The Virtual Internet Gallery (TVIG)

3D visualization of a queryable art-database on the Internet

ANDREAS MUELLER[*], ERICH NEUHOLD[*]

tr-98-039

Abstract

The still rapidly growing Internet offers new ways to reach an increasing number of people in all areas of life. More and more companies take advantage of this fact by advertising and selling their products through this new electronic media. Art is a great example for using this new approach, because the visualization is the most important aspect and the physical presence of the exhibited object has just a secondary significance for the buying process, in contrary to other products (e.g. instruments, perfume, cars, etc.). This paper introduces an electronic service for galleries and artists to exhibit their artwork on the Internet easily and efficiently. The Virtual Internet Gallery (TVIG) utilizes a database to offer fast search functionality and performs a 3D visualization of the user's query result, applying VRML. Users, who are interested in the exhibited art, can contact the gallery or artist directly through the system.

[*] [anmuelle, neuhold]@darmstadt,gmd.de

TABLE OF FIGURES

1. Introduction

The Virtual Internet Gallery (TVIG) provides an electronic service for galleries and artists to exhibit their artwork on the Internet easily and efficiently.

Small galleries usually try to enlarge their clientele by selling printed catalogs presenting the work of artists they support. This method is rather successful with a significant number of paintings being sold in this manner. For example, interior designers wishing to decorate a public building (e.g. executive offices and hallways) can select paintings through catalogs rather than visiting the galleries individually. However, this procedure of selling art has its drawbacks: a lot of money and effort is involved in producing the catalogs and the artwork portrayed is fixed. Each time a gallery wishes to sell new or different artwork, the whole catalog has to be edited and produced again. Also, customers have to check many different galleries or their catalogs when they look for a specific piece of art.

Since the Internet is constantly growing with more and more people obtaining access to it, we believe that the process of buying art without visiting galleries can be improved through an electronic system. Many galleries and artists have finally recognized this trend and have begun to build their own Web sites. By examining such pages, we realized that most of them are just electronic versions of the ordinary printed catalogs that do not take further advantage of the new media.

The system developed in this project utilizes a database to be able to offer more and better features than a traditional catalog. The intention of the service is to mediate between the sellers of artwork (artists and galleries) and potential customers. Because an administrator handles parts of, or all technical procedures (e.g. entering describing information and scanning photographs of the paintings) this system considerably reduces the work to be done by a single gallery or artist to advertise their work while simplifying the selection process for the buyer. Such galleries and artists who have access to the Internet can also submit information about new offers online, which is reviewed by the administrator and added to the database.

Another advantage of electronic advertising is its relatively low cost when compared to conventional catalogs. Whereas the cost of production for a printed catalog increases with a growing distribution, the costs for the electronic gallery remain almost the same independently of the number of customers who access it. Exhibiting art in the virtual gallery may not cause the seller any additional effort, compared to the preparations that need to be done for a printed catalog, e.g. taking photographs of the paintings and writing the descriptions. Contrary to expectation, no technical knowledge is required to use our system since an administrator manages the system to reflect the personal decisions of galleries and artists on how their work is to be displayed.

Features like search engines enable the customer who wants to buy art to either search within a single gallery or the whole database. The query results are displayed in a dynamic virtual 3D gallery which can be explored freely. This produces a much more natural examination experience than what can be provided by two-dimensional illustrations. The structure of the virtual gallery building and the representation of the gallery itself can be configured by the buyer as well as the seller. In addition, visitors can directly contact the gallery or artist through the system and inform them of any special interest they have.

From a more technical point of view, the implementation can be seen as an attempt to combine relational "state of the art" databases with 3D visualization technology VRML[1]. A remote database can be accessed through the Internet and a dynamically generated VRML scene visualizes the query results in the user's Web browser.

1.1 The Internet and its problems for data[2] visualization

It has become easier and cheaper for the consumer to gain access to the Internet and, since this trend is going to, a broader public will be reachable through this media. This could be seen as a step towards the human freedom of speech [Raj97]. On the other hand, this creates cultural and technical problems. For example, every single day a huge amount of unstructured and semi-useful information is being produced and published on the Internet, leading directly into search and consistency issues.

Even when the appropriate information can be found on the Internet, it may be such a huge amount of data that it is difficult to keep oriented while navigating through the information space. In such a case, visualization is of great importance in order to structure the content in a meaningful way. Today, the World Wide Web (WWW) utilizes the majority of the Internet's resources. It originally displayed the information as plain, static and linked text pages. Later, it started to use fixed images. Nowadays, Web pages can be more active and interactive with the help of Java applets, and even 3D content can be integrated into Web pages using VRML. Currently, 3D visualization is becoming an important alternative for displaying complex information. This can be seen as a very natural development, since our daily experiences are based on the three-dimensional natural environment.

---

[1] VRML (Virtual Reality Modeling Language) is one of the main aspects of this paper and will be detailed introduced in Chapter 3

[2] The terms "data" and "information" are interchangeable throughout this paper

1.2 Databases

Data has always been a key corporate asset and this asset look is becoming intensified in the "Information Age". Companies have to store huge amounts of information about their employees and customers, as well as product documentation and catalogs. Formerly, all kind of information was archived in filing cabinets that were difficult to manage and update. Later, the data was stored in computer filesystems with the advantage of easier maintenance, less physical space and a faster and more efficient access. These first computerized record-keeping systems, though an improvement, still consisted of multiple files requiring considerable knowledge about the content and structure of each file. Early database management systems (DBMS) followed which required substantial programming for their use.

The development of relational[3] DBMSs created a breakthrough in information storage and access. Today's DBMSs can store all kind of data, for example, plain text, pictures, videos and other multimedia data. Especially in distributed environments with concurrent access, complicated problems, as for example, reservation systems for airlines have been solved. The huge and complex information spaces produced by the increasing processing power of new computer generations and the overall presence of the Internet require the use of databases in all kinds of business. New kinds of DBMSs, such as deductive and multidimensional databases, are developed to face these upcoming challenges (e.g. datawarehousing).

Our system is based on a relational "state-of-the-art" database since this kind of databases is the most commonly used today. This conventional technology is sufficient and powerful enough to store the information describing the paintings and to offer text-based search functionality. Almost all DBMS include a C-programming API and on the other side Java offers a generic database API that allows to access all kinds of databases of different vendors.

1.3 Overview

After this brief introduction, Chapter 2 will show the kind of design errors that can be found on existing art-related Web pages which motivate the need for new design and presentation concepts. Chapter 3 introduces the used technologies before the architecture and design of the implemented system are explained in Chapter 4. Chapter 5 then shows some implementation details and Chapter 6 describes the performance measurements of the over-all system. Finally, Chapter 7 provides an outlook on additional work that could be done in the future.

[3] The concept of relational databases will be introduced in detail in Chapter 3

2. Art And The Internet - Requirements

In this chapter, we will analyze what users should be able to expect when exploring and perceiving art collections through an electronic system and what kinds of technical and cognitive problems arise with publishing art in the Internet. In addition, it will be discussed, in detail, how efficient, user-friendly and easy it is to handle the exhibition of art on the Internet, using today's technologies. The results presented are based on [OL97], as well as our own research in the WWW. The authors of [OL97] postulate some basic requirements for art related electronic systems and describe what they really found (in 1997). Then it will be explored whether anything changed in the meantime based on our own experience. As a conclusion, we will propose some additional requirements for the success of such a system.

2.1 Problems with publishing art via electronic media

Due to technical limitations and psychology (missing physical presence of the art object), a virtual gallery will not be able to completely replace the experience of real art in the near future. First of all, limitations in the current technology of rasterdisplay and LCD devices lead to a modified perception of the original colors, especially in the case of highlights and shadows. Software algorithms can only simulate such effects, but they produce only approximate results. For example, dazzling effects cannot be achieved with the physical concepts of current computer screens. Another problem is, of course, the limited display-area. This aggravates the recognition and comparison of sizes. A zoom function should be available for very large paintings, to make details visible which are otherwise lost because of the resolution of the screen. The digitalization of the artwork has also some limitations such as the loss of the canvas structure when scanning paintings.

In spite of these difficulties, the exhibition of art on the Internet also offers many opportunities. In addition to the economical and technical advantages indicated in Chapter 1, there are some cultural effects. Distant artists and viewers (potential buyers) can communicate independent of temporal and spatial limitations.

2.2 Basic requirements

There is a minimum of basic requirements to make the experience of art through electronic media as close as possible to contemplating real paintings. [OL97] postulates the following three aspects which list only the basic needs of users and do not suggest any technical solutions:

a) The artwork itself should be easy to recognize.

b) Users of the system should be able to understand the context of the artwork and relations to other pieces of art.

c) Users should be able to interact with the object spontaneously and instinctively.

The first aspect is concerned with the display quality as described in Section 2.1. The work of art has to be displayed in appropriate size and users should be able to distinguish between paintings of different sizes. If possible, natural effects such as light and shadow should be simulated to make the experience more realistic.

The second aspect means that sufficient information about the artwork, the author, etc.[4] has to be provided so the object can be placed into a proper context. For example, the drawing style and technique, the size of the original painting, and additional information about the artist (if available) should be specified.

The third aspect postulates the freedom of viewers to change their point of view and to interact with the artwork in adequate ways, e.g., getting additional information by clicking on a painting. This aspect will be further discussed in Chapter 4.

In order to verify how far these demands have been fulfilled, all 253 Web sites of Yahoo's "Arts: Museums and Galleries" category were analyzed. The main measuring values were the quality of the images[5] and the amount and quality of the describing text, as well as the existence of additional features for user-interaction.

The results are more than disappointing: Referring [OL97], a considerable number of sites offer just a plain list of art objects without any images, or mostly plain images without textual description. Just nine percent tried, at least, to simulate the real experience of art by providing such features as controlling distance, direction, or speed of viewing in any way. Some sites present moving images, that can be controlled in a way similar to a VCR. There are possibilities to play, forward, rewind,

[4] In the following, we will refer to this kind of information describing the artwork as "metainformation"

[5] Unfortunately, the authors of [OL97] did not specify the parameters to define the 'quality of images'. In our context, 'quality of images' refers to such parameters as size, resolution and accuracy of colors.

and stop, but the set of available paintings is always predefined and fixed. Only five sites offered a Virtual Reality (VR) environment, which at that point of time was always static. A critical point for these more evolved presentations was the absence of any measurement for distance and direction, which allowed the user to get lost very easily. For the textual information just eight percent of the reviewed sites were rated as excellent in terms of quantity and quality.

As a general problem, it was observed that accents in the artists' names, countries and titles of paintings were missing or misspelled because of the predominant use of English on the Internet.

The unsatisfactory test results are concluded with the following sentence: "The experience of art by visiting these sites is so simple and insufficient that it cannot be compared with the real world at all."

2.3 Currently existing galleries and museums on the WWW

With today's widely propagated technologies like Java, VRML and database management systems, one could expect enhancements to the situation described in the previous subchapter, but actually no significant improvements to the former results can be found. Trying to review some references listed in [OL97], broken links or the described low-tech solutions are still predominant.

Yahoo's "Arts: Museums, Galleries, and Centers" category now has 779 entries now and is subdivided into 26 separate subcategories. This makes it even harder to find any useful links to galleries, because subgroups such as 'Literature', 'Architecture' and 'Dance' also belong to this category. Most of the reachable sites are still static 2D HTML pages. The few sites which try to use VRML have a fixed presentation set, are totally confusing and the performance is so bad in almost all cases that it is nearly impossible to get any acceptable results with a "state-of-the-art" graphics system of a standard PC and standard modem network connections.

2.4 Additional requirements

With respect to the above experiences, we postulate some further requirements in addition to the three basic needs for a virtual gallery as proposed earlier by [OL97]:

d) Users should face a consistent, easy to use system where it is possible to query for a wide range of art and art sources.

e) 3D visualization should be used to improve the exploration quality. VRML is a suitable technological basis, because it is designed for use on the WWW and offers an easy way to take advantage of features such as texture mapping and shading. Furthermore, it can be easily combined with Java.

f) The possibilities for user interaction should be improved. This will be exemplified by the system developed, after its architecture has been introduced in Section 1 of Chapter 3. We provide the use of a database to offer more powerful search functionality.

g) Users have to be treated as intelligent individuals. Therefore, it should be possible for them to configure how the query results will be presented. Visualization setups, predefined by the different galleries and selectable and adaptable by users would be very practical.

h) Apparently, it needs to be mentioned that average users cannot be expected to be the owner of a high-end workstation with a high-speed Internet connection. As far as possible, the system should offer tolerable performance on standard PCs or laptops with a simple modem connection.

3. The Applied Technologies

This chapter will briefly summarize the most important technologies that have been used to build our system[6]. However, it is just a short explanation of the history and current importance of each single technology. A more detailed discussion of particular aspects used in the implementation part of this project will follow in Chapter 5.

3.1 Databases

The main purpose of database management systems is to ensure the independence, integrity and consistency of a huge amount of stored data. Data independence means that the information has to be stored in its plain form, without any additional markup. This leads to application independence, because different applications can retrieve and use the same data without performing any further analyzing (e.g. parsing). To guarantee the integrity of the information, transactions for storing data into the system have to be atomic and protocolled. Atomic means that the single operations are indivisible and, therefore, the result is always defined. If an error occurs while such an atomic operation is executed, nothing is changed in the database. Using the protocols, most of the DBMS include a backup system which can also be configured to run as a scheduled backup. These precautions assure that no large amount of information gets lost because of technical problems. Consistency can be achieved by avoiding redundancy; otherwise applications have to take care of a correct and consistent update of the redundant information.

As described in Chapter 1, databases play an important role in all kinds of today's business. Just a few years ago, large databases could only be administrated on mainframe-based machines, which were unaffordable for a single person.

3.1.1 Distributed systems

With the growth of the Internet the idea of networked computing was brought into the foreground; today even small companies have their own internal networks. A situation known as "client/server" architecture is implicit in the use of the Internet. In such distributed systems, we find a few big, powerful and expensive machines (the servers). These servers offer services such as shared filesystems, printer or database access, as well as other software-based support to a large number of

[6] If readers is already familiar with any of the technologies introduced in the following, they may skip the corresponding section

less powerful and hence inexpensive machines (the clients). This principle of shared resources reduces the purchase and maintenance costs and lowers the administration effort at the client side. If the software at the client side can be run within a Web browser and, therefore, does not need any additional installation, it is called a "thin client architecture".

With today’s powerful, inexpensive workstations almost everybody can afford to set up a database server. Such a server can be accessed within Intranets and also through the Internet. Simple, but still powerful local databases, such as Microsoft’s Access, can even be run on low-end personal computers. To provide access through the Internet at least a DBMS in the range of Microsoft’s, Oracle’s, Informix’ or Sybase’s SQL-Server is required to provide a large scale permanently running service. Unfortunately filesystem-based lightweight databases like MS Access do not offer this possibility.

3.1.2 The relational model

After we have described the capabilities of databases in general one specific type of DBMS will be explained in more technical details. Relational databases are simply based on tables (which are described by columns and rows) and the relationships between the different tables. The data itself is represented inside single fields, uniquely identified by the corresponding row and column in a table. In this model m:n relationships can be very easily represented.

When there is no information stored in a field, it has the value "null". This is not a numerical value, but rather an indication that the value is still undefined. Different "views" to the information can be defined independent of the physical storage of the data in the system. A "view" is essentially a virtual table which represents the data in an alternative way. The common database methods can be applied to views as they are applied to ordinary tables. One of the most important database operations is the "join". It is used to gather and manipulate data from several tables. There are various types of joins which are all based on a cross product like combination of the rows of different tables as shown in Figure 1. In the basic "cross join" each single row of the first table is combined with all rows of the second table. More useful joins include the possibility to specify, which rows are combined. An "equal join", for example, combines only such rows where a shared column of both tables contains the same value.

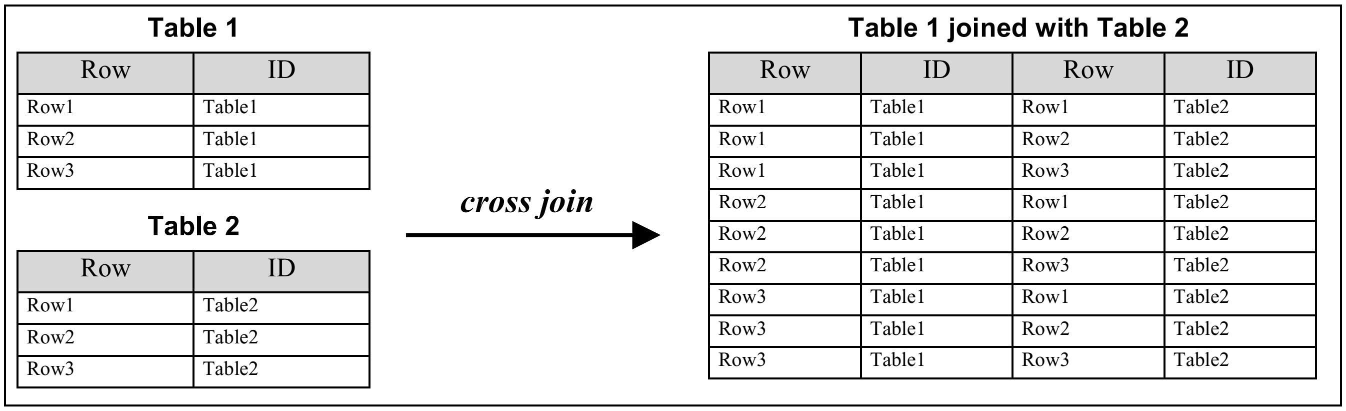


Table 1

Row	ID
Row1	Table1
Row2	Table1
Row3	Table1

Table 2

Row	ID
Row1	Table2
Row2	Table2
Row3	Table2

cross join

Table 1 joined with Table 2

Row	ID	Row	ID
Row1	Table1	Row1	Table2
Row1	Table1	Row2	Table2
Row1	Table1	Row3	Table2
Row2	Table1	Row1	Table2
Row2	Table1	Row2	Table2
Row2	Table1	Row3	Table2
Row3	Table1	Row1	Table2
Row3	Table1	Row2	Table2
Row3	Table1	Row3	Table2

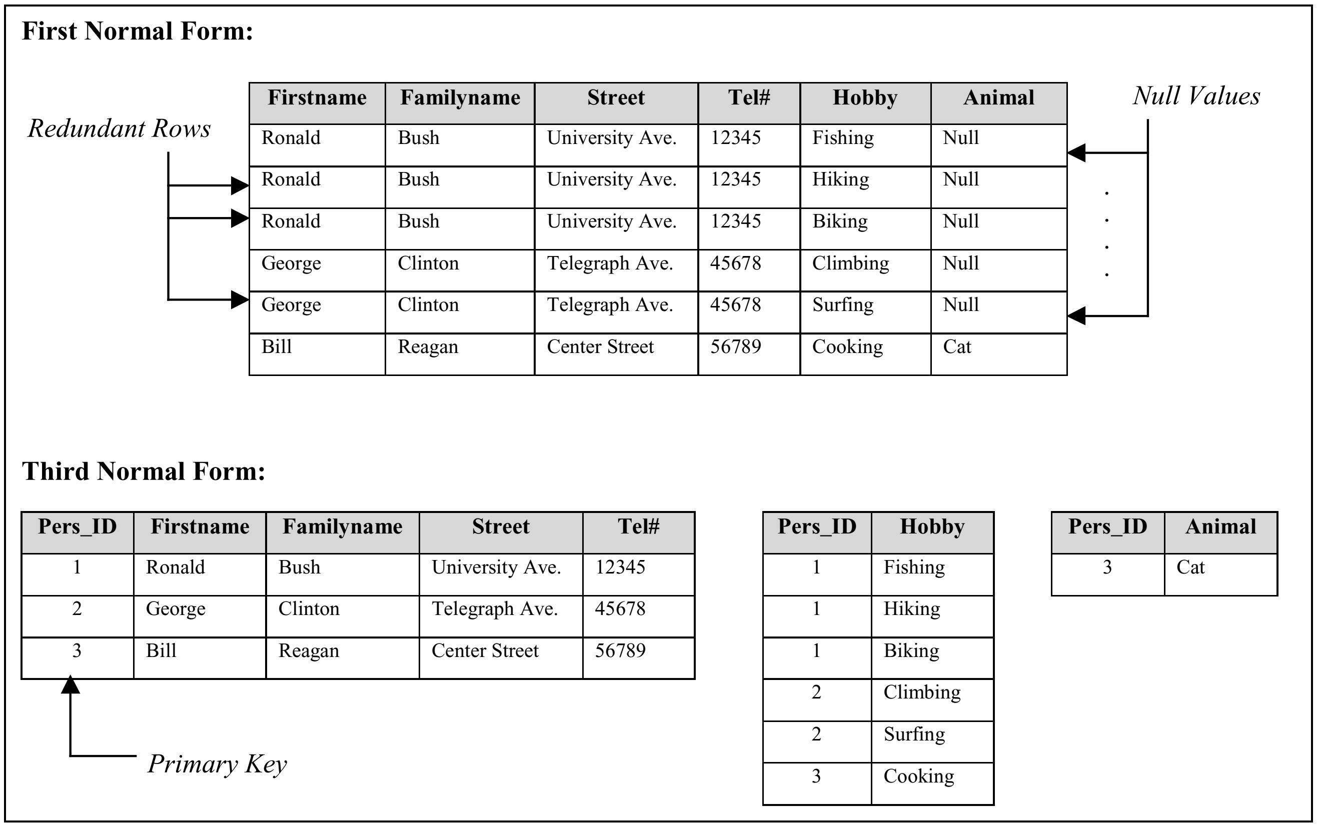
Figure 1: Example of "cross join"

One of the columns in each table has to hold unique information in order to distinguish between otherwise possibly equal rows. The identifiers in this column are called "Primary Keys". Specially marked columns can hold unique identification values of other tables which then are called "Foreign Keys". The description of how the data is split into the different tables is called the "Database Scheme" and the process of designing a "good" Database Scheme is known as "Normalization" (Figure 2). In the database context "good" means (referring to [Bu97]) that:

- there is no redundancy of information,
- as few null-values as possible have to be stored,
- no loss of unrelated information or access paths is produced by deleting database rows,
- no invalid information is produced by performing a join.

Different levels of normalization exist, but with a higher normal form the performance usually declines. This is because higher normalization produces a large number of very small tables and relations, and as a result the access paths extend and many more joins are needed to regain the original information. Therefore many database administrators abstain from using the higher normal forms in favor of a better performance.

The performance of a database scheme can be improved by applying index structures to often-accessed columns. This also improves the performance when sorting fields and can ensure referential integrity constraints.



First Normal Form:

Firstname	Familyname	Street	Tel#	Hobby	Animal
Ronald	Bush	University Ave.	12345	Fishing	Null
Ronald	Bush	University Ave.	12345	Hiking	Null
Ronald	Bush	University Ave.	12345	Biking	Null
George	Clinton	Telegraph Ave.	45678	Climbing	Null
George	Clinton	Telegraph Ave.	45678	Surfing	Null
Bill	Reagan	Center Street	56789	Cooking	Cat

Third Normal Form:

Pers_ID	Firstname	Familyname	Street	Tel#
1	Ronald	Bush	University Ave.	12345
2	George	Clinton	Telegraph Ave.	45678
3	Bill	Reagan	Center Street	56789

Pers_ID	Hobby
1	Fishing
1	Hiking
1	Biking
2	Climbing
2	Surfing
3	Cooking

Pers_ID	Animal
3	Cat

Figure 2: Example of a database scheme – Normalization

3.1.3 SQL

In the late 1970s SQL (Structured Query Language) was developed at the IBM Laboratories in San José. Its aim was to provide a single declarative language which would allow uniform access to relational databases. Therefore, it just describes which operations can be performed and not how the results are achieved by a particular database. Sets of rows can be processed instead of just one row at a time and automatic navigation through the data is provided. Because the programmer does not need to specify the access method to the data, it is easier to concentrate on obtaining the desired results. Most DBMSs use an internal optimizer to find the best way of accessing the data by taking advantage of existing indexes. The "Q" for Query gives a hint that SQL is used to formulate structured questions against a database. Beyond this obvious fact, SQL offers some additional features:

- Modifying the database’s structure, i.e. adding, deleting and altering tables,
- Changing system security settings,
- Adding user permissions on databases and tables,
- Updating the contents of a database, i.e. updating, adding, deleting rows of tables.

The most commonly used statement in SQL is the SELECT statement. This statement retrieves data from the database and returns it to the user. When such statements are embedded within the actual program code of any procedural language it is called "Embedded SQL". Using this mechanism, the program can completely control the database, including opening and closing connections. Such applications can provide an easy to use interface and the user can access a DBMS very comfortably without knowing any SQL syntax.

3.2 Java

Java is an object-oriented, platform-independent, multi-threaded, general-purpose programming language (or better environment), developed by Sun Microsystems Inc. The main purpose of Java was to simplify the world of network programming by providing platform independence and therefore simple portability of the code, without sacrificing system security.

Object orientation is the latest programming methodology and should not need any further explanations. Unfortunately, everybody tries to take advantage of this "hype" by using it for PR, whenever something can be described as an object in some way. For Java, object orientation means the basic principles of encapsulation, polymorphism, inheritance and dynamic binding. For further information concerning these keywords see any Java reference, e.g. [Eck98].

Besides the common object-oriented design goals, Java was intended to be simple, extensible, robust, secure, and portable. Simplicity was reached through the similarity to many former programming languages. This allows programmers who are already familiar with these other languages to learn it very easily. Extensibility is ensured by the consequent object-oriented design. Robustness was reached by using built-in memory management and exception handling. Security features are especially realized by enforcing restrictions on applet-programming. Since Java is a dynamically linked language, it is not compiled into the final binary code for the different platforms but into a machine-independent form, the "Byte-Code". This Byte-Code is interpreted by Java‘s "Virtual Machine" (VM) at runtime. A Java VM is available for all major platforms and even for handheld and other electronic devices.

Multithreading allows running several processes in parallel, which is very useful for many common tasks like providing a Graphical User Interface (GUI). Java supplies multithreading and goes even further by providing a package called "Advanced Windowing Toolkit" (AWT). This package allows developing applets and GUI‘s with minimal effort. Applets are Java programs which can be downloaded through a network environment and executed within a Web browser.

3.2.1 Accessing Databases through Java – JDBC

JDBC is a Java API for executing SQL statements. Although often thought of as standing for "Java Database Connectivity", it is actually not an acronym. The JDBC package allows the developer to write database applications by only using the pure Java API. The advantage is abstraction from the particular database and platform, i.e. the programmer only needs to develop one single application, which is able to run on virtually all platforms and can connect to a variety of databases.

In order to execute SQL statements and process the results, first, a database connection needs to be established. This is done through the so-called JDBC-drivers. There are four different types of these drivers used in two-tier and three-tier models.

The Type-1 drivers are called "JDBC-ODBC Bridge" and the Type-2 drivers are partly-Java drivers which use the database‘s native APIs (Figure 3). Both are designed to be used in a two-tier environment. This means that the applet or application connects directly to the database. If the database is located on another machine, it is a client/server configuration. In the case of these two types of JDBC drivers, the code for the connection and the database-dependent commands are translated into another syntax first, before they are executed at the database server. For the Type-1 drivers the syntax is converted into ODBC, the Microsoft C-API for database access. The Type-2 drivers directly access the databases native APIs which are also mostly written in C. But the JDBC programmer does not need to care about this fact, since the translation is totally transparent, and there is no difference at the JDBC-side when using different drivers.

Due to the use of C-APIs and the associated loading of binary code, the Type-1 and Type-2 drivers are limited to single platforms and they cannot be used for Internet access since ODBC needs to be set up by the operating system at the client side. As a result, these drivers are just useful for local database access or within an intranet where an administrator does the necessary ODBC-setup.

The Type-3 and Type-4 drivers are completely written in Java. These drivers translate the JDBC calls into a DBMS-independent net protocol, which is then converted to the particular DBMS syntax at the server. Since this process needs special knowledge about the different types of databases they are mostly created and provided by the database companies themselves.

The Type-3 drivers are meant to be used in a three-tier model (see Figure 4). This means that there is a middle tier of software, possibly on a third machine, which handles the different JDBC-calls by translating them into the specific DBMS-protocols. Then this server application forwards the SQL-statements to the DBMS and receives the results, passing them back to the client applications. This type of drivers is the most flexible solution, even if there is the drawback of installing the additional middle tier software.

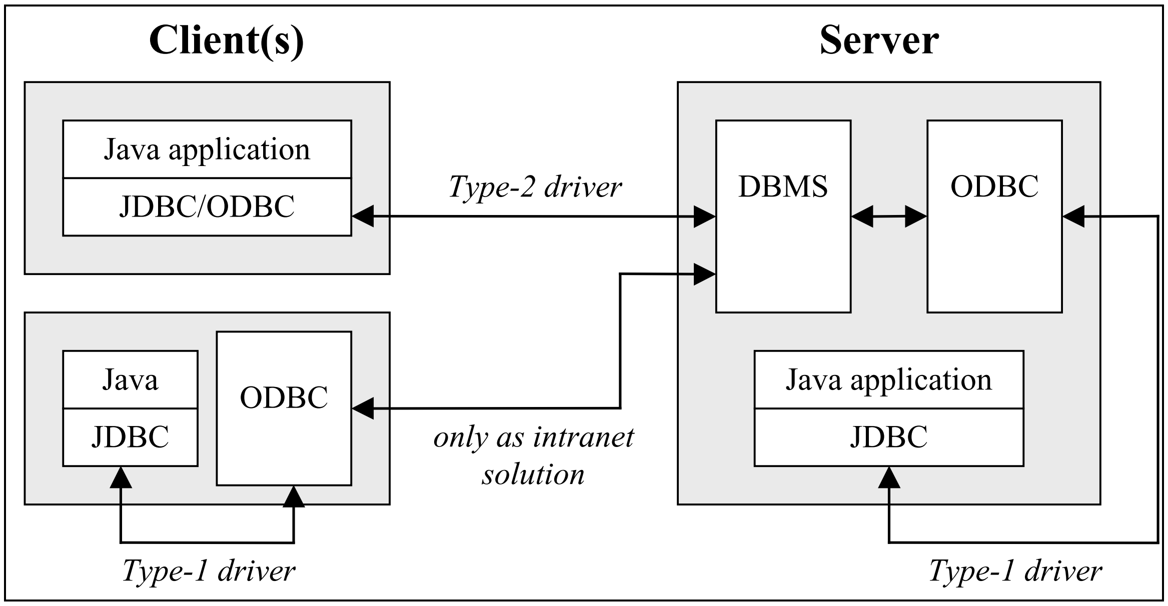


Figure 3: Two-tier model / JDBC drivers Type-1 and Type-2

The Type-4 drivers (two-tier) are the ideal solution, since they are pure Java, and therefore portable and do not need the additional software layer, like the Type-3 drivers. When a JDBC connection is established with an applet through the Internet, the drivers are downloaded to the client, together with the application classes. In doing so, the drivers themselves handle problems like being downloaded through a firewall. The programmer does not need to care about any network or database specific configurations.

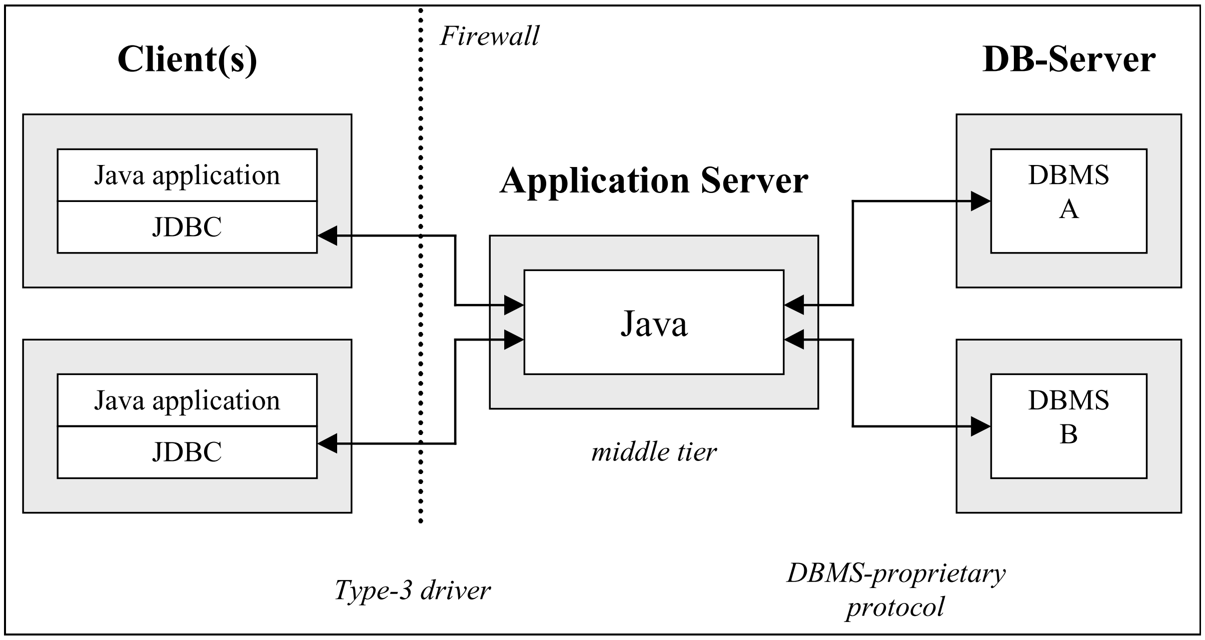


Figure 4: Three-tier model / JDBC driver Type-3

The above explanations show that the Type-3 and Type-4 drivers are the preferred solution, if available. The first two types should only be used, if a database vendor does not provide any appropriate drivers.

3.3 VRML

VRML, sometimes pronounced as "Vermal", is an acronym for Virtual Reality Modeling Language, although it is neither exactly "Virtual Reality", nor a "Modeling Language". Almost everything, from a little animated picture up to the million dollar simulation projects, is called "Virtual Reality" today. Even if this term is not well defined, it usually covers from simple up to far-reaching 3D experiences, such as the immersing behavior offered by head-mounted displays and 3D input devices. As a "Modeling Language" VRML should contain much richer geometric modeling primitives and mechanisms than it does.

At its very basis, VRML is simply a 3D interchange format based on Silicon Graphics "OpenInventor" file format. When VRML 1.0 was released in May 1995, it was intended to be a 3D graphics format for the WWW analogous to HTML. That is a simple, multi-platform, text-based scene description language for publishing 3D Web pages. This approach was inspired by the success of HTML, caused by its simplicity and interchangeability. The idea to transfer the successful design to a 3D graphics format was motivated by the fact that some information is best experienced three dimensionally, such as engineering and scientific visualization, educational information and architecture. The required intensive interaction, animation and user participation is beyond what is possible with the page- and text-based format HTML.

The main design goals for VRML are to:

- evolve the standard in single bigger steps at a time,
- keep it simple,
- standardize only such problems that are completely understood and reasonably solved,
- encourage experimentation and extensions at the frontier,
- not reinvent technologies that can be solved outside of VRML (e.g. HTTP).

If VRML does not seem to be progressive enough, it is because it is an open standard and the inventors did not want to create something completely new, but wanted to combine good and proven concepts based on already existing standards.

VRML 1.0 was only capable of describing static 3D scenes and the inventors had been aware, very early, of the need for further development through the feedback of many experimenting users.

3.3.1 The features of VRML 2.0

Possibilities to describe animations, interaction and programmable behavior have been added in the VRML 2.0 standard. Another big issue that is going to be approached in the next revision of VRML is to allow the description of shared multi-user worlds. Besides the above described design goals, new constraints were added:

- The VRML interpreting software should be able to process highly optimized results. Therefore features that seemed complicated to implement, were rejected. This step should ensure that many VRML browsers are being implemented as this will lead to better and faster implementations due to competition.
- It should be very easy to put files, created by several people, together to produce a new combined VRML scene. This characteristic is called composability.
- Scalability should be achieved on three different levels. Distributed worlds of any size can be produced, using composability in combination with the capabilities of the Internet. Such big scale worlds would need, of course, more processing power than we could provide today, but at least the mechanisms to handle these distributed worlds are included in the VRML 2.0 standard. VRML worlds should be executable on inexpensive machines, as well as powerful workstations (maybe with different levels of detail) and finally they should scale with network performance.

The basics:

VRML 2.0 consists of two fundamental parts: the scene description language and the mechanisms to create animations, user-interaction and the scene’s behavior. In the terminology of VRML, we find two basic data structures, the "Node" and the "Field". Nodes are the equivalent to Java classes and consist of a collection of fields. Fields can be compared with a variable and a method combined in one single entity. They are the basic properties of the 54 build-in nodes of VRML.

The logical structure of a VRML file is referred to as the "Scene Graph". It is an acyclic graph, where each node can contain other nodes in a parent-child relationship, except itself (see Figure 5). The fields in a parent node affect the properties of the child nodes. For example, the content of transformation nodes is summed up through the graph’s structure from the root to the leaves, which is called a hierarchical transformation. The 54 built-in nodes are categorized in nine different groups by their function. There are Grouping Nodes, Special Groups, Common Nodes, Sensors, Geometry Nodes, Geometric Properties, Appearance Nodes, Interpolators and Bindable Nodes. Grouping nodes are one of the most important categories because they contain a children field where all kinds of other nodes can be added. For all other nodes, the structure is fixed so that only certain types of nodes can be appended. The remaining node-types will be discussed in Chapter 5, as far as they are needed to understand the concepts realized in the implementation.

VRML also offers a mechanism to define a complete sub-graph as a new node. This node is identified by a name and can be reused somewhere else in the scene graph structure at any time (using the DEF/USE syntax). Furthermore, completely new types of nodes can be constructed using the prototyping concept. A prototype associates several different nodes and fields to create a new type-description, which can be instantiated with different content. A prototype has the type of the node that appears first in the definition and can therefore be used anywhere in the scene graph in a suitable context.

```
#VRML V2.0 utf8

Transform {                                                # Root Node
  children [
    Transform {                                            # First child
      translation 3 0 1
      children [
        Shape {
          geometry Sphere { radius 2.3 }
          appearance Appearance {
            material Material { diffuseColor 1 0 0 }
          }
        }
      ]
    }
    Transform {                                            # Second Child
      translation -2.4 .2 1
      rotation     0 1 1  .9
      children [
        Shape {
          geometry Box {}
          appearance Appearance {
            material Material { diffuseColor 0 0 1 }
          }
        }
      ]
    }
  ]
}                                                          # End of world
```

Figure 5: Example of VRML syntax

Animation (The event concept):

To animate a scene, information is passed around the scene by creating an event. Some fields are "event-driven", which means they can send and receive events. These special field types are "eventIn" and "eventOut" and they do not have any value associated with them, unlike the normal fields and "exposedFields". Their only meaning is to provide a way of communication between the other fields that hold the actual information. If an event driven field has a name starting with "set_" or ending with "_changed", it is automatically connected with the field which has the plain name without this prefix or suffix. When an event is sent to an eventIn, the value of the associated field is overwritten and when the value of a field has changed, the new value is sent through the connected eventOut. An "exposedField" combines the plain data field with both event-driven properties so that the eventIn and eventOut do not need to be explicitly written down in the definition of a node.

An event itself consists of two pieces of data. These are the data that has to be sent from one node to another and a time-stamp, which contains the time when the event was initially generated. This information can be used to treat incoming events in the correct sequence, which is very important for synchronization issues and real-time simulations.

Events can be routed between two specific fields or between any amount of source and destination fields. Even cyclic routing is allowed. If an event is forwarded to another field after it was received, the time-stamp remains the same. By comparing the time-stamps of generated and received events, cyclic routing can be recognized and treated in an appropriate way. In general, such situations should be avoided as they will slow down the execution speed of the whole system.

In combination with the scripting (to be explained later), the event concept would be sufficient for users to build their own animations. Nevertheless, VRML offers a set of different types of interpolator nodes (e.g. PositionInterpolator, OrientationInterpolator, etc.), which make it very easy to describe key-frame animations. Instead of calculating each single animation step, only few key values need to be defined and the node performs a linear interpolation. If a smooth motion is required, this behavior has to be programmed using a script node.

Interaction (sensor nodes):

The different sensor nodes allow users to interact with a VRML scene. When a sensor node occurs somewhere in the scene graph, the whole content of the substructure is sensitive in some way. In the case of a "touchSensor" node the whole visible geometry below the sensor node itself is touchable. As a result, an event is generated when users clicks on any part of this geometry, and this event can be routed to another node to trigger any behavior.

In addition to the "touchSensor" there are other sensor nodes like the "timeSensor" node. This node is aware of elapsing time, i.e. it can produce events in regular time intervals. This possibility is used combined with the interpolator nodes to produce a synchronized animation.

3.3.2 Controling VRML behavior (Scripting with JavaScript and Java)

With the event concept and the interpolator, as well as the sensor nodes, it is possible to create interactive animated and, therefore, dynamic VRML scenes, but without decision logic and state management this can still not be characterized as behavior. The scripting concept, already mentioned, enables the world author to program any desired behavior within a scene. For this purpose, VRML browsers have to support at least one script or programming language. Due to the design concept of not reinventing already solved problems, no new programming or scripting language was introduced. Instead, VRML provides an interface mechanism to existing languages, the "Script Node". There is no consensus on which language should be chosen as the standard for VRML scripting, but at the moment there are official language bindings for Java, JavaScript, and a subset of JavaScript, known as VRMLScript.

Script nodes allow the world author to insert logic in the middle of an event cascade. They are activated when they receive an event, and, therefore, perform calculations and logic decisions on the received data and produce another event as a result. Furthermore, they can keep track of information between the executions, i.e. they are capable to manage an internal state over time. The incoming events are handled in time-stamp order and the outgoing events have the same time-stamps as the triggering ones, which means that the script execution does not consume any time from a conceptual point of view.

The script node consists of three required fields. The author can define any amount of additional fields to determine the functionality of the script. The three mandatory fields are:

- mustEvaluate – This field takes care of performance concerns. The default value is "false", which allows the browser to delay the script execution until the results are needed elsewhere in the scene. If it is required that the result has to be produced at once, when the script is triggered by an incoming event, the mustEvaluate value can be set to "true".
- directOutput – If this value is "true" the script code can be used to affect other nodes of the scene, directly. This is only possible when another node is visible within the script node, i.e. at least one field of the script holds a reference which can, for example, be realized through an eventIn.
- url – In contrast to the two above field, the third one has no default value, and, therefore, it is required to specify one. The "url" field either points to a file (including the compiled code), or contains the inline code to be used for the script. This could be, for example, a Java class file or inline JavaScript code. Alternative locations can be specified, separated by a comma, to guarantee the desired behavior even when the network connection to the original server is corrupted while the world is being explored.

With the functionality of the script node, all kinds of behavior can be assigned to the objects in the scene. Already with the interpolator nodes, parts of the world can be moved around, triggered by user actions through sensor nodes. Now even more complex situations can be simulated. For example, a door which opens only when a correct code is entered can be realized with the new capabilities to hold information and perform comparisons.

There is still one thing missing to make the worlds completely dynamic. With all of the above features, it is merely possible to change the content of the fields in a fixed scene graph. One can for example easily change the color of an object, but, so far, we described no way to alter the scene graph structure itself, i.e. remove or add complete objects to the scene. The internal scripting interface for Java and JavaScript offers some methods to perform exactly these actions. Unfortunately, the internal scripting mechanism is not powerful enough, when a complete world should be created dynamically from within a script node, especially when communication with supporting processes is required. Such supporting processes would be necessary to achieve network or database access, which cannot be handled within the VRML browser. In such a case the External Authoring Interface, described next, is the better choice.

3.3.3 The External Authoring Interface (EAI)

The External Authoring Interface is a Java API that can be used by an applet to control a VRML browser and its content. The EAI is much more powerful than the internal scripting interface because it does not depend on being executed within the VRML browser and is therefore not subject to the extended security features of these browsers. Moreover, the EAI makes it much easier to create a user interface which can be used to control the VRML scene or to extract information from the scene and display it in an applet.

Such features are desirable because the only accepted input device for current VRML browsers is the mouse, and the browsers do not offer any plain text output. As long as the browsers determine the navigation style and as long as the 3D features are not sufficient to replace the common user interfaces (of Web browsers and applications), the use of VRML does only make sense in the context of an integrated interface. A persuading virtual reality interface would, for example, include a virtual person to guide users through the world, reacting to spontaneous and complex questions in an intuitive way. This would open a completely free interaction between the users and the software, similar to our daily experience of grasping information in the dialogue with a "real" person. Of course, this cannot be realized with the processing power of today’s personal computers and, therefore, we still have to rely on conventional user interface concepts and can only transfer parts of it, step by step, into 3D equivalents. As long as the above aim is not reached, new concepts require a learning phase for users to familiarize themselves with the new features. This means that the integration of known concepts with new, easy and intuitive features will provide the best user acceptance.

In the following, we will focus on the methods of the EAI, which allow us to establish a connection between a Java applet and the VRML browser and to undertake modifications of the scene graph structure.

Getting a reference to the browser and to nodes

Because the EAI classes are used as part of an applet, they cannot exist on their own and need a reference to the VRML browser in order to achieve any effects on the browser's content (e.g. adding or removing nodes from the scene). The static EAI method `Browser.getBrowser()`returns a reference to the browser instance that has to exist in the frame of the Web page where the applet calls the method (a technical problem that occurs using this EAI method will be discussed in Chapter 5).

To read and modify properties from the scene, a reference to the respective node is required. In contrast to the internal scripting interface, such a reference is not automatically available because the executed code does not belong to the scene and has no information about any nodes. By calling the method `getNode(String Name)` in the already requested browser instance, a reference to the corresponding node is returned. This requires that the node has been named in the original world file using the DEF syntax. Once the program using the EAI has a reference to a node, all its fields, including the "event-driven" ones, can be accessed similar to internal scripting.

If this was the only facility to keep track of the events that are produced in the scene, the applet would have to utilize a Java thread, always checking the "event-driven" fields of the particular nodes at regular intervals of time. Because this "busy waiting" is a bad design and leads to a reduced performance, we use the callback concept of Java to monitor the events that are produced. For this task, it is important that the callback method of an `EventOutObserver` class can be assigned to any number of eventOut fields. As soon as an event is generated and sent through an observed eventOut field, the associated callback method will be called. Since more than one field can refer to the same `EventOutObserver`, the `callback(EventOut value, double timeStamp, Object data)` method has a third argument besides the event itself and the time-stamp. By analyzing the third argument the callback method can distinguish between the different possible sources and react in the proper way.

Adding and removing nodes

To alter the scene graph by adding completely new geometry, first the internal representation that will be understood by the browser has to be produced from a string composed of the usual VRML syntax. This VRML parsing is supported by the `createVrmlFromString(String VrmlSyntax)` method of the `Browser` class. After the browser has parsed the string, the corresponding graph structure is returned, represented by one or more objects of the EAI's `Node` class. The parsing process itself does not display the new geometry within the browser. For this, the generated Node structure has to be passed to the "addChildren" eventIn of the parents Node. Every group node that can function as a parent for other nodes has the event "addChildren" and "removeChildren". Utilizing these two events, new children can be added to and old ones removed from existing nodes, passing the reference of the new (parsed) substructure to them.

In addition, the `Browser` class offers the `methods addRoute(sourceNode, eventOut, destNode, eventIn)` and `deleteRoute(…)` that can be deployed to generate and remove routes in the scene dynamically.

Equipped with all of the above features, it is possible to generate a totally dynamic, interactive and user dependent VRML world.

4. Our Approach – The Virtual Internet Gallery

After we have been discussing the basics of all the required technologies in Chapter 3, we can now focus our attention on a conceptual overview of our system “The Virtual Internet Gallery (TVIG)” and find out how the different parts are integrated to one single application.

4.1 The architecture of TVIG

Because TVIG should be easy to install, we wanted to take advantage of the existing Internet infrastructure and the protocols which belong to it. Therefore, we did not build a program that has to be installed permanently on the client machine. Instead, we developed a set of Java classes which are downloaded through the Internet and executed within the Web browser on the client side (thin client architecture). The only installation effort that appears on the client side is to download and setup a VRML plugin, which extends the Web browser with the capability to display VRML scenes. But even this expenditure will vanish in the near future because browser vendors plan to integrate such VRML plugins in their next product generation. The main advantage of the thin client architecture is that users do not need to download and install new versions of an application. As soon as a new version is placed on the server, all users automatically have access to the latest changes.

The database server is the core of the system. All system- and user-related information is stored and managed in this single server. This fact could be seen as the weak point of the system, since no user will be able to access the application when the server machine is down due to technical problems. On the other hand, consistency issues would arise with the distribution of information across several databases. Integrity constraints (recovery and backup mechanisms) of the DBMS will ensure that the database services are highly reliable and permanently working.

Supplementary to the DBMS, the server machine has to function as a WWW server. The Web server provides the HTML pages that embed the applications functionality, and holds the applets which, in turn, are downloaded together with the JDBC drivers when a user accesses the system. In addition, the images to be displayed are placed in the Web servers file directory instead of within the database. Although the pictures could also be managed by the database system, only the paths to the images are stored in the database. This is an important design decision and will be further explained in the context of the VRML plugin and its limitations.

As usual, the application's applets and Type-4 JDBC drivers are downloaded from the server and interpreted by the browser's integrated Virtual Machine. The required Java classes are loaded by the Virtual Machines class-loader, on demand. Therefore, the VM can decide whether the classes need to be downloaded via the network, as in the case of the JDBC driver classes, or whether they are to be loaded at the local system (client), as in the case of the EAI's classes that are delivered with the VRML plugin. This decision is made when the first instance of a new class is created.

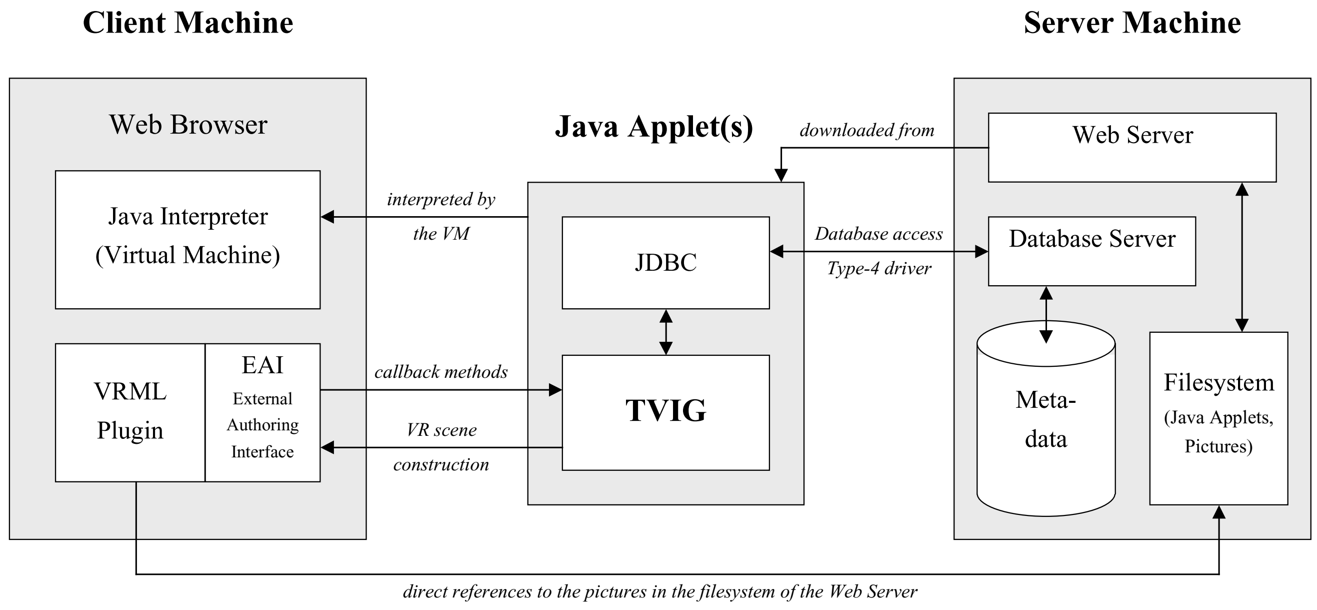

Client Machine
Server Machine
Web Browser
Java Interpreter (Virtual Machine)
VRML Plugin
EAI External Authoring Interface
Java Applet(s)
JDBC
TVIG
interpreted by the VM
callback methods
VR scene construction
downloaded from
Database access Type-4 driver
Web Server
Database Server
Meta-data
Filesystem (Java Applets, Pictures)
direct references to the pictures in the filesystem of the Web Server

Figure 6: Architectural scheme of TVIG

The application's classes have access to the database content via the JDBC drivers and utilize the EAI to construct the VRML scene, as described in Subchapter 3.3.3. The specification of the EAI does not allow writing image-data for textures directly into the scene, but needs to specify an URL from where the image-data can be read. Thus, it would be a disadvantage to manage the images in the database, instead of just storing them in the Web servers file system. The image-data would have to be read from the database and would have to be written back into a file directory on the server, where the VRML plugin would be able to find and download it again.

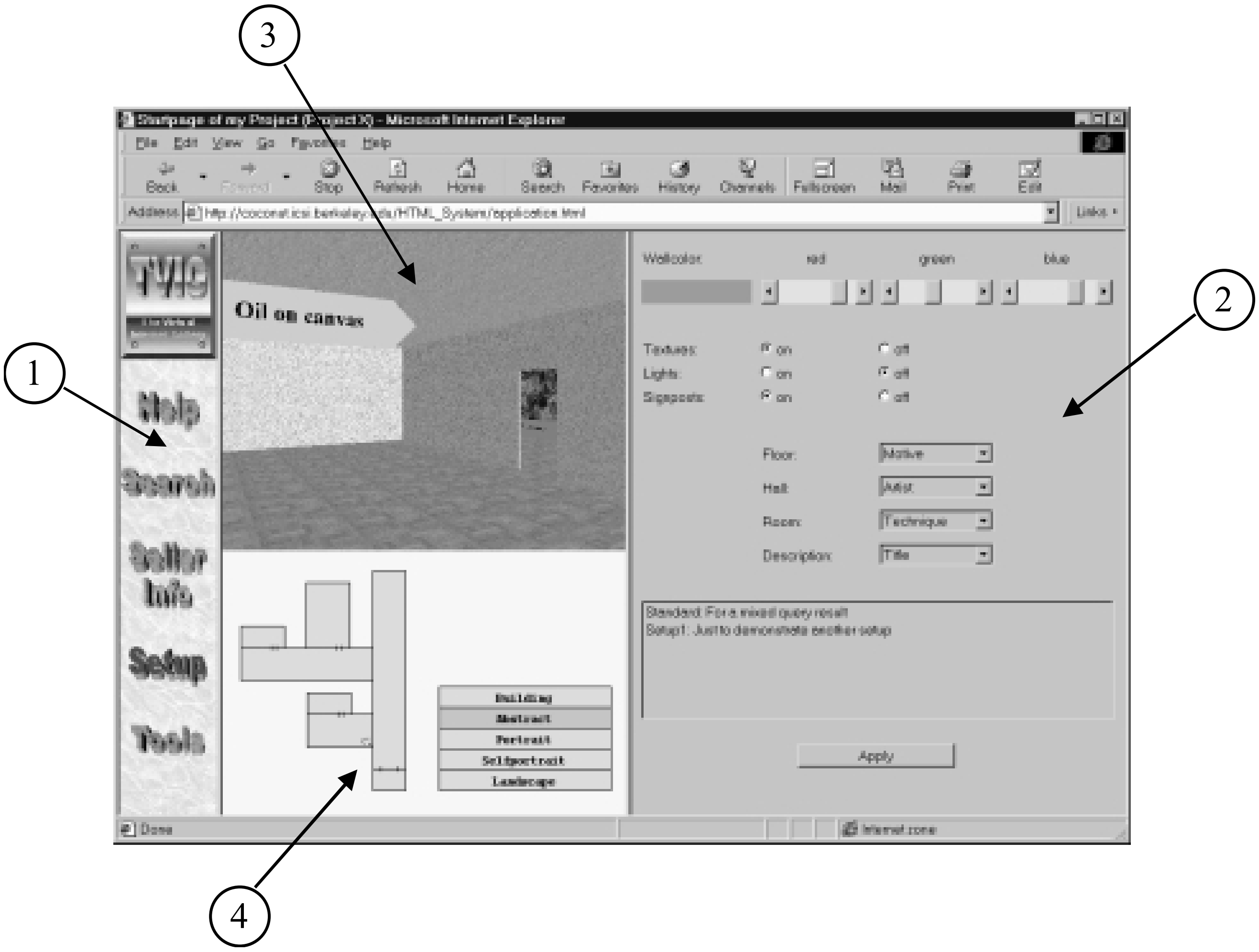
4.1.1 The user interface design

HTML already offers a good concept to guide users, i.e., the link hypertext mechanism. Single words and whole expressions can be marked as links. These marked words are connected with other positions in the same document, or even in a document that is placed on another server. When users click on such a word, they follow the link and the interrelated document will be displayed in the Web Browser. This mechanism, of course, interconnects the huge amount of HTML documents that are placed on servers, all over the world, and it inspired the name "World Wide Web".

When this possibility is used properly, it can guide the novice as well as an experienced person. When some users who read an HTML document, e.g., some technical brochure, do not understand some expressions they may follow the links to explanatory information, while others who are already familiar with the topic can read on without wasting time to read about the things that are already known to them. Unfortunately, authors of Web documents frequently misuse this great opportunity. Some pages include so many links that they disturb the reading process, especially when they lead to only slightly related information. By following such links, users are directed away from the actual points of interest and sometimes it is even difficult to navigate back to the original document.

Apart from this link mechanism, pure HTML pages cannot be seen as a user interface. To allow further interaction, a user interface has to provide some possibilities for user input (e.g. text-fields), for selections (e.g. choice boxes, lists and radio buttons), and for trigger actions (e.g. buttons). When the demand for interactive Web pages increased, JavaScript was introduced and provided some of the above features. By embedding Java applets in the Web pages, all these possibilities and even more advanced user interface elements, like progress-bars and menus, can be integrated, utilizing a Java package that supports "Graphical User Interface" (GUI) design (the AWT-package).

We used a combination of these traditional UI items (Java) and the VRML interface in order to provide a richer choice of interactions. The main aim was to offer users a structured and easy to manage interface, instead of overwhelming them with a huge amount of windows and options. In our system, users will always see only such options, that fit the immediate situation, and in addition, a context sensitive online help will guide the user.


3
2
1
4
Startpage of my Project (Project X) - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address http://coconut.icsi.berkeley.edu/HTML_System/application.html
Links
TVIG
Help
Search
Seller Info
Setup
Tools
Oil on canvas
Building
Abstract
Portrait
Selfportrait
Landscape
Apply
Done
Internet zone

Wallcolor: red green blue

Textures: [✓ on] off
Lights: on [✓ off]
Signposts: [✓ on] off

Floor: Motive
Hall: Artist
Room: Technique
Description: Title

Standard: For a mixed query result
Setup1: Just to demonstrate another setup

Figure 7: Screenshot of TVIG

Figure 7 illustrates how the interface (GUI) appears to users and what kind of features it actually offers. Point (1) refers to the main menu bar which is static and remains at this position offering the basic options that can be selected at any point of time. The options are the context sensitive help, the query interface which causes the scene construction and will be explained in detail later, the seller information, the setup tool to configure the scene construction (actually selected in the Figure), and a set of tools that can only be used by registered sellers to manage their offers. The frame marked (2) changes its content dependent on the selected option, whereas (3) and (4) always display the graphical representation of the gallery.

The scene construction setup

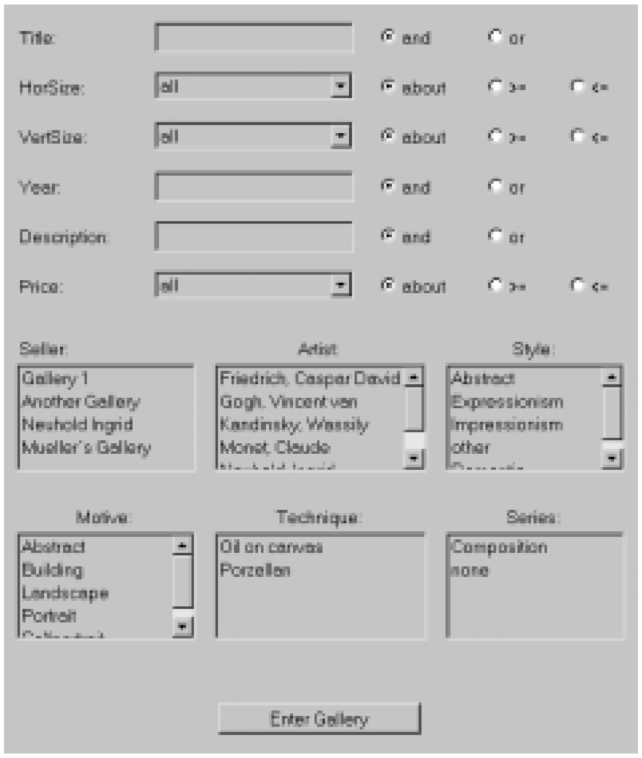
In Figure 7, the tool that can be used to configure the scene construction is currently active and can be found in the area marked (2). With the sliders labeled red, green and blue the color of the walls in the exhibition rooms can be specified. Below these sliders, additional textures, lights and signs (for orientation in the 3D scene) can be turned on and off. The usability of these features depends heavily on users' graphics hardware. In particular the signs in the scene require 3D-accelerator hardware, because they produce a large amount of text. Displaying text is a big performance problem for VRML browsers, since it has to be rendered through hundreds of polygons, to convey the impression of real text and make it readable. Users can experiment with different configurations to find out which setup produces the best graphical output with acceptable performance.

With the choice boxes in the central area of the tool, it can be defined in which way the query result will be mapped to the buildings representation. How this is done on a technical level, will be explained in Subchapter 5.2.2, which describes the scene construction mechanism. The obvious structure of a building (i.e. floors, halls and rooms) can be exploited to project some selected categories of the information, that describes a painting, on it. For example, different artists can be exhibited on various floors of the building, and the artwork of each single artist is sub-structured by the motives of the paintings on different halls and the painting techniques within the single rooms. When the query results are displayed in such a structured way, it is much easier for users to get a general idea of the retrieved data and to keep oriented while navigating through it. Nevertheless, some knowledge about the database scheme and the current population of the database is required to adjust this mapping configuration in such a way that it leads to a reasonable distribution of the paintings. Therefore, users can select from a list of settings which can be predefined by the galleries and the artists.

The query interface

After users selected the desired configuration for the scene construction and adapted it for their particular requirements, or when they are satisfied with the standard setup, they can select the "Search" option in the main menu to specify a query that will lead to the construction of the gallery.

The query interface (shown in Figure 8) is generated automatically, based on the database scheme of the database table that stores the information describing the exhibited objects (metainformation). Now, we will describe the possible query types and how they are graphically represented in this tool. All metainformation categories that are defined as queryable (some categories may be preferred not to be queryable by users, e.g., the price) have a related input item in the graphical interface. The applet will generate the corresponding SQL syntax for the specified query, whereby all the sub-queries of the input items are combined with a logical "and". Therefore, the tool can be seen as a filter that restricts the complete database content to the part requested by the users.



Title: [✓ and] or

HorSize: all [✓ about] >= <=

VertSize: all [✓ about] >= <=

Year: [✓ and] or

Description: [✓ and] or

Price: all [✓ about] >= <=

Seller: Gallery 1, Another Gallery, Neuhold Ingrid, Mueller's Gallery

Artist: Friedrich, Caspar David; Gogh, Vincent van; Kandinsky, Wassily; Monet, Claude

Style: Abstract, Expressionism, Impressionism, other

Motive: Abstract, Building, Landscape, Portrait

Technique: Oil on canvas, Porzellan

Series: Composition, none

Figure 8: Screenshot of the query interface

The first and simplest query type is named "free". When a category has this type it is represented as a plain text field, labeled with its identifier and without any further options. Users can then enter any single keyword in this field, and when the query process is started, the applet will try to match this expression exactly with the database content of the column which is specified by the identifier. This query type is the least powerful and requires very specific information about what users are looking for. An example for the use of this type, is an electronic telephone book where users knows an exact name and want to find the corresponding telephone number, or vice versa. Such a search type is most sensible, when one has mostly unique database entries.

The second type is called "search" and this name refers to a search within the fields of the database. The graphical representation is the same as for the first type with the additional option of the logical combinations "and" and "or" (see "Title", "Year" and "Description" in Figure 8). Users can enter a single word, as for example, "man", and the result will include all entries of the related database column which contains "man" as a sub-pattern, e.g. "policeman". Furthermore, users can enter several separated words and choose one of the possible combination types. If, for example, "police" and "man" was entered, a possible result would be the sentence "The police arrested a young man" as well as the expression "policeman".

The type "range" is used for non-discrete data, which fits best to numeric values. Users could for example be looking for a painting, which should be not bigger than 120x80 cm. Based on some additional values about the limits of the range and the size of the steps a choice-box is generated that offers a list of values from which users can select. With the additional check buttons on the right side it can be decided, if the resulting values should be bigger, smaller or about the same size as the value, that had been chosen.

The last two types are "select" and "multiselect" and they differ only slightly. These types should be used if there are not many possible values and the users does not know, which values could possibly exist in this column. The graphical representation is in both cases a list box, with the only difference, that for the "multiselect" type more than one value can be chosen. When an offer is added to the system that has a not already existing value for a "select" or "multiselect" category, this information is stored in a special index so that the list of selectable entries does not need to be recreated, each time the query interface is constructed.

After users have filled out the query form according to their interest, they just need to press the "Enter Gallery" button and the corresponding gallery will be constructed, based on the results of the query and the setup for the mapping to the scene. (The technical mechanism will be described in Subchapter 5.2.1)

4.1.2 The 3D interface

The 3D-style interface of VRML is only used in situations where it clearly offers an advantage over the conventional two dimensional user interfaces. This is, e.g., the case for displaying paintings and pictures of art objects, since it is an intuitive and natural way of exploring art and provides users a sense of depth and perspective. Referring to [CDTG97] spatial user interfaces are beneficial due to the natural human capacities for dealing with objects in space, behaving in space and attaching meaning to spatial dimensions, and the intrinsic memorability of spatially arranged information. Therefore 3D visualization offers natural access to complex ideas and concepts and promotes a quicker path to understanding. Especially our scenario with a large amount of highly structured data benefits from a 3D representation. The exhibition objects are described through a fixed database scheme which provides us with the opportunity to map the categories of this metainformation directly to 3D metaphors in the VRML scene (e.g. floors, halls and rooms). In this way we make the structure of the information visually explicit, by transforming relationships into spatial distances.

Two-dimensional GUIs have the problem of spatial limitation because the screen serves as a metaphor for a desktop. Therefore a large amount of information cannot be arranged in a reasonable way to support users who are browsing this information. It is very hard to gain a sense of the position in the information space because there is no higher-level overview representation of the information. On the other side, 2D GUIs can offer clear and simple layouts which are very useful for applications that display text or form elements as our query interface. In such cases a 3D interface would be "over-kill". Nevertheless, VRML people currently think about allowing the use of Java applets as Textures in the world. This would preserve the 2D user interface concepts and integrate them into the VRML scenes and is much more intuitive than integrating 3D spaces into plain Web pages.

The VRML plugin

Point (3) of Figure 7 marks the area of the VRML plugin where the gallery is displayed after it was constructed as a result of the database query. When users move the mouse over this region, the mouse pointer changes its shape, and the mouse control is transformed into a movement in the VRML scene. This control is predefined by the plugin, and the functionality is similar for the different available VRML browsers. When the left button is pressed down and held in this position, the following mouse-movement is translated into a movement in the scene, relative to the point where the button was pressed down, until the button is released again. When the mouse points at a touch sensitive object, the shape of the mouse pointer changes again (a circle appears around it). This is just a sign, that the referred object is touchable but does not trigger the action. In order to trigger the connected action, the left button has to be pressed and released while the mouse points at the concerned object.

One problem with the VRML plugin is the build-in navigation mechanism for the mouse which cannot be adapted in any way. For inexperienced users, this can cause difficulties when walking around in the scene. One possibility to escape the plugin’s interface constraint for fixed navigation is to take advantage of the programmable behavior and insert some own navigation facilities into the scene. Because users are expected to have some problems with the navigation in the beginning, for example, to position themselves in front of a painting, we offer the possibility to click on the title text below the painting. This will automatically move users in front of the work of their interest. Clicking at an image triggers another in-scene action. It will cause the browser to display the detailed information about the work of interest as HTML.

The overview maps

An additional approach to relieve navigational limitations of the VRML browser is the hybrid user interface that has been designed to support users with additional features, through Java applets. The applet in the area marked with (4) in Figure 7 displays a 2D overview-map of the building from the top (floor plan) and the whole building from the side (the different floors). These maps can be used as an additional facility for navigation by clicking on any position. As a result, users will immediately be placed in the new position in the scene.

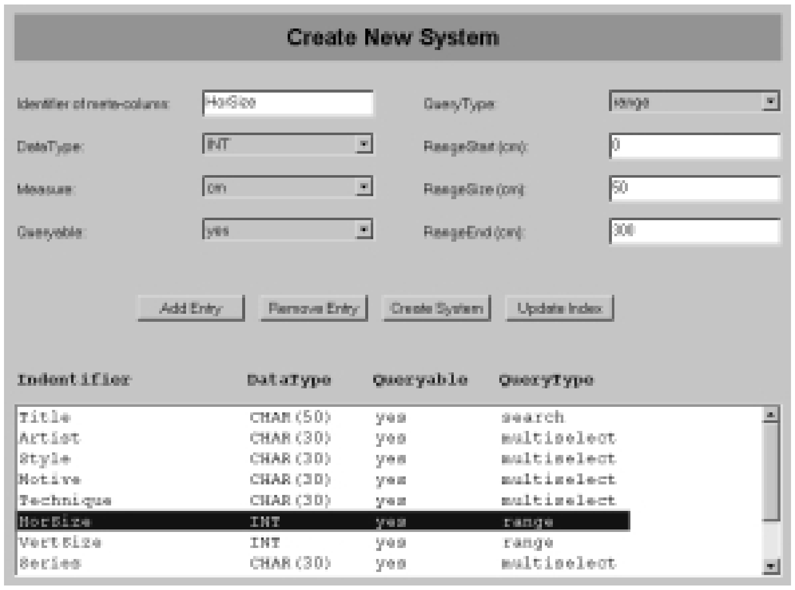
Besides this navigation purpose, the maps can, of course, be used for orientation. A small arrow on the map which is updated whenever users move in the scene, marks the users' position and orientation. The maps do not only show the spatial arrangement, but do also provide information about the content of the geometry (i.e. room, hall and floor) when users moves the mouse over them.

4.2 Administration Tools

As we now know, how the system's interface appears to the users and how they can interact with the system, it needs to be explained in which way new information is stored into the database. In general, we could enable every single user of the system to manipulate the database content, i.e., register as a new seller, add or delete offers, etc. But, as in every shared environment, this would lead into uncontrolled chaos, and, therefore we need some authority which monitors applications of new sellers and checks if the describing information about an offer is sufficient and correct. This authority are the administrators who are supported by tools, which will be described in the following.

4.2.1 The metainformation management

The whole system was designed to allow the easy exhibition of other objects besides 2D paintings and images of 3D art objects. The mechanism to construct and display the scene in the VRML browser will stay the same and can be used to show, for example, videos or objects which are modeled with VRML themselves. This will be further discussed in Chapter 7. The part that will experience the biggest changes, when other subject areas are to be exhibited, is the metainformation and therefore the underlying database scheme. If, for example, photographs are to be exhibited, instead of paintings, columns for the used lens, the exposure time and sensibility of the film would be needed, instead of, e.g., the painting technique. To guarantee the extensibility of the system, the database scheme can be set up for different scenarios. The specification of the database scheme is only required once for each new system and has to be performed by the administrators, using the tool shown in Figure 9.



Identifier of meta-column:	HorSize	QueryType:	range
DataType:	INT	RangeStart (cm):	0
Measure:	cm	RangeSize (cm):	50
Queryable:	yes	RangeEnd (cm):	300

Indentifier	DataType	Queryable	QueryType
Title	CHAR(50)	yes	search
Artist	CHAR(30)	yes	multiselect
Style	CHAR(30)	yes	multiselect
Motive	CHAR(30)	yes	multiselect
Technique	CHAR(30)	yes	multiselect
HorSize	INT	yes	range
VertSize	INT	yes	range
Series	CHAR(30)	yes	multiselect

Figure 9: Screenshot of the "Create New System" tool

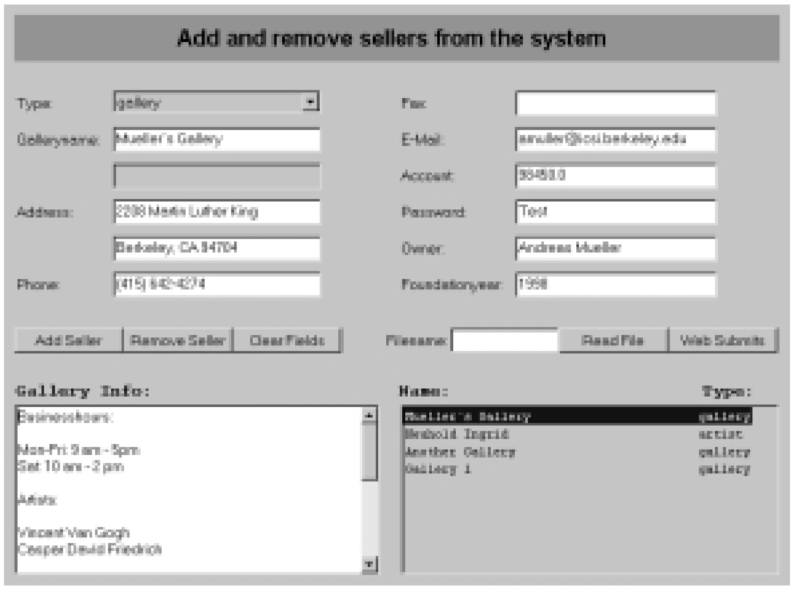
With the "Create New System" tool, administrators define all categories that are required to sufficiently describe the objects to be exhibited. This list of defined categories (as they can be seen at the lower part of the tool) is stored in a separate database table and will be used to generate the table that will finally hold the actual metainformation for all the registered exhibition objects. Besides providing a unique name (Identifier) for each category (which should also be self-explanatory), the administrators have to select the data-type that is adequate for representing the information of this category. They can select from numeric types (INT, REAL), and strings of fixed size (CHAR(*)) as well as unlimited size (TEXT).

The specified information in the rows of the list is not only used to create the columns of the generic metainformation table, but it is also a reference for the automatic query interface generation, as described in Subchapter 4.1.1. In this context the remaining fields of the form in Figure 9 can be understood. The "Queryable" flag determines, if the column should be queryable at all, or if its content should only be displayed, when users select an object in the scene to get detailed information. If the column is set to be querable, a "QueryType" has to be selected. The possible types are "free", "search", "select", "multiselect", and "range" which describe the search functionality. They are all differently represented in the query interface. The "range" query type makes only sense in combination with a numeric data type, and requires additional values about the lower and upper bound of the range and the step size in which the values will be selectable in the query interface.

The "Update Index" button is also related to the automatic query interface generation. As explained in (4.1.1) an index is required for "select" and "multiselect" categories to avoid searching a complete database column for all possible values, each time the query interface is generated. This index would actually require updating, each time a new row is added to or deleted from the metainformation table. This would lead to performance problems, if, for example, a seller and therefore all his offers are to be completely removed from the database. In such a case, it is sufficient to update the index once, after all offers are removed.

If a numeric data-type is selected, additional information about the measurement of this category is required. When all fields are filled out and the "Add Entry" button is pressed, the applet will check if the identifier is unique and add the complete form as a new row to the list. By pressing the "Create System" button, the table that stores the metainformation for all the offers will be newly constructed, based on the rows in the list.

4.2.2 Managing the sellers


Add and remove sellers from the system

Type:	gallery	Fax:	
Galleryname:	Mueller's Gallery	E-Mail:	amuller@icsi.berkeley.edu
		Account:	384580
Address:	2208 Martin Luther King	Password:	Test
	Berkeley, CA 94704	Owner:	Andreas Mueller
Phone:	(415) 642-4274	Foundationyear:	1998

Gallery Info:

Businesshours:

Mon-Fri: 9 am - 5pm
Sat: 10 am - 2 pm

Artists:

Vincent Van Gogh
Casper David Friedrich

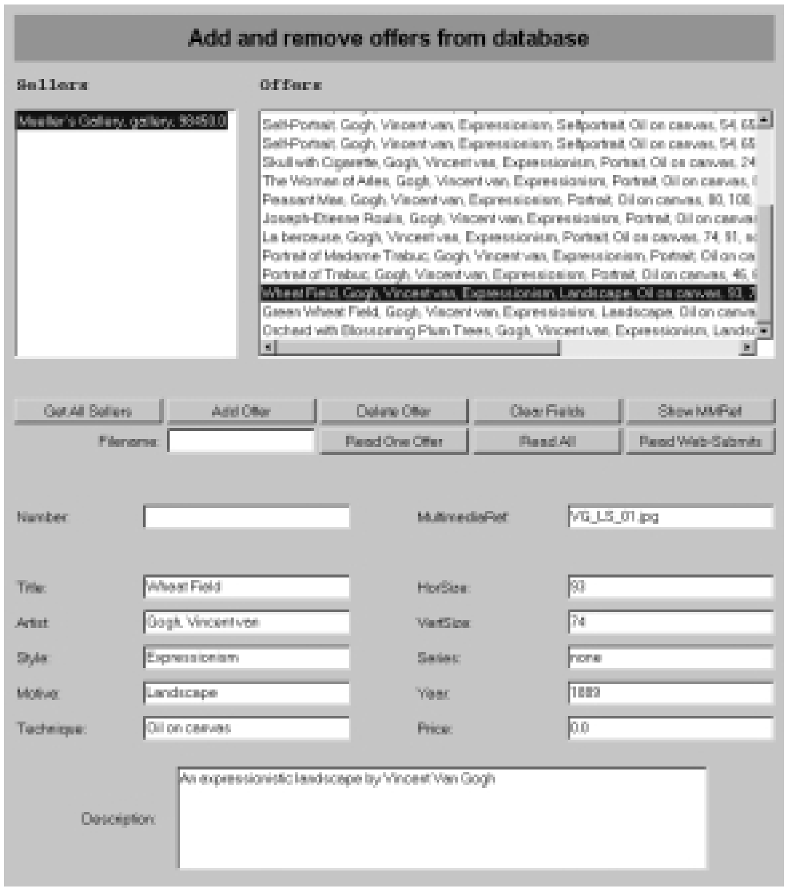
Name:	Type:
Mueller's Gallery	gallery
Meinbold Ingrid	artist
Another Gallery	gallery
Gallery 1	gallery

Figure 10: Screenshot of the "Edit Sellers" tool

Because galleries and artists using this system are not required to have access to the system via a computer, there has to be a way to register them in the database when they apply to become sellers. To do so, they can request a submission form via mail and send it back to the system's administrators, filled out with the required information. Administrator will then transfer the information into the tool shown in Figure 10 and press the "Add Seller" button to write the entry into the database. As soon as the sellers have received their access password, they can start to add offers to the system, also possibly online. Because filling out the standard fields of such forms is error-prone the administrator can merge many new sellers in one specially marked up text-file and import this file using the "Read File" button.

The much more common way for galleries and artist who have direct access to the electronic system will be, to fill out the online submission form and send it to the administrator by simply clicking a button. This is one of the password-secured tools that can only be used by registered sellers. When the administrator presses the “Web Submits” button, such online registrations will be parsed and displayed in the form. Then, the administrator can check if the information is complete and accept the application or request further information.

4.2.3 Managing the offers



Moeller's Gallery, gallery, 93453.0

Self-Portrait, Gogh, Vincent van, Expressionism, Selfportrait, Oil on canvas, 54, 65
Self-Portrait, Gogh, Vincent van, Expressionism, Selfportrait, Oil on canvas, 54, 65
Skull with Cigarette, Gogh, Vincent van, Expressionism, Portrait, Oil on canvas, 24
The Woman of Arles, Gogh, Vincent van, Expressionism, Portrait, Oil on canvas,
Peasant Man, Gogh, Vincent van, Expressionism, Portrait, Oil on canvas, 80, 100,
Joseph-Etienne Roulin, Gogh, Vincent van, Expressionism, Portrait, Oil on canvas
La berceuse, Gogh, Vincent van, Expressionism, Portrait, Oil on canvas, 74, 91, s
Portrait of Madame Trabuc, Gogh, Vincent van, Expressionism, Portrait, Oil on ca
Portrait of Trabuc, Gogh, Vincent van, Expressionism, Portrait, Oil on canvas, 46, 6
Wheat Field, Gogh, Vincent van, Expressionism, Landscape, Oil on canvas, 93, 7
Green Wheat Field, Gogh, Vincent van, Expressionism, Landscape, Oil on canva
Orchard with Blossoming Plum Trees, Gogh, Vincent van, Expressionism, Landsc

Number:		MultimediaRef:	VG_LS_01.jpg
Title:	Wheat Field	HorSize:	93
Artist:	Gogh, Vincent van	VertSize:	74
Style:	Expressionism	Series:	none
Motive:	Landscape	Year:	1889
Technique:	Oil on canvas	Price:	0.0

Description: An expressionistic landscape by Vincent Van Gogh

Figure 11: Screenshot of the "Edit Offers" tool

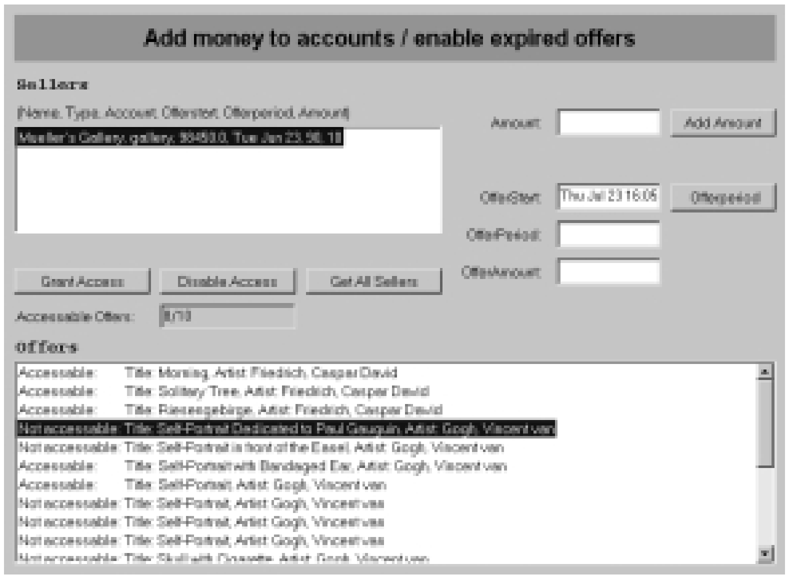
The behavior of the "Edit Offers" tool (Figure 11) is similar to the "Edit Sellers" tool. The biggest difference is that the input mask is dynamically generated, because the names and types of the categories for the metainformation values are not fixed. Besides the metainformation values, the number of available copies of the exhibited subject and a reference to the image has to be specified. As with the "Edit Sellers" tool, the administrators can read the new entries from a text-file or directly from the database where the online submissions are stored. Before administrators accept an offer with the "Add Offer" button, they can check if the correct image is assigned to the offer, by using the "Show MMRef" option. This will open another browser window and show the referenced image.

4.2.4 The accounting system

The introduced system is not only designed for exhibition purposes (in which case it would be a virtual museum), but also to enable the contact between a customer of the service (e.g. the exhibiting and selling gallery or artist) and the viewer or potential buyer. For this purpose, the system offers electronic commerce functionality for the gallery or artist which goes beyond the plain display features.

For this purpose, the database also stores the contact information about people that are interested in a painting. The sellers can check for these interests through the system and directly contact the interested persons, for example, via e-mail. If galleries or artists have no access to the electronic service, but want nevertheless to exhibit and sell artwork using this system, the system administrators will inform them via conventional communication facilities. Another important feature is that the system keeps track of users’ accesses. Each time a painting belongs to a query result and is therefore retrieved from the database (or somebody clicks on a painting to receive further information about it) the access information is stored together with the metainformation of the painting. Using a password-secured tool, sellers can check the statistics of their offers and find out, which kind of art produces the highest interest. Based on this profile, sellers can adjust the exhibited paintings to optimize the offer.

The last administration tool (Figure 12) is used to add money to a sellers account. Gallery or artist who are offering a number of paintings for a defined period of time, are charged for this service, whereas end users of the system can browse the gallery for free. Sellers can add any amount of offers to the system, independent of the number of offers they are contracting for. This eases the manageability of the system, because sellers do not need to add a new offer to the system each time a painting is sold. Displaying one of these reserved offers will be sufficient. With this tool, administrators have the possibility to act for such sellers who cannot do it themselves (e.g., if they do not have online access). In addition, the tool checks if the paid for offer periods have expired, and if so, sets all offers of the concerned seller to "not accessible".



(Name, Type, Account, Offerstart, Offerperiod, Amount)

Moeller's Gallery, gallery, 99450.0, Tue Jan 23, 98, 18

OfferStart: Thu Jul 23 16:06

Accessable Offers: 8/10

```
Accessable:     Title: Morning, Artist: Friedrich, Caspar David
Accessable:     Title: Solitary Tree, Artist: Friedrich, Caspar David
Accessable:     Title: Riesengebirge, Artist: Friedrich, Caspar David
Not accessable: Title: Self-Portrait Dedicated to Paul Gauguin, Artist: Gogh, Vincent van
Not accessable: Title: Self-Portrait in front of the Easel, Artist: Gogh, Vincent van
Accessable:     Title: Self-Portrait with Bandaged Ear, Artist: Gogh, Vincent van
Accessable:     Title: Self-Portrait, Artist: Gogh, Vincent van
Not accessable: Title: Self-Portrait, Artist: Gogh, Vincent van
Not accessable: Title: Self-Portrait, Artist: Gogh, Vincent van
Not accessable: Title: Self-Portrait, Artist: Gogh, Vincent van
Not accessable: Title: Skull with Cigarette, Artist: Gogh, Vincent van
```

Figure 12: Screenshot of the "Edit Accounting" tool

The tools for the sellers

All of the above features for entering new information into the database, checking the access statistics and the interests in pieces of art, can also be accessed online by the galleries and artists that are registered in the system (except for altering the database scheme and adding money to an account). The tools which allow this remote access will not be further explained. They are just the counterpart to the previously described tools and write the new and yet uncontrolled data in a separate table of the database where they can be read and verified with the administration tools. These seller tools are accessible with the "Tools" option in the main menu (see Figure 7).

5. Implementation Details

Since we are dealing with a rather complex system which consists of nearly 40 Java classes that use inheritance and “uses” relationships, we will attempt to structure the explanations in a way that allows all the explicit as well as implicit relations to be understood easily. First, the basic concepts will be introduced that need to be clarified, before any other parts of the system can be analyzed. Afterwards the system’s internals, including the scene construction concept, will be investigated.

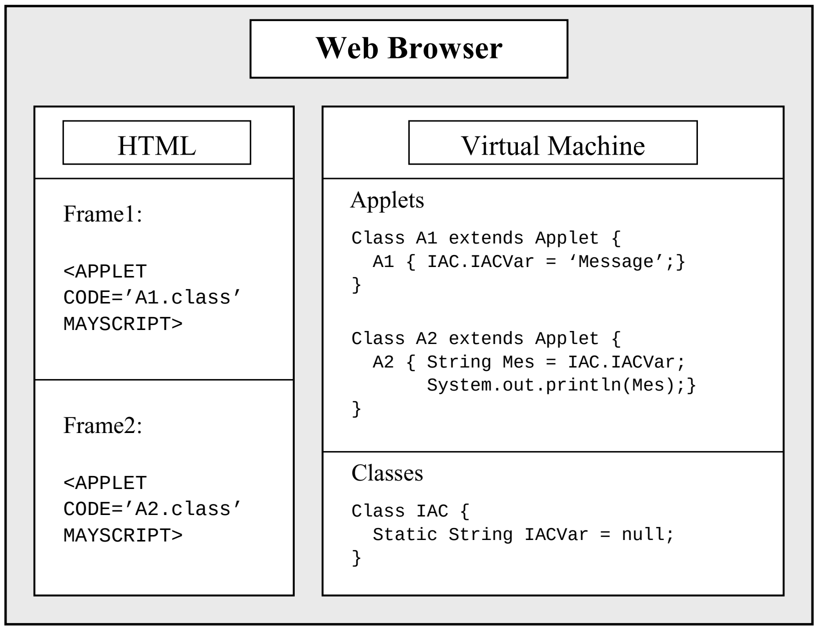
5.1 General concepts

There are four underlying principles, which are of inherent importance and which cannot be assigned to any special part of the system. The first is the "Inter Applet Communication". The others are the use of Java classes to write HTML content into a Web browsers frame, the realization of the generic database scheme, as described in Subchapter 4.2.1, and finally the set of wrapper classes used to simplify the frequently needed database accesses.

5.1.1 Inter Applet Communication (IAC)

Inter Applet Communication (IAC) is, as the name implies, a mechanism to pass information from one applet to another. This can be done by directly accessing variables in another applet’s class instance, but to apply this method a reference to the targeted applet is required. For this purpose, we could use the method `this.getAppletContext().getApplet(Sting Name)` on the sending applet’s instance, but, unfortunately, this method works only with applets which are embedded within the same Web browser frame. Because we want to provide an easy to use interface, we are utilizing only a single browser window, instead of many unrelated windows. This window is subdivided into a set of frames which change their content dependent on the users’ actions. Therefore, we need another mechanism that allows us to perform IAC between applets which are placed in different frames.

One possibility would be to use the Java-JavaScript interface to retrieve the destination applet’s reference through JavaScript, but this is just a partial solution since this interface is only implemented for Netscape’s Navigator/Communicator. If we want to offer a more general solution, we have to employ a trick which takes advantage of the fact that the actual Web browsers can be forced to use the same Virtual Machine and class-loader to instantiate all applets and associated classes, independently of where they are embedded. Because all classes exist in the same address space, we just need to define one class which consists exclusively of static variables and then can use this shared class to exchange information between all applets. This works even when the applets exist in different browser windows, as long as the MAYSCRIPT attribute is set in the EMBED-Tags of all the classes which should be handled by the same VM.



```
<APPLET
CODE='A1.class'
MAYSCRIPT>

<APPLET
CODE='A2.class'
MAYSCRIPT>
```

```
Class A1 extends Applet {
  A1 { IAC.IACVar = 'Message';}
}

Class A2 extends Applet {
  A2 { String Mes = IAC.IACVar;
       System.out.println(Mes);}
}
```

```
Class IAC {
  Static String IACVar = null;
}
```

Figure 13: Simplified scheme of Inter Applet Communication using static classes

Figure 13 gives a simplified impression of how the concept of using static classes works. In addition, we need to make sure that the shared information is consistent, by applying the `synchronized` keyword to the static variable. And if information is exchanged more than once, the receiving class has to employ a thread which checks from time to time if a new message has arrived.

5.1.2 Using Java to write HTML into a Web browser frame

We are integrating HTML, applets and the VRML interface within one single browser window to take advantage of the special strength of each single technology and, at the same time, to circumvent some weak points of the individual approaches. For example, since HTML offers much better possibilities to format and display text than Java applets, we want to write HTML, dynamically generated by an applet to a Web browser frame. Java's `AppletContext` class offers a method `showDocument(URL)`, which seems to perform a similar task, but it is not exactly what we need. Applying this method, we could request an already existing HTML page from a Web server and let the browser display it in the same frame where the applet was embedded. A related method uses a second argument which defines the frame that should display the new Web page. But if we want to display HTML text generated on the fly with these features it, would require to first write the generated HTML back to the Web server, and then call the method which would bring the HTML back. If we want to avoid this network overhead, we have to find a possibility to pass the generated HTML directly from the applet to the browser at the client side.

This problem can be solved with the assistance of the IAC described earlier in combination with the JavaScript-Java connection. This is not to be confused with the Java-JavaScript connection, which is only available as a Java package for Netscape's browser products. The connection in the other direction is part of JavaScript and works with Netscape browsers, as well as with Microsoft's Internet Explorer. That is, using JavaScript, we can access applet variables and pass their content as an argument of the JavaScript `document.write(String HTML)` method, which produces exactly the desired feature.

In the applet where we want to output some HTML within a browser frame, we just need to assign the HTML text to a String variable in the shared static IAC class (HTML in the example). The method `showDocument(URL)`is then called on the `AppletContext` object, related to the applet requesting to output HTML in an abitrary frame. The argument to this method is the URL that contains the document shown in Figure 14.

```
<HTML>
<HEAD>
   <SCRIPT language="JavaScript">
   function WritePage()
   {
     if(document.iac != null)
     {
       var input = document.iac.HTML;
       if(input != null)
       {
         document.write(input);
       }
     }
   }
   </SCRIPT>
</HEAD>
<BODY onload="WritePage();">
<APPLET code="IAC.class" width="0" height="0" name="iac" MAYSCRIPT></APPLET>
</BODY>
</HTML>
```

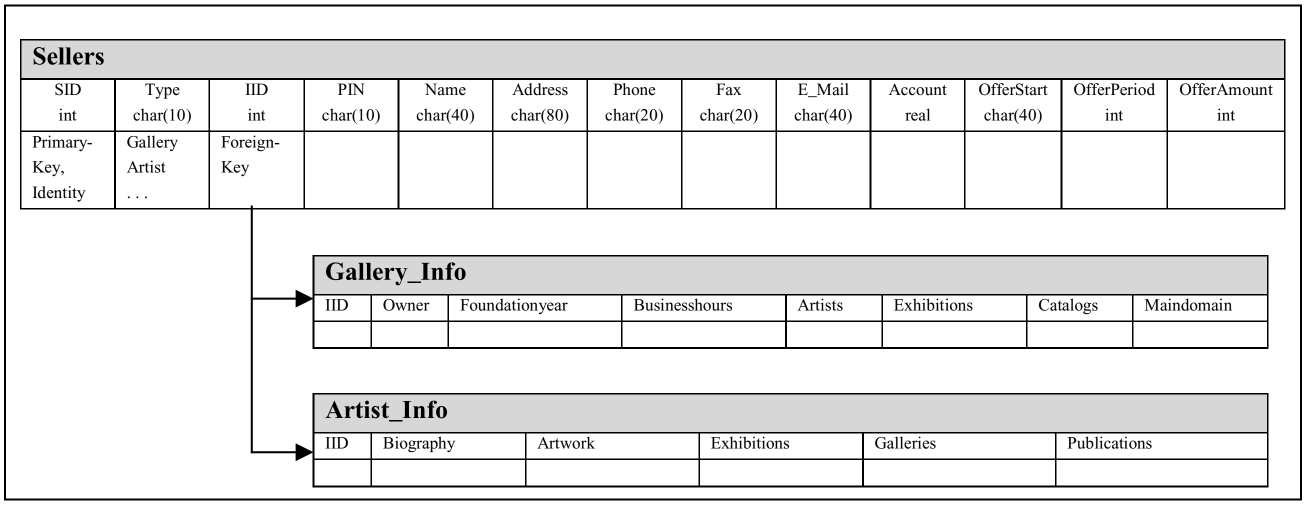
Figure 14: Writing HTML from an Applet directly into a Web browser frame

The illustrated example Web page does nothing but embedd the IAC class with zero size, so that it cannot be seen on the page, and as soon as the IAC applet is instantiated, the JavaScript method

`WritePage()` is called, because of the `onload` attribute of the `BODY`-Tag. This method checks if there is content in the HTML variable, by accessing the embedded IAC class, and calls the JavaScript method `document.write(String HTML)` to display the HTML information in the current frame.

5.1.3 The database scheme

In the following, the underlying database scheme will be described. First, we need to think about what kind of information will be required and how it can be stored, using the given database-types. Because we are providing a service where a customer can contact the seller or vice versa, we need to store information about the sellers, as well as about the possible customers. For our current special system, we have two possible types of sellers: the galleries and the single artists. Some information is shared by both categories, but each has to provide some additional specific information, reflected in the "Sellers" table. Using the following scheme, we can easily add other types of sellers, which would be described through different additional tables.



Sellers

SID int	Type char(10)	IID int	PIN char(10)	Name char(40)	Address char(80)	Phone char(20)	Fax char(20)	E_Mail char(40)	Account real	OfferStart char(40)	OfferPeriod int	OfferAmount int
Primary-Key, Identity	Gallery Artist . . .	Foreign-Key										

Gallery_Info

IID	Owner	Foundationyear	Businesshours	Artists	Exhibitions	Catalogs	Maindomain

Artist_Info

IID	Biography	Artwork	Exhibitions	Galleries	Publications

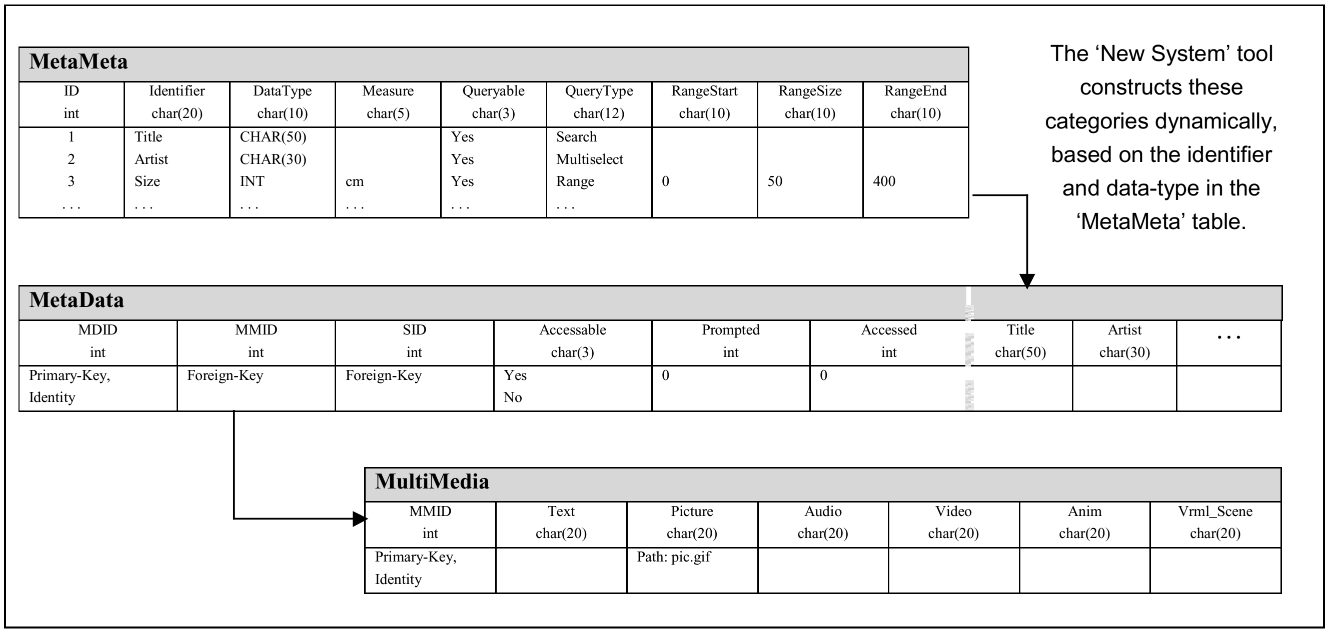
Figure 15: The "Sellers" table

Checking the type-information (Type field of the "Sellers" table), we can find out from which table we have to retrieve the special information specified by the value in the IID (Info-ID) field. The information about start-date, period and amount of offers is required to enable the access to registered paintings and to check for the expired offers, as described in Subchapter 4.2.4. The table for the customers is basically the same as the "Sellers" table, but without the information about the account and the offer contract.

As mentioned in the context of the "New System" administration tool (4.2.1), the table which holds the describing categories for the actual individual offered objects is created dynamically at the startup of the system. Because we cannot know which fields will be required when the subject of the exhibition changes, the structure of this "MetaData" table (see Figure 16) is more complicated, i.e. not complete. When the administrator has specified all the categories which are sufficient to represent the exhibition objects properly (e.g. Artist, Title, etc.), we could theoretically directly use these categories to complete the missing columns of the "MetaData" table.

But there is also some additional information regarding these categories which is required to generate the query interface dynamically. This is the reason to create another database table that holds information about the describing categories and is, therefore, called "MetaMeta" table. This table stores exactly the information about the query-type and corresponding parameters that are specified by the administrator (using the "New System" tool) and later used for the query interface construction.

When the descriptive information for the "MetaData" table is complete, the "New System" tool will be used to generate the basic setup of this table, which will then remain unchangeable, after the initial system configuration. For this purpose, the identifier and data-type of each row in the "MetaMeta" table are used to create one column in the "MetaDate" table (see Figure 16).


The 'New System' tool constructs these categories dynamically, based on the identifier and data-type in the 'MetaMeta' table.

MetaMeta

ID int	Identifier char(20)	DataType char(10)	Measure char(5)	Queryable char(3)	QueryType char(12)	RangeStart char(10)	RangeSize char(10)	RangeEnd char(10)
1	Title	CHAR(50)		Yes	Search			
2	Artist	CHAR(30)		Yes	Multiselect			
3	Size	INT	cm	Yes	Range	0	50	400
. . .	. . .	. . .	. . .	. . .	. . .			

MetaData

MDID int	MMID int	SID int	Accessable char(3)	Prompted int	Accessed int	Title char(50)	Artist char(30)	. . .
Primary-Key, Identity	Foreign-Key	Foreign-Key	Yes No	0	0			

MultiMedia

MMID int	Text char(20)	Picture char(20)	Audio char(20)	Video char(20)	Anim char(20)	Vrml_Scene char(20)
Primary-Key, Identity		Path: pic.gif				

Figure 16: "MetaMeta" and "MetaData" table

Each row in the "MetaData" table represents an offer of a seller, who is referred to by the "SID" value. The individual offers are uniquely identified by the "MDID" value. It also holds the values for access statistics, besides the object describing metainformation in the dynamically generated columns. The "Promted" column stores how often a particular painting was retrieved from the database (part of a query result), and the "Accessed" column keeps track of how often users requested further information. In addition, we provide a flag that controls the accessibility of each single offer, so that the sellers will be able to specify exactly which of their offers should be exhibited in case they have registered more offers in the database than they are contracting to display.

Actually, we could directly add a column to the "MetaData" table which would hold the associated paths to the respective objects representation on the Web server. But, in order to provide easy extensibility for the future, we are keeping the physical location of the image in a separate table. With this concept, it would be a minimal effort to add, for example, an explanatory audio file to each offer that should be played when users view the painting. Or, in the case of other subjects than paintings for which it is intuitive that they are exhibited as an image on the wall, we could provide another type of representation, e.g. a video, an animation or even a 3D object that is modeled in VRML, itself.

5.1.4 Database Wrapper classes

The concept of wrapper classes is a well-known and easy way to access complex data-structures in order to avoid repeating tasks, such as nested initializations and type casts. For example, if it is known that two classes always occur in combination and that we have to initialize both classes each time they are needed, we could wrap both classes into a "wrapper" class, which handles the initialization, for both classes.

The same mechanism was used to wrap the content of single database rows of each table into a seperate class. This produces a set of advantages. First of all, we do not need to keep a reference to JDBC's `ResultSet` class, which is rather complicated to handle. If, for example, we need to modify a row of a table, it would be necessary to retrieve the data into such a `ResultSet.` Then, we would have to get each single piece of information from the `ResultSet` with a seperate method call, perform the required update and finally construct an SQL-statement which writes the new information back into the database. Instead of always executing this whole sequence of methods, we can encapsulate all the SQL-syntax in a wrapper class and provide an interface which allows easy access to the database rows for all kind of purposes. The wrapper classes for the different tables all have some common basic methods to retrieve a row from the database and to write it back. The other methods are table specific and can be used to modify the information which is held in the wrapper class. This allows easy creation, manipulation and deletion of database rows, anywhere in the system, without explicitly using the SQL-Syntax.

Another advantage can be identified when we want to delete a database substructure consisting of multiple rows. Figure 17, for example, shows "Table A", which refers to "Table B" by means of a foreign key. If we would like to delete the second row of "Table A", we would first need to retrieve the foreign key value (4) for "Table B". Then we would delete the row of "Table B" that is identified by this reference, before we could take care of the originally targeted row in "Table A". Such complex, but repeating steps can be automated and encapsulated within the methods of the wrapper class. If, for example, the `deleteFromDB()` method is called on an instance of the A_Wrapper class, the complete substructure, including foreign key references, will be removed from the database, by this single command.

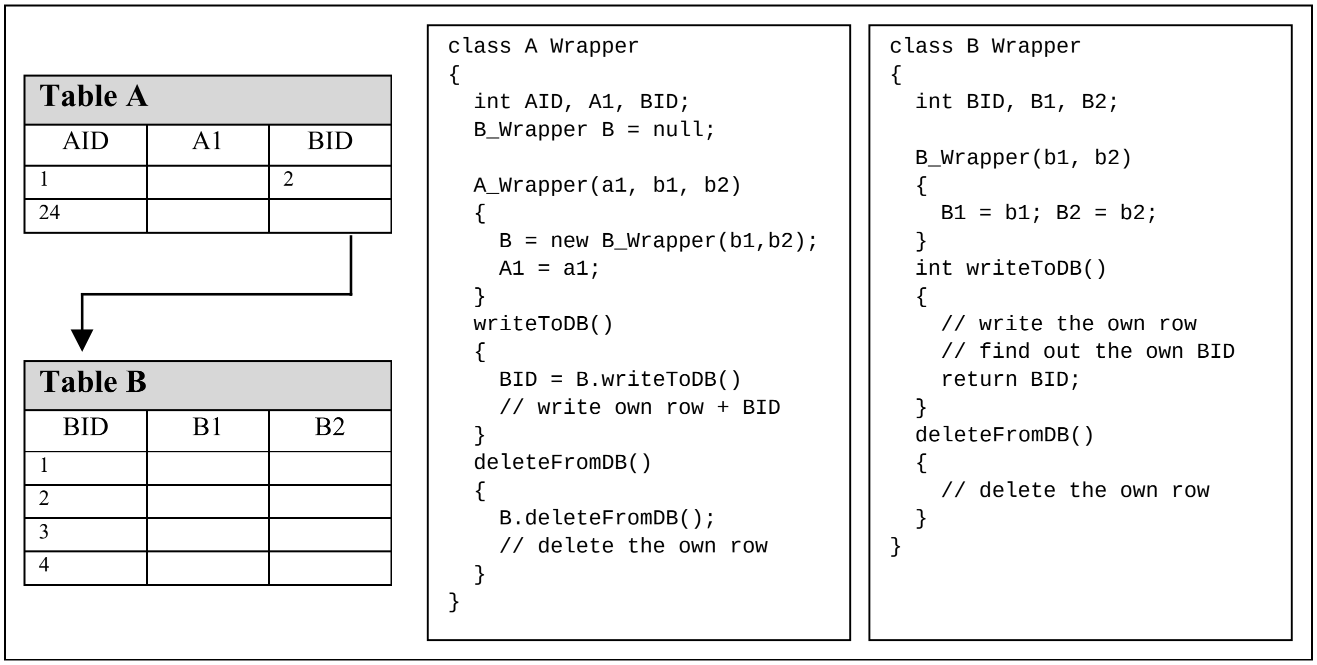


Table A

AID	A1	BID
1		2
24		

Table B

BID	B1	B2
1		
2		
3		
4		

```
class A Wrapper
{
  int AID, A1, BID;
  B_Wrapper B = null;

  A_Wrapper(a1, b1, b2)
  {
    B = new B_Wrapper(b1,b2);
    A1 = a1;
  }
  writeToDB()
  {
    BID = B.writeToDB()
    // write own row + BID
  }
  deleteFromDB()
  {
    B.deleteFromDB();
    // delete the own row
  }
}
```

```
class B Wrapper
{
  int BID, B1, B2;

  B_Wrapper(b1, b2)
  {
    B1 = b1; B2 = b2;
  }
  int writeToDB()
  {
    // write the own row
    // find out the own BID
    return BID;
  }
  deleteFromDB()
  {
    // delete the own row
  }
}
```

Figure 17: Example of a wrapper class for a database row

5.2 The system's internals

We will now explain such parts of the implementation that are not just supporting concepts, but have their own independent relevance and can be triggered by the user. This is the scene construction concept and algorithm, the implementation of the overview maps, and the trigger events in the VRML scene, as well as all related mechanisms.

5.2.1 The IAC-listener thread

We will now investigate what exactly happens after the SQL statement has been constructed by the query interface (4.1.1), executed, and the related query result has been retrieved. When the execution of the SQL statement has returned a non-zero result, the rows of the `ResultSet` are stored in a vector of "MetaData" wrapper classes to make the information easier to access for the following methods. This vector is then passed to an applet in the frame where the VRML plugin is embedded. It was decided to construct the internal scene representation in this second applet, instead of performing this task instantly, because we do not have a VRML browser reference in the query interface applet. Such a reference would be required to pass the geometry of the gallery to the VRML plugin, after the scene graph has been constructed. We cannot use EAI's `Browser.getBrowser()` method to obtain this reference because this method works only when the requesting applet is placed in the same frame as the plugin. However, EAI also offers another method, which should deliver the browser reference even when the requesting applet belongs to another frame, but unfortunately this method does not work properly in the current implementation of the EAI.

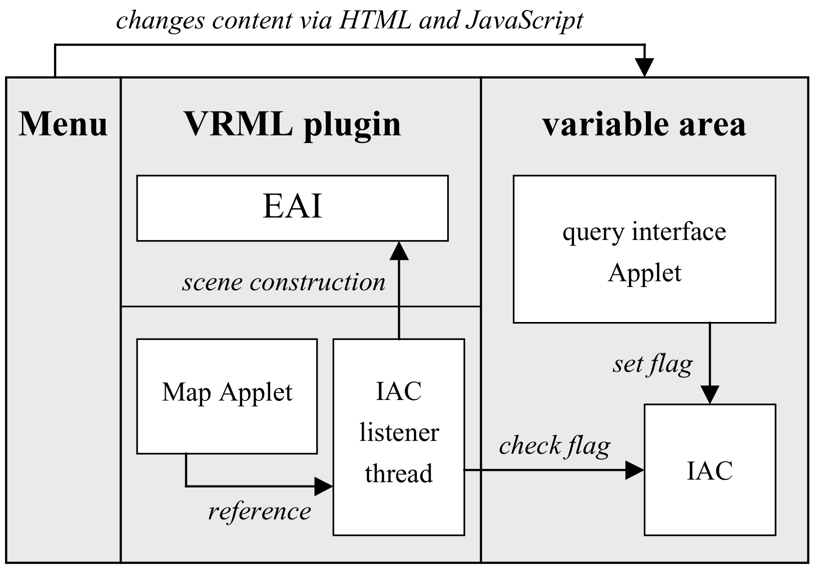


Figure 18: Functionality of the IAC-listener thread

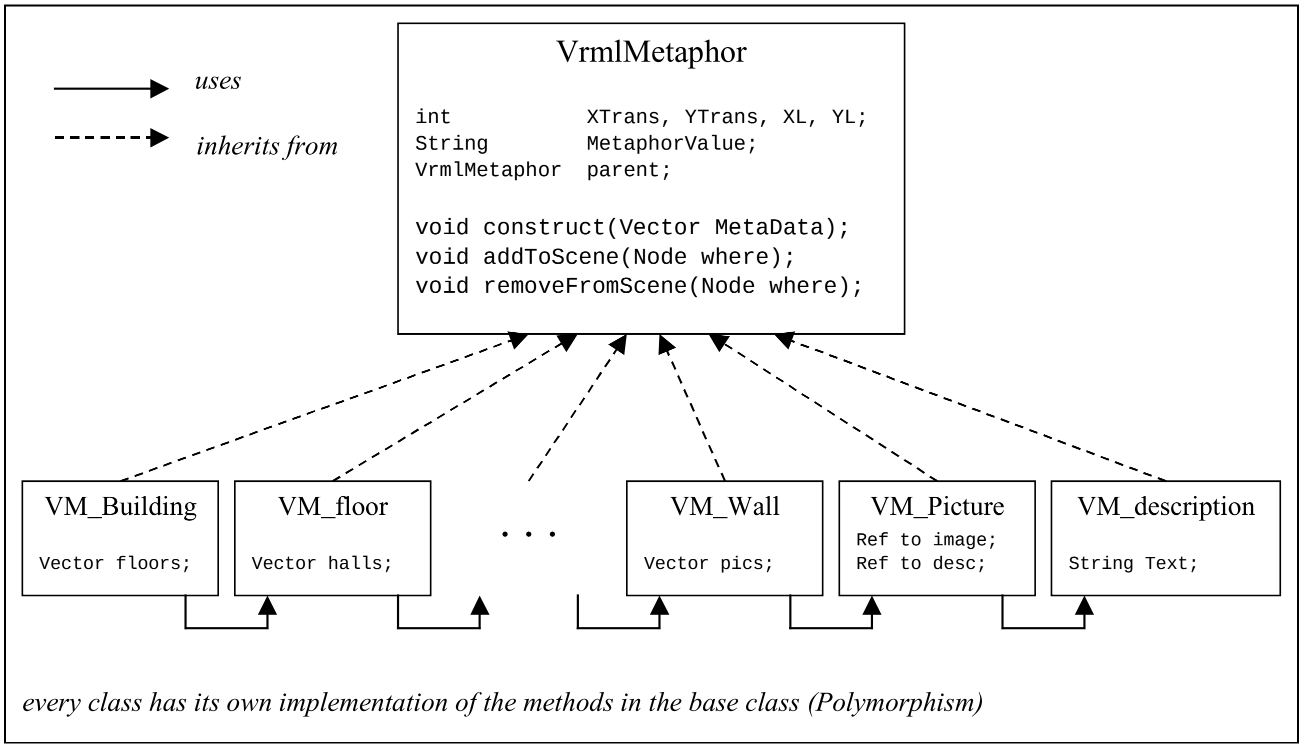
To pass the vector that contains the wrapper classes from the query interface applet to the part which does the scene construction, we employ the IAC mechanism. The applet that has the intrinsic task to display the overview maps instantiates the thread `IACListener` and holds a reference to it (see Figure 18). This applet has also retrieved the VRML browser reference and passes it to the `IACListener` thread, as an argument of the constructor.

The static class IAC holds a reference to a vector of database wrapper classes which is used to pass the query result from the query interface to the scene construction method. It has also a "boolean" flag that indicates whether a new query result was retrieved. Once the `IACListener` thread is started, it does nothing but check the flag of the class IAC at regular time intervals. This is realized by the thread's `sleep(long millisec)` method. This method induces Java's runtime system to process another thread and not to continue with the original one, till the specified amount of time is elapsed. We could also utilize the methods `suspend()` and `resume()` of Java's `Thread` class. The method `suspend()` interrupts the current program execution and switches to another thread, until the method `resume()` is called on the suspended thread. But, because we are only checking one single "boolean" variable, the performance of the system is not significantly reduced with the first method. As soon as the IAC listener recognizes that a new query result has arrived, it initiates the scene construction.

5.2.2 The scene construction

The scene construction itself is based on a concept called "visual classes" and widely follows the mechanism which was used in the "Virgilio" project [GMD991, GMD1093, GMD_R15]. The concept of "visual classes" means that we utilize a set of classes that represent the complete visual scene and can be used to construct, alter and delete geometric elements (i.e. single walls, rooms, halls, floors). After we have created a complete internal representation of the scene graph within our applet, we just need to write the corresponding VRML syntax to the plugin. For this purpose, each single class that represents a visual object in the scene provides a method which constructs the VRML syntax of its own content, calls the same method on all children elements and passes the VRML representation to the plugin, using the EAI. It may seem to be an additional effort to build a second internal scene representation within the applet, besides the scene graph that is implicit in the VRML structure. But this strategy is necessary if we want to reach an acceptable performance for large query results. Instead of constructing the whole scene in the VRML browser, which would lead to a very bad performance, we can take advantage of one important fact. The user can at any time only exist at one single point in the scene. If we can keep track of the user's position in the scene, we just need to visualize the immediate surrounding, to get a much better performance. But this concept makes only sense, when we have access to the information about the structure and content of the users surrounding, otherwise we would need to recalculate this information, each time the user moves in the scene, e.g. from one room into another. If we pre-calculate the whole structure of the scene and hold this information internally in the construction class, it is very easy to add the geometry which is in the immediate surrounding of the user, and to remove the elements which are out of sight from the VRML scene. The VRML representation of removed scene parts is stored internally, so that, e.g., adding a room to the scene for a second time, can be performed much more efficient than the first time.

The different geometric elements are inherited from one single base class (Figure 19) which provides a common interface. Therefore, it is very easy to add new graphical representations to the hierarchy (shelves, cupboards, video screens, etc.). The base class "VrmlMetaphor" enfolds variables for the translation of the element in the scene (relative to the parent element) and values concerning the spatial extension. The variable “MetaphorValue” defines the category or single entry of the database that it represented by the current instance of the different classes (e.g. “Van Gogh” for a floor and “Water Lilies” for a description). This base class also defines the methods that are used to construct, delete and display the elements in the scene. Each single category of classes (wall, room, hall, floor) implements these functions in its own way, and it is decided at runtime which method has to be called (Polymorphism).


uses
inherits from
VrmlMetaphor
VM_Building
VM_floor
. . .
VM_Wall
VM_Picture
VM_description
every class has its own implementation of the methods in the base class (Polymorphism)

```
int           XTrans, YTrans, XL, YL;
String        MetaphorValue;
VrmlMetaphor  parent;

void construct(Vector MetaData);
void addToScene(Node where);
void removeFromScene(Node where);
```

```
Vector floors;
```

```
Vector halls;
```

```
Vector pics;
```

```
Ref to image;
Ref to desc;
```

```
String Text;
```

Figure 19: Concept of the "visual classes"

In the following, we will investigate what a possible query result could look like in the internal representation of the "visual classes", using a simplified example. Subsequently, we will describe the algorithm which splits the result set into this structured form, based on the VRML metaphor configuration.

Figure 20 shows parts of the internal structure after the scene construction algorithm [`void construct(Vector MetaData)`] was executed. The result of this algorithm is a tree structure of `VM_Metaphor` objects with the actual exhibition objects (images), at the bottom. On the left side, we see a possible distribution of the instances as nodes in the tree. The nodes (boxes) are labeled with the `MetaphorValue` variable of each concrete instance or with a name that was assigned by the construct method, if the `VM_Metaphor` class is just a grouping type and does therefore not explicitly represent a MetaData category. On the right side, next to the tree structure, we find the different classes that are derived from `VM_Metaphor` and on which level they occur. In our current structure, the inclusion sequence for the classes is fixed (except the VM_Wall class, that can occur as a child for the VM_Hall class as well as for VM_room). With the Polymorphism principle, it is very easy to keep this structure totally variable, e.g. to add images to the walls in the halls. Or, we could, for example, construct a completely new object as a cupboard which can be placed anywhere in the scene structure.

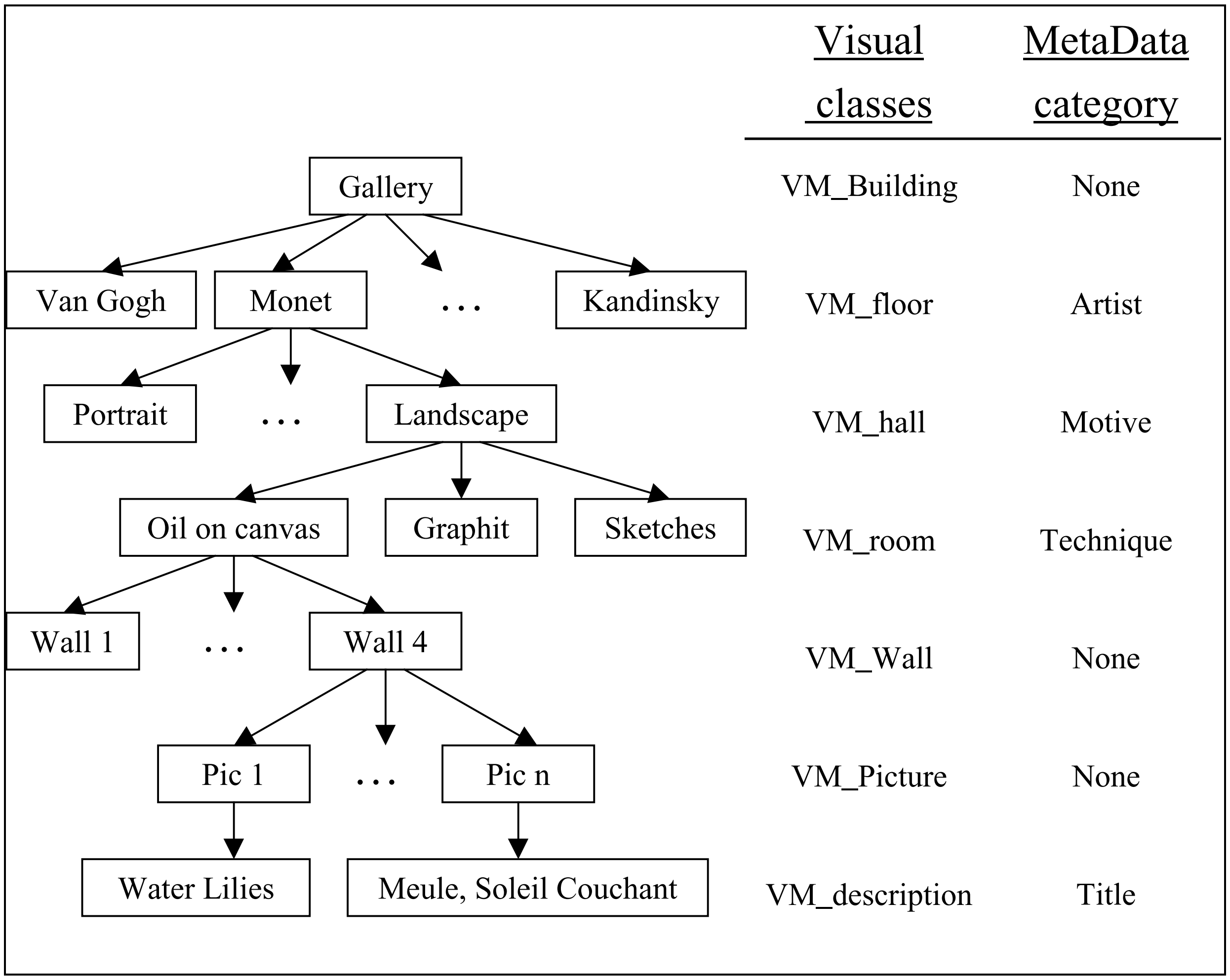


Figure 20: Example of the internal scene representation

At the right we can see which MetaData category was defined for the different types of classes with the metaphor configuration tool. The names of the grouping nodes start with a capital letter after the "VM_"-prefix, and they do not have any MetaData category assigned to them. Instead, the constructing algorithm has assigned a name to the `MetaphorValue` variable of such objects to identify them uniquely. The VM_Picture class holds a reference to the physical position of the image on the Web server, besides the VM_description class.

The `void construct(Vector MetaData)` method which produces the object structure shown in Figure 20 is defined for all VM_Metaphor classes. Each superior class calls the same method recursively on its children. To start the whole scene construction process, an object of the VM_Building class has to be instantiated and then the `construct` method has to be called on this object with the vector of MetaData wrapper classes, that was passed from the query interface, as an argument.

Until now we did not say anything about how the rooms are distributed in space. In the current implementation, the rooms are just cubic and scale with the amount of pictures that belong to the room. Furthermore the rooms are directly placed one next to the other. This simple concept can be replaced by more complicated methods that construct the shape randomly and apply a room-optimizing algorithm to fit the rooms and halls to a floor. This extension is easy to realize because the shape of the rooms is not predefined and can be adapted in any way, using the VM_Wall objects.

5.2.3 The overview maps

Besides the VRML plugin, the applet which generates the overview map is the only part that remains constantly visible on the screen and is always executing. It is placed in one frame together with the VRML browser, so that it can get the browser reference by simply calling EAI's `Browser.getBrowser()` method. It also instantiates the IACListener thread and sets up the routes for the EventOutObserver classes which are required to establish an efficient communication mechanism between the VMRL scene and the controlling applet. This concept will be described in the next section.

The applet has to provide the 2D overview-map of the gallery in order to give users a reference about his current position in the scene. This is realized by a small arrow which shows the current position and orientation on the map. The overview map of a single floor scales automatically, so that the complete floor plan always fits into the fixed space of the display area. The maps can also be used to gather further information about the content of rooms, halls and floors, by moving the mouse over the geometry on the map.

The need to always display the complete floor map is another reason to pre-calculate the distribution of the paintings and holding a reference to the internal representation of the scene. By doing so, we can apply a simple mechanism to draw the map. Each object in the internal scene representation (room, hall, floor) provides a method called `drawToMap(Graphics g, int XOrg, int YOrg)` which draws itself onto the transferred `Graphics` object, in consideration of its own translation and by means of the current scale factor which can be different for each floor. With this tool, we just need to call the `drawToMap` method on the currently active floor, and the function will recursively traverse the children objects of the internal scene representation and draw the complete floor.

The label with the content information about the geometry has to be updated with the movement of the mouse and needs to be added after the complete floor plan has been drawn, in order to avoid the overlapp of graphical elements. Therefore, we assigned this task to a seperate method, instead of adding it to the `drawToMap` method. Every object in the internal scene representation has an additional boolean flag and a method `checkMousePosition(int x, int y)` which sets the flag to "true" if the mouse points to this geometry. When the active geometry has been marked by recursively applying this method, we just need to call the `drawDescription` method at the root of the scene tree, and it will traverse the structure till it finds the marked object and draws the appendant information onto the map.

The last task that the map applet has to perform is to offer an alternative way of navigating through the scene, besides the usual mouse-movement-based navigation style, which is built into the plugin. If the users click somewhere on the map, they will immediately be placed at this point in the scene. This is realized by means of VRML’s viewpoint concept. We just need to create a new viewpoint at the selected position and bind the user to it. The only problem that can occur is that the user could decide to enter a room which does not yet exist in the scene, by just clicking inside of it. In such a case, users are placed directly in front of the room, the visual representation is added to the scene, using the EAI, and the door opens as soon as the plugin has finished adding the new geometry.

5.2.4 The EventOutObserver

The EventOutObserver class (interface) is one of the most important implementation details. Using the `callback(EventOut value, double timeStamp, Object data)` method is a very efficient way to receive feedback from the scene about any changes in eventOut fields (see Subchapter 3.3.3). We utilize this feature to keep track of several time-critical, very dynamic, and therefore unpredictable changes in the scene.

Position tracking

First of all, we will investigate, how the tracking of the users position in the scene was realized. Unfortunately, there is no EAI browser method which allows asking directly for the users position or orientation in the scene. But there still exists a way to retrieve the required information. The original intention of VRML's `ProximitySensor` was to measure the distance between the viewer and an object in the scene. But, if we use a `ProximitySensor` which is big enough to enclose the entire scene, we just need to check the `position_changed` and `orientation_changed` eventOuts of this node, to keep track of the user's position and orientation (Figure 21).

```
# Part of the basic Root.wrl file, to keep track of the
# viewers position and orientation in the scene

DEF PS ProximitySensor {
  enabled TRUE
  center 0 0 0
  size   10000, 10000, 10000
}
```

Figure 21: ProximitySensor - enclosing the complete scene

The following Java source code excerpt requests a reference to the ProximitySensor, placed in the basic Root.wrl file. The method `getNode(String name)` of the EAI's Browser class can be used, because the node was named "PS", using the DEF statement. Then, the node reference is used to register the eventOuts of the ProximitySensor at the callback method of a class that implements the EventOutObserver interface which is defined in the EAI specification. In addition, we pass an identifying Integer value so that the callback method can distinguish between the two possible sources.

```
Node PosSens = BrowserInstance.getNode("PS");
PosSens.getEventOut("position_changed").advise(PSObs, new Integer(0));
PosSens.getEventOut("orientation_changed").advise(PSObs, new Integer(1));
```

Figure 22 shows, how the callback method distinguishes between the two possible sources by checking the content of the data object. The ProximitySensor changes the position and orientation values several hundred times a second, when a user moves around in the scene. Therefore, we would block the processor if we repaint the complete overview map each time the callback method is called. Because the sensed scale of the position values is much finer than it can be displayed in the map applet, we need to update the position and orientation of the reference arrow only a few times each second. This can be assured very easily by comparing the time-stamps of the incoming events.

```
class PSObs extends Applet implements EventOutObserver
{
  public void callback(EventOut value, double timeStamp, Object data)
  {
    int Source = ((Integer)data).intValue();
    if(Source==0) // The Position was changed
    {
      if(timeStamp > LastPosChangedTime+0.3d)
      {
        LastPosChangedTime = timeStamp;
        // do the update of the position variables
        repaint();
      }
    }
    else if(Source==1) // The same as Choice==0, just for the Orientation
  }
}
```

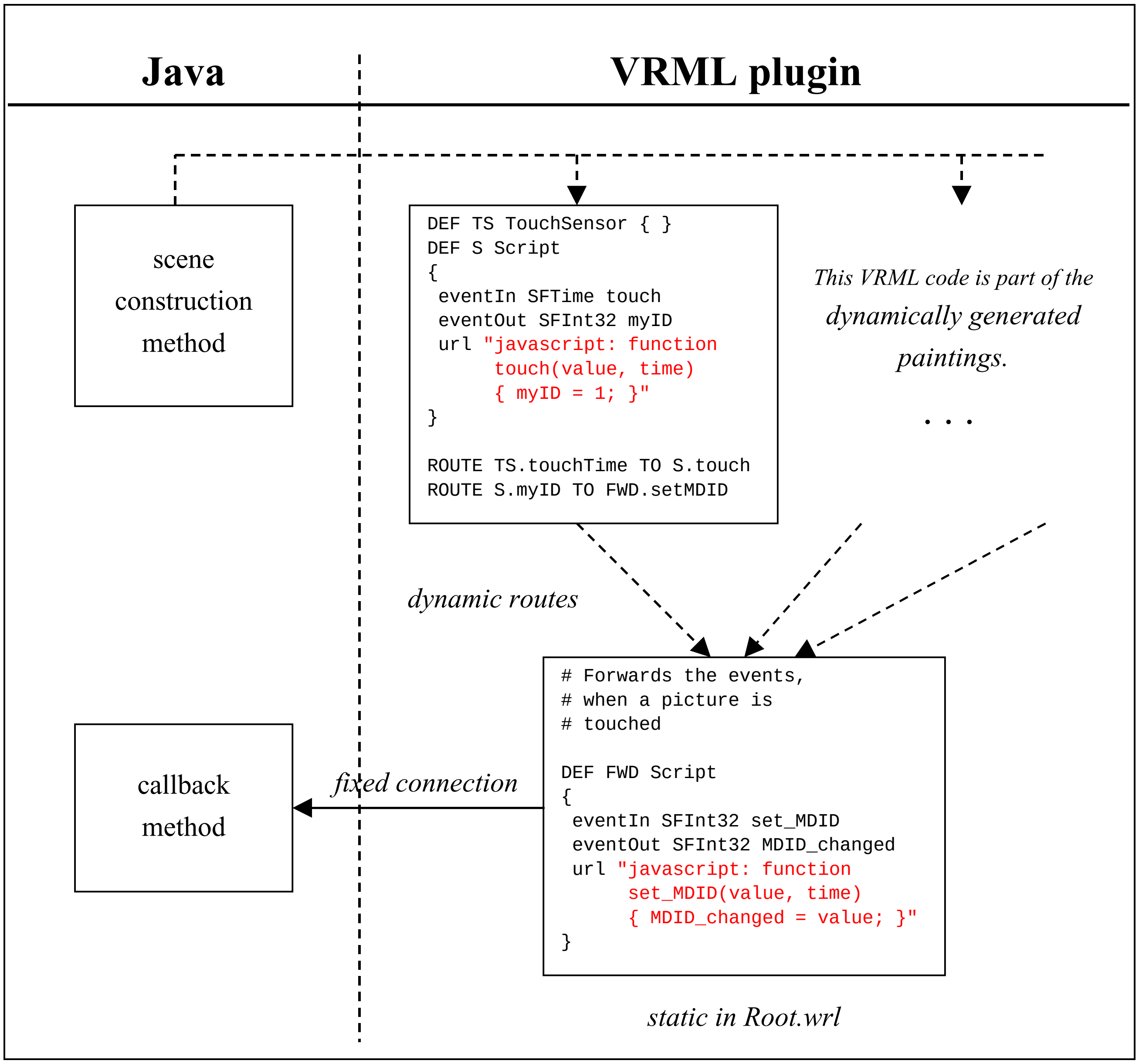
Figure 22: The callback method of the EventOutObserver interface

Interaction with the paintings

As already described in Subchapter 4.1.2, users can click on the description text (Title) of the painting to be moved in front of it. Furthermore, the image itself can be touched to retrieve further information which is displayed as HTML text. The concept for this kind of interaction with the paintings in the scene is a little bit different, because there is not just one single sensor node that needs to be registered with a callback method. Instead, we now have a large amount of sensor nodes which all trigger the same action, just with different arguments. The argument which is the ID of the MetaData row of the touched painting can be encoded in the eventType itself, but there is still the problem that we need a mechanism to add and remove the sources (TouchSensors) dynamically, together with the paintings.

Figure 23 illustrates how this problem is solved. Similar to the ProximitySensor for the user tracking, we add a Script node to the basic Root.wrl file. This Script node does nothing else than forwarding incoming events from an eventIn to an eventOut field. The callback method of the applet is registered with the eventOut field of the forwarding script, in the same way as with the position tracking. The crucial point of this architecture is that we are now able to add and remove internal VRML routes to the eventIn of the forwarding script dynamically, together with the paintings. Using this technique we need just one callback method that can identify the triggering object (in this case painting) by analysing the passed argument (ID).

The scene construction method has access to the MetaData information of each single painting and, when the VRML syntax for a painting's representation is generated, it writes the current MetaData's ID to the script code and adds the route to the forwarding script node dynamically to the scene. Applying this method, we can add and remove any number of paintings in the scene which have all their unique MetaData ID. Analyzing the content of the incoming event the callback method can find out which painting was clicked. Again there are different sorts of sources that are processed by the same callback method, so that we need to apply different identification values with the registration of the callback events. Besides the events from the paintings and the description areas, also the eventOut from the doors, that can be opened by clicking on the door knobs, are processed with the same method.


Java
VRML plugin
scene construction method
This VRML code is part of the dynamically generated paintings.
. . .
dynamic routes
callback method
fixed connection
static in Root.wrl

```
DEF TS TouchSensor { }
DEF S Script
{
 eventIn SFTime touch
 eventOut SFInt32 myID
 url "javascript: function
      touch(value, time)
      { myID = 1; }"
}

ROUTE TS.touchTime TO S.touch
ROUTE S.myID TO FWD.setMDID
```

```
# Forwards the events,
# when a picture is
# touched

DEF FWD Script
{
 eventIn SFInt32 set_MDID
 eventOut SFInt32 MDID_changed
 url "javascript: function
      set_MDID(value, time)
      { MDID_changed = value; }"
}
```

Figure 23: Concept to route multiple events to one eventOut field

When a painting is selected, the callback method just uses the `toHTML()` method of the MetaData wrapper class that was identified by the incoming event and displays the HTML result directly in the rightmost frame, utilizing the method described in Subchapter 5.1.2.

When the description text is selected, the current position and orientation of the users viewpoint is linearly interpolated to a predefined position in front of the painting. These viewpoints are being pre-calculated when the room is constructed, and the distance to the painting is chosen in a way that the whole painting fits exactly into the display area of the VRML plugin.

A more complex mechanism is triggered when the user opens the door to a room. First, the door of the currently active room is closed automatically, and the geometry of this former room is removed from the VRML scene graph, whereas the internal Java-classes scene structure remains in the memory. Subsequently, the chosen room's content is constructed in case that it is entered the first time, or retrieved from the internal cash. After the room is then added to the VRML scene graph, the door opens.

In this chapter, we have explained the basic concepts and some details of the most important parts of the implementation of TVIG. At several points, we mentioned the generic and open design of the architecture, which makes it very easy to adapt the system for new applications and future developments (see also Chapter 7).

6. Performance Measurement

In Subchapter 2.3 we quoted "good" performance as a design goal of the TVIG system. Now, we will measure the performance of the implemented system which is composed of three main parts: The performance of the underlying database system which primarily depends on the database scheme. The network performance and, finally, the performance of the 3D visualization which is split into the construction and the display part.

The network performance is variable and unpredictable, dependent on the current network connection and load. Of course, a 28.8 Kbit/s modem connection will lead to less satisfying results than a 10 Mbit/s Ethernet connection. Therefore, we performed the measurement locally at the server machine to ensure the network share to be constant. In this case, we can assume the data transfer time as nearly zero (compared with the database retrieval and scene-construction times). Nevertheless, one has to consider that a variable factor has to be applied in a realistic working environment.

The database performance usually depends on the database scheme and the applied indexes on the most frequently accessed tables. Our system utilizes a relatively small database scheme with less than 20 tables and, therefore, the system can easily manage several ten thousands of database rows. Of course, the retrieval time increases with bigger database content, but this factor can be considered linear, because we do not need to apply "joins" to retrieve the information from the "MetaData" table.

The performance concerning the construction and display of the VRML scene depends highly on the client machine's processing power. First of all, there are performance differences between the various implementations of the VRML plugins. The time for the scene construction is partly used in our own implementation of the internal scene representation and partly by the implementation of the EAI, i.e., the internals of the browser. Displaying the VRML scene is the most calculation-intense part of the whole system, and the displaying speed can only be increased by additional 3D graphics hardware.

The following numbers have to be seen as a qualitative measurement, because the numeric values vary with the network load, processing power and graphics hardware. For the measurement, we enabled all additional VMRL features (textures, lights and signposts) to survey the upper limitation of the scene construction algorithm.

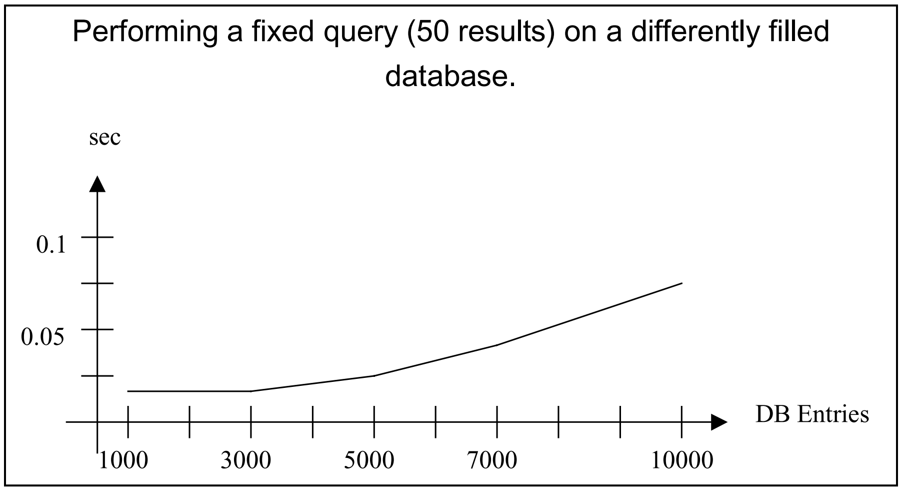


Figure 24: Measurement of the database scheme

The first test result (Figure 24) is interesting as well as surprising. The same query that retrieves 50 rows from the database was performed on the database with different number of entries. The query executed a search on only one column of the "MetaData" table, but even when the database contains 10000 entries the retrieval time is still below one tenth of a second. For such simple queries, we can assume that even several hundred thousands rows would not cause any unbearable retrieval times. We also performed more complex queries on the "MetaData table", filled with 10000 entries. A query that requires different kinds of searches on all columns of the table and has only 3 results is, for example, still executed in less than 2 seconds.

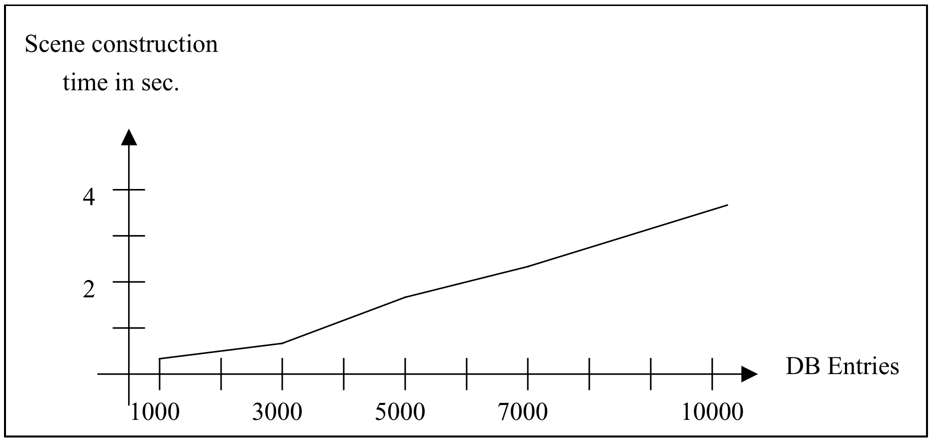


Figure 25: Measurement of the scene construction algorithm

The second test (Figure 25) measures the performance of the scene construction algorithm and shows that it could easily construct a gallery with more than 10000 paintings, without taking more than 5 seconds. This result is more of theoretical value, since the user is not supposed to formulate queries that return more than about 100 database rows. We have also measured the time that is required to initialize the MetaData wrapper classes and to store them in a Hashtable. This time already reaches an unacceptable value of 20 seconds, with just 1000 database rows in the resultset. This proportionally high value is caused by the calculation of the hash-value when an object is added to the Hashtable. If a better performance for displaying more than 500 results is required, the Hashtable should be replaced by a vector in the next version of the implementation.

The following table measures the performance of the EAI methods that were used to parse the VRML syntax and to add the new geometry to the scene. These time values are caused by the internal implementation of the VRML plugin and cannot be improved (except by better implementations of VRML browsers). For this test, the time for adding a room to the scene was measured, i.e. parsing the VRML String and using the addChildren method. The time was protocolled for several rooms with different amounts of paintings. Of course, the time depends on the size and resolution of the images, and usually the download time for the network has to be added.

# paintings	Parsing	adding the nodes
3	6.388	0.501
4	6.579	0.531
6	9.173	1.172
8	11.317	1.031

Figure 26: Measurement of the EAI methods (in sec.)

The whole system consists of 39 Java classes (271kb). It was tested on several PCs and it even reacts with an acceptable speed, running on a Pentium Laptop with a 28.8k modem connection. One point that we were not able to investigate in the present version is the performance of massive concurrent access, because the currently used JDBC driver does only support two concurrent database connections. But no unpredictable limitations need to be expected since most DBMSs are designed to handle multi-user access very well, and all the visualization related computations are performed on the client side.

7. Conclusion

Throughout this paper, we have discussed the advantages of 3D visualization on the WWW and how technologies such as Java and VRML can be employed to improve the quality of exhibiting art objects through an electronic system. VRML is still under rapid development, and one can be excited about future enhancements and possibilities. The next revision of VRML will extend the current standard with features for describing multi-user environments. This will open completely new application fields.

We presented a first implementation of our art-gallery service. The objective of this first implementation was to evaluate up to what extend the new technologies could meet art-display requirements. Regarding these requirements, we can summarize that we developed one consistent and easy to use system with query functionality (database access) and 3D visualization for the query results. The application is highly interactive such that the user can configure the scene construction and specify individual queries. In addition, the user can interact with the paintings in the scene. Finally, as the last chapter showed that the performance is sufficient to run the system on a Pentium Laptop with a 28.8k modem connection.

Our system utilizes a generic database scheme and a generic scene construction algorithm for flexibility reasons. It can easily be used for other visualization and exhibition needs. For example, it is conceivable to integrate video and audio sequences or even to add shelves to the rooms, where objects modeled in VRML can be exhibited. Such changes just require the development of new visual classes but no changes to any part of the scene construction algorithm. Because of this generic design it would, for example, be easy, to build a virtual supermarket as a future application of the system.

However, additional work could improve the system. In art-related systems users could benefit from extended search functionality that is not just plain text- but also content-related. If, for example, we apply such image retrieval algorithms (histogram functions) for shape and color similarities, as described in [WWF97, WWF97b], users could search for paintings with a sunset just by drawing a search pattern and/or specifying the corresponding colors.

Another enhancement for the sellers (galleries and artists) would be a built-in electronic cash transaction. This would reduce the time wasted by the administrator waiting for confirmation of payment before he registers new offers to the database. However, e-cash transactions have to be secure and are still an active research area, although there are already some first applications on the Internet which perform such transactions.

An easy to implement improvement is the capability to mark interesting paintings while exploring the gallery, to display them altogether in one room or to compare them in a slide-show presentation. This would ease the comparison of spatially distant paintings. Another improvement could be a predefined tour through the gallery which could help novice users who are not familiar with the navigation capabilities of VRML browsers.

8. References

[Bu97]	Buchmann, A.: Skriptum for “Datenbanken I” / Kapitel 6: Entwurfstheorie TU-Darmstadt
[CB97]	Carey, R.; Bell, G.: The Annotated VRML 2.0 Reference Manual Addison Wessley, Developers Press, 1997
[CDTG97]	Crossley, M.; Davies, NJ; Taylor-Hendr, RJ; McGrath AJ: Three-dimensional Internet developments, BT Technology Journal Vol 15 No 2 April 1997
[Dai97]	Dai, F.: Von Animation und Walkthrough zu lebendigen virtuellen Welten
[Date95]	Date, C.J.: An Introduction to Database Systems, Addison Wesley 1995
[Eck98]	Eckel, B.: Thinking in Java, Prantice-Hall PTR, New Jersey 07458, 1998
[GMD991]	Paradiso, A.; Lorenzo, Saladini; Hemmje, M.: Architecture and System Specification of VIRGILIO, Arbeitspapiere der GMD 991, May 1996
[GMD1093]	Paradiso, A.; Hemmje, M.: A Generic Refinement of the Virgilio System’s Desing and a Prototypical ArchitectureVirgilio, Arbeitspapiere der GMD 1093, September 1997
[GMD_R15]	Marcello, L’Abbate; Hemmje, M.: VIRGILIO – The Metaphor Definition Tool GMD Report 15, May 1998
[HMTL]	The Web Publisher's Illustrated Quick Reference
[JDBC97]	Sun Microsystems, Inc.: JDBC Guide, Getting Started Mountain View / CA 94043-1100 U.S.A, 1997
[MKIA]	Marrin, C.; Kent, J.; Immel, D.; Aktihanoglu, M.: Using VRML Prototypes to Encapsulate Functionality for an External Java Applet
[OL97]	Oh, J.; Lim, G.: Problems and Possibilities of the Experience of Modern Art through the Internet (Paper for INET97)
[Raj97]	Rajah, N.: Art After the Internet: The Impact of the WWW on Global Culture (Paper for INET97)

[RCRRB97] Roehl, B.; Couch, J.; Reed, C.; Rohaly, T.; Brown G.: Late Night VRML 2.0 with Java, Ziff-Davis Press, Emeryville / CA 94608, 1997

[SQL95] Morgan B., Perkins, J.: Teach Yourself SQL in 14 Days
SAMS Publishing, Indianapolis, Indiana 46290, 1995

[Sra98] Sracco, C. M.: Universal Database Management
Morgan Kaufmann Publishers, Inc., San Francisco / CA, 1998

[WWF97] Wang, J. Z., Wiederhold G., Firschein O.: Content based image indexing and searching using Daubechies' wavlets
Springer, International Journal On Digital Libraries ¼ 1997

[WWF97b] Wang, J. Z., Wiederhold G., Firschein O.:
Wavlet-Based Image Indexing Techniques with Partial Sketch Retrival Capability,
Stanfod University, Stanford / CA 94305, 1997