DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Relatório Técnico
Technical Report
DCA-001/97

Efficient Visualization in a Mobile WWW Environment

Alberto B. Raposo[1], Luc Neumann[2], Léo P. Magalhães[1,3], Ivan L. M. Ricarte[1]

[1]UNICAMP (State University of Campinas)
FEEC (School of Electrical and Computer Engineering)
DCA (Dept. of Computer Engineering and Industrial Automation)
{alberto,leopini,ricarte}@dca.fee.unicamp.br

[2]Computer Graphics Center (ZGDV)
Dept. Mobile Information Visualization
neumann@zgdv.de

[3]University of Waterloo — Computer Science Dept. — CGL
lpini@cgl.uwaterloo.ca

February 1997

Abstract

The facility of access to information in the World-Wide Web (WWW), the expanding availability of information technology, and the recent developments in the handling of multimedia data are all important steps towards a Global Information Infrastructure accessible to anyone, anywhere in the world. However, in order to achieve this accessible infrastructure, one should consider the aspects related to efficient communication. Some of these aspects are addressed in this work. A mobile WWW rendering application using VRML is introduced, the related problems are pinpointed, and approaches to overcome them are proposed. As a first result we have developed an application that filters VRML scenes to render only parts selected by the user.

Key words: World-Wide Web Application, Mobile Computing, VRML, Resource Adaptive Distribution.

Contents

1 Introduction **3**

2 Challenges of Mobile Computing **5**

3 Supporting Technology **8**

3.1 The World-Wide Web 8

3.2 The Java Language 8

3.3 Virtual Reality Modeling Language (VRML) 9

3.4 CORBA Technology 10

4 Application Scenario and Solutions **12**

4.1 Reducing the Scene Complexity 12

4.2 Task Distribution 14

5 Results **19**

5.1 Implementation details 20

5.2 Specification of the script 28

5.3 Deficiencies Encountered 30

6 Conclusions and Future Work **33**

1 Introduction

The emergence of the World-Wide Web (WWW) caused an impressive growth of the global information space, integrating and expanding the services offered by the Internet and other networks. However, the number of users is increasing rapidly and the information they need is becoming more and more complex, requiring new ways of communication and interaction with these networks. The challenge is to reach the Global Information Infrastructure (GII) [6], a step beyond the WWW which will allow more efficient interaction with the information and will be globally accessible.

Our paper focuses on the accessibility issue, particularly concerning the technology of mobile communications. Due to this rapidly expanding technology, mobile users with portable computers may access information anywhere and at anytime. These mobile data terminals are becoming increasingly integrated into the WWW [20, 22].

In general, a mobile WWW application uses mobile devices such as Personal Digital Assistants (PDAs) or notebooks to access remote WWW services. The required connection to the stationary WWW server can be established with communication facilities provided by the mobile device itself (such as the Nokia 9000 Communicator) or with a data capable (cellular) phone. The scenario is illustrated in Fig. 1.

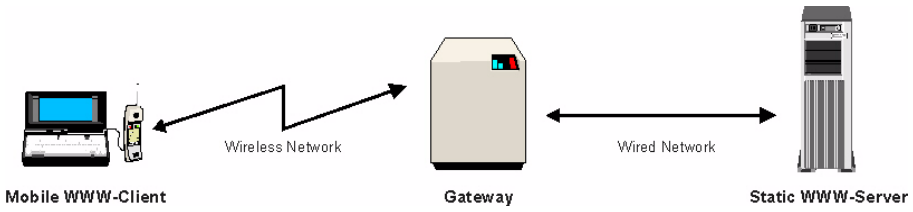

Figure 1: Mobile Application Scenario.

This new communication style significantly affects the requirements of WWW applications and faces new challenges [4, 17]. One challenge is the design and implementation of highly interactive applications that include the handling of non-textual bulky information. The main problems in handling such applications in a mobile environment are the resource constraints of the mobile data terminal and the narrow bandwidth of wireless networks. Thus, sophisticated concepts for the management and optimal utilization of resources within a mobile system are needed.

In this paper we address the remote access of VRML 2.0 [21] worlds from a mobile client, typically a laptop-like environment. In the section **Challenges of Mobile Computing** we present the peculiarities of mobile computing and its challenges. In the **Supporting Technology** section we introduce the tools used for the development of our soluction, which is introduced in the section **Application Scenario and Solutions**. Preliminary results are presented in the **Results** section, and in **Conclusions and Future Work** we come to the conclusions and propose future improvements on this work.

2 Challenges of Mobile Computing

A mobile application accessing information somewhere in the WWW should be regarded as a distributed application. However, a mobile application with the partially wireless communication link between the mobile device and the stationary server has to take into account different properties than an application using a wired stationary environment. A mobile environment is characterized by at least the following properties [13]:

- Limited resources of the mobile devices in terms of storage, battery power, memory and processing power relative to non-portable devices.
- Low bandwidth, low reliability and high costs of wireless narrowband wide-area networks (WANs). Typical bandwidths of currently available cellular systems are 9.6 Kbps or 19.2 Kbps. A new system offering up to 2 Mbps is supposed to be available after the year 2000. Table 1 shows bandwidths of many current and future mobile communication networks. Furthermore, a wireless network is less reliable than a fixed one due to temporary disconnections.
- Imbalance of resource availability between the mobile device and the stationary servers.
- Users have no fixed location and can move during a connection.

The introduction of interactive applications — such as a WWW application — used within a mobile environment will only be successful when the requirements arising from the handling of non-textual bulky data (*e.g.*, graphics, animation) can be fulfilled. In order to specify the requirements of an application and to estimate the degree of satisfaction one can use Quality of Service (QoS) parameters. In Table 2 some sample values for a number of QoS parameters for different media types are given. It can be inferred from the table that current data rates over wireless WANs remain poor compared with required data throughputs for multimedia applications.

A simple approach to overcome these problems might be to apply all the well-known mechanisms for a distributed application in a wired network also for a mobile application. However, they are designed for a higher bandwidth and richer resources at the end device.

Network	Switching-Technology	Bandwidth
GSM	Connection-oriented Packet-oriented	2.4, 4.8, 9.6 Kbps 9.6 Kbps
CDPD	Packet-oriented	19.2 Kbps
Modacom	Packet-oriented	9.6 Kbps
UMTS	Packet-oriented	$n \times 64$ Kbps (2 Mbps max.)

Kbps = Kilobit per Second
Mbps = Megabit per Second
CDPD = Cellular Digital Packet Data
GSM = Global System for Mobile Communication
UMTS = Universal Mobile Telecommunication System
(supposed to be available after year 2000)

Table 1: Bandwidths of mobile communication wide-area networks.

It will work, but the requirements in terms of throughput, delay, jitter, or response time will not be fulfilled. This especially holds for the handling of distributed interactive multimedia applications. Therefore, it is a challenge to develop appropriate techniques such as compression, progressive refinement, previewing, etc. for a mobile environment to solve these problems.

As pointed out by Watson [22], there are several communication optimization techniques:

Data compression increases the effective bandwidth compressing the information to be transmitted. Distillation [5] is an example of data specific compression developed in order to improve response times. However, even with the best compression techniques, the bandwidth of wireless communication is too narrow for an efficient transmission of non-textual bulky data.

Caching uses local storage to reduce communication. Coda [18] is an example of mobile system that uses caching to reduce bandwidth requirements and even to work disconnected. This technique increases the requirements on local storage resources.

Prefetching tries to antecipate data requests. This technique hides network latency, starting data transmission before the user requires it. However, if the prediction fails, communication and storage resources are unnecessarily used.

QoS Medium	Max. Delay (s)	Max. Jitter (ms)	Throughput (Mbps)	Bit error rate	Packet error rate
Voice	0.25	10	0.064	$< 10^{-1}$	$< 10^{-1}$
Video	0.25	10	100	$< 10^{-2}$	$< 10^{-3}$
Compressed Video	0.25	1	2–10	10^{-6}	10^{-9}
File Transfer	1	—	2–100	0	0
Real-time data	0.001–1	—	< 10	0	0
Image	1	—	2–10	10^{-4}	10^{-9}

Table 2: Sample values of QoS [19].

Data reduction filters the information to be transmitted. This technique is normally used in the context of resource negotiation (*e.g.*, a color video can be transmitted as greyscale when the client display does not support colors). Data reduction is also used to maintain the bandwidth requirements below a critical threshold. An example of a system that uses this technique is Odyssey [18], providing "application-aware adaptation".

Usual solutions address two main aspects related to the specific properties or limitations mentioned above. First, the transferred amount of data has to be as small as possible, which requires that parts of the application data should be processed and stored on the client side. The second aspect refers to the use of local resources such as processing power and storage space, which should be the least possible. Thus, the client should provide a presentation front end and communicate frequently with servers that process computing intensive parts of the application.

It can be easily seen that solutions for one aspect are counterproductive for the other aspect. A solution providing an appropriate balance between both aspects would overcome the most serious problems — namely the narrow bandwidth and limited resources — of a mobile environment.

3 Supporting Technology

In this section we present the tools used in the development of our solution.

3.1 The World-Wide Web

The WWW [1] is the most used resource of the Internet, incorporating support to multiple protocols (through the Hypertext Transfer Protocol, HTTP) and multiple digital data formats (through links in a hypertext written in the Hypertext Markup Language, HTML).

The WWW has a client/server structure, where the client side is composed by a navigator and browser tool, and the server side is composed by a daemon which receives requests from clients and delivers HTML files (usually called *pages*) and requested data. The client specify the Internet address of required information through an Uniform Request Locator (URL), which locates the server, the file with the data, and the protocol to be used.

When the client is linked to the internet through a slow connection, as it happens to be the case with mobile clients, some of the techniques described in the previous section are used to improve its performance. Caching may be used to store hypertexts and images locally in order to speed up reloads. Data reduction may be achieved by setting options such as *supress images*. However, such strategy would not be effective when manipulating VRML worlds through WWW clients.

3.2 The Java Language

Java [2, 8, 9] is a high-level programming language developed by Sun Microsystems, Inc., which is becoming very popular for the construction of highly interactive pages in the WWW. Small Java programs (applets) embedded in hypertext pages are transmitted through the Internet and executed in the local computer when these pages are accessed by a Java-enabled browser.

Java is platform-independent. A source program written in Java is translated into bytecode, which is interpreted by virtual machines installed in each platform. HTML provides the mechanisms to specify the URL of an applet bytecode, which is accessed by the server, transmitted to the client, and locally executed. Therefore, the same applet can be executed in distinct platforms, making it suitable for heterogeneous environments such as the Internet and mobile environments.

Java is also a general-purpose object-oriented language, designed to be small and simple. Java interfaces and classes are grouped into packages, with standard packages (`lang`, `util`, `io`, `net`, `awt`, and `applet`) providing the basic resources of the language.

Some of the provided resources are not so basic. The `java.lang` package supports multithreaded programming, *i.e.*, it allows programs to have many threads of execution at the same time, each thread carrying its local data and sharing global information. This is an important facility to multitask and distributed programming. The `java.net` has classes for manipulating socket connections and URLs — Java was the first language designed assuming that distributed computing resources are always available.

Since Java bytecode has to be interpreted by virtual machines, the performance of Java programs is still significantly worse than that of programs written in a platform-dependent language. However, based on its features, on the success it has obtained, and on the performance improvements it will certainly gain, we can expect that Java is going to be the one of the dominant languages for WWW and GII applications.

3.3 Virtual Reality Modeling Language (VRML)

The Virtual Reality Modeling Language (VRML) [21] is a file format for describing interactive 3D objects and worlds in the WWW, just as HTML describes hypertext pages. It allows hyperlinks to other VRML worlds or media, such as sound, animation, and hypertext.

The current version of VRML, 2.0, supports interaction with objects, prototyping, and animation resources. Interaction is provided by the definition of sensors and de-

tectors, such as time sensors, touch sensors, and collision detectors. Prototyping makes created objects available to anyone who wants to use them. Animation resources includes keyframe and scripting facilities that support custom protocols for many scripting languages, specially Java and JavaScript[1].

In order to view a VRML world a specific browser is necessary. Such browser may be an independent program, a Java applet, or a plug-in to a conventional Web browser. We used the Liquid Reality package [3], a set of Java class libraries that implements the VRML 2.0 specification, providing everything needed to write out, to render, and to manipulate the scene graph described by a VRML script.

3.4 CORBA Technology

The Common Object Request Broker Architecture (CORBA) is the product of the Object Management Group (OMG) that includes hundreds of computer-related companies. The CORBA standard [14, 15], first published by the OMG in 1991, defines an open object infrastructure to support application interworking across diverse architectures and infrastructures. It provides a distributed object environment that supports the location transparent invocation of methods on object and includes services — so called CORBAservices and CORBAfacilities — to build complex distributed systems more easily.

The Object Request Broker (ORB) and the Interface Definition Language (IDL) are the basis of this open object infrastructure. The ORB represents an object bus enabling client objects to make requests to and to receive responses from other local or remote objects. Furthermore, it provides means to manage the objects, advertise them, and describe their metadata.

IDL provides a unique notification to specify an Application Programming Interface (API) of a component. It is a neutral and declarative language, allowing to separate the interface specification from the implementation. In other words, a client application can invoke server functionality without taking care about the way the server is implemented (*e.g.*, programming language, underlying hardware). The implementation of object meth-

[1] JavaScript is a scripting language, with syntax similar to Java, supported by some browsers which enables to include the source code of a script into an HTML page. In this case, text — and not bytecode — is transferred to the client.

ods specified in IDL can be written in any language for which an IDL mapping exists. Currently mappings for C, C++, Smalltalk, and ADA are available. The mapping between Java and IDL is currently under work. IDL can be regarded as the glue that allows the interoperation of client and server objects across networks and operating systems independently of the programming language.

Using such an open object infrastructure in order to design a mobile application allows to think in terms of distributed objects. Instead of being only partitioned into a client and server component, as is the case for current WWW applications, a mobile distributed application can now be composed of several distributed objects.

4 Application Scenario and Solutions

As already stated, we focus on a rendering scenario where mobile users connected to the Internet via a narrow bandwidth communication channel request for the visualization of a certain VRML world, either static or animated. The description of the world is stored somewhere on an information server. This description will be retrieved, rendered, and displayed on a mobile client. However, the narrow bandwidth and the restricted processing power of the mobile data terminal request for intelligent strategies to enable an interactive handling of the rendered scenes. Simply stated, the straightforward approach of retrieving the scene description and rendering on a mobile client is not an adequate solution, especially if we think of more complex scenes.

In this section, we present two complementary approaches for the solution of this problem. In the first approach, **Reducing the Scene Complexity**, we simplify the rendering process in the mobile client by reducing the complexity of the scene to be rendered, allowing users to select which elements of the scene they want to visualize. The second approach, **Task Distribution**, complements the former, improving performance of the rendering process in the mobile client by distributing parts of the related tasks among the resources of the stationary servers in the wired network.

4.1 Reducing the Scene Complexity

Rendering is notably a process that requires a large amount of processing power. Due to the limited resources of mobile devices, techniques are necessary to simplify that process. One straightforward approach is to filter the data to be transmitted and to render only the parts of the scene that are actually of interest.

In order to demonstrate this strategy, we have developed an application capable of selecting the elements (*i.e.*, geometric objects, light sources, and cameras) of a remote VRML world, which are going to be rendered and visualized in the mobile client [16]. This application runs in a mobile terminal, which is connected to an application server. In this server there is a program capable of reading the VRML world, located in any WWW server, parsing it and sending its hierarchical structure back to the client. The

client provides a user interface adapted to the hierarchical structure, enabling the user to select the elements of the world he/she wants to see. The selected elements are then sent to the application server, which is capable of parsing the original VRML file and extracting from it only the desired elements, sending a valid “sub-VRML” world to the client, which can finally render the scene. This approach is illustrated in Fig. 2. The application is executed in the client as a Java applet downloaded with an HTML page.

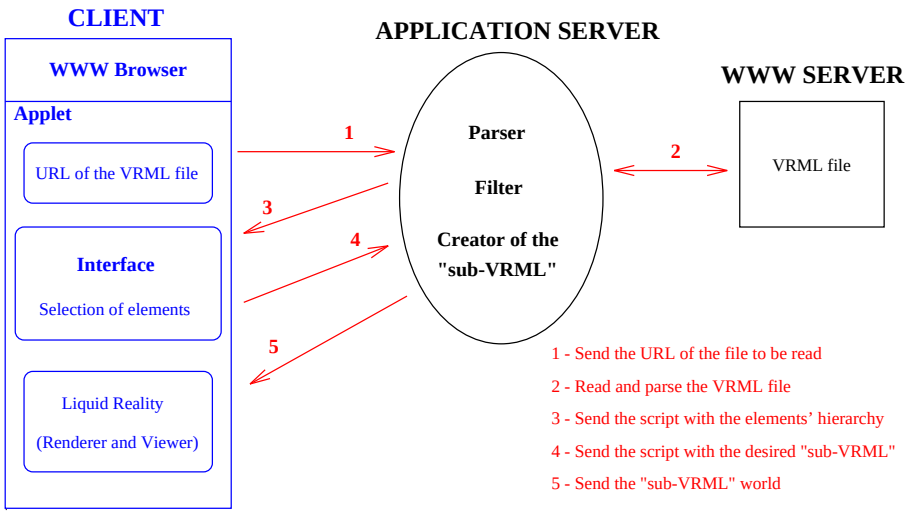

Figure 2: Strategy to visualize only a subset of a VRML 2.0 world.

The process begins by calling the application using any Java-enabled WWW browser. At this moment, a connection is established between the client and the application server. The next step is to send the URL of a VRML world to the application server (this is represented by arrow labeled 1 in Fig. 2). The application server will then use Java resources to connect with the server of the VRML file, read and parse it (arrow 2 in Fig. 2). The first output of the application server is a text script, representing the hierarchy of the elements of the VRML world, which will be sent to the client (arrow 3).

Based upon the received script, the applet creates an interface, which allows the user to select the desired elements of the world. After this selection, a new script is sent to the application server, describing the selected elements (arrow 4 in Fig. 2). Using this new script, the application server can create a new VRML world from the original one, by extracting only the desired parts of it. This final sub-VRML world is then sent to the client (arrow 5), that visualizes the results using methods of the Liquid Reality library [3].

Details of the implementation are presented in Section 5.

The division of the program into an applet (executed in the client) and an application server is needed due to security restrictions of some WWW browsers (*e.g.*, Netscape). Because of these restrictions, the client would only be able to read VRML worlds located in the WWW server of the applet page. To overcome this limitation, the application server was developed so as to act as a proxy server [11], a service to grab information from the WWW and send to whoever has requested it. The application server is able to grab that information because it is executed as an independent program, not subject to restrictions imposed to applets.

Another aspect that need to be considered is a good trade-off using both client and server resources in the rendering process. This is the topic of the following section.

4.2 Task Distribution

In this section we present a solution to optimize the rendering process by the distribution of the rendering work (tasks) among the available resources of the mobile client and the stationary servers. The optimization will primarily focus on a fast feedback to the user even over a narrow bandwidth network. The main approach is to use the knowledge about the application semantic data and the environment resources to distribute the tasks. Furthermore, user preferences defining the preferred trade-off between quality and transfer time will be used to control the distribution. This knowledge will avoid overloaded clients and servers and provide the best achievable quality of service combined with a short response time for the user.

Keeping the application scenario mentioned above in mind and considering a multi-tiered application architecture, then the rendering application can roughly be separated into the following components:

- Presentation logic;
- Application (or Rendering) logic;
- Data management.

This means that the rendering process is separated from the data and the user interface. That type of architecture is already well known in the area of business applications, and is depicted in Fig. 3.

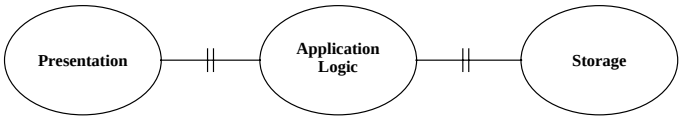

Figure 3: Three tier application architecture.

A straightforward approach to map this application functionality on a multi-server environment is to locate the presentation logic on the client, the bulk of the application logic on an application server, and let the data management to be done by a third server. This offers different partitioning possibilities, that range from a thin up to a thick client respective server.

Besides, the presentation logic should be separated into two further different functional objects as it is defined by the Model View Control (MVC) paradigm in Smalltalk [7]. Keeping all this together, a mobile client/server application is defined as a composition of objects that play together as illustrated in Fig. 4.

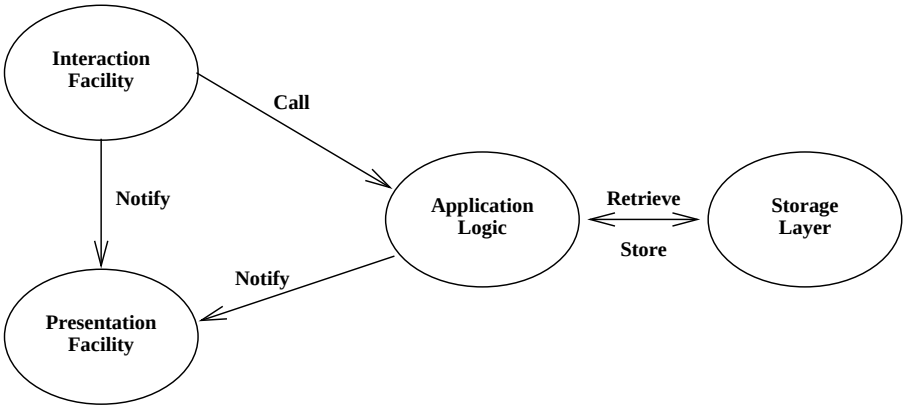

Figure 4: Basic Mobile Application Model.

For the best distribution of the different tasks to the suitable resources we propose an architecture with a *Resource and Task Manager* (*ResTaMan*) [12] that distributes and controls the rendering process using application semantic information. *ResTaMan* can

be regarded as a proxy located between the different stationary servers and the mobile client. In order to handle the application semantic information the filter introduced in the previous subsection can be used. It extracts the relevant parts of a scene to identify the tasks for the different renderers.

The establishment of such an architecture is based on two main points: the introduction of a semantic content header for the scene description and the application of Java-based ORBs to integrate the distribution architecture into the WWW. Independently of the separation strategy the distributed rendering architecture will encompass at least

- a representation facility (e.g., located on the mobile client),
- several VRML renderers with different properties (on several stationary servers),
- a VRML analyzer and a distribution manager (part of the *ResTaMan*) to identify and distribute the different tasks, and
- a composing and synchronization tool for the presentation (*ResTaMan*).

The VRML analyzer reads the semantic header and the world description. The header contains additional information about the content of the scene. It can describe which objects are in the background (BG) or in the foreground (FG), define different rendering qualities for the scene, or establish the priority of objects. The semantic header can be regarded as metainformation about the scene supporting the identification of subtasks to be distributed. Here, the filter introduced in the previous subsection may be used to extract the relevant parts of a scene for the different renderers. In combination with an Open Distributed Processing (ODP) Trader [10], which is a yellow page service knowing the properties of the environment resources, a good utilization of the rendering resources on the mobile client and in the fixed network can be achieved. Finally, an image composer and synchronizer is necessary to display the result.

Fig. 5 shows the different components and the dataflow between them. In this case the metainformation describes the foreground and background of the scene. The task for the foreground rendering is done on the mobile client and the background rendering is performed on several renderers on the fixed network. The Request Manager receives the requests from the mobile client, invokes the VRML analyzer and distribution manager.

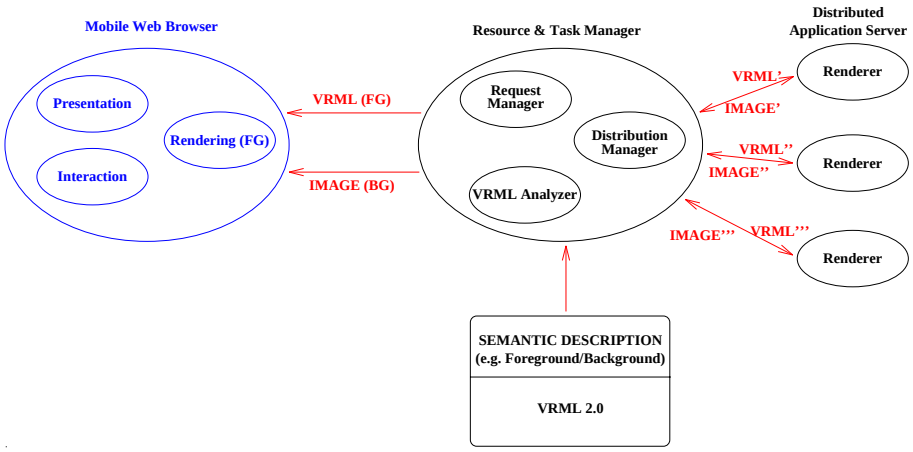

Figure 5: Distribution of VRML Rendering.

The distribution manager delegates and schedules the identified subtasks. It can be seen as a master component, that instructs various resources as workers to perform a subtask.

For the integration of *ResTaMan* in the WWW environment we will use Java ORBs. They enable the WWW browser to gain access to arbitrary CORBA services. That means it allows the direct integration of computational services in the WWW. A Java applet can be downloaded to the WWW client and serve as CORBA client accessing services using the Internet Inter-ORB Protocol (IIOP) [15]. In our approach it contacts the *ResTaMan* object that represents a metaserver from the viewpoint of the client. The entire rendering application consists of a set of interworking objects playing together via an ORB. For the transmission of images or image sequences we use a stream oriented connection between the Applet and *ResTaMan* that may be established with an IP socket connection. Thus, for the bulk data transfer the stream connection is used and the control operations are transmitted via the IIOP. Fig. 6 illustrates the integration of our architecture with the WWW.

Nevertheless, VRML also describes interactive worlds. Therefore, we need a distributed event mechanism to send information about the occurrence of a specific event in the client to the appropriate remote object. CORBA introduced an *Event Notification Service* allowing objects to register their interest for specific events or to inform interested parties about the occurence of events. This is all controlled via a specific object called Event Channel. We can use this event channel object in our environment to handle events

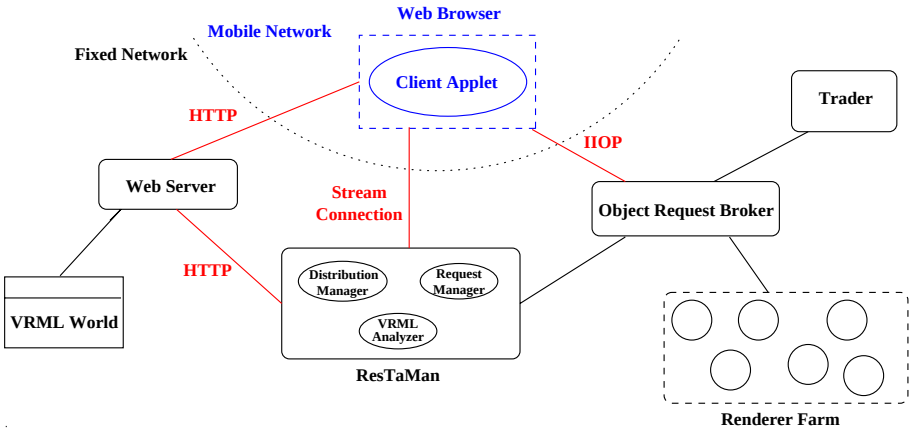

Figure 6: Integration of the Distributed Rendering Architecture into WWW.

and to inform the respective rendering object.

5 Results

An application (written in Java) was developed in order to test the script filtering strategy. This application is designed to run in a terminal (fixed or mobile), which is connected to the application server. In this server there is a program capable of reading the VRML world (located in any WWW-server), parsing it and sending its hierarchical structure back to the client (the mobile terminal). The client, based upon the data received from the server, creates a simple graphical interface that gives the user the opportunity of selecting part of the world he/she wants to see. The selected sub-tree (or sub-trees) is then sent back to the application server, which is capable of parsing the original VRML file and extracting from it only the desired parts, sending a valid sub-VRML file to the client, which can finally call a VRML 2.0 viewer.

The application runs in the client as a Java applet downloaded with a HTML page. The process begins by calling the application home page using any Java compatible Web browser, such as recent versions (higher than 2.0) of the Netscape Navigator. At this moment, a socket is created between the client and the application server[2]. The next step is to write the URL of a VRML world and send it to the application server. The application server will then use Java resources to connect to the server of the VRML file, read and parse it. The first output of the server is a text script, representing the hierarchy of the objects of the VRML world, which will be sent to the client. A brief specification of this script is in Section 5.2.

Based on the received script, the applet creates a simple interface which allows the user to select the desired sub-tree(s) from the file. After this selection, a new script is sent to the server, describing the selected parts. This script is similar to the previous, having simply the keywords *true* or *false* appended to the objects in order to indicate whether they are selected or not. Based on this new script, the application server can create a new VRML world from the original one, by extracting only the desired parts from it. This final sub-VRML world is then sent to the client, that will finally be able to see the desired world, using methods of the Liquid Reality library.

Partitioning the program into an applet (executed in the client) and an application

[2]Currently running in `caolho.dca.fee.unicamp.br:8079`.

server is needed due to a security restriction of the Netscape browser. As stated in [2], "applets run on the computer browsing your Web page, but they can only access files and sockets on the computer serving the Web page." Because of this restriction, the client would only be able to read VRML worlds located in the WWW server of the applet's page. In order to overcome this limitation, the application server was developed so as to act as a proxy server, a service that "grabs requested information from the Web and sends it to whoever requested it." The application server is able to grab that information anywhere in the Web because it is executed as an independent program, not subject to the applet restrictions.

5.1 Implementation details

The proposed approach can be used to accelerate the visualization of VRML worlds in mobile terminals by reducing the data transmission involved in this process. The scripts are small text files and the sub-VRML file is also a text file that can be small, depending upon which sub-trees were selected.

The developed application can also be seen as a starting point for a larger application, which would parse not only the objects structure, but also the textures, geometry, animation, and other aspects of the VRML files.

The main source files of the application are:

VRMLServer.java is the application server main program. It has an infinite loop that waits for a connection on its port. When this connection occurs, it creates a new thread, which deals with the client communication protocol, reading the VRML file, translating it into a script, and doing the final parsing of the VRML (*i.e.*, removing the undesired nodes from the original VRML), as requested by the client.

ReadVRMLFile.java is the program executed in the application server that reads the VRML file and translates its structure into a script.

ParseVRMLFile.java is the program, also executed in the application server, responsible for creating the sub-VRML file after the selection of objects by the client.

Interface.java is the client program, an applet. It deals with communication and interfacing, and is also responsible for the translation of the script into a tree data structure.

SimpleViewer.java is the renderer, which uses methods of Liquid Reality library.

The application server

The server is implemented by the `VRMLServer` class, whose main methos simply waits for a connection to the server at the specified port[3]. When this connection occurs, a socket is created between the client and the application server, and a new instance of the class `ServerControl` is established, using this socket. The new instance of `ServerControl` class will be executed in a separated thread, allowing more than one client be connected to the server simultaneously. The piece of the code created to do that is shown below:

```
// Creating a server, that waits for a connection
ServerSocket ss = new ServerSocket(8079);

// Continuously monitoring for connections
for(;;)
{
    // Creating a socket when a client asks for a connection
    Socket incoming = ss.accept();
    // Starting a new thread, that will run class ServerControl
    new ServerControl(incoming).start();
}
```

The `ServerControl` class manages the communication protocol between the client and the application server.

The communication starts with a signal from the client, which wants to read a VRML world. When the application server receives this signal, it becomes prepared to read the URL of the VRML world, which is the next information the client is going to send (arrow 1 in Fig. 2). After reading this URL, the application server connects to the WWW server where the desired VRML file is located, and reads this file (arrow 2 in Fig. 2).

[3] In the example, port 8079.

After these initial steps, the application server has already downloaded the original VRML file, which is now stored in a string variable of the program. This variable will be used to the creation of the script that will be sent to the client (`ReadVRMLFile`) and in the final parsing of the file, when the undesired nodes will be eliminated (`ParseVRMLFile`).

Creating the script

This task is performed by the `ReadVRMLFile` class. After reading the VRML file, the application server parses it to write a text script, that will be sent to the client. The application server reads each word of the VRML file (stored in a string variable) and calls the method `getKeyWords`, that will compose the script correctly, while the interested keywords are being found in the original file. The `getKeyWords` method is a method of the `ReadVRMLFile` class.

This method looks for the following keywords in the VRML file:

Shape establishes the beginning of a `Shape` node, *i.e.*, a geometric object of the scene. When this word is found, it is written in the script. Inside a `Shape` node, the method also looks for the auxiliary keyword `geometry`, that describes the geometric characteristics of the node. When this word is found, it is written in the script, and the program reads the next word in the file, that is the name of the geometric node representing the geometry of the object (`Sphere`, `Cone`, `Text`, and so on). This word is also written in the script.

PointLight, DirectionalLight and SpotLight are the keywords for illumination nodes. When one of these words is found, it is written in the script, and the program looks for the auxiliary keywords location and direction, describing some features of the light source.

Viewpoint establishes the beginning of a camera node. This word, when found in the original file, is written in the script, and the program starts the search for auxiliary keywords for the camera: position and orientation.

children indicates a level change. Every time this word is found in the file, we entered a deeper level of hierarchy (*i.e.*, the following nodes are children of the current one).

When this word is found, a global variable indicating the current hierarchical level is updated and the value of the new level is written in the script. The children nodes are enclosed by brackets. Because of that, [and] are also considered keywords (a] can determines the return to an upper hierarchical level).

Switch indicates the beginning of a `Switch` node, *i.e.*, a node that allows a choice among the various nodes defined inside of it. This word is written in the script and the method looks for the auxiliary keyword `whichChoice`, that indicates which node of the `Switch` is going to be visualized.

LOD is the keyword for the *Level Of Detail* node.

DEF is used to give names to specific nodes, that can be later reused. This word is considered by the method if it is related to any of the previous key nodes (`Shape`, `Viewpoint`, `PointLight`, etc), otherwise it is ignored. In the case of relating to a key node, the word is written in the script, followed by the name given for the node and the node itself (followed by auxiliary keywords, as stated previously).

USE indicates the use of a predefined node. This word is considered only if the equivalent `DEF` was not ignored (*i.e.*, `USE` will be written in the script only if it is related to a key node).

An important observation about the keywords `Switch`, `LOD` and `DEF` is that their definitions are enclosed by braces ({ and }). For this reason, these symbols are also considered keywords (like the brackets, for the children nodes).

After the creation of the script, it is transmitted to the client (arrow 3 in Fig. 2), which creates an interface for the selection of the desired objects. This interface and the details of the client side will be discussed in the next section.

The client

`Interface.java` is the program being executed at the client. It is an applet embedded in the application home page. When this applet is initialized, it opens a socket connection to the application server, that should be waiting for a connection (as explained

in the description of `VRMLServer`). At this moment, the applet also creates a simple user interface, with which the user will be able to select the VRML world and the objects of this world he/she wants to see.

On the top of the applet window (Fig. 7) there is a text field to write the Uniform Resource Locator (URL) of the VRML file the user wants to work with. When this name is correctly written in the text field, the user can click one of the two buttons labelled *Read it* to send this URL to the application server. After that, the application server will read that VRML file, parse it, and send the script back with the hierarchical structure of the VRML file (all this process was discussed in the previous sections).

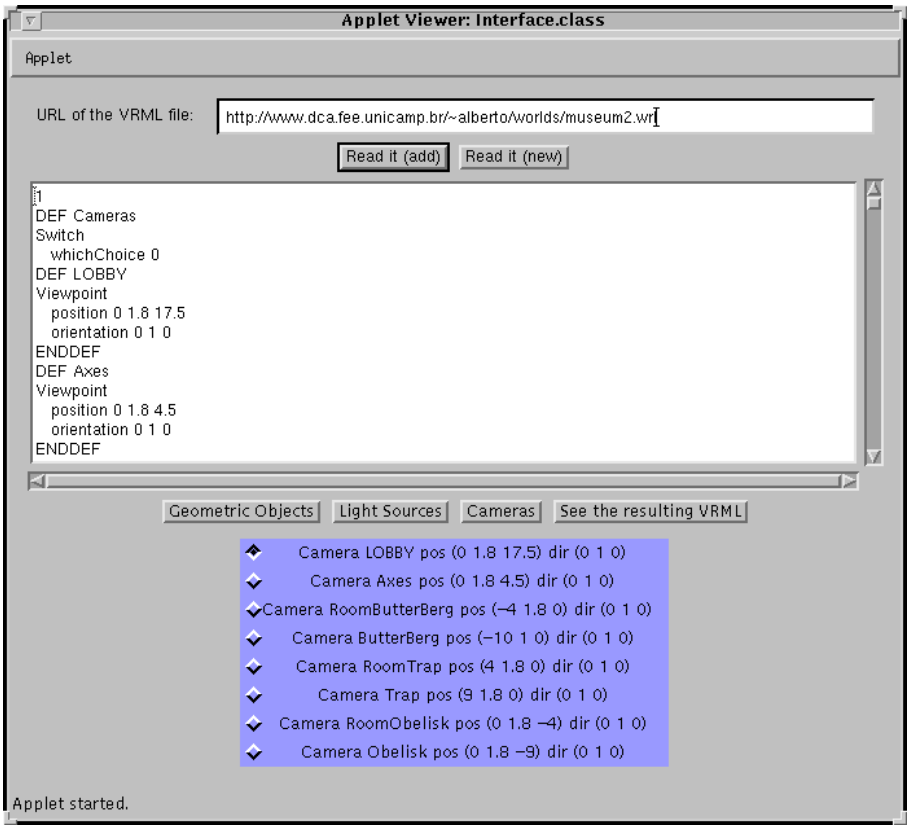

Figure 7: Interface of the developed application.

When the client receives the script, it is shown in the large text area below the two buttons. In fact, the script is shown in the applet only for demonstration purposes, since the user does not have to work on it. Actually, the user does not even have to know about

the existence of this script.

The user can select the desired objects of the scene using the buttons below the text area. The user should click the button *Geometric Objects* to select among the geometric objects of the scene, *Light Sources* to select among the light sources, and *Cameras* to select among the cameras[4]. When one of these buttons is clicked, a colored panel appears, with the name of all the objects of that kind in the first hierarchical level. If there are objects of the selected kind in a deeper hierarchical level, a button *Next Level* can be used to change the panel contents to the names of the objects in the next level.

By default, all the objects are chosen; the user has to click only on the objects he/she does not want to visualize. If the user deselects (selects) an object, all its descendants will be automatically deselected (selected).

Once the user has deselected the objects he/she does not want to visualize, he/she can click the button *See the resulting VRML* to visualize the created sub-VRML world. At this moment, the user work is finished; he/she must only wait for the image.

After the button *See the resulting VRML* is activated, the client rewrites the script, adding the words *true* or *false* after each object, indicating whether the object is selected or not. This new script is sent to the application server (arrow 4 in Fig. 2), which will create the sub-VRML world based on it. This is the topic of the next section.

A final observation about the applet interface should be made, regarding the difference between the two buttons *Read it.* The button *Read it (new)* makes the application server read the VRML world as a new file, throwing away the information about the sub-VRML world previously created. The button *Read it (add)* makes the application server maintain that information, merging two sub-worlds. In this way, the application allows that objects of different VRML worlds to be joined in another world — the tool can be used not only as an object-extraction tool, but also as a merging tool.

[4]It is important to note that the `Viewpoint` node (that defines the camera) of VRML 2.0 is a blindable node, *i.e.*, there may be many instances of it in a scene. However, only one instance can be active at any instant of time. If more than one instance is active, the renderer will use the last active viewpoint.

Creating the sub-VRML world

This task is performed by the `ParseVRMLFile` class. When the application server receives from the client the script containing the information about which objects were selected , it calls the method `searchInFile`, that functions similarly to the `getKeyWords` method, described in the previous section. The main difference is that the `searchInFile` method rewrites the VRML file (or, in other words, writes the sub-VRML file) based on what is read from the script, while the `getKeyWords` composes the script based on what was read from the original VRML file.

`searchInFile` is a method of `ParseVRMLFile` class that, like the `getKeyWords` method, searches the following keywords in the received script:

Shape, PointLight, DirectionalLight, SpotLight, Viewpoint: when one of these keywords is found in the script, the program reads the next word to know if this object was selected or not. If the word is *true* (*i.e.*, it was selected), the node is copied from the original VRML to the sub-VRML. If the word is *false*, the node was not selected and will not be copied to the sub-VRML file.

DEF: when a definition is found in the script, an additional search is needed to know if there is at least one selected node in that definition. A definition itself is not a node, but comprises one or more nodes. If there is at least one selected node in that definition, it should be written. Otherwise, if no node of a definition was selected, the definition will not be copied at the moment, but should be stored in a vector of unused `DEF`s, because it can be reused later.

USE: if this node was not selected, it will be simply skipped. However, if it was selected, the program must search the vector of unused `DEF`s. If it is the case of using a definition previously skipped (*i.e.*, stored in the vector of unused `DEF`s), the definition should be written in the place of the `USE`, and removed from the vector.

whichChoice: this word is considered as a keyword here because, based on the choice made in the client, the corresponding `Switch` node could have its value changed — the interface at the client side allows that the user changes the choice of such node. The new choice is transmitted in the script. So, when this word is found, it

cannot be copied from the original VRML file, but from the script received. This is a first experiment using the idea that the client could also be able to change features from the original VRML.

All other nodes that are not treated in this application will simply be copied from the original VRML file to the sub-VRML.

The dynamically created sub-VRML world is stored in the application server having the name `subVRML`*ddmmyyhhiiss*`.wrl`, where *ddmmyy* is the current date (day, month, year) and *hhiiss* is the time (hour, minutes, seconds) when it was created. This name will appear in the applet window at the client when the resulting image is visualized, and the user will be able to download the sub-VRML world he/she created.

Now that the application server has stored the sub-VRML world, it can send the name of this world to the client in order to allow the visualization (arrow 5 in Fig. 2).

Visualization of the results

To visualize the resulting sub-VRML world, the program `SimpleViewer.java` uses classes of the Liquid Reality library [3]. Liquid Reality is a set of Java class libraries that gives the user VRML functionality. In other words, Liquid Reality implements the VRML 2.0 specification, providing everything needed to write out VRML, to render, and to manipulate the scene graph. Although it is a commercial product, there is currently a beta version freely available on the Web[5].

When the client receives the URL of the created sub-world, it executes a method of `SimpleViewer` class, that utilizes the `BrowserView3D` class of Liquid Reality. The latter class is suitable for implementing browsers, since it supports the essential functionality of a browser, and parses the appropriate input. In fact, this class acts like a “black box” where the input is the URL of the VRML file and the output is a frame showing the rendered world; this frame appears in the applet window at the client.

In the `SimpleViewer` frame the user is able to visualize the sub-world he/she has

[5] `http://www.dnx.com/products/lr`

created and to “walk” through it — *i.e.*, with the mouse one can change the observer’s position, simulating a walk in the world.

5.2 Specification of the script

The script proposed to send the information regarding the hierarchical structure of a VRML world is simply a text file that represents each hierarchical level by a number. Every time the word `children` is found on the VRML file, a deeper hierarchical level is defined and when the definition of the child is finished, we come back to the upper level.

Besides the information about the hierarchical level, the script gives information about the objects of the scene (here, the term “objects” refers to geometric objects, light sources and cameras). For example, the script contains information about the geometry of the objects, the position and direction of light sources, etc. This information, although not directly necessary to select the sub-trees, can be used to help the user in knowing with which object he/she is dealing and to facilitate future extensions of the application — for example, the client could be able to change the characteristics of an object. To transmit this information, the script uses the same keywords as the VRML, thus ensuring consistency.

Fig. 8 represents a possible structure of a VRML file. The script for the file represented in this figure would be:

```
1
PointLight
   location x y z
2
DirectionalLight
   direction x y z
1
SpotLight
   location x y z
   direction x y z
0
Viewpoint
   position x y z
```

```
   orientation x y z
1
2
3
Shape
   geometry Sphere
Shape
   geometry Text
2
3
4
Shape
   geometry Cone
3
2
1
0
```

The x, y and z in the previous script are only meta-symbols, representing the numbers in the actual script.

Instancing (keywords `DEF` and `USE` in VRML) is also treated by the script — it delimits the definitions and transmits the reuses of them. The following example defines a sphere called `ball` at level 1 and uses it further in another hierarchical level.

```
1
DEF ball
Shape
   geometry Sphere
ENDDEF
2
3
USE ball
2
1
0
```

The nodes `Switch` and `LOD` of VRML are also considered in the script. The former allows to choose one among several children nodes — all the children and the choice are passed in the script. `LOD` node allows to give different definitions for an object, according to the distance from it the observer is located. All these definitions are passed

in the script. The next example shows a script with the `Switch` node used to select between two cameras (the first one is selected) and a `LOD` node, defining two shapes for an object.

```
1
Switch
   whichChoice 0
Viewpoint
   position 10. 10. 20.0
   orientation 0 0 1
Viewpoint
   position -10. 10. 15.0
   orientation 0 0 1
ENDSwitch
0
1
LOD
Shape
   geometry Sphere
Shape
   geometry Cone
ENDLOD
0
```

5.3 Deficiencies Encountered

The tests made with many VRML worlds have detected some problems in the application that have not been solved yet, but that should be corrected in a near future:

- The application does not consider nodes declared in other VRML files, *i.e.*, `Inside` nodes are ignored in the model of the hierarchical structure of the VRML. `Inside` nodes are used to make a link with a node that is located in another VRML file.
- A `Switch` node must have all its nodes of the same type — either geometric objects, or cameras, or light sources. Although generally a `Switch` node is used to

select among nodes of the same type, there is nothing in the VRML 2.0 specification that obligates all the `Switch` children to be of the same type. If this happens, the selection among the children might not work correctly.

- If a `Switch` node follows another `Switch` node with the same type of children, in the same hierarchical level, things may not work properly.
- If the number of nodes of the same type in a hierarchical level is too large, there could be not enough space for all of them in the client's interface window.
- The user interface is still very simple. A possible improvement is to present graphically the hierarchical structure of the world.

Other deficiencies of the application are related to problems of the current beta version of Liquid Reality. These problems are supposed to be solved in the final version of the tool:

- Liquid Reality does not run properly under Netscape; however, it works properly under the appletviewer of the JDK, the Java Development Kit[6].
- Liquid Reality does not implement the complete specification of VRML 2.0 yet. For instance, `Text` nodes and some kinds of textures are not implemented in the current beta version.

[6] `http://java.sun.com/products/jdk/`

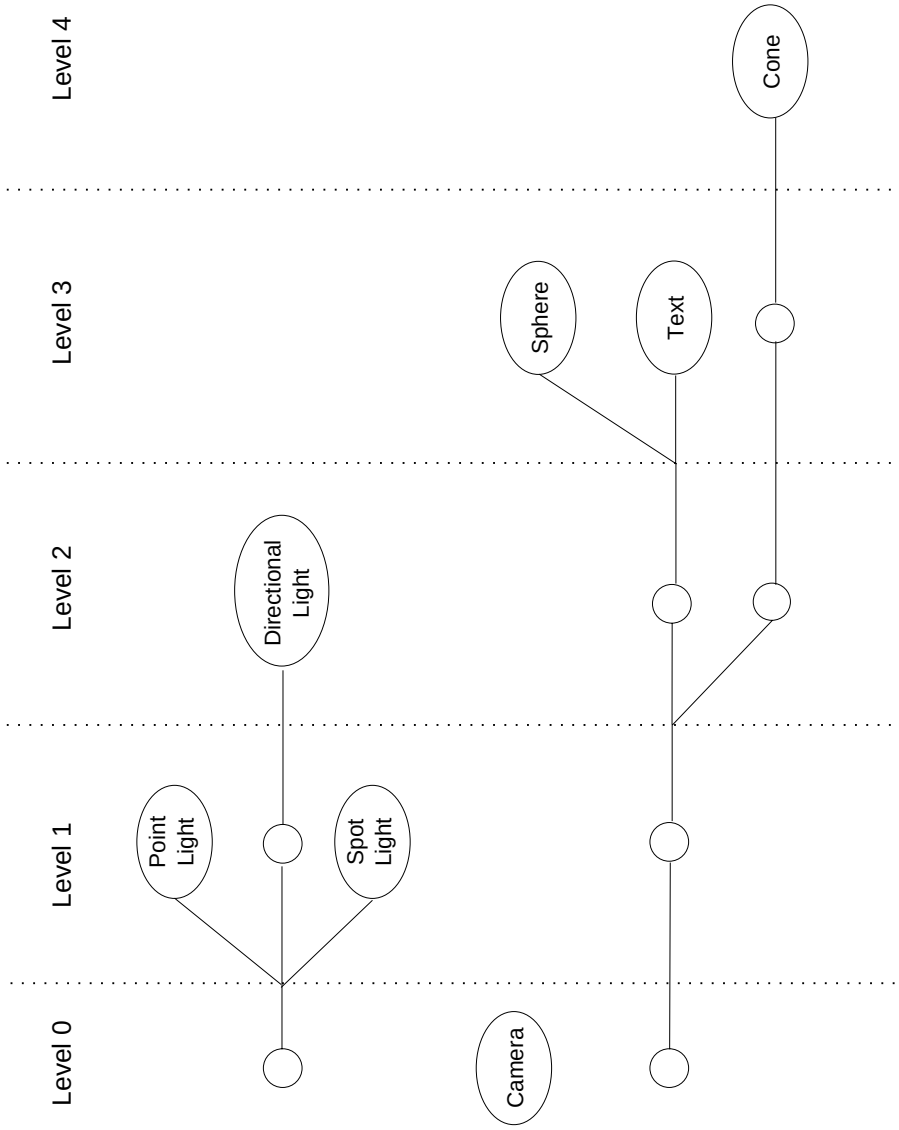

Figure 8: Example of a VRML 2.0 hierarchical tree.

6 Conclusions and Future Work

There is a growing need for tools supporting the interactive access, manipulation, and visualization of distributed multimedia information to realize the vision of “all information at your fingertip”. One great challenge is the improvement of the accessibility in the WWW using mobile devices. The main problems are the limited resources on the mobile terminal and the narrow bandwidth of the wireless link.

In this report we presented solutions aiming to provide a good trade-off between bandwidth and resource requirements. We developed an application enabling the user to select elements of a VRML world and proposed an architecture for the adaptive distribution of tasks in a mobile environment.

The application has been implemented to demonstrate and analyse our filtering approach. It reduces the utilization of client resources and the amount of data transferred. The next steps will be in the direction of enhancing the user interface making it more intuitive. This can be achieved for example with a graphical representation of the hierarchical structure of the VRML world. Further, we will work on other possibilities of the application like that of mixing elements of different worlds.

Our experience, however, showed that a technique to avoid the overload of the client using server resources was necessary. Therefore, we proposed an architecture in order to distribute tasks among the mobile client and the servers in the fixed network using application semantic data and environmental information. It divides the tasks to be done among several stationary servers to reduce the utilization of the client resources and to produce very fast first results. In principle the partitioning of the tasks can be related to the screen division or the scene content. Possible strategies are **quality level** or **scene partitioning**.

In the **quality level** strategy, one can define different quality levels that vary in the amount of work items for a rendering task. The scene can be rendered in parallel with different qualities from several renderers. Assuming the higher quality level, the longer it takes to render the image, quality can increase step by step like progressive refinement. The different qualities can refer to shading method, texture mapping, resolution of the image, or level of detail.

Using the **scene partitioning** approach the scene is subdivided into smaller parts to be rendered in parallel. One obvious approach is to subdivide the pixels among the renderers so that each of the involved renderers is responsible for a specific subarea of the entire image. If we are using the content of a scene, then we can distribute subparts of a static scene (*e.g.*, foreground and background) or subparts of an animation (*i.e.*, dividing into frame sequences).

The main advantage of our approach is the integration of efficient transmission and user interactivity. The former is achieved by the data filtering and distribution techniques presented. The interactivity of our solution is represented by the user selection of the elements he/she wants to visualize. In the current prototype, this interaction still requires some knowledge of VRML by the user, but the goal is to use semantic information (defined by the author in the *semantic content header*) to achieve as much transparency as possible in the process. In this way, the user would be able to select objects by the role they played in the world (as defined by the author). Another possibility is to have the author assigning levels of priority to objects in the world, with a high level to objects which are essential to the scene and a low level of priority to details and textures, for example. In this case, the user would have only to set up to which level of priority he/she is willing to wait.

Acknowledgements

This paper is a result of ProBrAl 002/94, a cooperation project between UNICAMP (State University of Campinas) and the Technical University of Darmstadt, sponsored by CAPES (Brazil) and DAAD (Germany). We deeply appreciate the support granted by these institutions. We would like to thank FAPESP and CNPq (Brazil) for the sponsorship of some authors. Special thanks to José M. De Martino, Dr. Rüdiger Strack and Jian Zhang for helpful pointers and clarifying discussions.

References

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, August 1994.

[2] G. Cornell and C. S. Horstmann. *Core Java*. SunSoft Press, 1996.

[3] Dimension X. *Liquid Reality*, 1996.
(http://www.dnx.com/products/lr/).

[4] George H. Forman and John Zahorjan. The Challenges of Mobile Computing. *IEEE Computer*, 27(4):38–47, April 1994.

[5] Armando Fox and Eric A. Brewer. Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation. In *Fifth International World-Wide Web Conference*, Paris, France, May 1996.
(http://www5conf.inria.fr/fich_html/papers/P48/Overview.html).

[6] N. Gershon, J.R̃. Brown, et al. Computer Graphics and Visualization in the Global Information Infrastructure (special report). *IEEE Computer Graphics and Applications*, 16(2):60–75, March 1996.

[7] Adele Goldberg and David Robson. *Smalltalk-80, The Language*. Addison-Wesley Series in Computer Science. Addison-Wesley, September 1989.

[8] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification — Version 1.0*. Java Series. Addison-Wesley, 1996.
(http://java.sun.com/doc/language_specification/index.html).

[9] James Gosling and Henry McGilton. The Java Language Environment — A White Paper. May 1996.
(http://java.sun.com/doc/language_environment/).

[10] *ISO/IEC DIS 13235: Information Technology — Open Distributed Processing — ODP Trading Function*.

[11] Ari Luotonen and Kevin Altis. World-Wide Web Proxies. In *First International World-Wide Web Conference*, Geneva, Switzerland, May 1994.
(http://www1.cern.ch/PapersWWW94/luotonen.ps).

[12] L. Neumann, J. Zhang, and R. Strack. Concepts for the Efficient Handling of Multimedia Data within a Mobile Environment. MOMID Deliverable 3, Computer Graphics Center, July 1996.

[13] L. Neumann, J. Zhang, and R. Strack. Evaluation of the Existing and Forthcoming Mobile Infrastructure. MOMID Deliverable 2, Computer Graphics Center, March 1996.

[14] Object Management Group (OMG). The Common Object Request Broker: Architecture and Specification; Revision 1.1. OMG TC Document 91.12.1, OMG, December 1991.

[15] Object Management Group (OMG). The Common Object Request Broker: Architecture and Specification; Revision 2.0. OMG TC Document, OMG, July 1995.

[16] A.B. Raposo. Extracting Objects in a VRML2.0 file. Technical report, ProBrAl 002/94, 1996.
(http://www.dca.fee.unicamp.br/projects/prosim/publiPS.html).

[17] M. Satyanarayanan. Fundamental Challenges in Mobile Computing. In *15th. ACM Symposium on Principles of Distributed Computing*, Philadelphia, PA, May 1996.
(http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html).

[18] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1), February 1996.
(http://www.cs.cmu.edu/afs/cs/project/coda/Web/docs-coda.html).

[19] Wassim Tawbi. *La Qualité de Service Dans les Systèmes de Communication Multimédia: Un Cadre d'Etude et Spécification d'un Protocole de Négociation entre Applications*. PhD thesis, L'Université Pierre et Marie Currie, Décembre 1993.

[20] Geoffrey M. Voelker and Brian N. Bershad. Mobisaic: An Information System for a Mobile Wireless Computing Environment. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, December 1994.
(http://www.cs.washington.edu/homes/voelker/mobisaic/papers.html).

[21] *The Virtual Reality Modeling Language Specification — Version 2.0, ISO/IEC CD 14772*, August 1996.
(http://vrml.sgi.com/moving-worlds).

[22] Terri Watson. Application Design for Wireless Computing. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, December 1994.
(http://snapple.cs.washington.edu:600/wit/presentations.html).