend on the implementation details of the system and therefore are not adequate to design an information system with.

The conceptual level however is independent on both the interface and the implementation. Neither changes to the user interface or changes in the implementation affect the conceptual level, which makes this level the most stable level to describe our information system.

Since the conceptual level expresses the system in concepts natural to a novice user, conceptual schemas are useful as a tool to talk about the information system in a way that is easy to understand by the end users of the system.

## 2.2 Conceptual schemas

A conceptual schema describes the Universe of Disclosure (UoD) of a certain information system in a way that is natural and unambiguous. The UoD is expressed in terms of concepts, which are familiar to the end user of the application. It completely specifies all the permitted states and transitions of the UoD.

A conceptual schema is independent of the Database Management System (DBMS) and user interface used. When the DBMS is changed and replaced by another one, the conceptual schema will remain unaffected. Since the schema does not specify how the data is displayed it is also independent of the user interface.

Another advantage of conceptual schemas is that they provide a natural way to communicate with end users about the information system. The closer we stay to the conceptual level, the easier the communication with novice users will be. Because of this, I will later on make use of conceptual queries, which are, as the name implies, queries formulated on the conceptual level.

## 2.3 Modeling methods

Several modeling approaches exist, such as Entity-Relationship (ER) modeling, extended ER (E-ER) and Object-Role Modeling (ORM).

### 2.3.1 ER

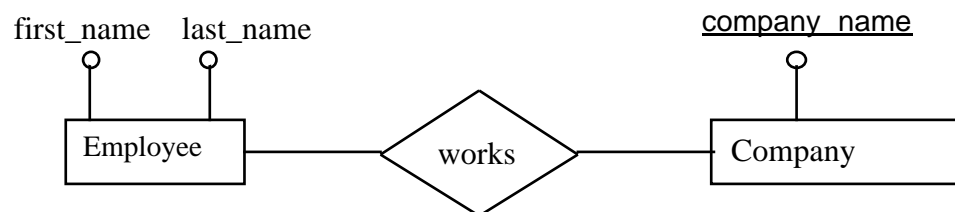
The Entity-Relationship model (see eg [25]) views the UoD as a set of basic objects (entities) and relationships among these objects. An entity is represented by a set of attributes.

Each attribute has a certain domain (a set of permitted values). For example, an entity *employee* can have as attribute his *age*, which can range from 20 to 100.

Basic constraints can be applied to relationships:

- *Cardinality*: specifies how many relationship instances any entity can participate in.
- *Participation*: indicates whether each entity must participate in a relationship or not.

The graphical notation of an ER diagram looks like this:



**Figure 2: ER diagram**

Extended ER diagrams (see eg [36]) allow more details/constraints in the UoD to be recorded such as:

- Composite attributes
- Derived attributes
- Subclasses and superclasses
- Generalization and specialization

### 2.3.2 ORM

Because I use ORM as our modeling method, it will be discussed in greater detail. The reason why ORM is preferred above ER will also be explained.

#### What is ORM?

ORM is a conceptual modeling approach that views an IS in terms of objects and the roles they play. A typical ORM schema consists of *object* playing *roles* (individually or in *relationships*).

For each object, or entity, there is a set of all possible instances. An entity is an instance of a particular entity type. For example, the entity type Book is the set of all books we want to talk about in the context of our IS. A book in particular is an instance of the Book entity type.

In contrast to other modeling techniques, ORM does not make use of attributes (opposed to ER and Object Oriented methods). Instead, everything is expressed by means of relationships. This makes ORM more stable since we do not have to worry about the question when an attribute becomes an entity or vice a versa.

ORM is also more expressive since its role-based notation makes it easy to express a wide range of constraints.

ORM foresees an automated mapping from the conceptual to the logical level (Rmap algorithm outlined in [14]). This means that a conceptual model can be mapped to a concrete implementation in a straightforward manner.

## ORM notation

The following graphical notation is usually used for ORM:

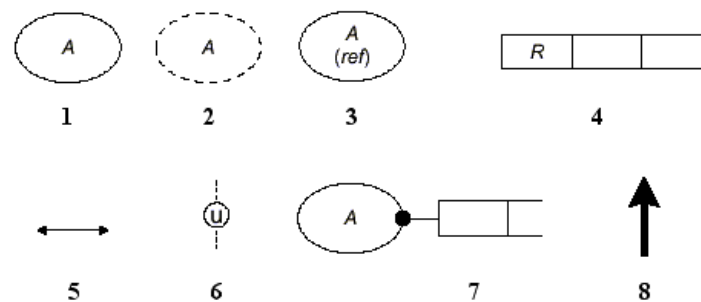


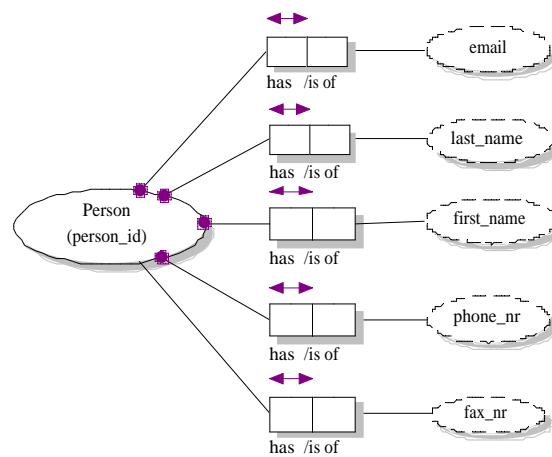
Figure 3: Common ORM symbols

1. Entity type
2. Value type
3. Abbreviated reference scheme
4. Ternary predicate compromised of three roles
5. Internal uniqueness constraint



6. External uniqueness constraint
7. Mandatory role constraint
8. Subtyping

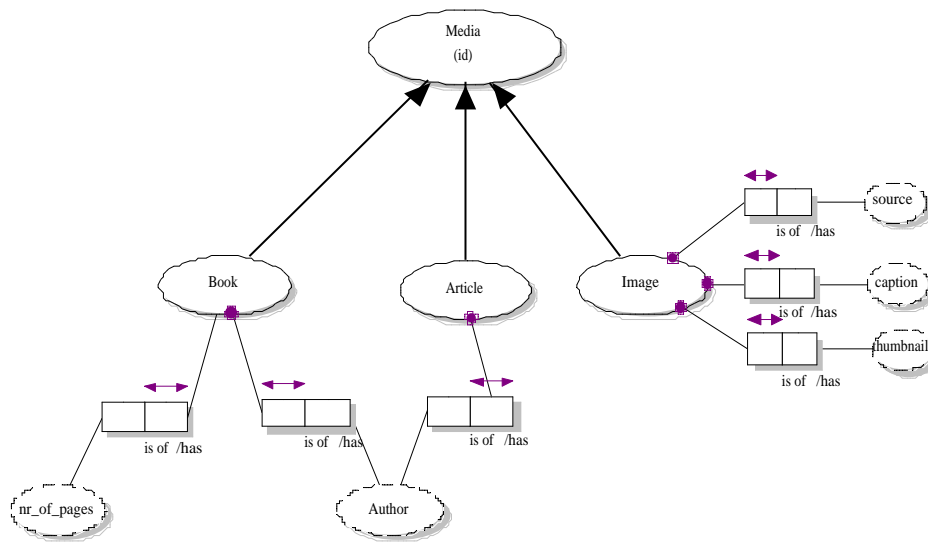
In order to illustrate the use of ORM some examples are shown here:



**Figure 4: ORM example 1**

Figure 4 shows an ORM schema of a *Person*, referenced by a *person\_id*. A *Person* has exactly one (specified by the external uniqueness constraint) *email* address, *first name*, *last name*, *phone number* and *fax number*. All of these roles except for the fax number are mandatory.

Figure 5 shows another example, making use of inheritance.

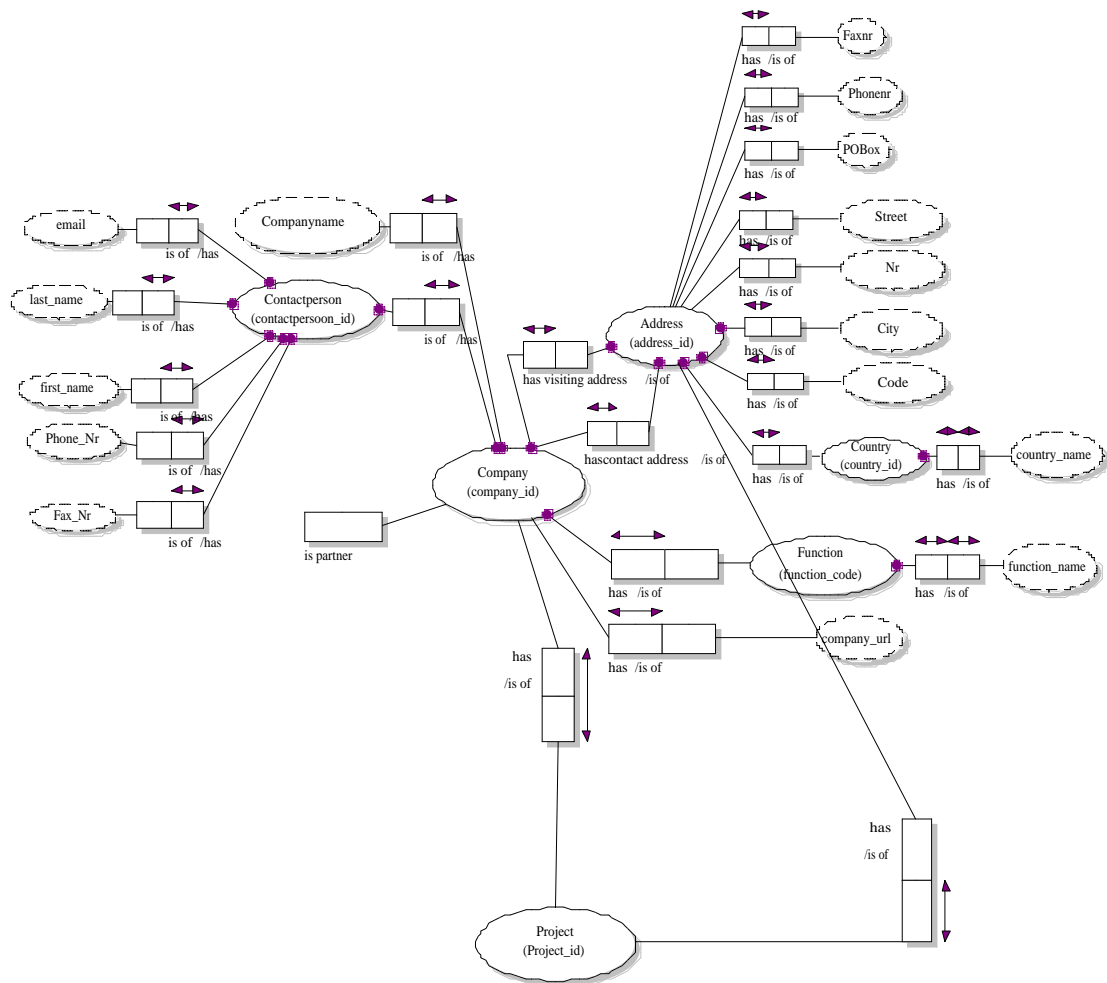


**Figure 5: ORM example 2**

The object type *Media*, has several subtypes, namely *Book*, *Article* and *Image*. Each of these subtypes has their own specific properties.

### TensiNet ORM diagram

The complete TensiNet ORM schema that will be used as an example is given in figure 6 and 7.



**Figure 6: TensiNet ORM diagram (1)**

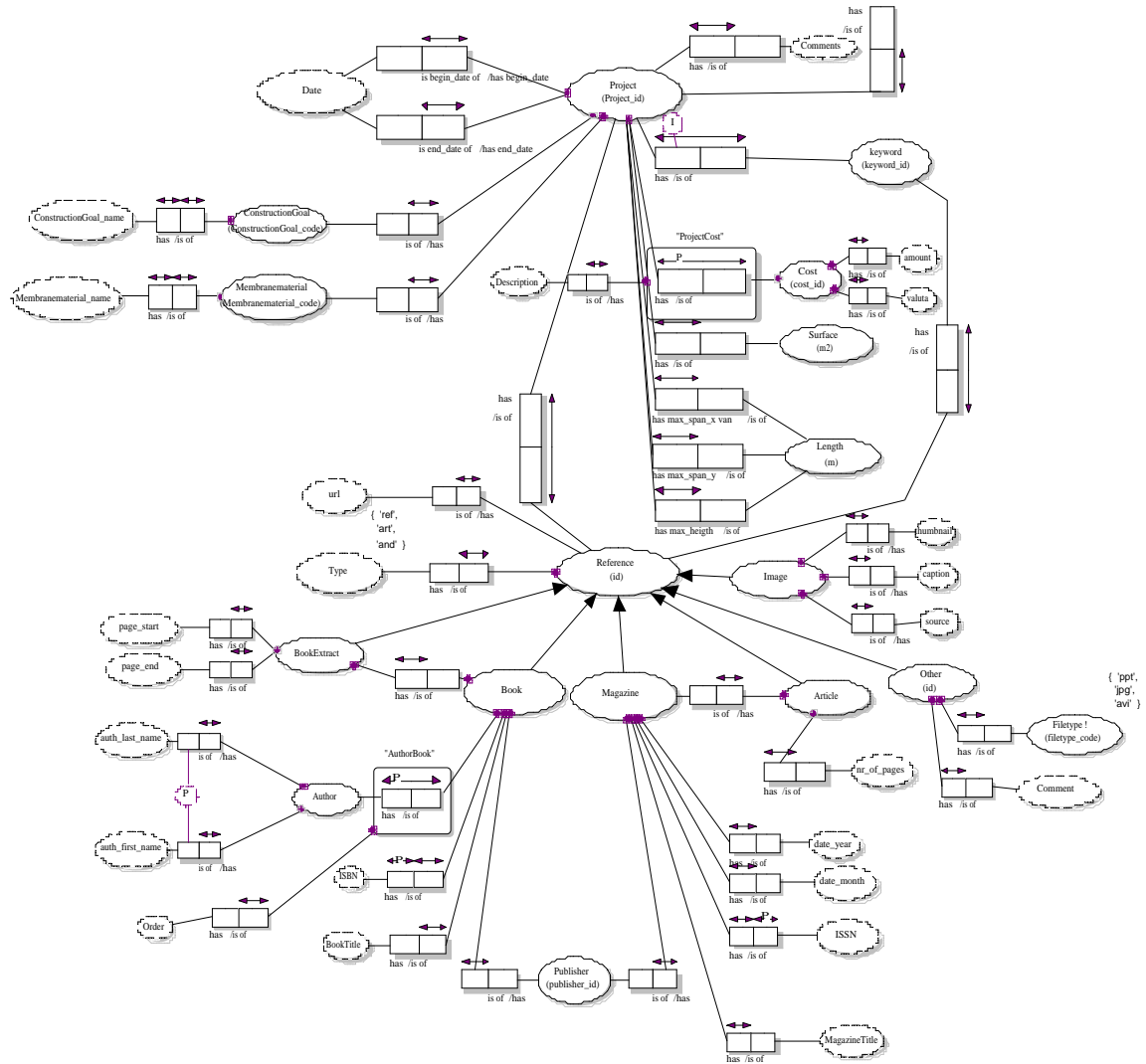


Figure 7: TensiNet ORM diagram (2)

*Projects*, identified by their *id*, form the kernel of this diagram. Each project has a set of properties associated, such as the used *membrane material*, the *span* of the construction, overall *cost* and so on.

Several *companies* may be involved in a project, an *architect*, an engineer, a *producer* of membranes and a *fabricant* of membrane material. Of each company, the *address* and *contactperson* are saved as well as some additional information.

A set of *references* is also stored in the system and associated with projects. Reference material can be either a *book*, a *book extract*, a

*magazine*, an *article* or an *image*. All of these have specific properties of their own that need to be recorded. For example, of an image we want to keep a *caption* and a *thumbnail* while a book requires us to store the *author*, *title*, *ISBN number*, ... .

The complete ORM diagram is too big to fit on one page and thus loses some of its coherence. This is a well-known problem regarding ORM schemas and will be further discussed below.

## Shortcomings of ORM

“The usefulness of any diagram is inversely proportional to the size of the model depicted” [23].

This notion is very important since large information systems become difficult to handle when using ORM diagrams. For small-scaled schemas ORM works fine and users can quickly obtain a complete overview of the diagram. When facing large diagrams, the user will be drowned in details and will not be able anymore to grasp the complete view of the system.

ORM schemas offer a global, one-dimensional view of the information system. While they provide a detailed model of the system, their efficiency quickly degenerates when dealing with larger (wall-sized) diagrams. This is caused by the fact that every part of the system is treated as equally important. The whole system is depicted at the same level. It is clear however that certain parts of an ORM schema are more important than others.

When we want to get an overview of a large system, we should be able to create several abstraction layers of this system, each at a more detailed level. ORM does not offer this possibility however. Several solutions will be discussed in the next section.

## 2.4 Multi-layered Conceptual Schemas

Our first challenge is thus to create a multi-layered schema given a traditional ORM diagram.

Different techniques for conceptual modeling on have been proposed in the literature:

- Viewpoint relative abstraction [22]
- Absolution abstraction hierarchy [22]
- ER Model Clustering [23]
- Nested ER (NER) [24]

Viewpoint relative abstraction:

Only a portion of the schema within a certain distance of a focussed object type is considered.

Absolution abstraction hierarchy:

Iteratively non-key concepts are removed from successive layers of abstraction. Non-key concepts are parts of the schema, which are deemed less important than other ones.

ER Model Clustering:

A hierarchy of successively more detailed Entity Relationship diagrams, with a lower-level diagram appearing as a single entity type on the next higher-level diagram.

Nested ER (NER):

Recursively grouping entities according to the cohesion among the entities involved.

The Nested ER model extends the data modeling capability of the basic entity relationship approach through a multi-level approach. Each object in the Nested ER model can be an attribute, an entity, or a relationship. Each of these objects can be either simple or complex.

- a Simple Entity is a collection of simple attributes.
- a Simple Relationship is a collection of simple attributes which describe a semantic connection between entities.
- a Complex Attribute is defined by a lower level entity or relationship.

- a Complex Entity is defined by a lower-level Nested E-R diagram together with a collection of simple and complex attributes which may be derived from the lower level diagram.
- a Complex Relationship amongst various entities is defined by a lower-level Nested E-R diagram involving the same entities, together with a collection of simple and complex attributes which may be derived from the lower level diagram.

Most of these techniques share a common element, namely the notion of “major object types” (although each technique has its own specific term for these objects types, e.g. key concepts, major entity types, maximal objects, dominant objects).

We can define major object types as the object types within a conceptual schema, which are considered to be of higher semantic importance because they keep the graph connected.

None of the abstraction methods mentioned provides an automated algorithm to distil these major object types. In all of the presented algorithms human guidance and intuition is required to select the major object types.

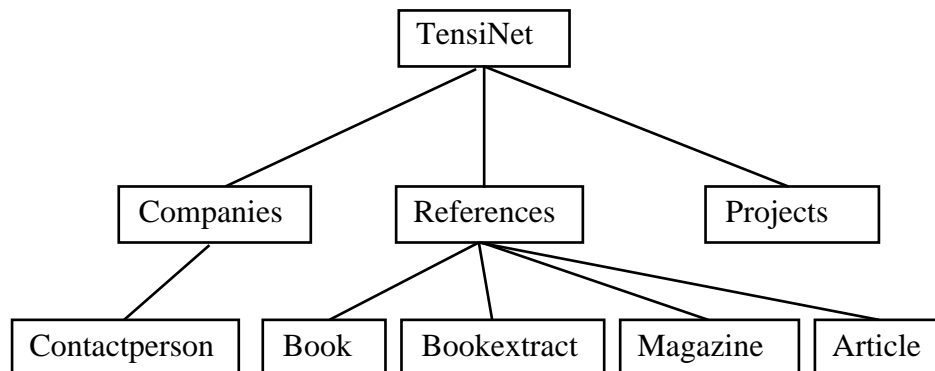
In [1], Dr. Terry Halpin proposes an algorithm that claims to automatically retrieve major object types. It also allows human intervention in case the results need to be manually corrected.

By weighting a schema according to a set of weighting rules that associates a weight to each fact type role based on the context of these roles, the major object types can be automatically retrieved.

As far as this thesis is concerned, I will suppose we have an abstraction of the existing ORM diagram, which can be obtained by any of the techniques discussed above.

This multi-layered abstraction will be organized as a tree structure, where the most important object (the object on the highest level) from our abstraction is the root node of the tree. Since we want our tree to have one root, a new top root node will be introduced in case there is more than once object at the highest abstraction level. This top node will be given the name of the information system itself.

The abstraction of the TensiNet ORM schema is as following:



**Figure 8: TensiNet schema abstraction**

At first sight, this might be a surprising abstraction, as project seems to play a very central role in the ORM diagram at first sight. A naïve solution would be to make Projects the top-node, and Companies and Reference child nodes of it.

Why have I opted for this alternative abstraction?

This solution is much more user-oriented compared to the first solution which is data-oriented. From a user's point of view, there are three important sources of information in the database: projects, involved companies and reference material. In this context, all three of them are equally important and is the abstraction as seen in figure 8 much more suited.



## 2.5 Conclusion

The whole procedure of constructing a web query interface is based on conceptual schemas. The motivate behind this is the fact that a description of the information system on the conceptual level is most easily understood by novice users. It describes the information system in a natural way and using concepts familiar with the end users.

Using an ORM diagram as a starting point, I have introduced the notion of multilayered diagrams that prove their usefulness when dealing with wall-sized diagrams.

Finally, these multilayered diagrams are structured like a tree structure and will be the base of visualization maps as will be seen in Chapter 3.

The original flowchart diagram is now extended and looks like this:

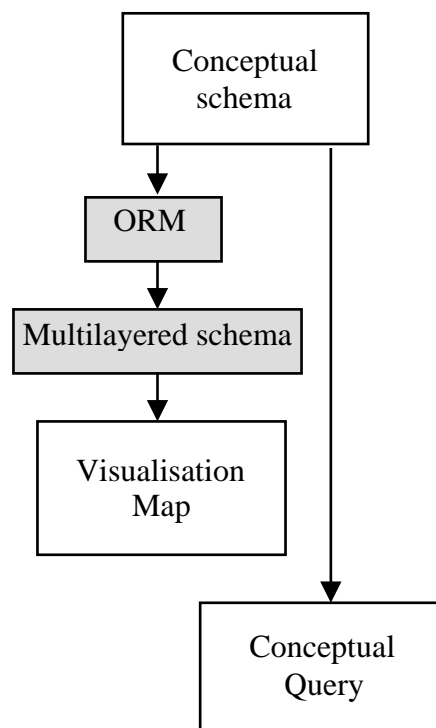


Figure 9: Creation of a user-interface based on a conceptual database schema (2)

## Chapter 3

# Visualisation Maps

Getting an overview of the available information is the first step in the process of searching for data. Especially because the user in general does not know what information is stored and how it is stored, it is important to provide the user with a clear overview of the structure of the data that can be retrieved.

Showing large data collections using a limited display surface (i.e. the computer screen) poses some serious problems as it becomes difficult to show all information at once.

The conventional approach maps all the information onto a region that is larger than the display and then uses scrolling to move around in the region. This approach has the problem that the user cannot see the relationship of the visible portion of the data to the entire structure (without auxiliary views). It would be useful to be able to see the entire hierarchy while focusing on any particular part so that the relationship of parts to the whole can be seen and so that focus can be moved to other parts in a smooth and continuous way.

A multi-layered ORM diagram provides us with a hierarchical view of the information system. A tree-like structure presents itself as the most natural way to visualise this hierarchy.

There is a multitude of representations available for tree structures, each with their own advantages and disadvantages. The most commonly used ones will be analysed below.

## 3.1 Criteria for visualisation

### 3.1.1 Introduction

In order to define a good visualisation technique, I will list some criteria which should be met by any technique.

### 3.1.2 Focus+context

One of the main goals we would like to achieve is to give the user a complete overview of the system at one glance. Since this is usually not possible with larger systems we will have to find a way to give the user at least a partial overview without him losing sight of the overall view.

A well-know technique to achieve this is called “focus+context”, originally developed by Xerox PARC. Focus+context is also given the name “fisheye view” which will become clear later on.

Focus+context allows one to zoom in (focus) on a area of interest while maintaining the surrounding context (but not in as much detail) at the same time.

The picture below illustrates an application of focus+context:

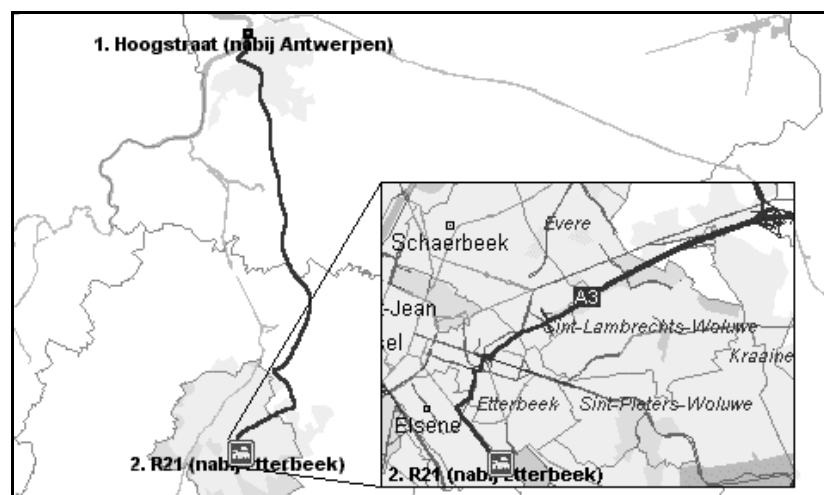


Figure 10: Roadmap

Imagine a route planner showing the route from location A to location B. The roadmap of an area is displayed. A so-called lens-view shows the destination area of the route in greater detail (the focus).

While focussing in on the destination area, the complete route is still visible to the user (the context).

### 3.1.3 Metaphors

Metaphors can provide a visual aid to the user to better illustrate parts of an information system.

These metaphors should be:

- *consistent*: objects that are similar in the database should also look similar.
- *easy to understand*: the user should immediately understand what the visual objects represent and how to interact with them.
- *powerful*: they should be able to represent all kind of objects and data contained in the database.
- *general*: their meaning should be easily understood by anyone, they should not be bound to culture or religion.

A classic example of bad metaphor is the recycle bin in the Apple OS. Dragging a floppy drive icon to the bin results in ejecting the floppy from the disk drive.

An example of a good metaphor would be the use of an image as an envelope to represent an e-mail. In the real world, an envelope has the same characteristics as what it describes in a computer environment.

### 3.1.4 Navigation

Another important aspect will be the ease of interaction with the interface. The user will be required to navigate through the hierarchy. Therefore, it is important this is as easy and natural as possible for the user.

## 3.2 Standard visualisation techniques

### 3.2.1 Folders

Well known as a standard GUI component in many operating systems is the standard folder structure as seen in figure 11. This might be the most used representation for hierarchies in classic user interfaces, it is also the most ineffective one. Although there is a hierarchy, each node is displayed likewise and seems to have the same importance.

Novice users quickly lose insight on the whole structure when this method is applied to larger hierarchies.

Starting out with a tree where all the nodes are collapsed, a user will have to click through a path of nodes (expanding each node with a click) in order to reach a node at a certain depth level.

Showing the tree with all its nodes expanded will result in an overload of information showed at once. Because of its ineffective use of screen-space, only a small part of the tree will be visible at once.

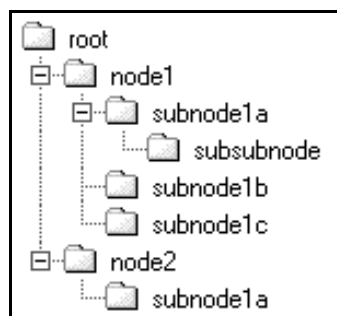
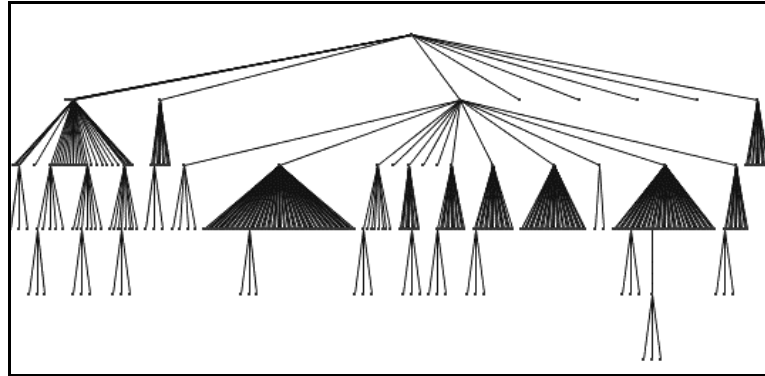


Figure 11: Folder view

### 3.2.2 2-D trees

The traditional presentation of hierarchies usually consists of a 2-D representation where child nodes are positioned under their parents in a wedge-like formation. Representing trees in this manner severely limits both the depth and breadth of the tree that one can view at a single time. Furthermore, navigating and finding specific nodes in such a structure can be confusing and disorienting.

As seen in figure 12, the usefulness of 2-D trees quickly degenerates when the size of the tree grows.



**Figure 12: 2-D tree**

### **3.2.3 Conclusion**

Standard visualisation methods such as the ones described here are sub-optimal solutions.

They provide a flat view of the hierarchy which is too disorienting for a non-knowledgeable user and do not match any of the criteria I have defined before.

### 3.3 Advanced visualisation techniques

#### 3.3.1 Cone-trees

Cone-trees, as the name suggest, display the tree in a cone-like formation, where each parent node has its child-nodes below him in a cone shape. They are basically the three-dimensional extension of regular 2-D trees. An example is given in figure 13.

The introduction of a third dimension effectively allows us to show more information using the same screen space. You will see that this third dimension is used in other techniques as well for exact the same reason.

Targeting a web-based interface, VRML (Virtual Reality Modeling Language) could be used to display a cone-tree.

One of the drawbacks is the limited interaction possibilities offered by VRML which makes navigating the tree a rather tedious experience.

This navigation can be eased as follows:

The tree is laid out horizontally (as seen in figure 13). A mouse-click on the red node will cause its child-nodes to rotate around the axis of the cone until the next child-node is shown up in front.

Clicking in a node will also make that node be shown in the centre of the screen (simply by moving the tree horizontally and eventually scaling it up or down).

In addition, colour, graphics (metaphors) and different node sizes can be used as additional visual aids. For example, all nodes that belong together according to some criteria would be coloured likewise, root nodes and other nodes of importance would be shown bigger than others, ...

When using images they must be carefully selected to match the metaphor criteria listed before. In addition, we have to make sure the images do not obstruct the view of the tree too much. Too many images may give a chaotic impression. A rule of thumb is to only use images for the most important nodes.

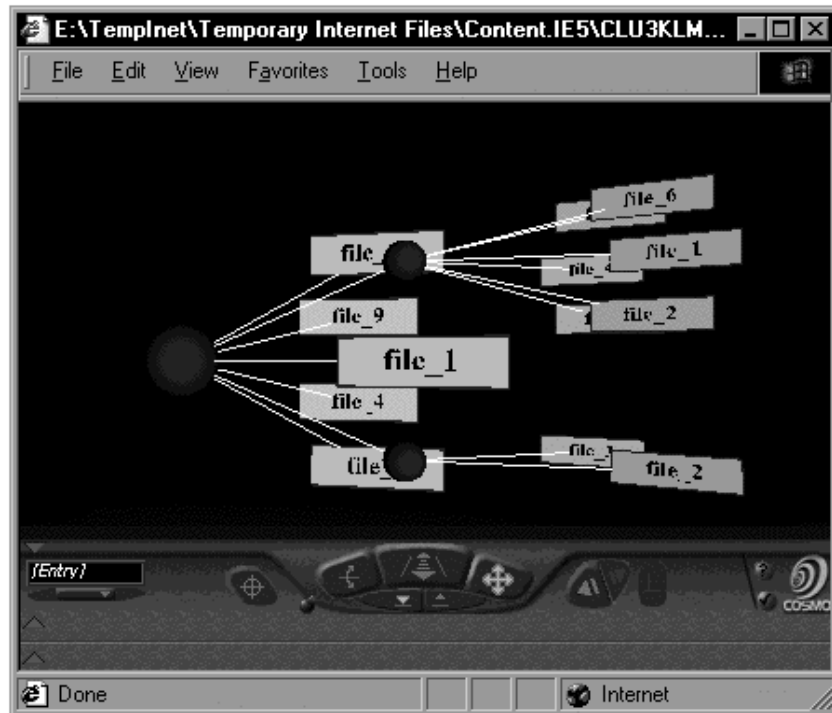


Figure 13: VRML cone-tree. The arrow shows the direction of the rotation

#### Node layout algorithm:

An important aspect of cone-trees is the layout of its nodes. They should be laid out in a way that the cones do not overlap each other and yet do not take up too much screen space.

A cone-tree is built as following:

The root node is placed in the middle with all children nodes distributed equally around their parent node along the base of the cone. This process is repeated for every node, diminishing the size of the cone according to the depth of the node in the tree. Different algorithms exist to determine the radius and height of the cones.

Two algorithms are proposed here.

The first algorithm (see [40]) is very basic and useful for small trees. It uses the following formula:

$$R(h) = \frac{R_i}{h+1}$$



$R(h)$  represents the radius of a tree at height  $h$ , and  $R_i$  some fixed initial radius. This function gives adequate results, but with an increasing number of hierarchies and nodes, it can lead to overlapping subtrees.

When confronted with larger hierarchies the previous algorithm comes short and formulas that are more complex are required.

The algorithm in [21] begins at the bottom of the tree (level  $n$ ), with a cone radius based linearly on the number of nodes in the cone. At all the levels above, the circumference for a cone at level  $n-1$  is estimated by:

$$C_{n-1} = 2 \sum_i r_{i,n}$$

where  $r_{i,n}$  is the radius of child  $i$  at level  $n$ . The radius is then calculated with:

$$r_n = \frac{C_n}{2\pi}$$

The arc length required for a child will be estimated by:

$$s_i = r_{i-1,n} + r_{i,n}$$

Subtrees are placed around the cone, with angles defined by:

$$\theta_i = \frac{s_i}{r_n} = \frac{r_{i-1,n} + r_{i,n}}{r_n}$$

This algorithm makes sure every cone occupies as much space as it requires in its parent's node.

### 3.3.2 Hyperbolic trees

Hyperbolic trees differ from the other methods that they do are not constructed in three dimensions, but rather on an hyperbolic plane which is then mapped to a circular display region.

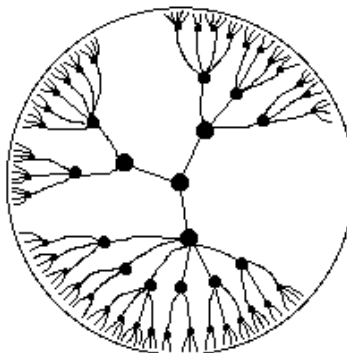
As seen in figure 14, the space available to display nodes grows exponentially going outwards as the radius of the circle grows. Especially since trees tend to grow exponentially with depth, this results in a uniform distribution of the tree on the hyperbolic plane.

A Java-applet can be used to show the hyperbolic tree, with the root node centred on the screen.

When the user clicks a node, it will float to the centre with the surrounding nodes regrouped and resized around the centred node.

The focussed node and its surroundings will be shown in full detail and centred on the screen. The other nodes will be located at the edges of the circle.

The farther away a node is from the node in focus, the smaller it will appear. It does allow however to display many of these minuscule nodes. Looking at figure 14, it is now clear why focus+context is also called a fisheye view.



**Figure 14: Hyperbolic tree**

Advanced interfaces such as Inxight's Site Lens Studio (see [41]) provide a smooth-varying focus+context view. This means the user cannot only click nodes to focus them but also freely drag nodes into focus and out of focus.

For large hierarchies, hyperbolic trees turn out to give the best results since they allow showing up to 10 times more nodes than other visualisations using the same screen space.

As with cone-trees it is also possible to use colour, size and images to visually enhance the comprehensibility of the system.

### **Node layout algorithm:**

Start with the root node placed on the hyperbolic plane.

Place all his children nodes at a fixed width, spaced evenly over a angle of available space (for the root node this is  $360^\circ$ )

For each of these nodes the process is iterated. The angle of available space however will diminish when going deeper down the iteration.

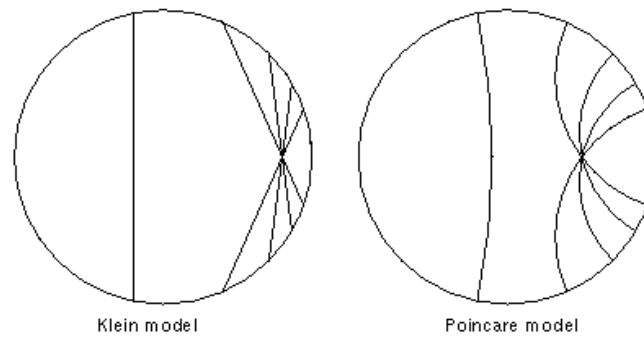
The iteration stops when there are no nodes left of a certain depth is reached. Once this is done, the hyperbolic plane is mapped to a 2-D plane (the screen).

Two different mapping methods can be applied. The Beltrami-Klein model (or simply the Klein model) and the Poincare model, map the hyperbolic plane to a focus at the centre of the disk while the rest of the hyperbolic plane fades off perspective-like toward the edge of the disk.

The points of both models are represented by points within a circle in the Euclidean plane.

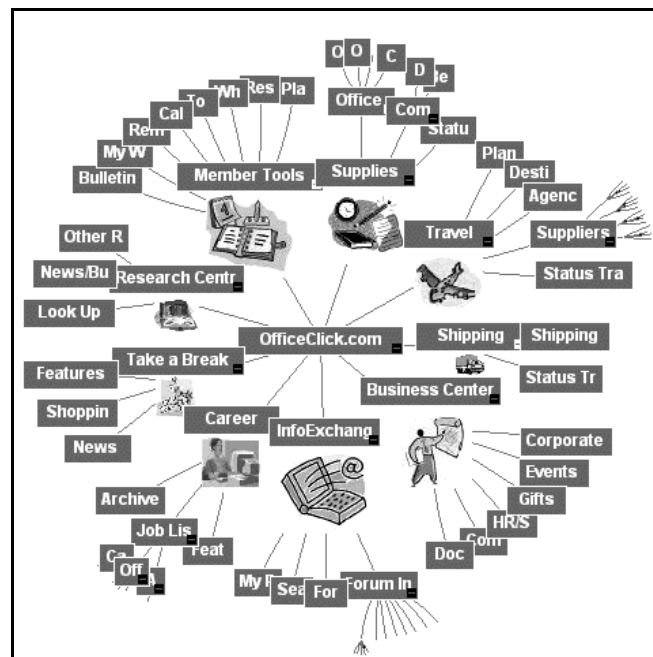
The hyperbolic lines in the Klein model, a conformal mapping, are represented by chords of the circle, distorting the angles from the hyperbolic space.

The hyperbolic lines in the Poincare disk model, a projective mapping, are represented by interior circular arcs that are perpendicular to the bounding circle and preserving the angles from the hyperbolic space. The Poincare model works better for visualizing hierarchies, because it preserves the fan-out shapes at nodes and uses the screen space more efficiently.



**Figure 15: Klein model vs. Poincare model**

An excellent example of a hyperbolic tree can be seen in figure 16, which depicts a site map using images as an additional visual aid.



**Figure 16: Inxight Site Lens Studio**

### 3.3.3 3D Landscapes or cityscapes

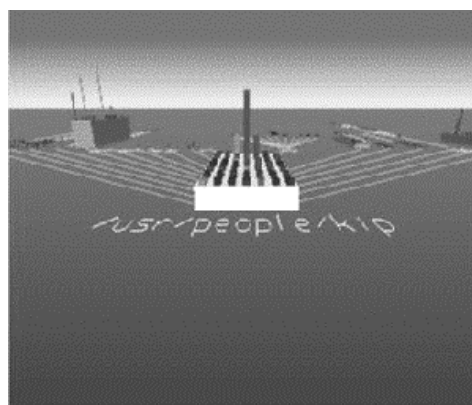
The hierarchical structure is spread out over a 3 dimensional landscape. Nodes are represented by 3-D structures on a flat plane, connected to each other by lanes.

The size and height of the structures can be used as an extra visual aid. For example, important nodes can be represented with taller structures. The use of a landscape allows to rigorously using real world metaphors for our nodes to enhance the recognizability of the system.

Navigation through the information is accomplished by “flying” over the landscape and go to the area of interest. The disadvantage is the fact that a 3-D environment needs to be mapped to a 2-D plane (i.e. the screen). This mapping causes a barrier between the user and the environment since all interaction is based in 2 dimensions. Given VRML’s limited interaction possibilities, navigating through a 3-D landscape becomes a rather tiresome experience. VRML and its possibilities will be discussed in depth in Chapter 4.

In the future, virtual reality devices might overcome this obstacle but for the time being, this causes quite some restrictions on the usability of 3-D landscapes as a viable tool to visualise hierarchies.

One of the first examples where this technique was applied was the File System Navigator, which comes with SGI Workstations (see figure 17). The directory structure of the disk is laid out over a map. The size of the files determines the size of the node as it is perceived in the three dimensional environment.



**Figure 17: SGI File System Navigator**

### 3.4 Conclusion

Up to now the field of applications for visualisation maps in the context of the World Wide Web was site maps. Nodes represented HTML pages and the connections between them acted as hyperlinks.

Visualisation maps turn out to be excellent tools to visualise information systems as well. Given the fact that we are designing a web query interface, either VRML or Java applets can be used to display maps.

Considering the three advanced visualisation techniques I have discussed, both cone-trees as hyperbolic trees are viable solutions to represent a hierarchic view of an information system. Due to the lack of decent interface tools, 3D landscapes are still too user-unfriendly to be used widely.

Based on the size of the system to be displayed a choice can be made between cone-trees and hyperbolic trees. Cone trees are very apt for small-scaled structures while hyperbolic trees are capable of efficiently displaying larger structures.

## Chapter 4

### Visualisation Map

### Implementation

Since our abstraction of the TensiNet system is a rather small-sized hierarchical structure I will use cone-trees to visualise it. As we concluded in the previous chapter we have the choice between cone-trees and hyperbolic trees. The advantage of hyperbolic tree only stands out when used for large trees. A cone-tree on the other hand is better suited for small trees. This cone-tree will be implemented using VRML since our query interface is aimed to be used in a web environment.

In chapter 2 I have introduced an abstraction of the original database using a hierarchical data structure. I will now define a data format in which the structure of the tree is stored using XML. In case our tree would grow in the future and cone-trees are no longer a viable option, the same XML data file can be used as input for a hyperbolic tree. In our case, the XML file will be translated to a VRML scene displaying a cone-tree.

First a small introduction about XML will be given, followed by the actual implementation of the abstraction of the ORM diagram in an XML document.

Since we will use VRML, I will discuss the most important aspects of VRML. To conclude this chapter a mapping from an XML document to a VRML cone-tree will be given.

## **4.1 XML representation**

### **4.1.1 XML**

XML (eXtensible Markup Language) is a markup language for any structured data on the Web. Unlike HTML, XML doesn't specify a tag set, nor semantics. In fact XML is a meta-language for describing your own markup languages. The semantics of an XML document is defined by the application using the document. An XML processor is used to read XML documents and provide access to their content and structure.

XML documents are made up of nested storage units called entities, which contain either parsed or unparsed data. These entities contain data and are referenced by their entity name. Each document has one entity called the document entity, which is the root for the whole document.

Because of its nested structure, XML lends itself very naturally to be used to model our hierarchical model of the ORM diagram.

Details about the XML syntax can be found at [37].

### **4.1.2 Document validation**

The purpose of a Document Type Definition (DTD) is to define the legal tags of an XML document. It defines the document structure with a list of legal elements.

XML provides an application independent way of sharing data. With a DTD, independent groups of people can agree to use a common DTD for interchanging data. An application can use a standard DTD to verify that data that you receive from the outside world is valid. One can also use a DTD to verify his own data.



### 4.1.3 XML processors

As mentioned before, an application uses an XML processor (a parser) to access the content of an XML file. Two major API's are available:

- SAX (Simple API for XML)
- DOM (Document Object Model)

#### SAX:

Sax uses an event-driven approach to parse XML data. The parser invokes the callback methods we have defined ourselves as it reads the XML data. Obviously this allows the parser to read large files, since they are read in a streaming sort of way.

SAX is useful when linearly reading an XML document is sufficient and there is no need of going back- or forward to other places in the document randomly.

#### DOM (Document Object Model):

DOM builds up a tree-structure of the XML document where each node contains one of the components of the XML structure. Because of this it is not recommended for use with large XML documents since the whole document will be stored in memory.

The advantage of DOM is that once the XML document is parsed and stored in memory, DOM provides random access to the parsed XML document. Each node can arbitrarily be modified, deleted or rearranged. In addition, new nodes can be added at any time.

Both API's clearly offer their own distinctive advantages and disadvantages. SAX forces us into a depth first tree transversal. When constructing our cone-tree this is not practical since we want a breadth first traversal, reading all nodes at the same level and placing them in a cone below their parent node located at a higher level. Therefore, DOM will be our XML processor of choice in this context.

#### 4.1.4 Format

Our hierarchic view of the original ORM diagram consists of nodes. This structure will be stored using XML. In order to do so we will first create a DTD in which nodes and their properties are defined.

Some of the information stored in our XML file comes from the conceptual diagram of the database, namely the caption of the nodes and their relationship with other nodes.

Additional information such as colour, images, ..., not found in the conceptual diagram are manually added.

A node has a caption, an identification number, a certain size, colour and an associated picture. Some of those properties are mandatory, others are optional.

This XML document is valid according to the following Document Type Definition (DTD):

```
<!DOCTYPE NODE [
  <!ELEMENT NODE (NODE*)>
  <!ATTLIST NODE caption CDATA #REQUIRED>
  <!ATTLIST NODE id      CDATA #REQUIRED>
  <!ATTLIST NODE color   None|Red|Green|Blue|Yellow)
  "None">
  <!ATTLIST NODE size    (small|regular|large)
  "regular">
  <!ATTLIST NODE image   CDATA "">
]>
```

As seen in the DTD, there is only one tag defined, namely <NODE>.

A <NODE> has the following attributes:

- Caption
- Id
- Color
- Size
- Image

Caption: (required)

Logically each node needs some text to explain what it stands for

Id: (required)

When the user clicks an end-node, (one without subnodes) he will be redirected to a web page where he can perform queries. The ID will be passed to this page so it knows which node has been selected.

Color: (optional)

A node can be colour to enhance the visual appeal. A standard set of colours is available as well as the possibility not to set a colour. In the case that no colour is specified, the application will decide the colour to use.

Size: (optional)

The concept of “focus+context” already displays the node in focus bigger than the other nodes. The “size” attribute however allows to manually influence this.

A straightforward example would be to set the size of the root node to “large”. This will cause that it will be drawn larger than other nodes and therefore less quickly disappears from the context.

This attributes is optional and defaults to “regular”.

Image: (optional)

Using images as metaphors. Although we will not be using images in our cone-tree, this attribute is included since it can be useful later on if we would implement another visualisation map where images can be included.

This attribute is optional. When omitted no picture will be displayed.

### 4.1.3 TensiNet XML document

Below is the TensiNet XML document<sup>2</sup> conforming the DTD I have defined:

```
<?xml version="1.0"?>
<NODE caption="TensiNet" id="0" color="Red">
  <NODE caption="Projects" id="1" color="Yellow">
  </NODE>
  <NODE caption="Companies" id="2">
    <NODE caption="Contactperson" id="4"></NODE>
  </NODE>
  <NODE caption="References" id="3">
    <NODE caption="Book" id="5"></NODE>
    <NODE caption="Bookextract" id="6"></NODE>
    <NODE caption="Magazine" id="7"></NODE>
    <NODE caption="Article" id="8"></NODE>
  </NODE>
</NODE>
```

As one can see, this corresponds with the multi-layered diagram of the TensiNet database, we defined in chapter 2, figure 8.

---

<sup>2</sup> Note that the reason we store our hierarchic view of the information system in an XML document is that we can use it to map it to different visualisation techniques.

## 4.2 VRML

### 4.2.1 VRML basics

I will use VRML to display the cone-tree. The data in the XML file will be translated to a VRML file, which can then be viewed using a web browser.

VRML is neither a programming language nor a scripting language. Instead, it describes a (interactive) 3-D world that can be shown in a web browser (given the proper plugins are installed). A commonly used plugin is CosmoPlayer [27].

VRML is organized into objects called nodes, which have their own fields and attributes.

A node has the following characteristics [26]:

- What kind of object it is. A node might be a cube, a sphere, a texture map, a transformation, etc.
- The parameters that distinguish this node from other nodes of the same type. For example, each Sphere node might have a different radius, and different texture maps nodes will certainly contain different images to use as the texture maps. These parameters are called Fields. A node can have 0 or more fields.
- A name to identify this node. Being able to name nodes and refer to them elsewhere is very powerful; it allows a scene's author to give hints to applications using the scene about what is in the scene, and creates possibilities for very powerful scripting extensions. Nodes do not have to be named, but if they are named, they can have only one name. However, names do not have to be unique-- several different nodes may be given the same name.
- Child nodes. Object hierarchy is implemented by allowing some types of nodes to contain other nodes. Parent nodes traverse their children in order during rendering. Nodes that may have children are referred to as group nodes. Group nodes can have zero or more children.

A complete specification of the VRML language can be found in [26], [28] and [29].

Example: the following Shape node defines a green box:

Shape

```
{
  appearance Appearance
  {
    material Material
```

```
    { diffuseColor 0 1 0 }
  }
  geometry Box
    { size 5.2 1.7 1.9 }
}
```

#### 4.2.2 VRML advanced

It is clear once complex scenes are built, one will have the need to reuse certain nodes. When building a city for example, you could build a few houses and then reuse them to ‘populate’ the city.

VRML offers an option to create predefined nodes using the DEF command.

```
DEF mySphere Sphere { radius 0.2 }
material Material
{
  diffuseColor 0 1 0
}
```

A named node *mySphere* has been defined, which can later on be used in the scene using the USE command.

```
USE mySphere
```

In addition, parametrized predefined nodes (prototypes) are also possible, using the **PROTO** command. The following VRML code defines a box which color can be defined as a parameter. The default color will be blue (RGB value = 0,0,1).

```
PROTO DrawBox [ SFCOLOR boxColor 0 0 1 ]
{
  Shape
  {
    appearance Appearance
    {
      material Material { diffuseColor IS
boxColor}
    }
    geometry Box { size 5.2 1.7 1.9 }
  }
}
```

Once this node is defined it can be used like this:

```
DrawBox { boxColor 1 0 0}
DrawBox { boxColor 0 1 0}
```

The above code will create two boxes, a red one and a green one.

### 4.2.3 Sensors, events and routes

Most nodes represent something you can actually see in your browser, but others, like the **TouchSensor** node for example, have no immediate visual effect themselves but can control other nodes. **TouchSensors** are not shown in the world but can be used as tools to interact with the user. In the case of **TouchSensors** they are used to respond to mouse-clicks (touch) of the user, generating an event every time they are clicked.

Another sensor I will use is the TimeSensor which generates events at certain time intervals.

What exactly is an event? An event is a data message send by one node to another one. The connection between the node generating the event and the node receiving the event is called a *route*. A node that produces events of given type can be routed to a node that receives events of the same type using the following syntax:

```
ROUTE NodeName.eventOutName_changed TO
NodeName.set_eventInName
```

#### 4.2.4 VRML shortcomings

As mentioned before, the major drawback of VRML in this context is the difficulty of navigation through a 3-D scene. Other drawbacks such as the inability to render high quality, photo-realistic scenes with proper lighting effects are not an issue here. Since our goal is to design an effective and user-friendly interface, the issue of navigation is very important however.

The CosmoPlayer plugin only allows 3 separate types of movement:

- *Tilt*: tilt the camera up or down or sideways.
- *Go*: move the camera around its vertical axis and move forward or backward.
- *Slide*: change the height of the viewpoint and strafe sideways.

Each of these movements needs to be performed separately. It is impossible to tilt the camera's viewpoint when moving around.

These limitations will be taken into consideration when designing our cone-tree.



### 4.2.5 Cone-tree design

Our goal is to create a tree as seen in figure 18. Taking into consideration the limited navigation possibilities of VRML, I will provide our own interaction method.

Instead of making the user move around the tree, we will allow him to manipulate the tree itself. Suppose the user wants to see the node that is behind *Sub9* in the picture, he can click the *Root* node. All of the child nodes of the *Root* node will now rotate down around the axis of the cone. At the end of the rotation, *Sub9* will be placed where *Sub1* was, and the node behind *Sub9* will become visible. Experiments with novice users have shown this is a very natural and above all easy to use technique to navigate the tree.

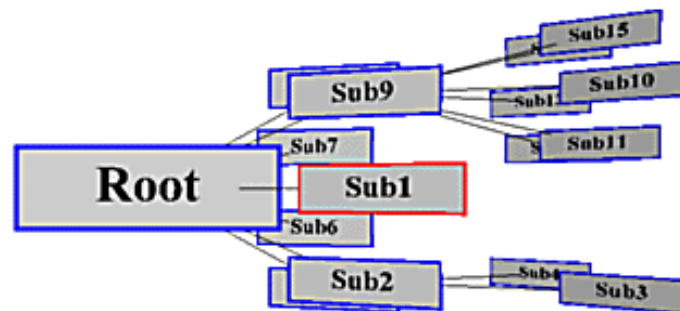


Figure 18: concept drawing of cone-tree

### 4.2.6 XML to VRML mapping

As seen in figure 18, our cone-tree will consist of textboxes connected to each other with lines.

The information in the XML document will be mapped to a VRML scene.

I will define three prototypes: a textbox, a connection line and a touch sensor.

## Textbox

A textbox is constituted out of three nodes, namely two rectangles, a coloured one and a somewhat smaller white one and a text object (figure 19).

I will define a textbox as a prototype with three parameters, being the scale, text and colour. Being able to set the scale and colour of the textbox is important when we want to map the data about the nodes contained in the XML document to our VRML cone-tree.

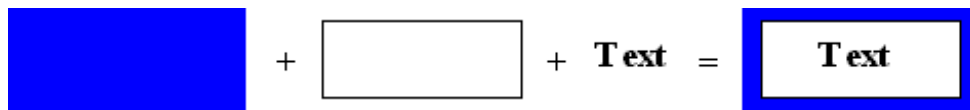


Figure 19: a textbox

A Textbox will consist of a group of nodes contained in a Billboard. A Billboard is a special grouping node. All child nodes will turn to face to the camera, which makes that the textbox will always face the user with the text side towards the user.

```
PROTO DrawTextbox [field SFVec3f scale 1 1
1 field MFString text "" field SFCOLOR
boxColor 0 0 1 ]
{
  Transform
  {
    scale IS scale
    children
    [Billboard
```

Once this is defined, the nodes that make up the actual textbox are given.

The coloured rectangle has its colour set to the colour that was specified as a parameter and has a fixed size. When the textbox node is scaled, all the child nodes will be scaled as well.

```
Shape
{
  appearance Appearance
  {
    material Material{diffuseColor IS
boxColor}
  }
  geometry Box { size 5.2 1.7 1.9 }
}
```

The white rectangle, with its colour set to white of course has a somewhat smaller size and is displayed a little in front of the coloured rectangle to prevent visual artefacts when rendering both rectangles.

```
Shape
{
  appearance Appearance
  {
    material Material { diffuseColor 1 1
1 }
  }
  geometry Box { size 5 1.5 2 }
}
```

The text node has its caption set to the text defined in the parameters.

```
Shape
{
  appearance Appearance
  {
    material Material { diffuseColor 0
0 0 }
  }
  geometry Text
  {
    string IS text
    fontStyle FontStyle
    {
      size 1.1
    }
  }
}
```

```

        justify "MIDDLE"
        style "BOLD"
    }
}

```

### Connection line

A connection line is a polyline, the coordinates of the line passed as an array of points.

```

PROTO ConnectionLine[field MFVec3f point [0
0 0 ]] {
    Shape
    {
        geometry IndexedLineSet
        {
            coord Coordinate
            {
                point IS point
            }
            coordIndex [ 0 1 -1]
        }
    }
}

```

IndexedLineSet uses the indices in its *coordIndex* field to specify the polylines by connecting together points from the *coord* field.

### Touch sensor / rotation

One last prototype is needed to allow the user to interact with the cone tree, namely a Rotation node that will be activated by a TouchSensor.

When a parent node is clicked, this node will rotate the cone-tree until the next child node is brought up in front of the camera.

A rotation node makes use of a TimeSensor. TimeSensors generate events at fixed time intervals. With each event of the TimeSensor the cone-tree will be rotated a little bit. This process ends when the next child node is shown up in front.

```
DEF RotTime TimeSensor
{
  cycleInterval IS rotateSpeed
  loop FALSE
  enabled TRUE
  startTime 1
}
```

In order to actually rotate the tree some code is required. VRML's Script node allows embedding code written in certain programming languages. In our case, we will use VRMLScript. Scripts typically respond to events. They are also able to generate their own events in return.

Our script will respond to the event given by the TouchSensor. It will initialise the parameters controlling the rotation and start the TimeSensor by generating a startTouch event. The *rotateSpeed* field defines for how long the TimeSensor will work (in terms of seconds).

The TimeSensor continuously sends out a *fraction\_changed* event to *Script.set\_fraction*, which is responsible to perform the actual rotation of the tree.

```
DEF Scrpt Script
{
  eventIn SFBool initProces IS aBool
  field SFBool b TRUE
  field SFInt32 child 0
  field SFFloat stepSize 0
  field SFInt32 nrChildren IS nrChildren
  field SFFloat nulRot 0.0
  eventOut SFTIME startTouch
  eventIn SFFloat set_fraction
  eventOut SFRotation set_rotation IS
set_rotation
  url "vrmlscript:
  function initProces(f, tm)
  {
    if (b)
    {
      stepSize = 6.28 / nrChildren;
```

```

        nulRot = child * stepSize;
        child = (child + 1)%nrChildren;
        startTouch = tm;
    }
    b = !b;
}
function set_fraction(f, tm)
{
    set_rotation[0] = 1;
    set_rotation[1] = 0;
    set_rotation[2] = 0;
    set_rotation[3] = nulRot + stepSize * f;
}
}
ROUTE Scrpt.startTouch TO
RotTime.set_startTime
ROUTE RotTime.fraction_changed TO
Scrpt.set_fraction
}

```

### Cone-tree

Once the building blocks (prototypes) have been defined, the actual cone-tree can be built. Size and colour of the nodes as defined in the XML document can be mapped to VRML using the prototype textbox. Our cone-tree will not support images however.

Using one of the node layout algorithms building the cone-tree is a trivial task.

Appendix B shows the complete listing of a sample cone-tree, as seen in figure 20.

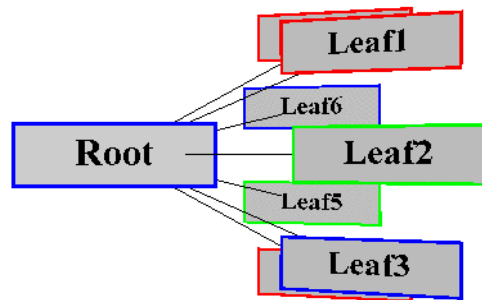


Figure 20: VRML cone-tree

The root node of the tree is TouchSensor containing two of our prototypes, namely a rotator node (*RotateProto*) and a textbox (*DrawTextbox*).

```
Group
{
  children
  [
    DEF Touch0 TouchSensor { }
    DEF Rotate0 RotateProto {
      nrChildren 7
      rotateSpeed 1.00 }
    DrawTextbox { scale 0.75 0.75 0.75 text
"Root" }
    DEF Trans0 Transform
      { translation 3.00 0 0 children
      [
```

The child nodes of the tree are defined as following:

```
      Transform
      { translation 0 -0.01 1.70 children
      [ ConnectionLine
        {point [ 0 0 0, -3.00 0.01 -1.70
1 ] }
      DEF Touch1 TouchSensor { }
      DrawTextbox
      {scale 0.38 0.38 0.38
      text "Leaf1"
      leafColor 1 0 0}
      ]
```

```
}

```

A connection line is drawn from the root to the child node. The coordinates of the child nodes are calculated using one of the algorithms described in this thesis as said before.

A TouchSensor (*Touch1*) is added to override the TouchSensor (*Touch0*) defined in the parent node.

Finally the textbox itself is drawn.

Remember the XML notation of a node:

```
<NODE caption="Leaf1" id="0" size="small"
color="Red">

```

This node translates to the following VRML code:

```
DrawTextbox
{scale 0.25 0.25 0.25
text "Leaf1"
leafColor 1 0 0}

```

Appendix C has the complete VRML description file of the TensiNet cone-tree.



### 4.3 Conclusion

In this chapter, the transformation from our abstraction of the database to a visualisation map using a cone-tree in VRML is explained.

First, an XML data format to store our multi-layered conceptual database schema is defined.

This XML file is mapped to a VRML scene depicting a cone-tree. Finally, some extra features were added to let the user easily navigate the tree.

Our flowchart diagram now looks like this:

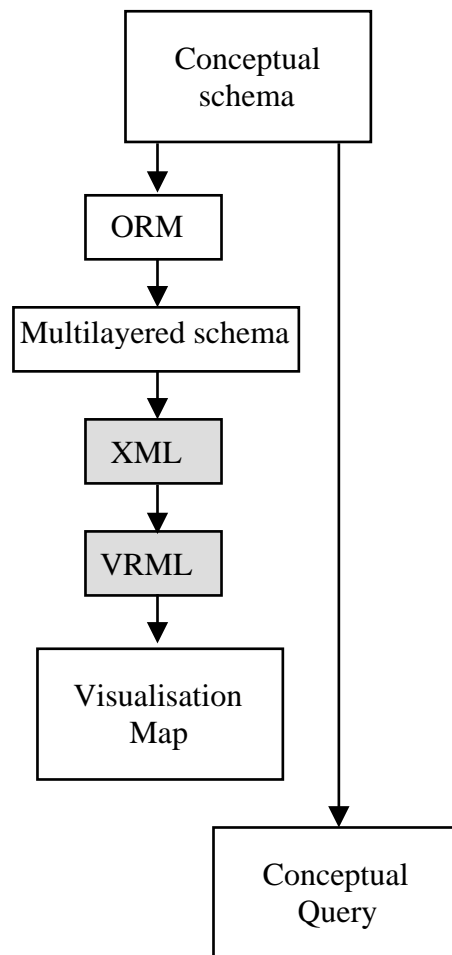


Figure 21: Creation of a user-interface based on a conceptual database schema (3)

## Chapter 5

# Database Query Methods

We applied visualisation maps to guide the user through the database structure and let him focus on a specific part. Our next goal is to guide the user through the process of formulating a valid query.

Traditionally, queries are formulated either at the external level, using forms, or at the logical level, using a language such as SQL.

Form-based queries, where the user is presented with a form and can fill in certain conditions, are typically very limited in expressiveness. In addition, when the external interface evolves the form-based query needs to be adapted.

SQL queries are more expressive but for the non-technical user SQL can be too difficult to master. Another drawback of SQL queries is the fact that when the conceptual or internal schema changes, the existing SQL queries become obsolete.

As before, we will see the conceptual level turns out to be the most stable and natural way to issue queries for a novice user.

A short summary of possible query methods will be given. I will then introduce a new query method that takes the analysis of the other methods into account.

## 5.1 Query methods

Several approaches exist to query a database, varying in expressiveness and ease-of-use.

### 5.1.1 Form-based queries/Query-by-Example (QBE)

Form-based queries are popular method used for websites.

They allow the user to fill out a form and specifying all kinds of search criteria. The form will be processed and an SQL query to retrieve the data the user specified will be generated.

The advantage is that the user does not have to learn a query language and that he is guided in his process of making a query; he is shown which search criteria can be used in his query.

While this is a very user-friendly method, it lacks the expressiveness we require for a decent query method. The simplicity of forms prohibits the use of complex queries.

Another disadvantage is the fact that form-based queries are situated on the external level: for each and every query a separate fill-out form needs to be made.

The image shows a screenshot of the eBay 'Smart Search' interface. It features a search bar with a 'Search' button and a 'tips' link. Below the search bar, there are several expandable sections for refining the search:

- Search Title** (NEW!): Includes a text input field, a dropdown menu set to 'All of these words', and a checkbox for 'Search title and description'. A link 'To find more items!' is also present.
- Words to Exclude**: A text input field for specifying words to exclude.
- Price Range**: Two input fields for 'Between \$' and 'and \$'.
- View Results**: A dropdown menu currently set to 'All items'.
- Payment** (NEW!): A checkbox for 'Accept eBay Online Payments by Billpoint' with a small icon.
- Search in Categories**: A dropdown menu set to 'All Categories'.
- Item location**: A dropdown menu set to 'All of eBay - include all regions'.
- Sort by**: A dropdown menu set to 'Items ending first'.

Figure 22: Form-based query found on ebay.com

### 5.1.2 Structured Query Language (SQL)

SQL [35] is a standard query language that can be used for a wide variety of relational databases. For experienced users, familiar with the database, SQL provides a relatively easy and yet powerful way to query databases.

Our target audience in this thesis is however a novice public. This public may not be familiar with SQL and its statements and more importantly, they do not know how the information is structured in the database.

Employees table

id	first_name	last_name
1	John	Carmack
2	Paul	Steed

Figure 23: Employees table

Figure 23 depicts a table from a fictional relational database. The SQL statement to retrieve information about the employees with first name 'John' would be this:

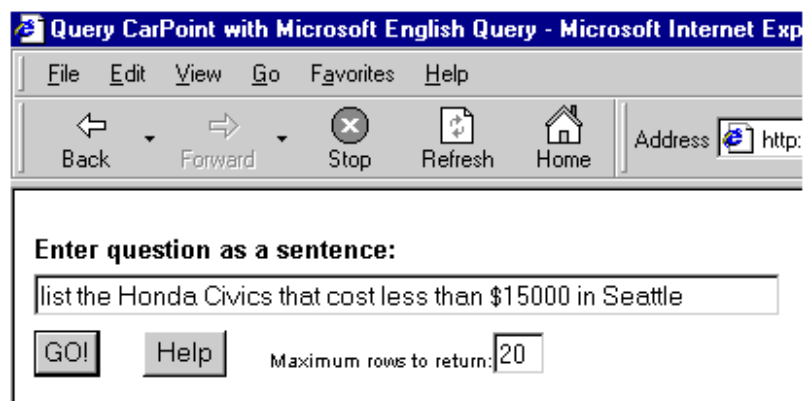
```
select id, first_name, last_name from Employees where  
first_name='John'
```

As shown in the example, not only does the user have to know the SQL language, he also needs to know the table structure. While SQL is a very powerful language for expert users, it is too difficult to use for a broader audience.

### 5.1.3 Fifth Generation Query Languages

Fifth Generation Languages (5GLs) use a natural language such as English to formulate queries.

Microsoft's English Query [39] is one of these fifth generation languages of which an example application can be seen in figure 24.



**Figure 24: Microsoft English Query**

The user has the possibility to query the database using a natural language (i.e. English). The phrase entered by the user will be mapped to an SQL statement that can be executed by the DBMS. It is clear the main benefit is that the user does not need to learn a specific query language.

Nothing assures us however that the user poses correctly formulated questions. An additional problem is that the user most likely does not know what kind of information is stored in the database. Remember that we solved a part of this problem by using visualisation maps to give the user an idea of what information can be retrieved. Now the user is focussed on a part of the database we do not want him to feel lost again. With MS English Query, instead of guiding the user through his query (as we will see in other query methods), he is left on his own.

An additional problem is that the database needs to be augmented with extra semantic information about the contents of the database. This has to be done manually and therefore causes a breach in our automated process from conceptual database diagram to web query interface.

### 5.1.4 Conceptual queries

A conceptual schema expresses an information system in terms of concepts, which are familiar to the end-user of the application.

Following this idea, allowing users to pose queries at the conceptual level would result in much more user-friendly and natural queries as opposed to queries formulated in a language such as SQL. On the other side conceptual queries should provide use more expressiveness than form-based methods.

Compared to other query languages, conceptual query languages, or CQL's for short, are still in their infancy. As far as query languages are considered, a relational language such as SQL is much more established and matured. On the other hand, at the conceptual level, most attention was given to modeling methods and less to CLQ's.

Different conceptual query languages (or CLQ for short) exist, among which RIDL [5] and ConQuer [6], [7] (both ORM-based). I will discuss these, focussing on their user-friendliness and usability.

#### RIDL

Although RIDL is a very powerful ORM-based CLQ, its advanced features are not easily mastered as it uses a command line interface. The user has to learn the RIDL syntax before he would be able to formulate queries.

An example of a query formulated in RIDL:

```
LIST EMPL-NAME OF ANALYST ( WRITING MODULE
OR
( RESPONSIBLE-FOR SET-OF-ROUTINES
AND
NOT ATTACHED-TO PROJECT WITH '?'
)
)
```

## ConQuer

ConQuer overcomes RIDL's lack of visual interface by providing the user with a GUI.

Traditionally, GUI's tend to degrade the expressiveness of a language. ConQuer however, claims to provide an intuitive interface for constructing almost any query without restrictions. An example of a query being constructed using ConQuer can be seen in figure 25.

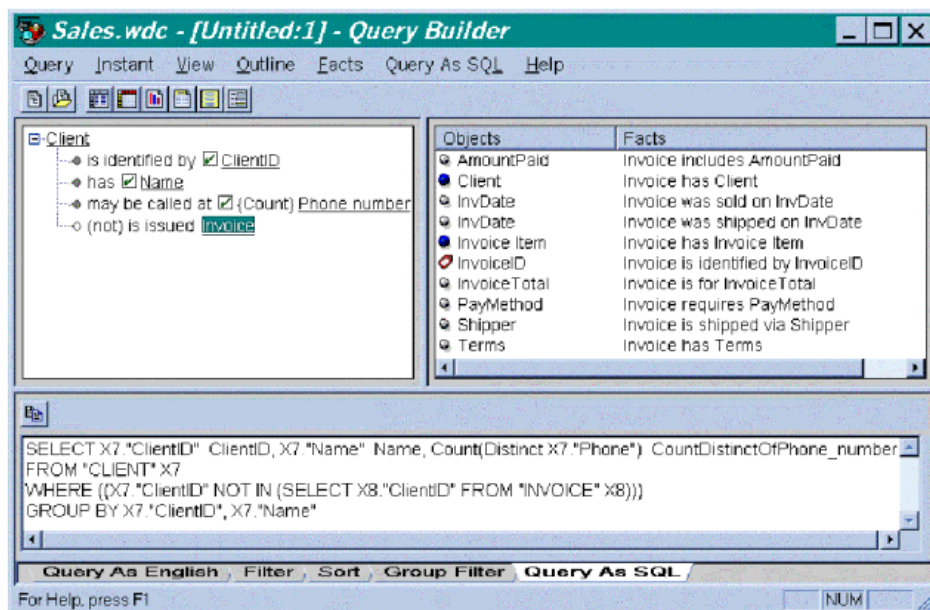


Figure 25: ConQuer GUI

As one can see, the interface consists out of 3 panes.

The left pane, seen in detail in figure 26 displays the object that is the subject of the query.

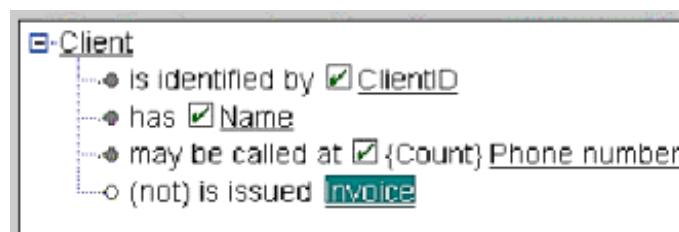
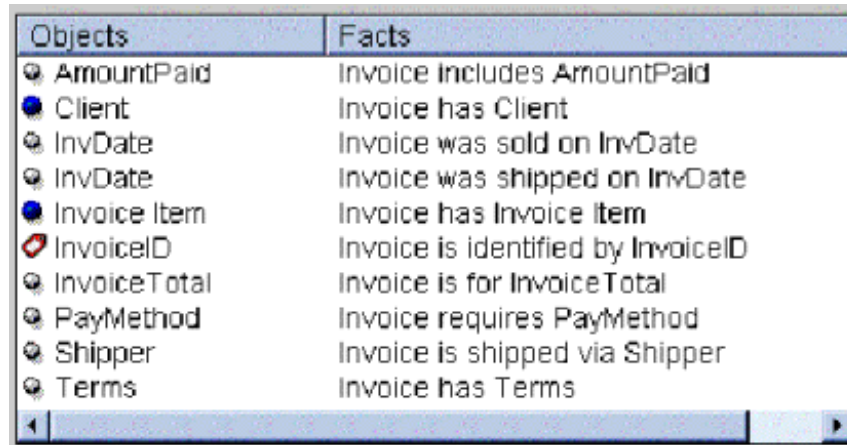


Figure 26: ConQuer GUI - close-up (1)

The right pane (figure 27), displays the roles played by the current object. In this example, the right pane shows the roles played by Invoice, which is the object highlighted in the left pane.



Objects	Facts
AmountPaid	Invoice includes AmountPaid
Client	Invoice has Client
InvDate	Invoice was sold on InvDate
InvDate	Invoice was shipped on InvDate
Invoice Item	Invoice has Invoice Item
InvoiceID	Invoice is identified by InvoiceID
InvoiceTotal	Invoice is for InvoiceTotal
PayMethod	Invoice requires PayMethod
Shipper	Invoice is shipped via Shipper
Terms	Invoice has Terms

**Figure 27: ConQuer GUI - close-up (2)**

Items to be displayed as the result of the query are checked in the left pane.

By dragging roles from the right pane to the left pane, additional search criteria can be defined.

Suppose we are looking for a client who has been issued a certain invoice. We start out with the Person object in the left pane. We drag the ‘is issued an invoice’ role to the left pane. Invoice is an object itself and by highlighting it, its roles will appear in the right pane. This is how we can put a restriction on the invoice we are interested in. Dragging the ‘is identified by InvoiceID’ and filling in the InvoiceID, we have set up our search criteria.

The bottom pane shows the SQL statement corresponding with our query. The bottom pane can also be switched to another view, verbalizing our query in English. This is very useful to give a user a clear view on what he has constructed.

If we compare this approach with a 5GL language such as implemented in English Query, we see that ConQuer offers the guidance throughout the query construction, we lack in English Query.



ConQuer is however by no means perfect. In the following section I will propose some changes and additions to the ConQuer interface which should improve ConQuer's usefulness<sup>3</sup>.

## 5.2 ConQuer interface improvements

In figure 26, we see how some of the branches of the Client object are ticked and have no constraints. These are the properties of the object that will be displayed in the result of the query.

The left pane serves two purposes: selecting which fields the user wants in the result query as well as creating search criteria. It would be a better idea to separate these two elements.

The left pane's purpose could be reduced to setting up search constraints. Once the user has constructed his query, he can execute it. Right before execution another window (figure 28) will be displayed, listing all the fields of the current object, asking the user to select which fields he wants to see displayed.

Objects	Facts
<input checked="" type="checkbox"/> AmountPaid	Invoice Includes AmountPaid
<input checked="" type="checkbox"/> Client	Invoice has Client
<input type="checkbox"/> InvDate	Invoice was sold on InvDate
<input type="checkbox"/> InvDate	Invoice was shipped on InvDate
<input checked="" type="checkbox"/> Invoice Item	Invoice has Invoice Item
<input checked="" type="checkbox"/> InvoiceID	Invoice is identified by InvoiceID
<input checked="" type="checkbox"/> InvoiceTotal	Invoice is for InvoiceTotal
<input type="checkbox"/> PayMethod	Invoice requires PayMethod
<input type="checkbox"/> Shipper	Invoice is shipped via Shipper
<input type="checkbox"/> Terms	Invoice has Terms

Figure 28: Field selection

---

<sup>3</sup> For a detailed resume about ConQuer, I refer the user to [6] and [7].

In the right pane, we propose to omit the Object column and simply list the Facts (roles played by the highlighted object) as seen in figure 29.

Facts
Invoice Includes AmountPaid
Invoice has Client
Invoice was sold on InvDate
Invoice was shipped on InvDate
Invoice has Invoice Item
Invoice is Identified by InvoiceID
Invoice is for InvoiceTotal
Invoice requires PayMethod
Invoice is shipped via Shipper
Invoice has Terms

**Figure 29: Right pane simplification**

## 5.3 Conclusion

Conceptual query languages (CLQ's) are very well suited to be used by novice users. However, compared to other query methods, the development of CLQ's has not been given as much attention as that of a relational query language such as SQL.

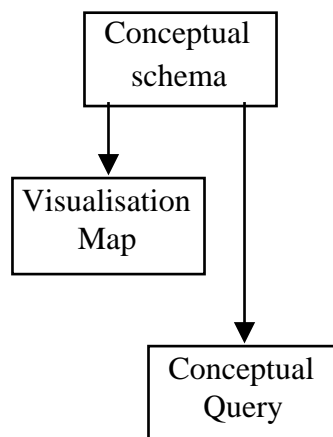
ConQuer is the only ORM-based CLQ with a graphical user interface. Because ConQuer is based on ORM, it fits in our framework to generate a query interface using ORM as a base.

Finally, I have outlined a set of changes and additions to ConQuer's interface, aiming to improve its usability.

## Chapter 6

### The Complete Process

In the first chapter, I outlined the process I propose to create a web-based query interface based on a conceptual database schema. Throughout the thesis, this process has been described in more detail. Looking back at the original diagram (figure 30), there are two distinct steps to be taken before a database query can be formulated.



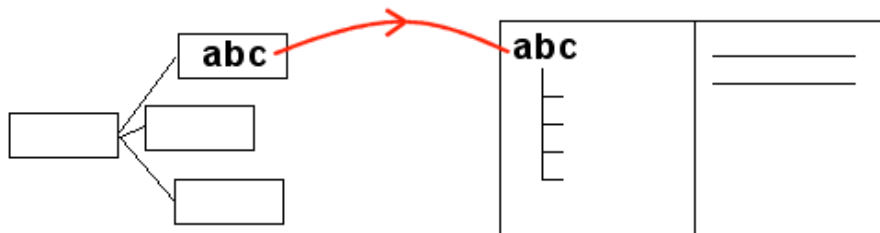
**Figure 30: Creation of a user-interface based on a conceptual database schema**

This process is a solution to the two-fold problem we encountered when novice users were confronted with an unfamiliar information system:

- β A novice user does not know the structure of a database he is not familiar with: we have used visualisation maps to depict the structure of the database in a user-friendly manner.
- β A novice user does not necessarily know a language such as SQL to formulate queries: constructing conceptual queries with the aid of a graphical user interface proves to be user-friendly for novice users.

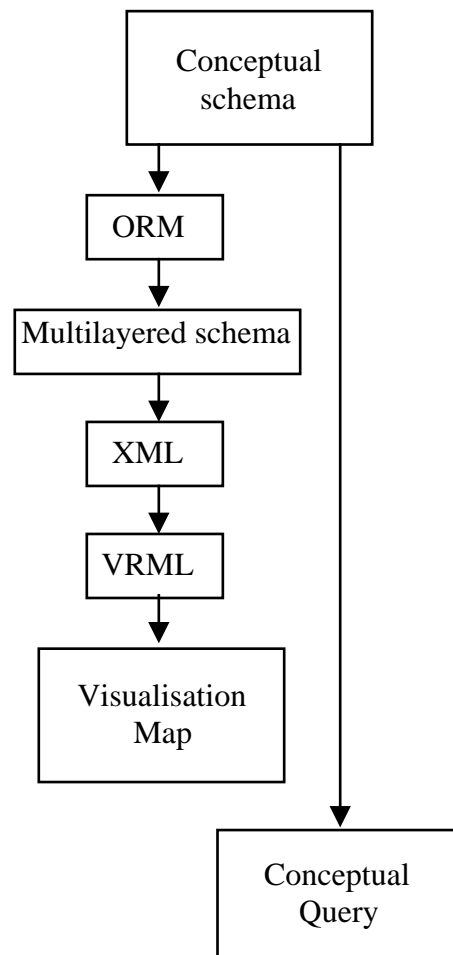
In a first step, a visualisation map is generated based on the conceptual schema of the database. The goal of the visualisation map was to give the user a clear view on the type of data stored in the information system. By means of the visualisation map, the user could focus on a certain part of the IS. The end result of the visualisation map is an object type from the database in which the user is interested.

In the second step, a GUI based conceptual query language is used to make the user pose a query about that particular object the user has chosen using the visualisation map.



**Figure 31: From visualisation map to conceptual query**

A complete view of all intermediate steps used in our user interface generation process is shown in figure 32.



**Figure 32: Creation of a user-interface based on a conceptual database schema - detailed view**

## Chapter 7

## Conclusion

To conclude my thesis, I would like to recapitulate the different issues discussed in this graduation thesis.

In chapter 1, the subject of this thesis is introduced: generating a web-based query interface for an information system. I outlined my own two-step process: first exploration, using visualisation maps, followed by searching using conceptual queries.

In chapter 2, I introduced conceptual schemas and motivated their use as a fundamental base.

The third chapter gave an overview of different visualisation techniques for large hierarchies.

In Chapter 4, I presented my own VRML implementation of a cone-tree.

Chapter 5 presented several query methods and gave an overview of their main characteristics. Additional improvements to the ConQuer interface were given as well.

In the last chapter, a complete overview was given of the two-step process I proposed.

In this thesis, various research topics have been used and explored, varying from conceptual schemas and query languages to visualisation techniques, VRML and GUI design.

Some fields were used as they are, for others we have proposed extensions. Although the contributions in these fields are not big, the power of the proposed system is in the combination of the existing techniques and methods.

The result is a new approach to allow novice users to explore large information systems through the Web.

# Bibliography

- [1] L.J. Campbell and T.A. Halpin and H.A. Proper , “Conceptual Schemas with Abstractions – Making flat conceptual schemas more comprehensible”, *Data & Knowledge Engineering*, 20(1), 39-85, 1996
- [2] John Lamping and Ramana Rao, “Visualizing Large Trees Using the Hyperbolic Browser”, Xerox Palo Alto Research Center, 1996
- [3] Dick Stenmark, “To Search is Great, to Find is Greater: a Study of Visualisation Tools for the Web”, Department of Informatics, Göteborg University Sweden, 1997
- [4] Martin Dodge, “An Atlas of Cyberspaces, Maps of Sites, Martin Dodge”, 2001
- [5] O. De Troyer, R. Meersman, F. Ponsaert , “RIDL User Guide”, 1984
- [6] A.C. Bloesch and T.A. Halpin, “Conceptual Queries using ConQuer-II”, *ER’97: 16<sup>th</sup> International Conference on Conceptual Modeling*, 1997
- [7] A.C. Bloesch and T.A. Halpin, “ConQuer: A Conceptual Query Language”, *Prod. ER’96*, 1996
- [8] C. Robert Carlson, Wenguang Ji and Adarsh K. Arora, “The Nested Entity - Relationship Model. - A Pragmatic Approach to E-R Comprehension and Design Layout”, *8th International Conference on E-R Approach*, 1989
- [9] John Lamping, Ramana Rao, and Peter Pirolli, “A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies”, *CHI ’95 Proceedings*, 1995
- [10] Tamara Munzner, “Visualizing the Structure of the World Wide Web in 3D Hyperbolic Space”, The Geometry Center, University of Minnesota, Paul Burchard - Department of Mathematics, University of Utah
- [11] Elliotte Rusty Harold, W. Scott Means, “XML in a Nutshell : A Desktop Quick Reference (Nutshell Handbook)”, O'Reilly & Associates, 2001
- [12] Cheng-Zen Yang and Chiun-How Kao, “Visualizing Large Hierarchical Information Structures in Digital Libraries”, *Proceedings of the Second Asian Digital Library Conference*, 1999
- [13] Nahum Gershon and Steve Eick, “Visualization's New Tack: Making Sense of Information”, *IEEE Spectrum*, Nov. 1995.



- [14] Terry Halpin, "Conceptual Schema & Relational Database Design, 2<sup>nd</sup> edition", 1999
- [15] Stuart K. Card, "Visualizing Retrieved Information: A Survey", Xerox PARC - Computer Graphics and Visualization in the Global Information Infrastructure CG&A, Vol. 16, No. 2, 1996
- [16] "The Web3D Repository" - <http://www.web3d.org/vrml/vrml.htm>
- [17] Vincent Nikkelen, "ConeGen: a cone tree generator", <http://www.stack.nl/~vincentn/ConeGen/>
- [18] "Extensible Markup Language (XML) 1.0 (Second Edition) - W3C Recommendation", 2000
- [19] "Xerces - XML parsers in Java", <http://xml.apache.org/index.html>, 1999
- [20] "SAX 2.0: event-based XML parsing", <http://www.megginson.com/SAX/>, 2000
- [21] Jeromy Carriere and Rick Kazman, "Interacting with huge hierarchies: Beyond cone trees", University of Waterloo, Ontario, 1995
- [22] D.Vermeir, "Semantic Hierarchies and Abstractions in Conceptual Schemata", Information Systems, 8(2):117-124,1983
- [23] P. Feldman abd D. Miller, "Entity Model Clustering: Structuring a Data Model by Abstractions", The Computer Journal, 29(4):348-360, 1986
- [24] C.R. Carlson, W. Ji, A.K. Arora, "The Nested Entity-Relationship Model", 1990
- [25] C. Batani, S. Ceri, S.B. Navathe, "Conceptual Database Design – An Entity-Relationship Approach", 1992
- [26] G. Bell, A. Parisi, M. Pesce, "The Virtual Reality Modeling Language - Version 1.0 Specification", <http://www.web3d.org/VRML1.0/vrml10c.html>, 1995
- [27] "CosmoPlayer", <http://www.cai.com/cosmo/>
- [28] "The Virtual Reality Modeling Language Specification Version 2.0", <http://www.npac.syr.edu/projects/tutorials/VRML/spec/2.0/spec/>
- [29] "The Virtual Reality Modeling Language - International Standard", <http://www.web3d.org/Specifications/VRML97/>
- [30] "Web3D consortium", <http://web3d.org/>
- [31] "TensiNet", <http://www.tensinet.com>

- [32] R. Yerneni, H. Garcia-Molina, "Design of Efficient Query Interfaces for Web Sources", Stanford University,
- [33] Y. Xie, C. Shakeri, "A Visual Query Interface for Web-based Database Access", Advanced Database Systems, CS561, Worcester Polytechnic Institute, 1997
- [34] T.A. Halpin, "Conceptual Queries", Database Newsletter, vol26, no2
- [35] R. Ramakrishnan, "Database Management Systems", McGraw-Hill, 1998
- [36] Martin Gogolla, "Extended Entity-Relationship Model"
- [37] "XML", <http://www.w3.org/XML/>
- [38] Elissa D. Smilowitz, "Do Metaphors Make Web Browsers Easier to Use?", Claris Corporation, <http://www.baddesigns.com/mswebcnf.htm>
- [39] "Developing with Microsoft English Query in SQL Server 7.0", <http://www.microsoft.com/SQL/techinfo/development/70/developingeq.asp>
- [40] Andrea Lombardoni, "VRML Interface for Internet OMS", , Diploma Thesis, 1999
- [41] "Inxight Site Lens Studio", <http://inxight.com/>

# Appendices

## Appendix A – DrawTextbox

```
PROTO DrawTextbox [ field SFVec3f scale 1 1 1 field
MFString text "" field SFColor leafColor 0 0 1 ]
{
  Transform
  {
    scale IS scale
    children
    [Billboard
    {
      axisOfRotation 0 0 0
      children
      [
        Shape
        {
          appearance Appearance
          {
            material Material { diffuseColor IS leafColor }
          }
          geometry Box { size 5.2 1.7 1.9 }
        }
        Shape
        {
          appearance Appearance
          {
            material Material { diffuseColor .8 .8 .8 }
          }
          geometry Box { size 5 1.5 2 }
        }
      ]
    }
    Transform
    {
      translation 0 -.25 1.2
      children
      [
        Shape
        {
          appearance Appearance
          {
            material Material { diffuseColor 0 0 0 }
          }
        }
      ]
    }
  }
}
```

```
geometry Text
{
  string IS text
  fontStyle FontStyle
  {
    size      1.1
    justify "MIDDLE"
    style "BOLD"
  }}]]]]]]}}
```

## Appendix B – Cone tree example

```

Group
{
  children
  [
    DEF Touch0 TouchSensor { }
    DEF Rotate0 RotateProto {
      nrChildren 7
      rotateSpeed 1.00 }
    DrawTextbox { scale 0.75 0.75 0.75 text "Root" }
    DEF Trans0 Transform
    { translation 3.00 0 0 children
      [
        Transform
        { translation 0 -0.01 1.70 children
          [ ConnectionLine
            {point [ 0 0 0, -3.00 0.01 -1.70 ] }
            DEF Touch1 TouchSensor { }
            DrawTextbox
            {scale 0.38 0.38 0.38
              text "Leaf1"
              leafColor 1 0 0}
          ]
        }
        Transform
        { translation 0 -1.33 1.06 children
          [ ConnectionLine
            {point [ 0 0 0, -3.00 1.33 -1.06 ] }
            DEF Touch2 TouchSensor { }
            DrawTextbox
            {scale 0.38 0.38 0.38
              text "Leaf2"
              leafColor 0 1 0}
          ]
        }
        Transform
        { translation 0 -1.66 -0.38 children
          [ ConnectionLine
            { point [ 0 0 0, -3.00 1.66 0.38 ] }
            DEF Touch5 TouchSensor { }
            DrawTextbox
            { scale 0.38 0.38 0.38
              text "Leaf3"
              leafColor 0 0 1 }
          ]
        }
      ]
    }
  ]
}

```

```

    }
    Transform
    { translation 0 -0.73 -1.53 children
      [ ConnectionLine
        { point [ 0 0 0, -3.00 0.73 1.53 ] }
        DEF Touch6 TouchSensor { }
        DrawTextbox
        { scale 0.38 0.38 0.38
          text "Leaf4"
          leafColor 1 0 0}

      ]
    }
    Transform
    { translation 0 0.74 -1.53 children
      [ ConnectionLine
        { point [ 0 0 0, -3.00 -0.74 1.53 ] }
        DEF Touch7 TouchSensor { }
        DrawTextbox
        { scale 0.38 0.38 0.38
          text "Leaf5"
          leafColor 0 1 0}

      ]
    }
    Transform
    { translation 0 1.66 -0.38 children
      [ ConnectionLine
        { point [ 0 0 0, -3.00 -1.66 0.38 ] }
        DEF Touch8 TouchSensor { }
        DrawTextbox
        { scale 0.38 0.38 0.38
          text "Leaf6"
          leafColor 0 0 1}

      ]
    }
    Transform
    { translation 0 1.33 1.06 children
      [ ConnectionLine
        { point [ 0 0 0, -3.00 -1.33 -1.06 ] }
        DEF Touch9 TouchSensor { }
        DrawTextbox
        { scale 0.38 0.38 0.38
          text "Leaf7"
          leafColor 1 0 0}

      ]
    }
  ]
}

```

## Appendix C – TensiNet VRML file

```
#VRML V2.0 utf8
WorldInfo { title "TensiNet" info ["Tennisnet"] }
Background
{
  skyColor
  [
    0.0 0.0 0.0
    0.0 0.0 0.0
    0.0 0.1 0.3
  ]
  skyAngle
  [
    1.57,
    1.87,
  ]
}

PROTO DrawFile [ field SFVec3f scale 1 1 1 field MFString
text "" field SFCOLOR leafColor 0 0 1 ]
{
  Transform
  {
    scale IS scale children
    [
      Billboard
      {
        axisOfRotation 0 0 0
        children
        [
          Shape
          {
            appearance Appearance
            {
              material Material { diffuseColor IS leafColor }
            }
            geometry Box { size 5.2 1.7 1.9 }
          }
          Shape
          {
            appearance Appearance
            {
              material Material { diffuseColor .8 .8 .8 }
            }
            geometry Box { size 5 1.5 2 }
          }
        ]
      }
    ]
  }
}
```

```

translation 0 -.25 1.2
children
[
  Shape
  {
    appearance Appearance
    {
      material Material
      { diffuseColor 0 0 0 }
    }
    geometry Text
    {
      string IS text
      fontStyle FontStyle
      {
        size 1.1
        justify "MIDDLE"
        style "BOLD"
      }
    }
  }
]
}
]
}
]
}
}

PROTO ConnectionLine [ field MFVec3f point [ 0 0 0 ] ]
{
  Shape
  {
    geometry IndexedLineSet
    {
      coord Coordinate
      {
        point IS point
      }
      coordIndex [ 0 1 -1]
    }
  }
}

PROTO RotateProto
[
  field SFInt32 nrChildren 1
  field SFTime rotateSpeed 1
  eventIn SFBool aBool
  eventOut SFRotation set_rotation
]
{
  DEF RotTime TimeSensor
  {

```



```

cycleInterval IS rotateSpeed
loop FALSE
enabled TRUE
startTime 1
}
DEF Scrpt Script
{
eventIn SFBool initProces IS aBool
field SFBool b TRUE
field SFInt32 child 0
field SFFloat stepSize 0
field SFInt32 nrChildren IS nrChildren
field SFFloat nulRot 0.0
eventOut SFTime startTouch
eventIn SFFloat set_fraction
eventOut SFRotation set_rotation IS set_rotation
url "vrmlscript:
function initProces(f, tm)
{
if (b)
{
stepSize = 6.28 / nrChildren;
nulRot = child * stepSize;
child = (child + 1)%nrChildren;
startTouch = tm;
}
b = !b;
}
function set_fraction(f, tm)
{
set_rotation[0] = 1;
set_rotation[1] = 0;
set_rotation[2] = 0;
set_rotation[3] = nulRot + stepSize * f;
}"
}
ROUTE Scrpt.startTouch TO RotTime.set_startTime
ROUTE RotTime.fraction_changed TO Scrpt.set_fraction
}

Group
{
children
[
DEF Touch0 TouchSensor { }
DEF Rotate0 RotateProto { nrChildren 7 rotateSpeed 1.00
}
DrawFile { scale 0.75 0.75 0.75 text "TensiNet" }
DEF Trans0 Transform
{ translation 3.00 0 0 children
[
Transform
{ translation 0 -0.01 1.70 children

```

```

[ ConnectionLine { point [ 0 0 0, -3.00 0.01 -1.70 ]
}
  DEF Touch1 TouchSensor { }
  DrawFile { scale 0.38 0.38 0.38 text "Project"
leafColor 1 0 0}
  ]
}
Transform
{ translation 0 -1.33 1.06 children
  [ ConnectionLine { point [ 0 0 0, -3.00 1.33 -1.06 ]
}
  DEF Touch2 TouchSensor { }
  DEF Rotate2 RotateProto { nrChildren 2 rotateSpeed
1.00 }
  DrawFile { scale 0.38 0.38 0.38 text "boom" }
  DEF Trans2 Transform
  { translation 3.00 0 0 children
  [ Transform
  {
  translation 0 -0.00 0.85 children
  [
  ConnectionLine { point [ 0 0 0, -3.00 0.00 -0.85
] }
  DEF Touch3 TouchSensor { }
  DrawFile { scale 0.25 0.25 0.25 text "peer" }
  ]
  }
  Transform
  { translation 0 0.00 -0.85 children
  [
  ConnectionLine
  { point [ 0 0 0, -3.00 -0.00 0.85 ] }
  DEF Touch4 TouchSensor { }
  DrawFile { scale 0.25 0.25 0.25 text "appel" }
  ]
  }
  ]
  }
  ]
}
Transform
{ translation 0 -1.66 -0.38 children
  [ ConnectionLine{ point [ 0 0 0, -3.00 1.66 0.38 ] }
  DEF Touch5 TouchSensor { }
  DrawFile { scale 0.38 0.38 0.38 text "file_3" }
  ]
  }
Transform
{ translation 0 -0.73 -1.53 children
  [ ConnectionLine { point [ 0 0 0, -3.00 0.73 1.53 ]
}
  DEF Touch6 TouchSensor { }
  DrawFile { scale 0.38 0.38 0.38 text "file_4" }
  ]
}

```

```

    }
    Transform
    { translation 0 0.74 -1.53 children
      [ ConnectionLine { point [ 0 0 0, -3.00 -0.74 1.53 ]
    }
      DEF Touch7 TouchSensor { }
      DrawFile { scale 0.38 0.38 0.38 text "file_9" }
    ]
  }
  Transform
  { translation 0 1.66 -0.38 children
    [ ConnectionLine { point [ 0 0 0, -3.00 -1.66 0.38 ]
  }
    DEF Touch8 TouchSensor { }
    DrawFile { scale 0.38 0.38 0.38 text "file_10" }
  ]
}
Transform
{ translation 0 1.33 1.06 children
  [ ConnectionLine { point [ 0 0 0, -3.00 -1.33 -1.06
1.00 ] }
    DEF Touch9 TouchSensor { }
    DEF Rotate9 RotateProto { nrChildren 6 rotateSpeed
1.00 }
    DrawFile { scale 0.38 0.38 0.38 text "dir_2" }
    DEF Trans9 Transform
    {
      translation 3.00 0 0 children
      [ Transform
        {
          translation 0 -0.00 0.85 children
          [ ConnectionLine {point [ 0 0 0, -3.00 0.00 -
0.85 ] }
            DEF Touch10 TouchSensor { }
            DrawFile { scale 0.25 0.25 0.25 text "file_1"
          }
        ]
      }
    }
    Transform
    {
      translation 0 -0.74 0.42 children
      [ ConnectionLine { point [ 0 0 0, -3.00 0.74 -
0.42 ] }
        DEF Touch11 TouchSensor { }
        DrawFile { scale 0.25 0.25 0.25 text "file_2"
      }
    ]
  }
  Transform
  {
    translation 0 -0.74 -0.43 children
    [ ConnectionLine { point [ 0 0 0, -3.00 0.74
0.43 ] }
      DEF Touch12 TouchSensor { }
    ]
  }
}

```

```

    DrawFile { scale 0.25 0.25 0.25 text "file_3"
  }
  ]
}
Transform
{
  translation 0 0.00 -0.85 children
  [ ConnectionLine { point [ 0 0 0, -3.00 -0.00
0.85 ] }
  DEF Touch13 TouchSensor { }
  DrawFile { scale 0.25 0.25 0.25 text "file_4"
}
  ]
}
Transform
{
  translation 0 0.74 -0.42 children
  [ ConnectionLine { point [ 0 0 0, -3.00 -0.74
0.42 ] }
  DEF Touch14 TouchSensor { }
  DrawFile { scale 0.25 0.25 0.25 text "file_5"
}
  ]
}
Transform
{
  translation 0 0.74 0.43 children
  [ ConnectionLine { point [ 0 0 0, -3.00 -0.74 -
0.43 ] }
  DEF Touch15 TouchSensor { }
  DrawFile { scale 0.25 0.25 0.25 text "file_6"
}
  ]
}
]
}
]
}
]
}
]
}
]
}
}

ROUTE Touch0.isActive TO Rotate0.aBool
ROUTE Rotate0.set_rotation TO Trans0.set_rotation
ROUTE Touch2.isActive TO Rotate2.aBool
ROUTE Rotate2.set_rotation TO Trans2.set_rotation
ROUTE Touch9.isActive TO Rotate9.aBool
ROUTE Rotate9.set_rotation TO Trans9.set_rotation

```