Extending Scientific Visualization into the World Wide Web

by

Zhihong Gao

A Thesis Submitted to the
Faculty of Computer Science
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF COMPUTER SCIENCE

Major Subject: Computer Science

APPROVED:

Dr. Philip Cox, Supervisor

Dr. Trevor Smedley

Prof. Thomas Emodi

DALHOUSIE UNIVERSITY – DALTECH

Halifax, Nova Scotia 1998

To my parents……

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CGI	Common Gateway Interface
FTP	File Transfer Protocol
GIS	Geographical Information System
GUI	Graphical User Interface
GSC(A)	Geological Survey Canada (Atlantic)
HCI	Human-Computer Interaction
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
MIME	Multipurpose Internet Mail Extension
SciVis	Scientific Visualization
SQL	Structured Query Language
URL	Universal Resource Locator
VSP	Visualization Service Provider
VRML	Virtual Reality Modeling Language
VUI	Visual User Interface
VSP	Visualization Service Provider
WWW	World Wide Web

ACKNOWLEDGEMENTS

I would like to express gratitude to my supervisor, Dr. Philip Cox, for assisting me in my thesis work. By sharing his knowledge and software industry experience, providing encouragement, and arranging for financial support, this project was successfully completed. Dr. Cox's dedication to excellence has served as an inspiration to me.

Appreciation is due to Dr. Trevor Smedley and Prof. Thomas Emodi for serving as members of my guiding committee, offering suggestions, and reviewing the thesis.

In particular, I wish to extend my sincere thanks to Mr. Ron Macnab in GSC(A) (Geological Survey of Canada, Atlantic) for his management of this project. I am also indebted to Dr. Karl Usow in GSC(A) for his technical assistance. Many other people at Daltech and GSC(A) have provided helpful advice and moral support.

A grant from the Nova Scotia Innovation Corporation is gratefully appreciated.

Last but not least, I wish to thank my parents for their understanding, encouragement and love.

ABSTRACT

Scientific visualization (SciVis) software generates graphical representations of data as a means of gaining understanding and insight into the data. Most in-house SciVis applications exist in scientific organizations, but these useful legacy applications are not widely distributed, making it difficult for multiple users in different locations to access them. The dramatic increase in the use of the Internet gives these SciVis software owners an opportunity to give a wider range of users remote access to their SciVis tools.

We propose a simple and quick standard re-engineering method to provide remote access to such legacy applications, following a systematic study of the issues involved in extending scientific visualization applications in this way.

In approaching this re-engineering problem, we have built upon two existing models: DiBiase's sequence model for scientific visualization [5] and the visualization pipeline architecture [39, 11].

DiBiase's SciVis sequence model, which has a great influence on the design and development of SciVis systems, has been extended to account for the issues relating to remote access. A model is proposed to extend SciVis into the global realm and to provide a basis for SciVis tools to be used for Web publication and public service.

The visualization pipeline, a high-level design architecture for SciVis, is also extended in our work, forming the re-engineering foundation. According to our extended visualization pipeline architecture, a general client/server approach for re-engineering an in-house SciVis application into one suitable for public use is illustrated with the Web as a low-cost deployment medium.

Our approach is demonstrated by re-engineering existing display software for geographic data developed at the Geological Survey of Canada (Atlantic), to provide a visualization client for the Web.

Chapter 1 Introduction

1.1 Scientific Visualization

Scientific Visualization, sometimes referred to in shorthand as SciVis [9], has been defined as by MacEachren and Kraak as "the use of sophisticated computing technology to create visual displays, in which numeric values in data sets are represented visually as colors, shapes, or symbols, and the goal of which is to facilitate thinking, problem solving and decision making" [22]. Since people seek to understand data in SciVis, it is sometimes referred to as *visual data analysis*.

The origin of the term "scientific visualization" is often traced to the publication of the much-cited National Science Foundation Report on Visualization in Scientific Computing [27]. However, as pointed out in [29], the role of SciVis in the history of scientific discovery can be traced to Plato, Kekule and Kelvin. SciVis is a fundamental investigation tool in many sciences as an "expression of " or "tool for" thinking. For instance, plotting thousands of data points on the 2D plane might easily demonstrate data relationships, minima, maxima, discontinuities, etc. For complex data, images might provide assistance to human information processing, enhancing mental visualization and the comprehension of 2D and 3D spatial relationships.

Today's SciVis emphasizes the role of computer display technology in prompting mental visualization and subsequent insight. A lot of effort has been put into SciVis research involving computer graphics, image processing, high-performance computing, simulation and other areas. Also today's SciVis emphasizes interaction between the user and the data rather than just static pictorial displays [33]. In this way, SciVis has helped scientific researchers to gain insight into the systems they study in ways previously impossible, by revealing structure, extracting meaning, and navigating large and complex information worlds. Many increasingly powerful in-house SciVis systems, such as GIS-SciVis Integration [34], have been developed to support scientific investigation.

The focus of the SciVis discipline has been on designing and validating, by experiments with users, effective ways of presenting and interacting with data, and creating internal representations of information in the minds of the users. Nevertheless, the increasing power and accessibility of computing and the Internet have created new challenges for SciVis, including giving Web users access to existing scientific visualization systems, which is the target of this research work.

1.2 From Scientific Visualization to Information Visualization

Through the availability of increasingly powerful computers with increasing amounts of internal and external memory, people are obtaining and storing vast amounts of data either from measurement (as in remote sensing, for example) or simulation (as in computational fluid dynamics, for example). Also, as the use of the Internet rapidly increases, it is becoming easier to access varied and large data sources. One opposite result of the information explosion is termed "information saturation", which refers to situations in which users simply cannot cope with the volume of information [18]. This is hampering new discoveries in many sciences because users must invest considerable time and energy in locating and studying necessary information. One of the solutions to this problem is to borrow the SciVis idea, that is, transforming gigabytes of data into meaningful patterns. When SciVis is used in the context of solving the "information saturation" problem, it is often called *information visualization*.

Information visualization puts into practice the adage " A picture is worth thousand words". Once a lot of data has been compressed into one picture the user can "browse" it, detecting correlations between different quantities both in space and time. During this process, information visualization might furnish new visual representations in addition to those which have already been produced. This opens up the possibility of viewing the data selectively and interactively in `real time' [9].

Information visualization can be further evolved on the Web by *information drill-down* to retrieve more related information within a region of interest in the visualized data [17].

1.3 The Progress of Scientific Visualization

Before the advent of computers, SciVis was performed with primitive technical methods such as blueprints or line plots drawn on graph paper.

Computer-performed SciVis started from the old model of batch processing simulation on a centralized mainframe, producing images from predefined viewpoints which are saved to disk or video for later viewing [43]. During the late 1970s and 1980s advances in computing power and network bandwidth brought high performance to desktop computing, which allowed a more interactive approach to image displaying handled at the desktop, permitting the scientist to interact with the simulation in near to "real time".

The scientific visualization sequence model proposed by DiBiase [5] has a great influence on today's SciVis community, affecting the way in which visualization systems are built [13, 25]. This model is explained in more detail in section 2.2. In this model, it is assumed that SciVis is used by researchers to reveal unknowns. To date the vast majority of both commercial and in-house visualization systems have concentrated on providing non-distributed solutions. *Non-distributed solutions* involve building software to run on a workstation or mainframe at the same location as the user. These solutions make public access from different locations difficult. Here and in the following, by *public* we mean the professional or scientific community with an interest in the data being presented.

The development of a SciVis system usually is a long and expensive process. Nevertheless, the usage of many non-distributed visualization systems is quite low after completion of the projects for which they were developed, since SciVis systems are frequently associated with complex, expensive and sometimes unique hardware and software to which the wider public is unlikely to have access. Furthermore, collaborative visualization is often necessary for solving larger and more complex research problems. *Collaborative visualization* occurs when a multi-disciplinary group of experts, likely to be geographically dispersed, use SciVis to collaborate on problems.

Many legacy non-distributed SciVis systems can still be very useful and valuable for continuing research and education. From a commercial view point, the owners of databases and associated SciVis systems might become *visualization service providers* (VSP), providing public information visualization services at their sites and enhancing the usage of their legacy SciVis systems. In this case, such existing SciVis systems need re-engineering for public access or collaborative visualization.

1.4 A New Paradigm for Scientific Visualization

Lemay characterizes the World Wide Web (WWW) as “a global, interactive, dynamic, cross-platform, distributed, graphical hypertext information system that runs over the Internet” [20]. It has been a primary means of acquiring and disseminating information and has been hailed as a significant enabling technology. With new Web technologies, such as common gateway interface (CGI), interactivity has been brought to the Web. The most significant technological impact on the Web is likely to be the rapidly increasing use of Java, which further increases the potential for interactive interfaces. User interface and graphic handling can be fully specified within the language giving far greater control over graphic objects. Java threading and two-way data communication allow interactive controls. With Java-enabled browsers becoming the standard, many organizations are now experimenting with the Web as a possible vehicle for delivering applications both to their customers as well as internally. These advances also provide an opportunity for enhancing the capabilities and increasing the usage of SciVis both by developing new Web-based SciVis software [43] and by re-engineering legacy SciVis systems to make them more widely available through the Web (the major focus of our work). SciVis in conjunction with the Web can make visualization capabilities available to end users without requiring them to have high-performance rendering systems. Indeed, because of the platform-independent nature of Web-client technology, this combination makes visualization facilities available on a wide variety of personal computers and workstations.

1.5 Contribution and Organization

This research deals with re-engineering of legacy custom SciVis systems for wider remote access and its related issues. The correctness and/or suitability of the existing graphic representation in these systems are not examined or addressed here.

In chapter 2 we first investigate DiBiase's sequence model for using scientific visualizations in today's scientific research. We then propose an extension of the SciVis sequence model to account for the wider, public use of such visualizations.

In chapter 3 the current state of the art in SciVis software is surveyed. It is found that the structure of such software usually conforms to an architecture called the "visualization pipeline" and that the majority of such applications are non-distributed. It is expected that commercial visualization applications will be or are being evolved by manufacturers into collaborative or Web-based versions. However, many scientific organizations like the Geological Survey of Canada, have valuable in-house SciVis systems which they would like to make more widely accessible and therefore need to be modified into multi-user applications.

In chapter 4 several possible scenarios for SciVis on the Web are discussed. We demonstrate that with new Web technologies SciVis applications not only can be used for Web publication, but also can be implemented on the Web as an interactive SciVis public service. Following our earlier definition of "public", by *public service* we mean a widely accessible and interactive information service for specialists with an interest in studying and analyzing the data that the service presents.

Given the need for deploying existing visualization applications in this way, in chapter 5 we turn our attention to considering how to provide a cost-effective, efficient and general approach to re-engineering in-house SciVis software into multi-user applications for public service without necessarily changing the existing system.

In chapter 6 this re-engineering approach is demonstrated by the implementation of the Web-based GSC(A) Display. Over many years (dating back to the 1950s), the Geological Survey of Canada, Atlantic (GSC(A)) has collected large amounts of geographic data and developed in-house display software (called GSC(A) DISPLAY) on a Unix platform to create color images of selected sets of data, such as the regional topography and bathymetry for eastern Canada, the NW Atlantic, and the Arctic. GSC(A) staff are presently using these data and GSC(A) DISPLAY internally and wish to give external users remote access to this in-house SciVis system to create and view these color images. They also wish to access to this system on various different platforms within GSC(A) such as Windows machines or Macintoshes.

In chapter 7 we summarize our results and discuss potential enhancements and future work.

Chapter 2 The Scientific Visualization Sequence

2.1 Overview

First we review DiBiase's SciVis sequence model for how SciVis tools are used in the research sequence. This SciVis sequence has a great influence on the design and development of SciVis software. We then look at the evolution of computing architecture, which has an impact on the SciVis sequence model. DiBiase's model was suggested prior to the advent of the Web and needs to be modified and extended to account for the much wider publication potential of this new medium. Finally we characterize SciVis as a multidimensional space. These dimensions are important considerations in designing new SciVis systems and re-engineering legacy ones.

2.2 Dibiase's Model

DiBiase [5] proposed a SciVis sequence model (Figure 2.1) for depicting the uses of visualization in the context of scientific research. Although this model was oriented towards visualization of information in earth sciences, it is applicable to any discipline in which large amounts of data need to be displayed and analyzed. This model focuses on how SciVis tools are used and who uses them. It has been widely used to guide the development of SciVis software as tools in all aspects of scientific research, from initial exploration and hypothesis formulation through to the "final" presentation of results. It is generally agreed that SciVis is used in two realms, a private realm and a public realm, and that it is used for visual thinking and visual communication throughout the research process.

2.2.1 Four Stages in the Progress of Research

In DiBiase's model, research progress can be idealized as a sequence of four stages:

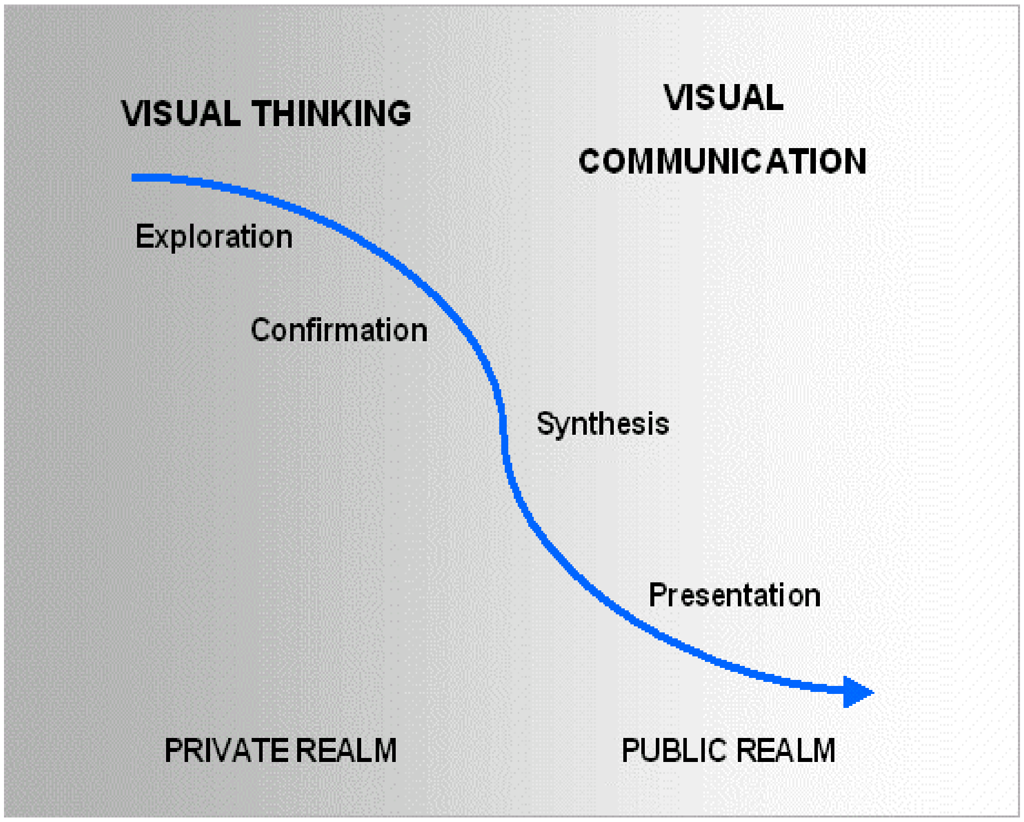


Figure 2.1 DiBiase's scientific visualization sequence model [5].

- *Exploration*: investigation of data to reveal pertinent questions and formulation of a formal hypothesis.
- *Confirmation*: confirmation of apparent relationships in the data in light of a formal hypothesis.

- *Synthesis*: obtaining new facts by generalization or inference.
- *Presentation*: presentation of the research at professional conferences and in scholarly publications.

2.2.2 Visualization Functionality and Realms

The intent of SciVis evolves in parallel with this progression. The intended use is indicated along the top of the diagram, while the stage of evolution is indicated along the bottom (refer to Figure 2.1). *Visual thinking* implies the generation of ideas through the creation, inspection, and interpretation of visual representations of the previously non-visible, while *visual communication* refers to the effective distribution of ideas in visual form. Visual thinking typically begins in a *private realm* of one or a few specialists who are intimately acquainted with the subject of the research. Finally, with the assistance of visual communication, the research is disseminated into a *public realm* consisting of the professional community with an interest in the data being presented.

2.3 Taxonomy of Scientific Visualization and Software Evolution

The Scientific Visualization Center, USA Department of Defense categorizes SciVis into three types [36]. They are:

- I See
- We See
- They See

This categorization is closely related to DiBiase's model as well as to the evolution of visualization software as described in the following.

2.3.1 “I See” Visualizations

“I See” visualizations correspond to the exploration stage of DiBiase’s model, in which a scientist or researcher investigates data collected from an experiment or other process. Initially, this researcher can use various “off-the-shelf” software applications to visualize the collected data in order to obtain a hypothesis. In advanced and complex projects, the researcher might need the support of software developers to build customized or integrated software. During the “I See” stage, the visualization is private and the software requirements may be changed or modified often for the initial exploratory purpose. Therefore the visualization representation can be simple, employing abstract symbols understandable only to the researcher, and the software is frequently operated by a simple command-line interface.

2.3.2 “We See” Visualizations

“We See” visualizations are responsible for the confirmation stage of DiBiase’s model. In the progress of the research project, other researchers might join the original solitary one, either for technical confirmation or to form a team as technical support. These people might be from different disciplines, have their own expertise, and do not have the knowledge the original researcher has. Therefore, the original researcher and the software developer must refine the visualization software to ensure that the representation is more concrete. Even through a command-line interface may still be used, the commands themselves must be more meaningful. In this way, the members of the research group can use the software by themselves for confirmation and further investigation after a short training. Using the visualization software, the research group can review and confirm the hypothesis of the original researcher as well as apply their own expertise to find new hypotheses about the visualized data.

2.3.3 "They See" Visualizations

"They See" visualizations are related to the synthesis and presentation stages of DiBiase's model. When a project reaches a final point, the research findings have to be synthesized (generated) and presented to the audiences with some interest in the results. These audiences also include funding officials or upper level management, who are likely to have less technical background. Visualizations tend to be the most elaborate in this phase, and often include videotape animation, slides, and other presentation quality formats which provide an overview of the key results. Such presentations frequently include a demonstration of the visualization software. Therefore, the research group and the software developer may need to work together to modify the visualization software to ensure that the images produced are suitable for the target audience. Because the presenter is usually the original researcher or a member of the research team, he or she is unlikely to have any difficulty in operating the software with the command line interface developed in the "We See" visualization stage. Since untrained users do not directly use the software, it usually remains at this level without any more attention. At most a simple GUI may be added for the trained internal users to perform routine work.

2.4 Evolution of Computing Architecture

Computing architecture has evolved from monolithic to distributed, with a consequent affect on the structure and usablity of software.

2.4.1 Monolithic Systems and Mainframes

The original computers were large mainframes, generally inaccessible to all except a small number of technical professionals. The first attempt to make computing more widely accessible consisted of time-sharing operating systems managing a number of simultaneously connected “dumb” terminals. Software systems written for mainframes were *monolithic*, that is, the user interface, logic, and data access functionality were all contained in one large application. The terminals used to access mainframes did no processing so the entire application ran on the mainframe itself. This is the architecture on which most legacy visualization applications are built.

2.4.2 Client/Server Architecture

The advent of microcomputers and workstations made possible a dramatic shift away from the monolithic architecture of mainframe-based applications. Whereas these applications require the mainframe itself to perform most processing, applications based on the client/server architecture allow some of the processing to be offloaded to workstations on the users’ desktops, making more efficient use of central services.

Client/Server applications typically distribute the software components so that the database resides on the server, the user interface resides on the client, and the logic resides in either, or both, components. When changes are made to parts of the client component, new copies of the client component have to be distributed to each user.

2.4.3 Web-Deployed Architecture

With the arrival of the Web the computing load moved back to the server as CGI scripts. Simple user interfaces in the *Form* format could be downloaded to the client on demand. This eliminated the requirement to deliver new copy to each user if the client component is updated. However, HTML and its extensions are not a programming language and can deal only with very simple tasks in the client. Moreover, this architecture created problems such as overload on the server and slow communication of large graphics.

With this initial Web architecture, SciVis Web publication is possible, but only in static form. *Web publication* means Web delivery of interim and final results in the context of SciVis.

2.4.4 Web Plus Java Applets

The adoption of Java has enabled web-based applications to become more distributed and more dynamic. One of the primary uses of Java is to write Applets, small applications that are downloaded to the client on demand through the Web. Applets, written in Java, a modern, powerful object-oriented derivative of C, can implement more complex user interfaces than are possible with HTML forms. They can also implement some of the logic of the overall application, thereby moving some of the computing load off the server. These capabilities of applets make a SciVis public service feasible.

2.5 An Extension of DiBiase's Model

To account for the wider publication potential offered by the Web, we take DiBiase's model one step further to include a *global realm*, as shown in Figure 2.2, within which visualizations of large bodies of data can be displayed to a wider audience, who can also benefit from visual thinking functionality of visualization (see section 2.2.2). It is important to note that by giving wider access to SciVis systems, one greatly increases the potential for further scientific discovery.

Considering the Web influence, we suggest a fourth level of visualization should be added to the three proposed by the Scientific Visualization Center outlined above, namely *World Sees* visualizations.

In this additional category of "World Sees" visualizations, the research group uses the Web medium to deliver to the world its newest research results with images without any delay due to the other traditional publication media, such as journals and conferences.

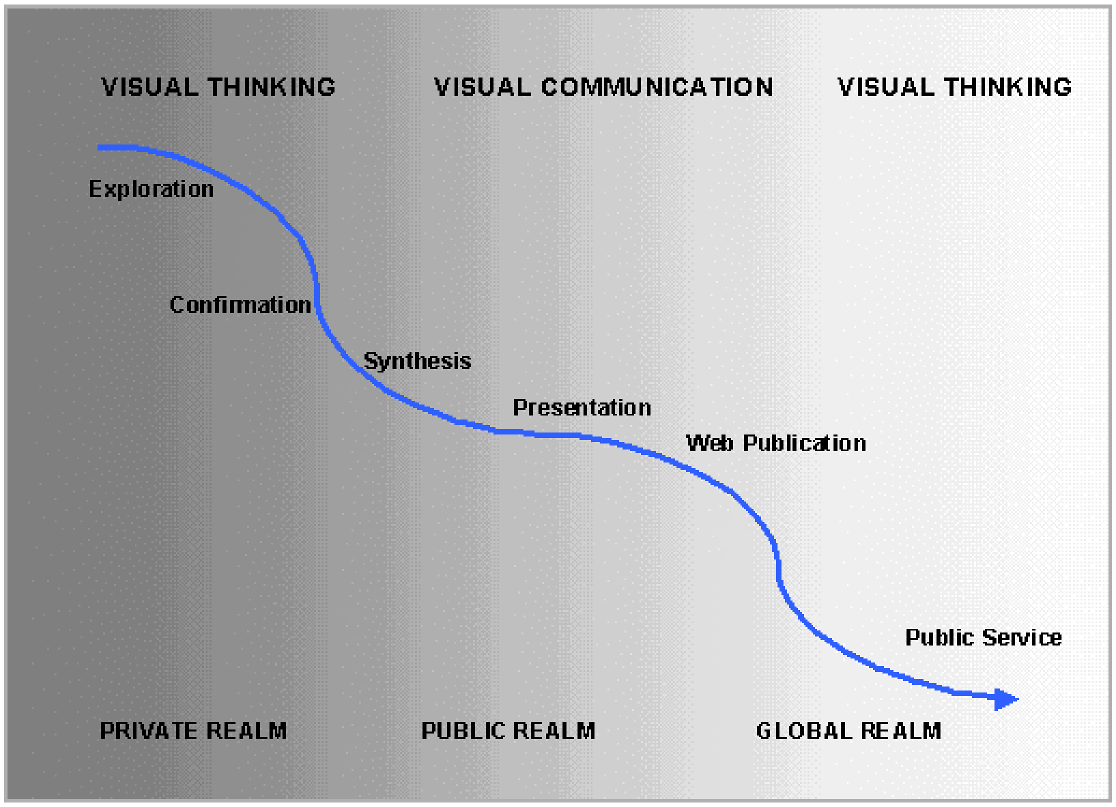


Figure 2.2 Extension of DiBiase's Model.

They use the developed SciVis system to generate high quality images and organize those important images with other related documents into a Web format.

However, "world sees" visualizations can serve a more important function. They can be further evolved to let the wider technical public obtain limited or total access to SciVis software for interactive visualization through the Web. In this way, other research groups or individuals can use the visualization software as a research tool to make further discoveries and hypotheses without repeating the same investment. At this "World Sees" stage, the visualization is public, so a high level of interactivity is required to provide a variety of users with interfaces that are so sufficiently sophisticated and intuitive that they can focus on the problem at hand rather than usability issues. It is our objective to re-engineer existing SciVis systems to this level.

2.6 Multidimensional Scientific Visualization Space

In order to extend SciVis into the Web, we must consider not only DiBiase's visualization sequence model but also the many variables affecting SciVis. We suggest that SciVis can be abstracted as a three-dimensional space (Figure 2.3).

DiBiase's model and the extension proposed here *directly* considers only one dimension of scientific visualization, "purpose/task" [23]. A *purpose/task* is an activity which a user of SciVis needs to undertake in order to achieve an objective. A SciVis system is developed primarily to help scientists explore masses of data, and can be further used for hypothesis confirmation, synthesis and presentation of ideas, Web publication and public service (Figure 2.2).

DiaBiase's model and our extension also *indirectly* consider another dimension, *user*. Users are not passive receivers but active individuals who filter and select information. Users vary in many ways, such as their ability to understand the conventions for human - computer interactions, their preference for the way they interact with a system and the way (format, method, amount) information is displayed. The typical values of this variable include *private users* (original researchers), public users and global users. *Public users* is the category defined previously, that is the wider technical audience with an interest in the data being presented. The *Global users* category includes these technical persons, and some non-technical ones, who are Web users and may access or browse SciVis occasionally.

The third dimension in the visualization space is the degree of human-computer interaction (HCI), which is not considered in DiBiase model. By *the degree of HCI*, we mean "degree of interactivity", i.e. the extent to which the user can inspect the data by direct interaction with the visualization. It is now generally appreciated within the information technology community that human-computer interface issues are fundamental to scientific visualization, where the ability to interact with the data is essential.

The corner of the cube in Figure 2.3 where the three dimensions have the values *original exploration*, *private user*, *low HCI* represents one extreme of of SciVis software providing "I See" visualization. The opposite corner, defined by the values *public service*, *global user*, *high HCI*, represents the other extreme providing "World Sees" visualization. How this cube can be used in practice will be explained later (see section 4.5).

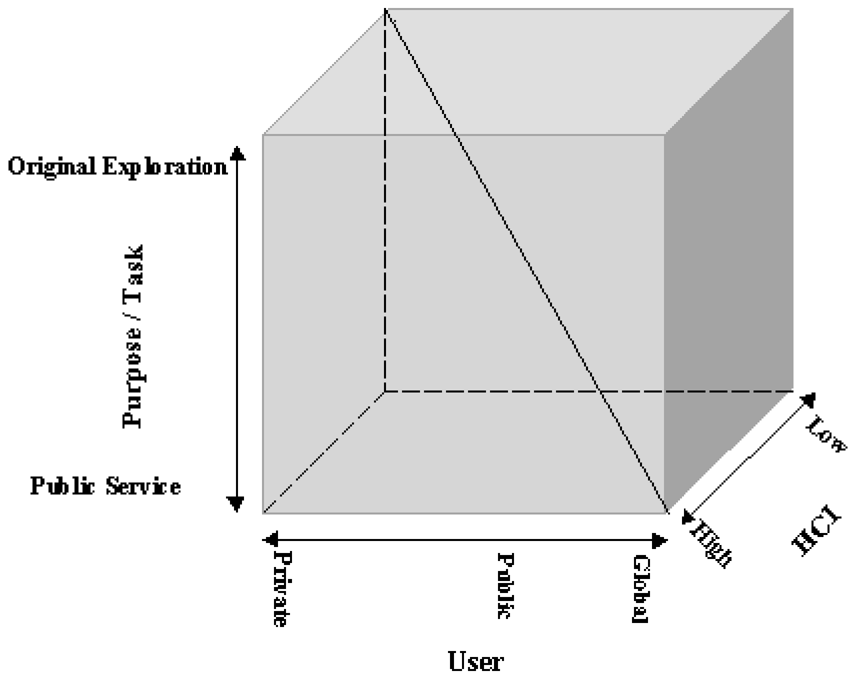


Figure 2.3 Multidimensional scientific visualization space.

Chapter 3 Scientific Visualization Software

3.1 Overview

Here we briefly describe the principles behind SciVis software. Most visualization software is developed according to the *visualization pipeline architecture*.

There are three categories of SciVis software, *turnkey visualization applications*, *application builders*, and *custom visualization applications*. Systems in the first two categories are commercial and the third category consists of in-house applications. At present, they are all designed as non-distributed applications. It is expected that the commercial ones will be modified into collaborative or Web based versions. The remaining problem is how to re-engineer in-house visualization applications cost-effectively as multiple-user applications for public service.

3.2 Visualization Pipeline Architecture

To help to understand how SciVis software works, it is useful to look at the general SciVis software architecture.

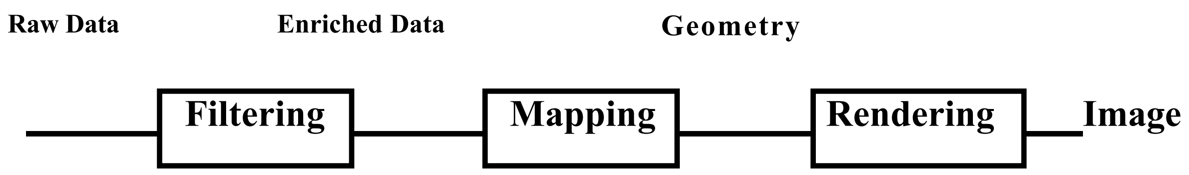


Figure 3.1 The visualization pipeline architecture

Most SciVis software is developed on the basis of the visualization pipeline architecture (Figure 3.1) proposed by Upson et al. [39] and Haber and McNabb [11]. The visualization pipeline is expressed as a sequence of processes, each of which does a specific well-defined task. The raw data, which may come from a computer simulation (as in computational fluid dynamics) or from measurement (as in remote sensing), is transformed by the following three processes.

- Filtering (filtering data into data) – This is a data enrichment or data manipulation process which algorithmically modifies the raw data into a more informative and perhaps less voluminous form for subsequent processes. If the raw data are assumed to be accurate, this is an interpolation process, fitting a mathematical function through the data points. If the raw data are inaccurate, an approximation process is required. For example various kinds of noise may be found in the data, and this can be reduced in this process by some smoothing operation.
- Mapping (mapping data into geometry) – This process transforms the derived data produced by the previous process into geometric primitives such as points, lines, polygons, surfaces and spheres for the rendering process. Haber and McNabb refer to this process as “mapping to abstract visualization objects (AVOs)” [11].
- Rendering (rendering the geometry into an image) – This process transforms the geometric data into a displayable picture in some image format. This will involve the preferred choice of colors and other issues specific to the appearance of the image. Rendering uses techniques of computer graphics and image processing. Typical rendering operations include transformations (rotation, translate, scale, perspective mapping and clipping) and optical models (shading, shadowing, hidden line/surface removal and anti-aliasing). In this process, speed and efficiency must be carefully balanced against the accuracy of alternative rendering algorithms.

Visualization means in general “to make visual”, “to bring something as a picture before the mind” [14]. This process is dependent on the perception and recognition of patterns in displays [24]. Cognitive psychologists believe that perception and pattern recognition are closely linked to memory. The representations in memory have analog qualities - mental images. The mental images vary with different cultures and scientific areas. Therefore SciVis systems might significantly vary with different scientific fields in visualization algorithms and implementations although they all have the general visualization pipeline architecture (Figure 3.1). During the design of a SciVis application, techniques from many disciplines such as perception, computing systems, mathematics, computer-graphic/image-processing, and the specific scientific area, are employed to generate the target image.

Although our research effort is focussed on custom visualization applications, for completeness we will briefly survey the other two types of SciVis software mentioned above, turnkey applications and application builders.

3.3 Turnkey Visualization Applications

Turnkey visualization applications aim at the *visualization end user*, the research scientist. Turnkey visualization applications offer fixed functionality to solve a limited range of specific problems. Their strategy is to conceal the pipeline architecture and to provide a predefined set of options for the user to select. These applications supply the main program and a simple interface with which the user provides the data and visualization options to the main program. Such applications have the following advantages:

- *Ease of use*: These applications do not require programming skill. Thus, one can obtain results quickly.
- *Speed*: Because they are very specific, such applications are usually faster than those built with generic application builders.

Turnkey applications also have some disadvantages, as follows:

- *Specific*: Many turnkey visualization applications are extremely application-specific and of limited use in other fields. For example*, Data Visualizer* (Wavefront Technologies Inc.)[1] is appropriate to some fluid flow visualization problems. Geographical information systems (GIS), such as *ARCVIEW* (Environmental Systems Research Institute, Inc.) [8] and *GRASS* (US Army Corps of Engineers) [40], deal only with geographical data. *PV-WAVE* (Visual Numerics Inc.) [41] is mainly used to organize and display plots, contours, histograms and tables.
- *No extensibility*: This kind of software is not extensible and, therefore, may often provide only part of the solution that the user requires.

3.4 Visualization Applications from Application Builders

Application builders, also called modular visualization environments (MVE), have emerged as attempts to provide general purpose SciVis tools [32]. Application builders target the *visualization programmer*, who can use them to build specific visualization systems. They make the visualization pipeline architecture visible to the programmer, and offer much richer functionality with the possibility of user extension. They are exemplified by four main commercial systems:

- Application Visualization System (AVS) from AVS Inc. [2]
- IRIS Explorer from Silicon Graphics Inc. [38]
- IBM Data Explorer from IBM [16]
- Khoros from Khoral Research Inc. [26]

Application builders offer the visualization programmer a library of predefined routines (modules for data input/output/control, filtering, mapping and rendering) mostly represented by graphical objects (glyphs). The visualization programmer can select these glyphs by dragging them into a workspace and combine them into pipelines using a graphical network editor. Data flows through the wires between modules to produce an image. The supplied modules can be replaced by third-party or user-written modules as required, providing they conform to the data input/output/control interface protocols.

These systems, while powerful and extensible, require special training and are therefore more difficult for a research scientist to use. A software developer is required to assist the research scientist to build an application which can generate images from the research data. However, if the programmer is familiar with the application builder environment, he or she can rapidly construct an application without writing code by manipulating icons on the screens and linking them with data paths.

The following are the characteristics of application builders:

- The user can connect modules by visual programming methods.
- The user can create and insert new modules.

The advantages of application builders are:

- *Flexibility*: These are general purpose SciVis tools [32]. Modules exist for supporting many different types of visualizations. Visualization problems are diverse and the modular approach provides great flexibility.
- *Extensibility*: The user can add modules. There are many third party vendors writing code for such systems. Furthermore, C, Fortran or C++ can be integrated into application builders simply by adding “wrappers” around the user’s existing subroutines.
- *Exchangeability*: The modules can be exchanged between disciplines.

The remaining problem for application builders is whether they are sufficiently general purpose to cover all SciVis solutions in all disciplines. The current application builders are quick approaches for prototyping new applications, but many visualization application domains have their own special requirements which present builders cannot meet. Therefore, in recent years, numerous graphics libraries have appeared (see next section).

3.5 Custom Visualization Applications from Graphics Packages

Graphics packages, such as GKS, openGL and the Mesa library, are collections of low level graphics operations. They have been developed in order to provide multipurpose and standard tools for visualization and computer graphics in general. These libraries represent the traditional, structured-language approach to programming.

Such graphics packages address the programmer and are far outside reach of the end user. Their effective use requires the visualization designer and the programmer to understand the graphics primitives and data structures in the library, the set of attributes that affect the appearance of the primitives, the rendering pipeline that the library implements, the administrative or housekeeping calls that are part of the library, and the system-level calls that are necessary to complete the non-graphics portion of an application. The designer and the programmer have to supply nearly all the pieces of the system, such as the user interface, data handling, geometry mapping, and the main program. Graphic libraries fail to hide the basic complexity of the visualization and place a huge software development burden on the programmer who must assemble a graphics application from low-level software.

The advantage of this type of system is its flexibility and direct control over the visualization requirements and patterns. It is the most useful type of software for the development of in-house SciVis systems for very specific solutions. Today, many organizations still use the graphics-package approach to developing SciVis systems for the solutions which current application builders cannot provide. However, this approach suffers from the large amount of time necessary to write, maintain and support the code. GSC(A) DISPLAY is in this category.

3.6 Collaborative Visualization Applications

The above three kinds of visualization applications all employ the visualization pipeline architecture and are non-distributed. However, as previously mentioned, many of today's research problems are too complex and too large to be tackled by a single person, and interdisciplinary teams are needed to work together to solve them. Therefore, collaborative visualization software may sometimes be a useful and cost-effective research tool. *Collaborative visualization* means visualization sharing, that is, people based at a number of geographically separated places can remotely use a single visualization resource.

In a demonstration of collaborative visualization in 1989, Haber and McNabb [12] employed a visualization resource located at National Center for Supercomputing Application (NCSA) labs in Champaign, Illinois, including supercomputing and rendering facilities, a video tape player for the relay of previously captured visualization results, and a camera to video the local user. The remote user (demonstrator) in Boston was equipped with a workstation and an input device. A private 96Kbps land line was used for the transmission of input commands from Boston to Champaign and an AT&T Telstar302 satellite link was used for returning the images from Champaign to Boston.

Although such elaborate equipment and communication facilities are not widely available, the problem of collaborative visualization has nevertheless received considerable attention.

In his PhD research [44], Wood suggests how commercial SciVis software (turnkey visualization applications and visualization builders) might be modified to support collaboration. It is expected that collaborative or Web-based versions of such commercial software will be available in the future. The Web provides a vehicle for this purpose. The use of SciVis in conjunction with the Web means that end users of SciVis do not necessarily require high-performance systems at their locations, and can use a variety of personal computers and workstations. In the next chapter, we explore this possibility.

Chapter 4 Scientific Visualization on the World Wide Web

4.1 Overview

The World Wide Web is not customarily viewed as a platform for developing and deploying applications. This chapter introduces the notion and demonstrates that SciVis applications can not only be used for Web publication but also be used to provide interactive SciVis public services. We also discuss two important aspects, the user interface and information drill-down. The user interface is an important consideration in creating SciVis public services because Web users are accustomed to and require easy access to services. Information drill-down is important to SciVis since in order to provide useful, high-level views of data, most visualization systems are likely to omit much detail. It is therefore necessary to provide the facilities for accessing embedded information.

4.2. Suitability of the Web for Interactive Visualization

The Web provides a paradigm for SciVis to be further evolved. Here we consider the reasons why the Web provides a suitable medium for publishing visualizations.

- *Connectedness*: The World Wide Web began as a hypertext information retrieval environment. It uses Internet services and merges the techniques of information retrieval and hypertext to provide simple access to dispersed information. The idea behind hypertext is that instead of reading text in a rigid, linear structure such as a book, one can easily skip from one point to another, obtain more information, go back, jump to other topics, and navigate through the text based on one's interests. With the development of Web technologies, the Web has incorporated much more than text. Technically, today's Web is not only a hypertext system but also a hypermedia system, which provides capabilities for graphics, sound, and video to be incorporated with the text. The reason that the Web has become so popular over other Internet services, such as Gopher, is its capability to display graphics in full color. This, together with the universal access it provides, appears to make it ideally suited to deploying SciVis applications in the global realm.

- *Platform Independence*: Access to the World Wide Web is obtained through Web browsers available on most popular personal computers and workstations. Consequently, for SciVis on the Web, no additional hardware is required, and any special capabilities can be dealt with by applets dispatched to the client by the server.

- *Universal access to specialized facilities*: Web information is distributed globally across thousands of different sites, each of which contributes hardware resources, software and databases for the information it publishes. SciVis usually requires special hardware, software, database, technical support and special setting-up. The publishing capabilities of the Web can eliminate these requirements for the visualization user by making specialized facilities widely available.

4.3 Concerns with the Web

There are three general concerns with the Web, interactivity, speed, and the variety of different end-users [4]. The introduction of SciVis into the Web means that these issues should be reconsidered in greater detail from somewhat different points of view. Where possible, we discuss techniques and technologies to address these issues.

- *Interactivity:* At the current state of Web technology, most user interactions with Web pages involve navigating hypertext links between passive documents. The addition of common gateway interface (CGI) scripts and forms (simple menus, buttons, text fields etc.) allows some interactions with computational facilities and data on the server. Although forms and CGIs have been the most widely used mechanisms for Web interactions, form interactions provide a very limited visual user interface, inadequate for complex systems such as SciVis applications. Java is beginning to provide the means to overcome this problem. Java's abilities have "activated the Web" via Java applets which can have highly interactive user interfaces with visual user-computer communication objects [15]. With further development of the Java Abstract Window Toolkit (AWT), event handling and networking, Java is making a significant technological impact on Web interactivity.

- *Speed*: The Web is a global infrastructure and does not deliver acceptable response speed on standalone machines or smaller networks. Highly graphic Web designs require high communication bandwidth in order to deliver information without unacceptable delays. Human-factors research has shown that for most computing tasks, the threshold of frustration is around 10 seconds [21]. Legacy SciVis software presents a speed problem here because it does not necessarily produce graphics in a compact format, so we need to consider the possibilities of adding some mechanisms at the server end to translate images produced by legacy systems into better formats.

- *Variety of end-users*: The Web is a public medium via which many kinds of end users can access information. A well designed system should be able to accommodate a range of potential users. Non-technical users depend on clear layout and easy access. They tend to be intimidated by many text inputs and complex menus and may be tentative about delving deeply into the information content if the user interface is not clearly arranged and graphically attractive. A glossary of technical terms, acronyms, abbreviations, and a list of frequently asked questions can be helpful to them.

Technical users, however, depend on the Web page to obtain information quickly and accurately, and tend to be very impatient with many text inputs and low-density menus that offer only a few choices at time. They generally have very specific goals in mind, and will appreciate fast information retrieval means and tools. It is a challenge for a Web user interface to meet the needs of both technical and non-technical users. By "technical user" here we mean technical in the sense of having an extensive computing background. Many end users of SciVis may not be technical in this sense even though they have a science background.

Today many people, including the users of SciVis systems, are used to the Web environment, therefore, Web conventions cannot be ignored. These conventions should be considered a high priority during user interface design. "Look and feel" conventions provide a general strategy for interface design to meet the needs of various users. Hypertext links, buttons, choice boxes and other familiar visual components are commonly understood and accepted.

4.4 Scenarios for Scientific Visualization on the Web

Using the Web to deliver visualization is a recent trend. At present there are several scenarios for SciVis on the Web [44]. In these scenarios, there are at least two participants: the *VSP* and the *user*. The VSP usually has a database and a visualization engine (i.e. SciVis application). The user at a different Internet location wishes to obtain the SciVis service from the VSP. Different scenarios then arise depending on who has access to and control over the visualization engine and the database. These scenarios can be divided into two categories: *static visualization* and *dynamic visualization*.

4.4.1 Static Visualization

In static visualization, fixed visualizations are created in advance by the VSP with its database and SciVis system and are posted on the Web server. In fact, this is simply Web publication (see Figure 2.2) where the user has no freedom to control the database and visualization engine. There are two scenarios in this category.

Provider creates the visualization

Currently most Web SciVis services provide simple browsing access to collections of static image files. These providers do not place their databases and visualization engines on line. They create visual representations and post them as images or as video sequences. The users can study these visualizations, either directly from the browser, or by downloading them and viewing via a helper application such as *xv*, a program on Unix for viewing images with various formats. The main limitation of this scenario is that these visualizations are static and the users have no opportunity to customize them.

Provider creates a 3D visualization

The second scenario of SciVis is that the provider supplies scripts in Virtual Reality Modeling Language (VRML) on the server, and the user looks at them using VRML-enabled browsers or plug-ins on the client. In this scenario, the rendering step of the visualization pipeline (Figure 3.1) has been migrated to the client.

VRML has become the standard for producing 3D graphics in Web environments. VRML is defined as a standard Multipurpose Internet Mail Extensions (MIME) data type and its scripts can be downloaded to a client through standard HTTP Web server mechanisms. This gives the user the option of rendering the 3D model locally from any viewpoint so giving more freedom to explore the result. However, the visualization result is still fixed by the VSP. Most parts of the visualization pipeline and the entire database are still off line, so this scenario is still in the Web- publication category (see Figure 2.2).

4.4.2 Dynamic Visualization

In dynamic visualization, the provider puts the database and the visualization engine on line, so the user can control the system to dynamically generate visualizations. This kind of visualization is in the public service category (Figure 2.2).

The development of the Web has created a new medium by which SciVis can be extended to this stage. One example of such a public service system is a dynamic map visualization service that appeared in 1994 [30]. This service produces displays dynamically on demand rather than just providing access to static files. With advances in SciVis systems and Web technologies, SciVis on the Web has the potential to become more dynamic and therefore an appropriate tool for exploration and analysis of data. Such SciVis tools should allow the user to specify parameters that determine exactly how the visualization should be produced: for example, which part of the data to use or how to display certain features. Technically, the advantages are unlimited parameter combinations and reduced requirements for disk space for image storage. The disadvantage is that a visualization engine can take a significant time to produce an image file, depending on the complexity of the processing and rendering algorithms it uses.

There are only a few such services on the Web at present. There are two scenarios for dynamic visualization.

User creates the visualization

In this scenario, the database and the visualization engine are on line. The VSP posts data on the server, and a visualization engine is installed on the client as a

"Visualization Plug-in" to the Web browser [17]. In this scenario, the whole visualization pipeline moves to the client. The data will be transferred on demand from the server to the client browser which launches the Visualization Plug-in once the data transfer finishes to perform the data manipulation and visualization locally at the client. This scenario has only been proposed and has rarely been seen on the Web for scientific visualization.

The advantages of this approach are as follows:

- The user can have real-time and interactive visualization. After the data is transferred to the client, the user operates the visualization tool and obtains the response time of a standalone local application.
- The user can have direct and interactive control of the actual underlying data (e.g. for statistical analysis). The issue of whether or not the direct interactive control provided is suitable for the wider audience is, as we have said previously, beyond the scope of our current work.

However, there are trade-offs involved. The disadvantages are as follows:

- If datasets are large, the time to download them may be unacceptably long.
- The user must have suitable visualization software, the plug-in that is, as well as the knowledge of how to install it and register it with the Web browser.
- If the visualization software is changed, a new version of the plug-in has to be re-distributed to clients for installation.
- The user must have the processing power available to run the plug-in.
- The user is given full access to data. The VSP may want to retain more control.

Provider creates visualization framework and user chooses the options

In this scenario, the VSP and the user share the responsibilities for creating the visualization. This scenario does not require the computing power and expertise at the client, required in the previous scenario.

Here the database and the visualization engine are on the server. The provider supplies the processing power and the basic visualization framework appropriate for the data concerned. The framework is sent to the client on demand where the user can control options within the framework. Control is then passed to the server for the visualization engine to generate an image or VRML script passed back to the client for display.

This is a compromise scenario. The complexity of the SciVis system remains with the VSP who has total control over the visualization, but the user is given the freedom to create visualizations within the framework.

Additional advantages of this approach are:

- No special installation is required on the client.
- The VSP does not need to worry about disclosure of its visualization engine and database.
- The VSP can give the user limited access to its visualization engine and database.
- The data sets do not need to be downloaded, so that the speed for processing very large data sets with small resulting visualization files can be improved.

The disadvantages are as follows:

- Long downloading time can result if the generated image file or the VRML file is large and the Internet bandwidth is low.
- The server may become overloaded.

Comparing the last advantage and first disadvantage, a flexible architecture with some intelligence is perhaps required where under some circumstances data is downloaded to the client which does the rendering, and in other situations just an image is sent.

4.5 User Interface

The user interface is extremely important in a Web-based SciVis public service. It has to be both simple and intuitive. The goal of user interfaces is to provide a communication pathway between users and computer software. The three-dimensional SciVis space (Figure 2.3) plays a reference role in the design of interfaces for SciVi applications. Usually an interface designer could identify the *user* and the *purpose/task* of the application under development, then picks the level of interactivity prescribed by the corresponding point on the diagonal of the cube indicated in Figure 2.3 as "Thinking … Communication … Thinking", and designs an interface with this level of interactivity.

Character-based interfaces provide adequate but inefficient interactions. During the initial development of in-house SciVis software for single user and original exploartion, however, they are frequently used, since the primary focus is on visualization rather the interaction. The GSC(A) DISPLAY software has such a character-based interface.

GUI provide a richer and more efficient means of HCI. For interactive and complex SciVis systems, it is important to design an appropriate GUI at some stage, providing user-friendly operations, in order to guide untutored users to explore and obtain desired visualizations. The term GUI strictly refers to interfaces relying on menus and forms although many people use it to mean anything other than a command-line interface. We will use the term GUI in the strict sense.

A user interface is not only a screen design but also a method of interacting with the software and its data. When people mention the “look and feel” of computers, they are referring to the traditional GUI. “Look” refers to the physical appearance of a screen to a user, whereas “feel” refers to the way the user interacts with the screen and hence the underlying software. A traditional GUI uses menus and forms. This makes the software more attractive and easier to use, but does not change the basic way users work with the computer, including typing, memorizing and referring to user’s guides or help. The users still need a lot of knowledge of and experience with the software, its data and its background to interact with the screen. GSC(A) has built a form-based GUI on the top of GSC(A) DISPLAY to replace the original command-line interface. This GUI requires a certain level of knowledge about geography and the software and expects the user to type in many parameter values in the correct format.

Jern recently proposed the term *Visual User Interfaces* (VUI) [17] for the category of interfaces that enable the user to interact more directly with the software by manipulating concrete graphical representations of objects. This concreteness includes not only spatial location and identity, but also appropriate image and behavior. In VUI, there is less need for the traditional pull-down menus and forms. This sense of immediate contact with screen objects is referred to as *direct engagement*. The VUI model eliminates the sense of an intermediary. There are many modern commercial software packages that employ the VUI approach, for example Prograph, Visual Basic, Visual C++, Visual Café, and VisualAge, all have user interface editors that allow programmers to drag and drop the GUI components as well as manipulate them with a mouse. Photoshop and PowerPoint make their tools visible as concrete graphical objects. Users can activate required tools, drag and draw the graphics components, as well as manipulate them directly by mouse.

The whole aim of the VUI is to create user interface components that can be easily manipulated by the user. By improving the visual communication that takes place in all of the elements of the user interface, VUI makes complex systems easier to learn, decreases learning time, and makes the user more productive. The greatest difference between a traditional GUI and VUI technology is that users interact directly with graphic objects on the screen in order to edit the data behind the graphic objects.

The importance of VUI in SciVis is that the purpose of visualization is to give a concrete representation of data. It is therefore critical to give the user the means to more meaningfully interact with this data.

4.6 Information Drill-Down

In a SciVis system, the data being visualized are frequently arranged hierarchically, since there are limits to the amount and scope of information that can be usefully depicted at a high level. Visualization shows us a high-level picture of some of the features of the data and lower-level detail is necessarily hidden for visual clarity. Moving down the hierarchy reveals more information in greater detail, which may be essential for validating conjectures made at the higher level, for example. Therefore, the success of SciVis services not only depends on the graphics which they produce, but also on the ability to drill down to retrieve more information related to a specific region of a graphic. Information drill-down and interactive data querying, together sometimes also referred to as "Visual Data Mining", are considered to be an important part of a Web-based SciVis service.

As an example, consider a graph of the history of the Dow-Jones industrial average. Such a graph might help the user to see market trends in very large data sets, however, for market analysis and reasoning, the serious user is unlikely to be satisfied with such an overview of the data. He or she needs the facility to retrieve a full and rich description of underlying database attributes and any related event information in any region of the graph. Related event information which affected market level might be stored somewhere other than in the database server, but may be important for the user to make investment decisions. Showing details only when they are requested is vital for decision-making and analysis. The user should be able to request details on demand by, say, clicking on a point in a visualization.

The image map is a technique implemented in web browsers to support information drill-down on the Web. An *image map* is a mapping from a set of regions on a screen to a set of links. Clicking on a region of the image will take the user to the location specified by the associated link. In cartography, the image map is also called a *hypermap*. The links are predetermined in the image map, therefore the image map is mainly limited to static information visualization.

Advanced Visualization Systems has implemented a dynamic image map creation in one of their products called *Gsharp* [35]. The graphics that are generated and displayed are just ordinary GIF or JPEG graphics; however, an additional file is generated and kept with the image called a *map definition file*. This is a textual file that contains the definition of where the active "clickable" areas on the image are located. This information is stored as regions with corresponding links. These regions can be defined as rectangles, circles and even arbitrary polygons. In addition to the image and map definition file, a CGI script is required to provide the connection between the browser and the map definition file. An image map definition file includes, among other things, lines starting with words like **rect**, **line** and **polygon** which represent the type of areas being defined. The CGI defines a function to compute a Universal Resource Locator (URL) from the map definition file and a mouse click location.

Chapter 5 Re-engineering In-house Scientific Visualization Software into a Public Service

5.1 Introduction

This chapter discusses a general approach to cost-effectively and efficiently re-engineering non-distributed in-house SciVis software into a public service available to many geographically dispersed users. We start from the requirement analysis.

5.2 Requirements for Re-engineering

The requirements for re-engineering an in-house SciVis application have to consider two distinct classes of people, the average end user and the VSP. The following requirements were obtained from the SciVis end users, system administrators and a SciVis software developer at the GSC(A).

5.2.1 Average End User

The average end user has a science background and requires data visualization services to assist with research and analysis of data associated with his or her primary field of expertise. The expertise of such users in using discipline-specific SciVis software may be quite limited, but it is assumed that they have the knowledge of how to use general-purpose software, such as Microsoft Office and a Web browser. Their criteria for such a system are:

- *Easy to use*: Average end users prefer that no special software installation is required since computer technical support is not usually available. They also need an appropriate visualization environment to promote rapid learning. End users should see the environment as an extension to the tools with which they are already familiar, allowing them to focus on the SciVis task rather than on learning how to use the system.

- *Interactive*: End users should be able to control important visualization parameters as well as directly manipulate the result. Also unfamiliar and unimportant parameters should be set to defaults and hidden from average end users. This requires a user interface that is tailored to the particular SciVis service.
- *Multiple platform*: The expertise of an end user may be limited to a particular platform. To provide a public service, it is necessary to re-engineer an existing system to run on most popular platforms, such as PC, Mac and Unix workstation.

5.2.2 Visualization Service Provider

This is the organization which owns an in-house SciVis application and also usually controls the related database. It is required that the VSP will be able to set up an appropriate network and have the capability to program HTML, CGI scripts (C/C++, Perl, shell scripts) and Java. From the point of view of the potential VSP, the re-engineering process should satisfy the following criteria.

- *Minimum investment*: The re-engineering process should be simple and flexible, allowing the legacy software and hardware infrastructures to be largely retained. This is an important consideration since such systems have usually evolved over a period of time, so there is likely to be a considerable investment in their development and maintenance. Furthermore, in many cases, some of the original developers may have left the organization, taking some of the intellectual history with them. Consequently, significant changes to or rewriting of such legacy systems is impractical and costly.
- *Keep resources private*: From the commercial point of view, the VSP needs to protect the SciVis software and data to maintain control of its intellectual property.

5.3 The Re-engineering Approach

In order for a non-distributed SciVis application to be re-engineered as a public service with dynamic visualization capability, we chose the scenario “provider creates visualization framework and user chooses the options” as the most appropriate for our purposes. This is a compromise scenario which balances the above requirements from the average end user and from the VSP. We also chose the Web as the deployment medium for publishing visualization because of the characteristics listed in section 4.2. These two decisions together suggested that an appropriate client/server strategy be used in our re-engineering approach.

5.3.1 Client/Server Architecture

In a client/server system, the client is a local processor, providing the user interface and communicating with the server which provides computation and data services. Thus client/server is fundamentally a distributed architecture, which means that resources are not typically concentrated in one place but spread over multiple computing platforms and locations. Communication between client and server can be provided by a variety of different mechanisms, for example, the Internet.

The inherent modularity of client/server systems leads to greater overall robustness, since computing responsibility is no longer concentrated in the server. This modularity also makes it easier to incorporate new technology. Therefore, the client/server model has provided an opportunity for many organizations to re-engineer their computing environments, and similarly provides an appropriate basis for re-engineering a non-distributed in-house SciVis system into a public service. The non-distributed application can still reside on the host computer, but with minimum modification, the host computer can become a server providing remote client computers with access to the SciVis service via the Internet. The Web infrastructure is ideal for deploying SciVis client/server services because of its existing and developing support for multimedia capabilities.

Conventionally, client/server deployment can have several configurations [28], which offer a very flexible architecture for designing and re-engineering different systems. The configurations outlined below (Figure 5.1) show three typical deployments of client/server architecture. These are often described as two-tier or three-tier client/server configurations. The terms *two-tier* and *three-tier* refer to the way in which a system may be partitioned. The three tiers are:

- *Presentation services* that manage the display and manipulation of data
- *Application services* that manage system rules and logic
- *Database services* that manage and supply the raw data

Two-tier client/server systems can be partitioned in one of two ways: with a thin or thick client running on a desktop computer connecting to a more powerful server computer. In *thin client* deployment, user-presentation services are localized on the client while database and application services are provided by the server. Conversely, in *thick client* systems, application services are provided by the client instead of the server. This two-tier configuration requires a more powerful desktop client with a fast processor, plenty of local memory, and disk resources to cope with the extra processing load being handled at the desktop machine.

Three-tier client/server systems separate application services from database services by interposing an application server between the client and database server. Three-tier client/server provides both more flexibility in system deployment options and more scaleability to cope with growth in system complexity because the configuration has been designed and partitioned with more granularity.

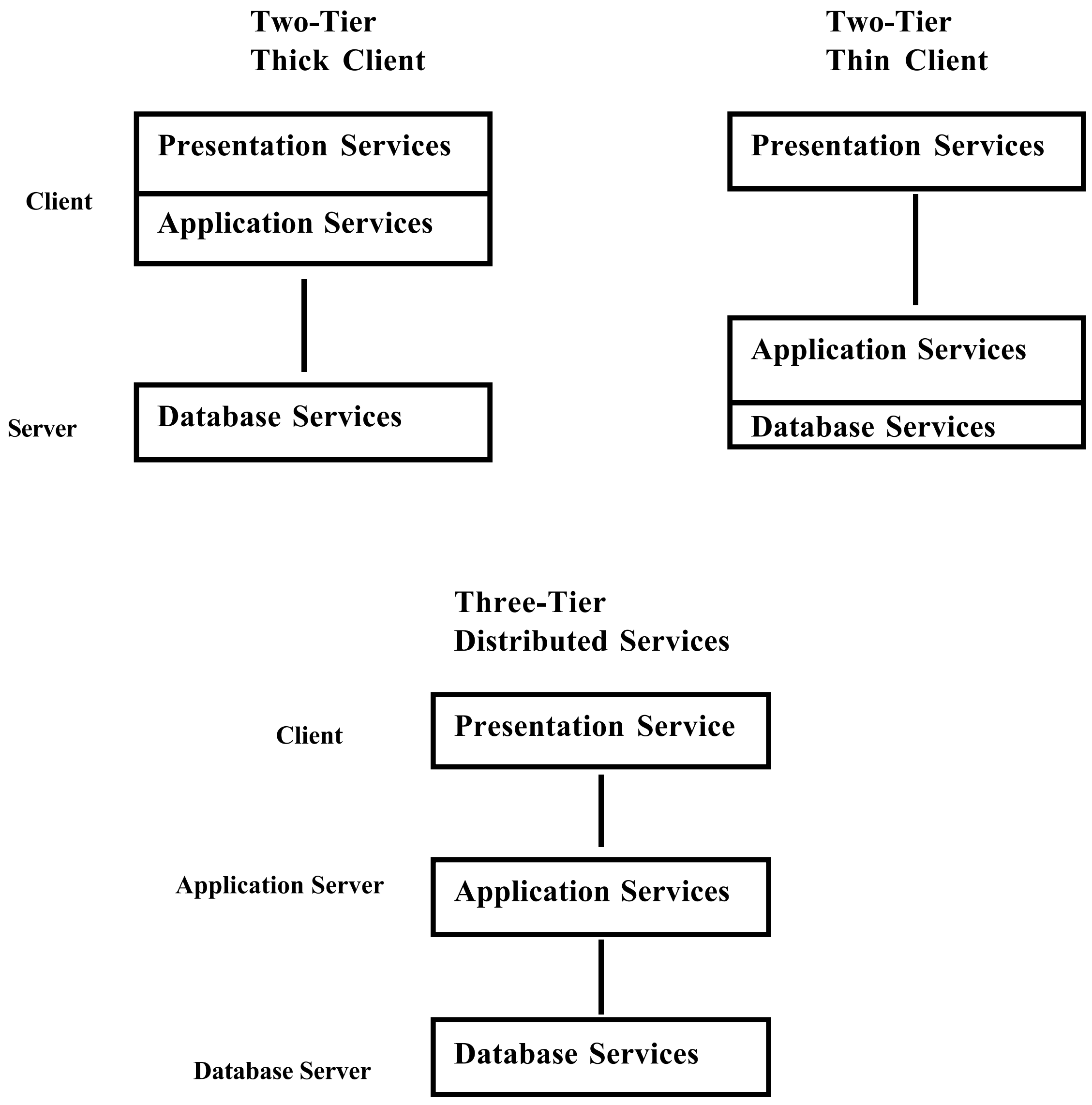


Figure 5.1 Three client/server configurations

It seems that this conventional three-tier configuration is the most appropriate one for client/server deployment. However, in reality, we used an alternative one (Figure 5.2) in our re-engineering in order to implement a Web-based solution for the chosen scenario which requires a user interface supporting domain-specific interactions.

The significant difference between this configuration and conventional three-tier is that the application services have been distributed over the three tiers, rather than located in only one. The client becomes thicker and contains not only the dumb Web browser for providing presentation services but also Java applet for providing some application services, according to a visualization framework defined by the VSP. The middle tier is the Web server, containing all communication rules (application services) between the client and the legacy system, which is the third tier in our modified configuration and consists of the legacy application (application services) and its database.

5.3.2 Extended Visualization Pipeline Architecture

In order to determine how the various software components get distributed over the tiers in the modified three-tier client/server model described above, we reconsider and extend the visualization pipeline architecture. With this extended architecture (Figure 5.3), a rapid re-engineering can be achieved that does not necessarily require changes to the existing in-house SciVis software, maintains the privacy of the VSP's database and visualization engine, and reaches the goal of zero installation on the client.

Generally, existing SciVis software can be front-ended by a suitable *Harness* (discussed below). The Harness is software developed by the VSP on top of the existing visualization system. It is stored on the server with the SciVis software, and can be

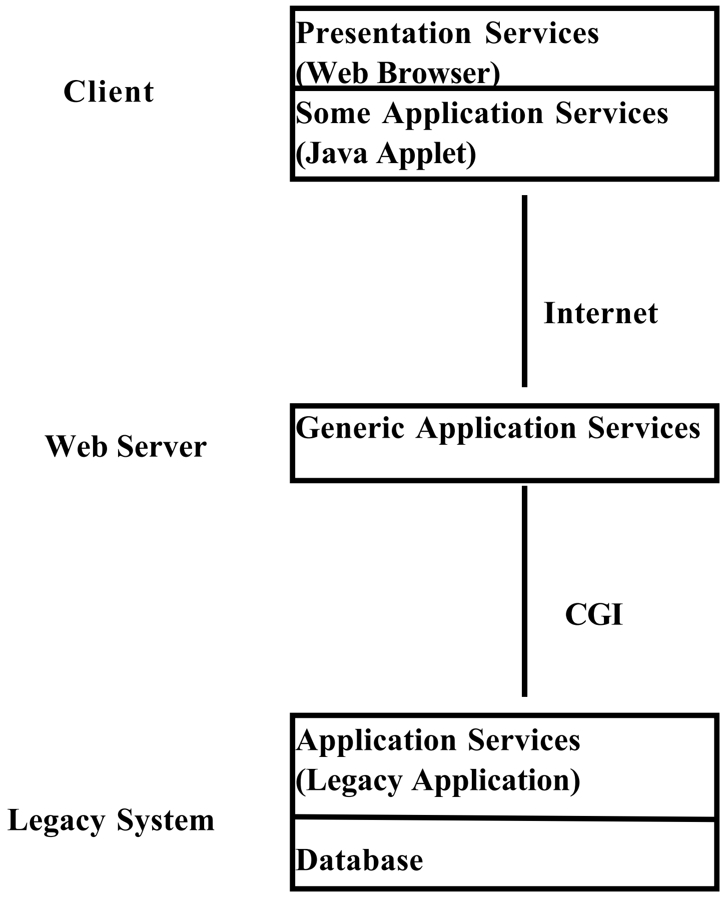

Client
Presentation Services (Web Browser)
Some Application Services (Java Applet)
Internet
Web Server
Generic Application Services
CGI
Legacy System
Application Services (Legacy Application)
Database

Figure 5.2 The modified three-tier configuration for re-engineering legacy systems

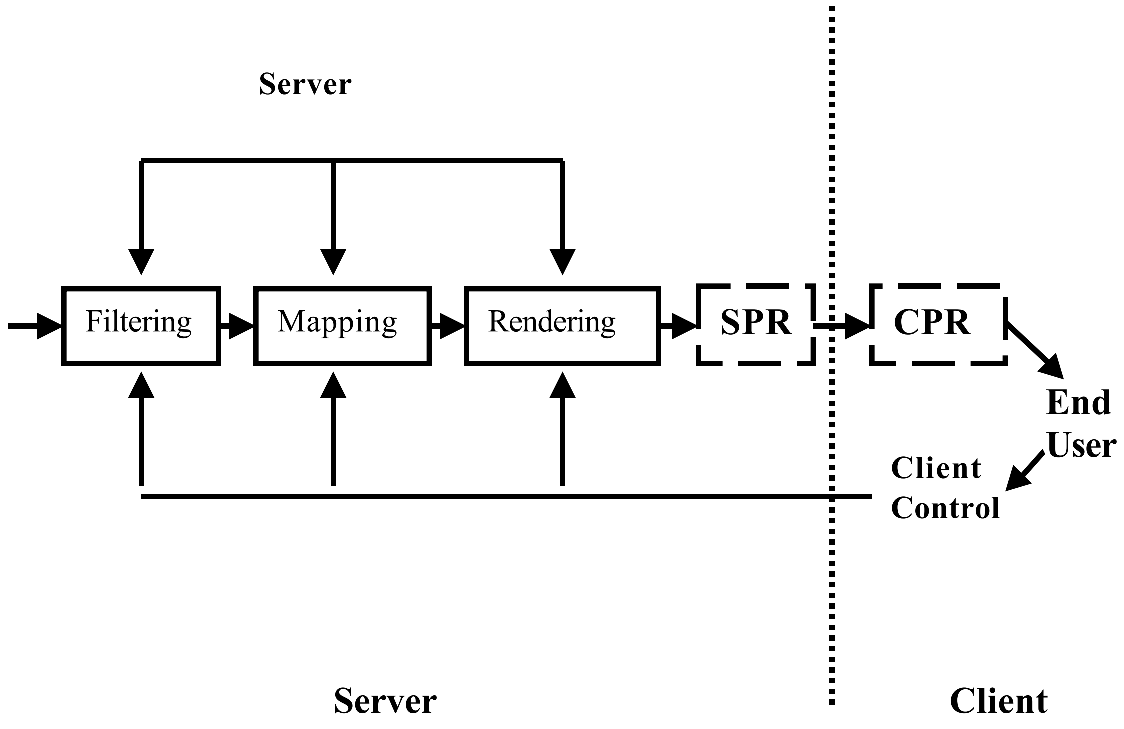


Figure 5.3 Extended visualization pipeline architecture.

dynamically downloaded to the client's Web browser on demand. It contains the visualization framework created by the VSP, and allows the user to specify parameter values (called "client control" in the architecture) for the control of visualization process. As discussed earlier, the VSP may set some parameters to defaults on the server (called "server control" in the architecture) if it is unnecessary to expose them to the average end user. The Harness collects the user's options and sends them to the server, which invokes the visualization pipeline with the selected options and the fixed defaults.

The visualization result, generated on the server using the legacy SciVis software, may be in various different formats (see Table 5.1). Although in theory one can arrange for images in any format to be displayed in a browser by writing an appropriate rendering applet, one may prefer to avoid this effort and simply rely on the display capabilities built into browsers. Current Web browsers, however, can directly render only the popular GIF and JPEG formats. In our case, the GSC(A) Display engine generates Postscript files. We do not want to transmit these via the Internet because of their large size. Nor do we want to have to write an applet to render Postscript files at the browser end. Hence, the visualization pipeline is extended with an optional server post-rendering (SPR) phase, which can convert the visualization result from one format to another format or do other image processing tasks with the utilities available on the server. The extended architecture also has an optional client post-rendering (CPR) phase on the client. CPR may perform various image processing operations, including cropping (extracting a rectangular piece of an existing image), RGB filtering (modifying individual pixels based on their current color and transparency), rotating, adding legends, and so on. The CPR phase resides in the Harness.

It is important to note that we are led to this structure by the fact that we are re-engineering legacy software. If we were implementing a SciVis system from scratch, then in cases where the data is small but the rendered image is large, it would make sense to send the data together with an applet to perform the rendering on the client.

Format	**Name**	**Comments**
GIF	Graphics Interchange Format 8 bits per pixel	Most commonly used image format.
JPEG	Joint Photographic Experts Group, 24 bits pre pixel	Used mostly for photographs, complex “photographic” illustrations, medical images, and other types of images where the JPEG compression processing does not severely degrade image quality.
PPM	Portable Pixel Map	Distributed with the X-window system.

Format	Name	Comments
	24 bits per pixel	
PGM	Portable Greyscale Map 8 bits per pixel	Distributed with the X-window system.
PBM	Portable Bit Map 1 bit per pixel	Distributed with the X-window system.
PostScript	Encapsulated PostScript Format (EPSF)	A page description language with sophisticated text facilities. For graphics, it tends to be expensive in terms of storage.
TIFF	Tagged Image File Format	Encompasses a range of different formats , originally designed for interchange between electronic publishing packages.
CGM	Computer Graphics Metafile	An ISO standard since 1987. Has the capability to encompass both graphical and image data.
XBM	X-window Bit Map	Standardized by the MIT X-consortium.
VRML	Virtual Reality Modeling Language	A standard language for 3D data interchange on the Web

Table 5.1 Some general visualization file formats. More formats, various format specifications and format conversion utilities are available on the Web (http://www.dcs.ed.ac.uk/~mxr/gfx/).

5.3.3 Description of the General Approach

The general re-engineering approach has been abstracted into a re-engineering architecture (Figure 5.4). This is a unifying solution that enables the re-engineered system to be constructed relatively quickly. Two important additions have been made to configure an existing in-house SciVis system into a client/server application. They are the Back End Manager, and the Harness mentioned briefly in the previously section. Both names were suggested by Edmonds et al. [7].

The Back End Manager provides the interface with the visualization engine and the database in a way that provides a clean and clear separation between the Harness and the existing application. This manager has three responsibilities:

- *Communication with Clients*: It waits for clients' connections, synchronizes multiple connections, and receives messages from clients.
- *Communication with Visualization Application*: It has all the information necessary to access and run the visualization application. Details of the location of the application, its server control (see Figure 5.3), input/output policies, etc. must all be made available to it. The Back End Manager must, therefore, be configured and maintained precisely in conformity with the actual application software available.
- *SPR*: It processes the visualization results on the server for delivery to the clients (see the discussion in section 5.3.2).

The central part of the re-engineered software is the Harness, which is the front end on the client. The Harness is often used in the re-use and integration of existing software. It has a range of responsibilities including control of the visualization process by end users, providing a visualization environment, dealing with presentation issues, and communicating with the server. The end users can access the Harness only through the visualization environment. The complexities of the Harness are hidden from the users and located in the presentation service and communication service of the Harness. The Harness consists of the following services:

- The *communication service* connects the Harness to the server and sends control messages to the Back End Manager.

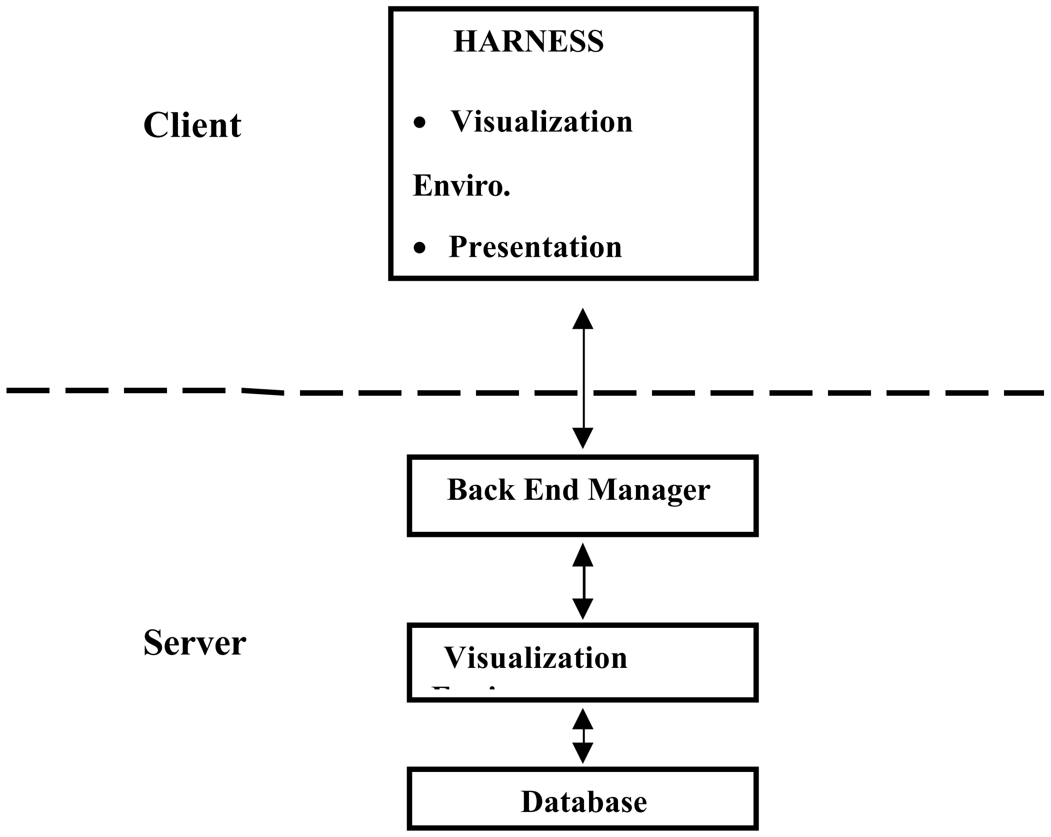


Figure 5.4 The re-engineering architecture

- The *presentation service* is responsible for the maintenance of the state of the user interface and its visual interaction objects, interpreting and processing all interface events. This service composes client control (see Figure 5.3) messages and passes them to the communication service for transmission to the server. It may also perform client post-rendering, as discussed in section 5.3.2, to improve the visualization. The presentation service plays a similar role to *controller* components in the *Model-View-Controller* (MVC) architecture for interactive applications used in *Smalltalk*. In MVC, a controller receives all relevant input events from the user and decides how they should affect the views and model associated with the controller [19].

- The *visualization environment service* provides the communication between the end user and the whole client/server system. It ensures that an end user is presented with a system that has a familiar, consistent and concrete style and organization, requiring little training overhead. In this way, complex in-house SciVis systems can be presented to a wider range of users, and learning times and training costs can be significantly reduced. This service provides functionality similar to that of the *view* component of MVC, which implements visual display [19].

5.3.4 User Visualization Environment

When a legacy SciVis system is deployed as a public service, the user visualization environment has to satisfy new requirements as follows.

- The end users have much more varied expertise, so although they may be familiar with the particular technical area the SciVis system deals with, they are not familiar with the system itself.
- The end users are interested in rapidly obtaining data visualizations to assist in analysis for scientific and technical purposes. They do not want to spend time on installing or learning special software.

Consequently, a visualization environment should rely as much as possible on standard interface conventions, and where appropriate, allow the user to interact directly with images that provide concrete representations of real-world objects rather than text or forms. Such concreteness, termed “closeness of mapping” by Green and Petre [10], is likely to significantly enhance the user’s ability to interact with the system.

Some of following characteristics which may be embodied in the visualization environment of an existing non-distributed SciVis system should be preserved in the re-engineering.

- *interactive display*: Some SciVis software packages provide a responsive visual environment in which displayed images can be dynamically generated.
- *appropriate tools*: Some SciVis software packages provide tools for input and display, such as a slider for inputting speed.
- *context presentation*: In order to prevent the user from losing track of how the information presented in a window relates to the underlying data when there are many windows on the screen, some software packages attempt to make the user aware of the context of the window by using informative window titles, and by displaying a brief description at the top or bottom of the window.
- *message boxes*: Some SciVis software uses message boxes for communication with the user, thereby increasing visual clarity.

Such preserved characteristics together with new direct-manipulation interface features are important not only to the end user but also to the VSP. A potential VSP possessing a rich database and proprietary display software is likely to be interested in generating revenue to maintain and further develop these systems by making them widely available.

In a dynamic SciVis public service on the Web, users need to select parameters to control the generation and rendering of images. This raises some issues for the user visualization environment.

Several visualization environments already exist on the Web, for example an interactive scientific display service due to Putz [30]. This service was created before the forms extension to HTML was available, so hypertext links were used to select map rendering parameters. Although hypertext links are one of the main Web characteristics, it is not appropriate to use them for selecting options. This user environment lacks the integration and active interactivity. One may have to go to a different Web page to find appropriate selections. In March 1996, the web version 1.0 of Generic Mapping Tools (GMT) was released for on-line map creation, which has since evolved to version 4.0 [43]. GMT is an interactive SciVis public service which displays maps on demand, and uses the forms approach in its interface. Although forms are a well accepted convention for entering parameter values, they are inferior to direct manipulation. For example, dragging and drawing a rectangle on a map is a much more effective and intuitive way for picking latitudes and longitudes in a GIS system than tediously typing four values into a form. In the forms environment, Web end users cannot take a more active part in the overall visualization process.

The advent of Java has greatly increased the degree of interactivity available in Web clients via applets downloaded transparently from the server. In fact, in contrast to HTML which is simply a page-layout and hypertext-link management language, Java is a true programming language, which, aside from any limitations imposed by security considerations, can accomplish any processing on the client. In particular, a Java applet can implement any style of user interface, providing appropriate tools for the direct-manipulation interfaces required by SciVis applications. Therefore, with Java applets, the user environment for Web-based SciVis can be implemented as a VUI (see section 4.5.2), providing natural, easy-to-use tools throughout the whole visualization process.

5.4 The Advantages and Disadvantages of the Re-engineering Approach

In our re-engineering approach, a modified three-tier client/server architecture is developed with the Web as a low cost medium to deploy this architecture. With this Web solution, we can effectively re-engineer a legacy non-distributed SciVis application into a public service and satisfy all re-engineering criteria from the end user and the VSP, enunciated in section 5.2.

The advantages of this solution are as follows.

- *Inexpensive to Deploy*: The Web browser software required to supply the interface and run applets is either inexpensive or free and is expected to become part of standard desktop operating systems in the future. The Web medium itself provides a low-cost and low-maintenance wide-area network (WAN) that can be used by any organization or individual. The legacy system is retained privately by the VSP without any change.
- *Inexpensive to Acquire*: No extra software is required for the client since applets implementing domain-specific functions are downloaded from the server and can run on multiple platforms. Applets can provide end users easy and interactive tailored manipulation of SciVis.
- *Easy to Maintain*: The necessary Java applets are stored on a central server and downloaded to the client. Therefore there is only one place to apply updates. This makes application maintenance easier and cheaper.
- *Extends Reach*: The Web can be accessed by anyone at any time from anywhere there is an Internet connection.

The disadvantages of this solution are as follows.

- *Slow Response Times*: The rapid growth of the Web and peaking of loads at various times of the day means that response times may vary widely.

- *Increased Complexity on the Server*: The use of applets requires a Web server plus the software that manages the communications among applets, Web server and visualization engine. This adds more complexity to the system and requires additional technical support.

5.5 Summary

In this chapter, we have presented an analysis of the re-engineering requirements for creating a Web-based public service from legacy SciVis Software. This analysis has led to two decisions related to the re-engineering. One is on choosing a compromise scenario for dynamic visualization and another is on using the Web as the deployment medium. These decisions have led us to suggest a modified three-tier client/server configuration which is suited to both the needs of visualization and the partitioning of functionality that is necessary for Web deployment. We have also proposed extension of the visualization pipeline architecture to tailor it to this configuration.

In our general re-engineering approach, there are two important components in addition to the existing in-house SciVis system. They are the Harness and the Back End Manager. Harness is a Java applet transferred from the server to the client, taking a range of responsibilities for an interactive and responsive visualization environment. The Back End Manager provides an interface between the Harness and the existing SciVis application.

Chapter 6 Implementation of Web-Based GSC(A) Display

6.1 Introduction of GSC(A) Display

The Geological Survey of Canada (Atlantic) (GSC(A)) is a division of the Federal Government of Canada. It engages in mapping and research in the marine areas adjacent to Atlantic and Arctic Canada in order to meet three primary objectives: (1) to determine the overall geological framework of the continental margin; (2) to evaluate the region's hydrocarbon potential; and (3) to understand the processes that affect the nature and the distribution of sediments on the sea floor. Since the 1960s, GSC(A) has been collecting large amounts of geographic data. In the middle of the 1980s, GSC(A) started to study the development and maintenance of computer procedures for the reduction, display, manipulation and interpretation of these large geographic data sets. The result is its in-house SciVis system called Geographically Oriented Display of Information on a Unix platform for Visualization and Analysis (GSC(A) Display). The system has helped the GSC(A) scientists to carry out various kinds of research, for example, into regional topography and bathymetry for eastern Canada, the NW Atlantic and the Arctic. The characteristics of this system are as follows.

Graphics Package Application

The GSC(A) Display software is a visualization application built with the aid of graphics packages (see section 3.5). There are many modules dealing with different data sets. The software runs on HP-UNIX and is written in FORTRAN using the Graphic Kernel System (GKS) library. Because of the problems of FORTRAN compiler compatibility, the FORTRAN-Graphics library binding, and the software organization structure, moving the software to another platform is too difficult to be practical. For example, at the beginning of this project, a GSC(A) software developer involved in the original design and development attempted to move part of the software to a Sun Solaris platform for the reengineering project, but failed after one month of effort.

Flat File Systems

The databases of the GSC(A) Display are flat files. By comparison with today's database technology, these file systems are relatively unsophisticated. Flat file systems require low maintenance, except when they go wrong, which means a database administrator is not required. However, the benefits of flat file databases do not outweigh the drawbacks unless the database is very simple and small.

For application software to manage such files, it needs to be aware of all low level details of the file structure such as what the type of a field is, where it begins in the file, and how long it is. The positional structure means that there are two key problems with flat file systems, namely, they are inflexible and difficult to access. They are inflexible because adding new field types or changing the length of existing field types immediately invalidates software used to manage, query, or report on data in the file. Flat file systems are difficult to access because each file is essentially unique; there is no standard means for accessing file data. In addition, flat file systems are expected to operate statically on a host computer not dynamically across a network. This design means a simple information request from a network workstation may require whole files to be transported from a server to the client. This is an inefficient way to manage data in a network environment. Hence, flat file databases do not support information drill-down over the Internet because of long transfer times for big files. For these reasons, flat file databases are rarely used these days. Instead relational databases are the most commonly used.

Nevertheless, GSC(A) is a legacy system, in which the visualization software is hard-wired to flat files. Although at this moment, GSC(A) has no plans to change their flat file databases into relational databases, in the future, these flat files could be changed into a relational database for effectiveness and flexibility, with the necessary corresponding modifications to the visualization software.

User Interface

During the development of GSA(A) Display, a "command line" user interface was designed by GSC(A) research scientists and software developers. In this interface, the user types inputs in response to prompts. Many inputs are required (Table 6.1), and if some input is wrong, the program will irrecoverably freeze, requiring the user to start over again. Although this user interface gives GSC(A) scientists many options, it is not user friendly at all, and quite unsuitable for the average end user.

6.2 Features and Functions of Web-Based GSC(A) Display

Great care has been taken to make the system user friendly. The user controls the visualization process via an interface supplied by the harness on which most options can be selected by direct manipulation of images and meaningful controls. As an Internet GIS service, this system has the following features:

- *Simplicity*: Without special installation, the users can run this system from a Web browser. The complexity of the server is hidden from end users.
- *Client server structure*: The client provides direct manipulation tools for end users to input parameter values and view the visualization results. The server processes requests from the client.
- *Platform independent*: Users can access to this system on different platforms inside and outside GSC(A).
- *Security*: Users can view the visualization result, but cannot save it on the client disk because of the security measures imposed by Java.

We now explain the features of the interface presented to the end user by the Harness applet.

Input prompts	Notes
Give name of input file:	Ask for the dataset file name and its whole path
Select the type of plot output:	X Window, postscript, etc.
Size:	The size of Printer paper for the plot output
Give name of the output plot file:	Ask for the output name and its whole path
Give plot-title (max 80 characters):	
Give the reference latitude:	Usually input default 0
Give area – lat – lat, lon – lon:	Ask for the latitudes and longitudes to select area
Rotate?	Portrait or landscape
Give Scale:	1. Scale with legend 2. larger plot
Give space between plot and frame:	The default is 0.0 cm
Select frame:	Ask for the type of the frame
Give color for frame:	
Plot legend?	The default is "yes"
Give the saturation:	Inactive for RGB color scheme
Give angle of sun:	The sun angle affects the relief shadow
Give line width factor and resolution:	Usually use the default (2.0, 0.054)
Specify the type of color bar:	Ask for the legend color bar type
Give name of color bar:	Ask for the color bar file name and its whole path
Give minimum and maximum shadow gradient:	The default is (-12, 12)
Give gradient interval:	The default is "4"
Give percent white contrast 0 to 100:	The default is "50"
Give percent black contrast 0 to 100:	The default is "50"
Plot land file?	The default is "yes"
Give color for the coast-line:	There are 14 colors for selection.
Give width for coast-line:	The default is 1.0 mm
Give option for coast line area and resolution:	Usually use the default settings, i.e. the default area is the whole world and the default resolution is medium

Table 6.1 Typical GSC(A) Display User Inputs for a Relief Map

6.2.1 The Initial Viewer Window

When the end user starts Web-based GSC(A) Display, a dialog box appears, prompting for the user name and password. Once these entries are validated, the initial viewer window (Figure 6.1) is presented.

Near the top of the window is a *drop down list* labeled **DataType**, which identifies the current data set and allows the user to choose another data set. A *button panel* is located below the drop down list, which provides an **InputManager** button to invoke input windows, a **Stop** button available once a visualization request has been dispatched to the server, and **Zooming** buttons available once the image arrives from the server.

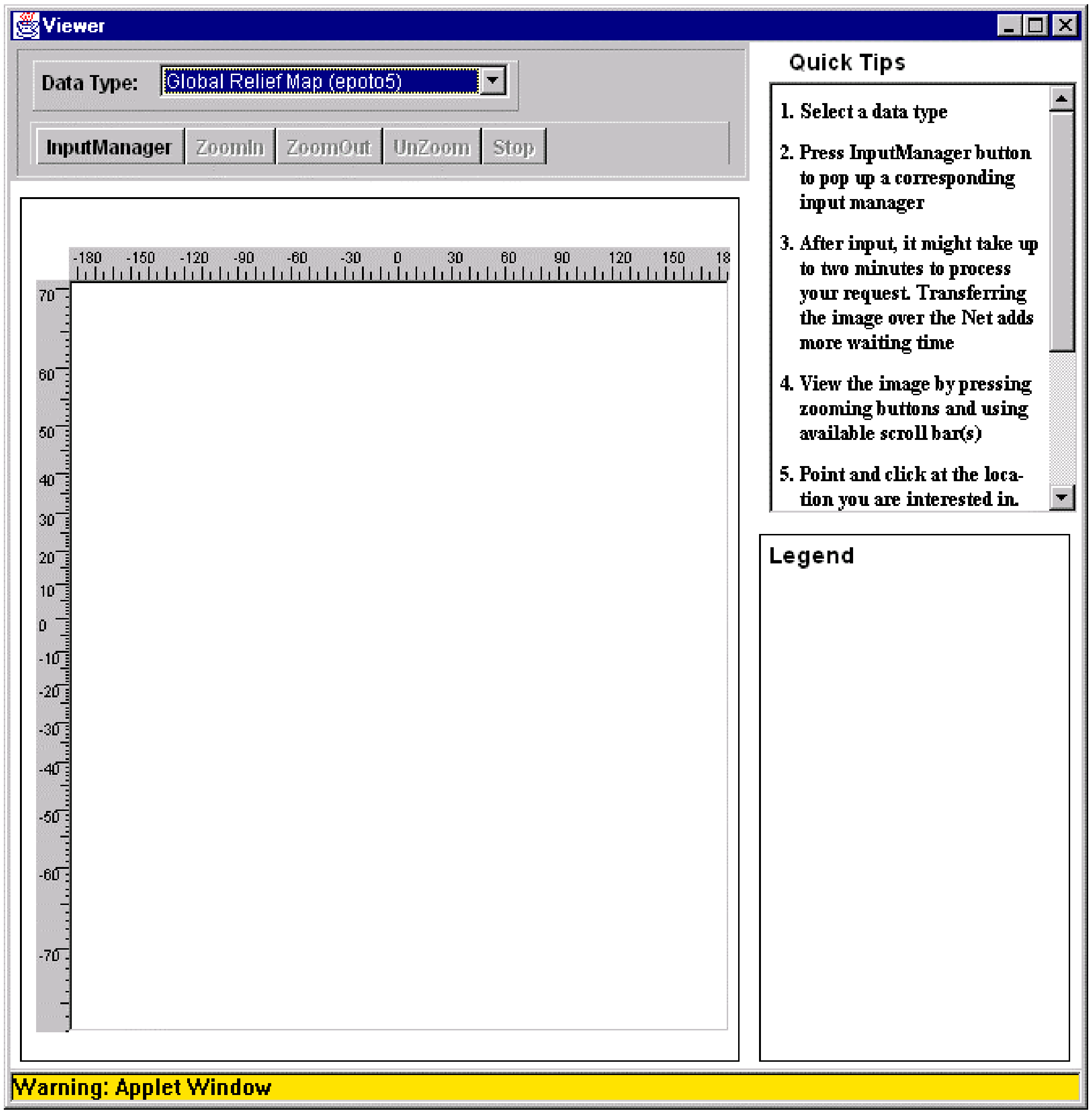
The large *viewing pane* with two *rulers* is used to display the images from the server. **Quick Tips** and **Legend**, located on the right, will be discussed later.

6.2.2 Selecting a Data Set and Invoking an Appropriate Input Manager

In the viewer window (Figure 6.1), the user selects the data set for which a visualization is required, using the *drop down list* labeled **DataType,** then clicks the **InputManager** button to pop up a new input window appropriate to the chosen data set.

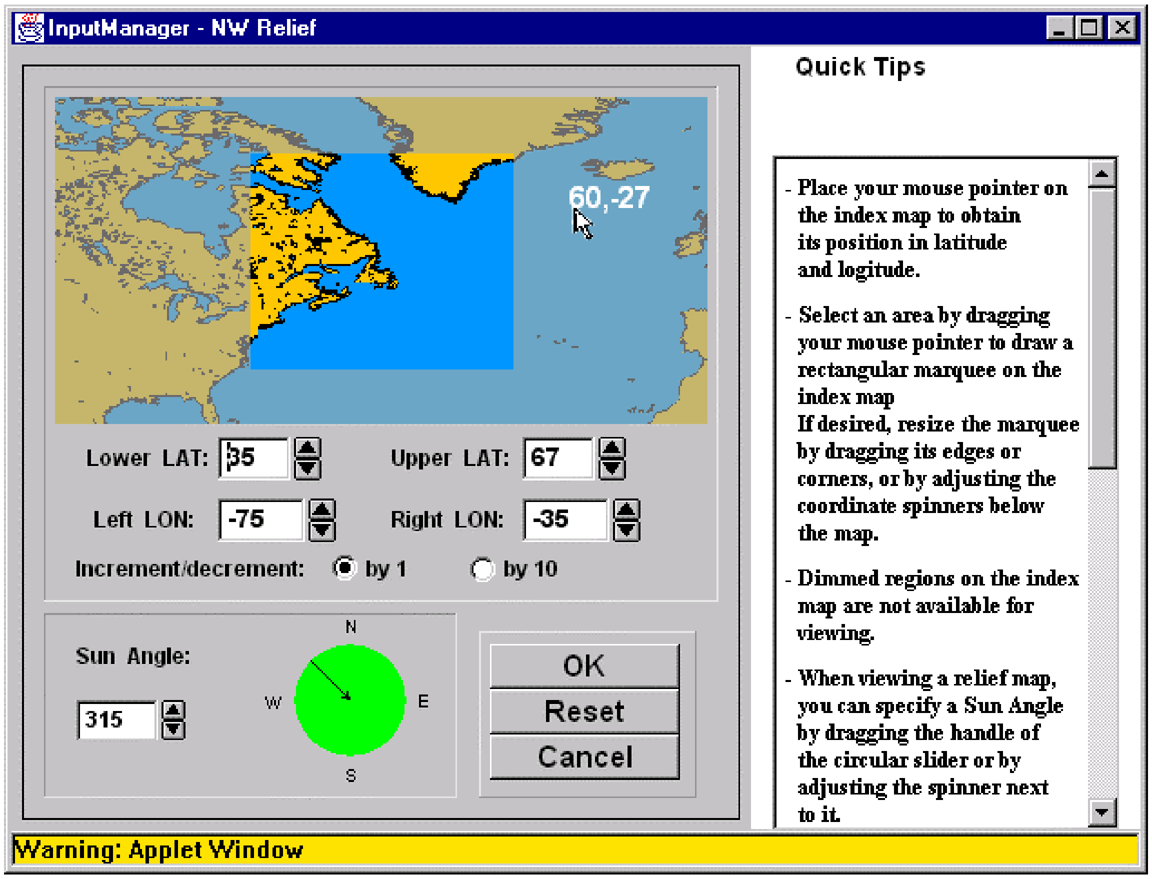
6.2.3 The Input Windows

The relief map input window is shown in Figure 6.2. This window consists of three main components, other than **Quick Tips**. These are an *index panel*, a *sun angle panel* and a *button panel*.



1. Select a data type
2. Press InputManager button to pop up a corresponding input manager
3. After input, it might take up to two minutes to process your request. Transferring the image over the Net adds more waiting time
4. View the image by pressing zooming buttons and using available scroll bar(s)
5. Point and click at the location you are interested in.

Figure 6.1 The initial viewer window.



Lower LAT: 35 Upper LAT: 67

Left LON: -75 Right LON: -35

Increment/decrement: [✓ by 1] by 10

Sun Angle: 315

Quick Tips

- Place your mouse pointer on the index map to obtain its position in latitude and logitude.

- Select an area by dragging your mouse pointer to draw a rectangular marquee on the index map
  If desired, resize the marquee by dragging its edges or corners, or by adjusting the coordinate spinners below the map.

- Dimmed regions on the index map are not available for viewing.

- When viewing a relief map, you can specify a Sun Angle by dragging the handle of the circular slider or by adjusting the spinner next to it.

Warning: Applet Window

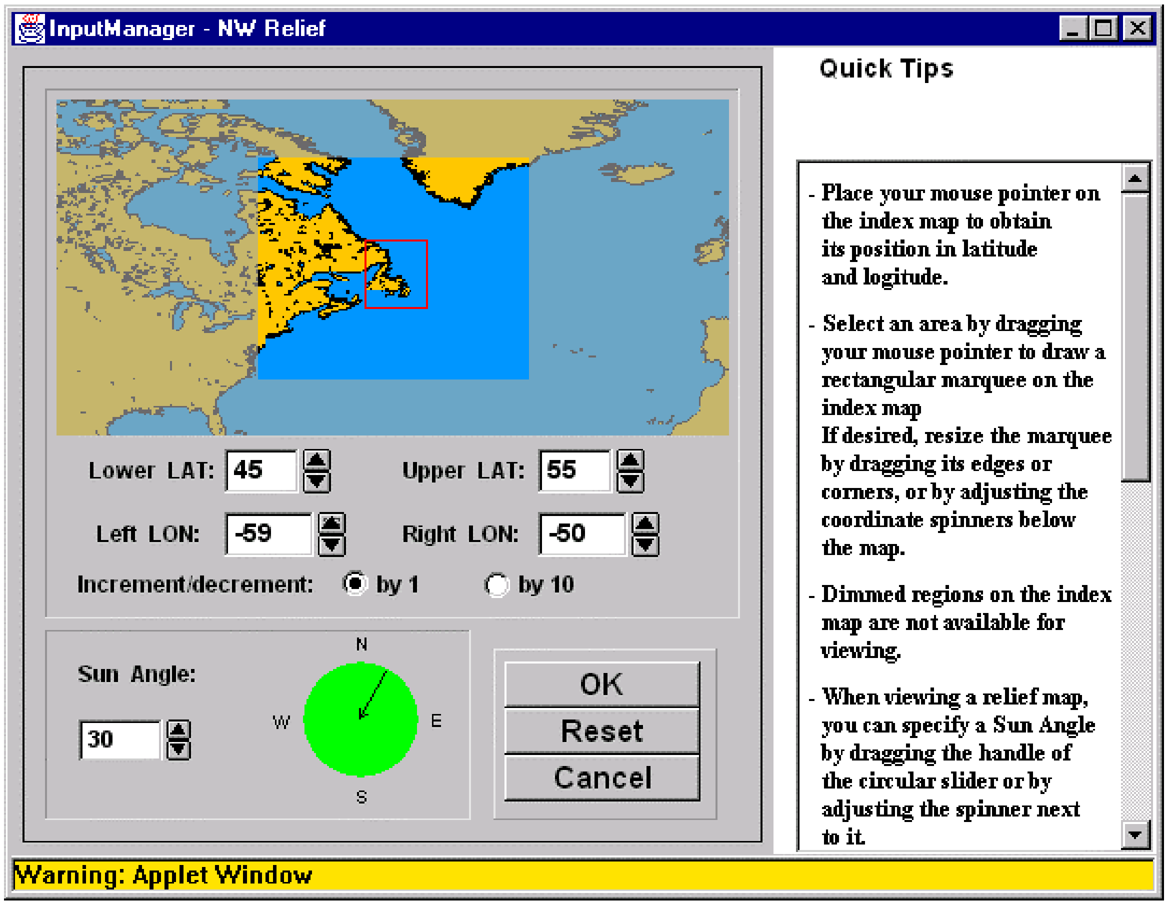
Figure 6.2 The input window for relief map.

The index panel is used for selecting the region for which a visualization (a relief map in this case) is required. The *index map* is located in the top of this panel. As the cursor moves over the index map, its position in latitude and longitude is displayed next to it (Figure 6.2). The user can select an area for viewing by holding the mouse button down and dragging a rectangular marquee on the index map. When the mouse button is released a selection is created, indicated by a rectangle as shown in Figure 6.3. If desired, the user can resize the selection by dragging its edges or corners. The dimmed regions on the index map indicate areas that are not available for visualization. The four *spinner controls* labeled **Lower LAT**, **Upper LAT**, **Left LON**, and **Right LO**N below the index map display the coordinates of the corners of the selected area rectangle. The user can edit these coordinates by incrementing or decrementing them using the scroll arrows. The increment/decrement step is determined by the setting of the *radio buttons* labeled **Increment/decrement**. As the values in the spinners are edited, the selection rectangle in the index map changes accordingly.

In the sun angle panel, the user can specify the Sun Angle for visualization of the relief by dragging the handle of the *circular slider* or by adjusting the *spinner* next to it.

In the button panel, the **Cancel** button is used to close the current input window. Clicking the **Reset** button resets all parameters to their defaults, and clicking the **OK** button requests the server to generate the required visualization.

Figure 6.4 shows the ship tracks input window, which differs from the relief map input window only in that the sun angle panel is replaced by a *color selector panel,* in which the user can specify track line color by clicking on the appropriate color button in the palette. The chosen color appears above the palette.


InputManager - NW Relief

Lower LAT: 45 Upper LAT: 55

Left LON: -59 Right LON: -50

Increment/decrement: (● by 1) ○ by 10

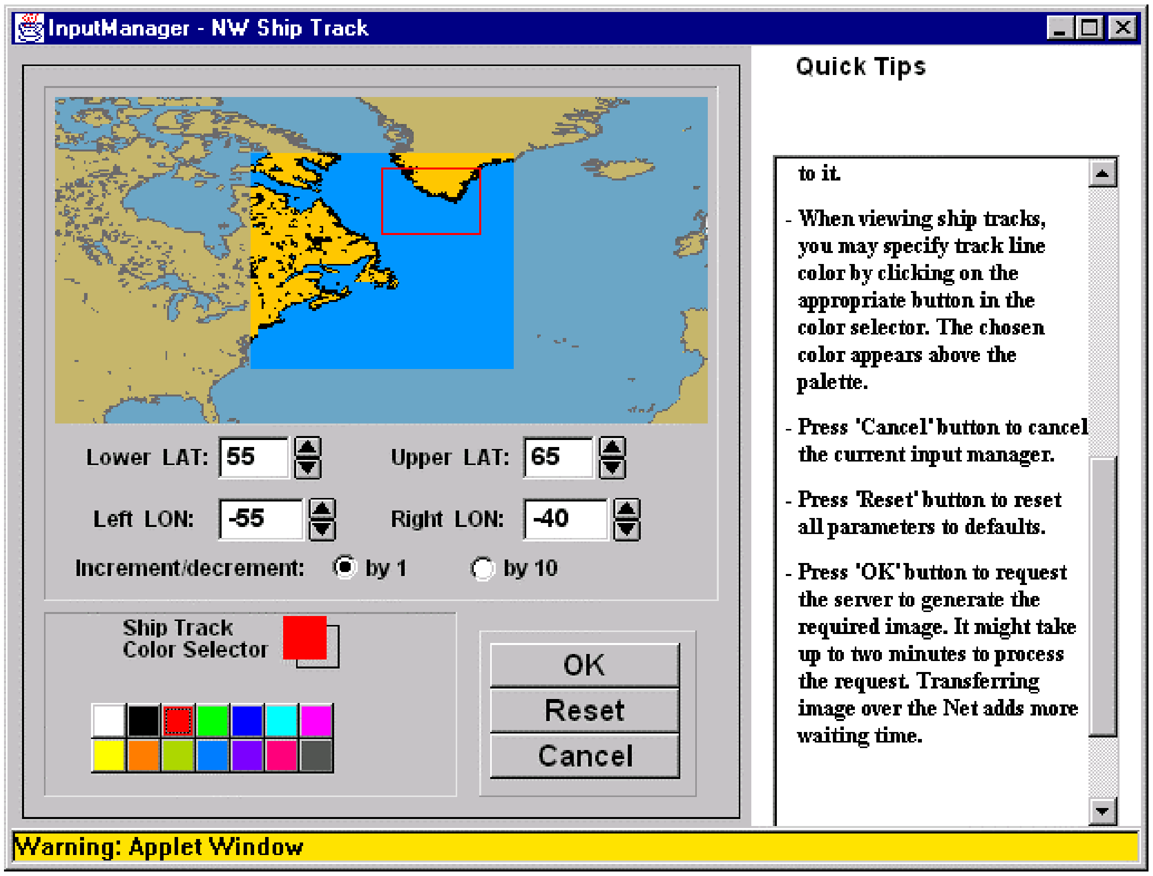
Sun Angle: 30

Quick Tips

- Place your mouse pointer on the index map to obtain its position in latitude and logitude.
- Select an area by dragging your mouse pointer to draw a rectangular marquee on the index map
If desired, resize the marquee by dragging its edges or corners, or by adjusting the coordinate spinners below the map.
- Dimmed regions on the index map are not available for viewing.
- When viewing a relief map, you can specify a Sun Angle by dragging the handle of the circular slider or by adjusting the spinner next to it.

Warning: Applet Window

Figure 6.3 Rectangular selection on the index map.



Lower LAT: 55
Upper LAT: 65
Left LON: -55
Right LON: -40
Increment/decrement: [✓ by 1] by 10
Ship Track Color Selector

to it.

- When viewing ship tracks, you may specify track line color by clicking on the appropriate button in the color selector. The chosen color appears above the palette.

- Press 'Cancel' button to cancel the current input manager.

- Press 'Reset' button to reset all parameters to defaults.

- Press 'OK' button to request the server to generate the required image. It might take up to two minutes to process the request. Transferring image over the Net adds more waiting time.

Figure 6.4 The input window for the NW Atlantic ship tracks with a rectangular selection on the index map and red selected as ship track line color.

6.2.4 Requesting the visualization

When the **OK** button in an input window is pressed, the window disappears and a request to the server is made for generating the required image.

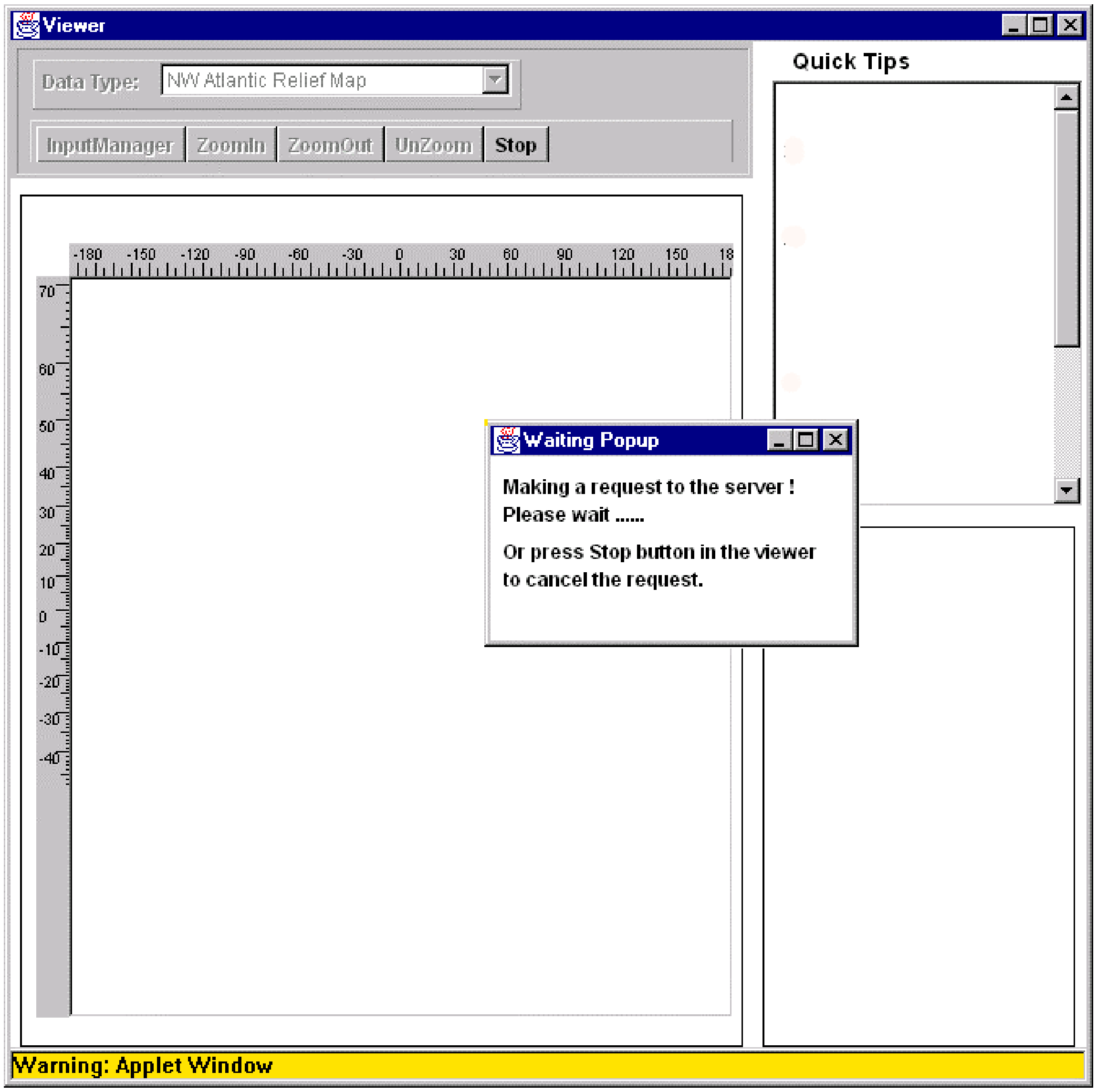
Creating the image and transferring it to the client is time-consuming, so a *waiting message box* is popped up (Figure 6.5). At this time, the **InputManager** button in the viewer window is not available, but the **Stop** button is available and can be pressed to cancel the request.

6.2.5 Viewing

When the required image is generated and transferred to the client, it is displayed in the viewing pane (Figure 6.6), the **Stop** button is disabled, and the **InputManager** and **Zooming** buttons are enabled. The image can be traversed by clicking the zooming buttons and scrolling. The rulers for latitude (nonlinear, Mercator Projection) and longitude (linear) change their values correspondingly. If the mouse button is held down while the cursor is in the image, information related to the corresponding map coordinates will appear if available, providing a simple form of information drill-down. For example, in Figure 6.7, the height or depth is shown on a relief map. This information will disappear when the mouse button is released.

6.2.6 Legend

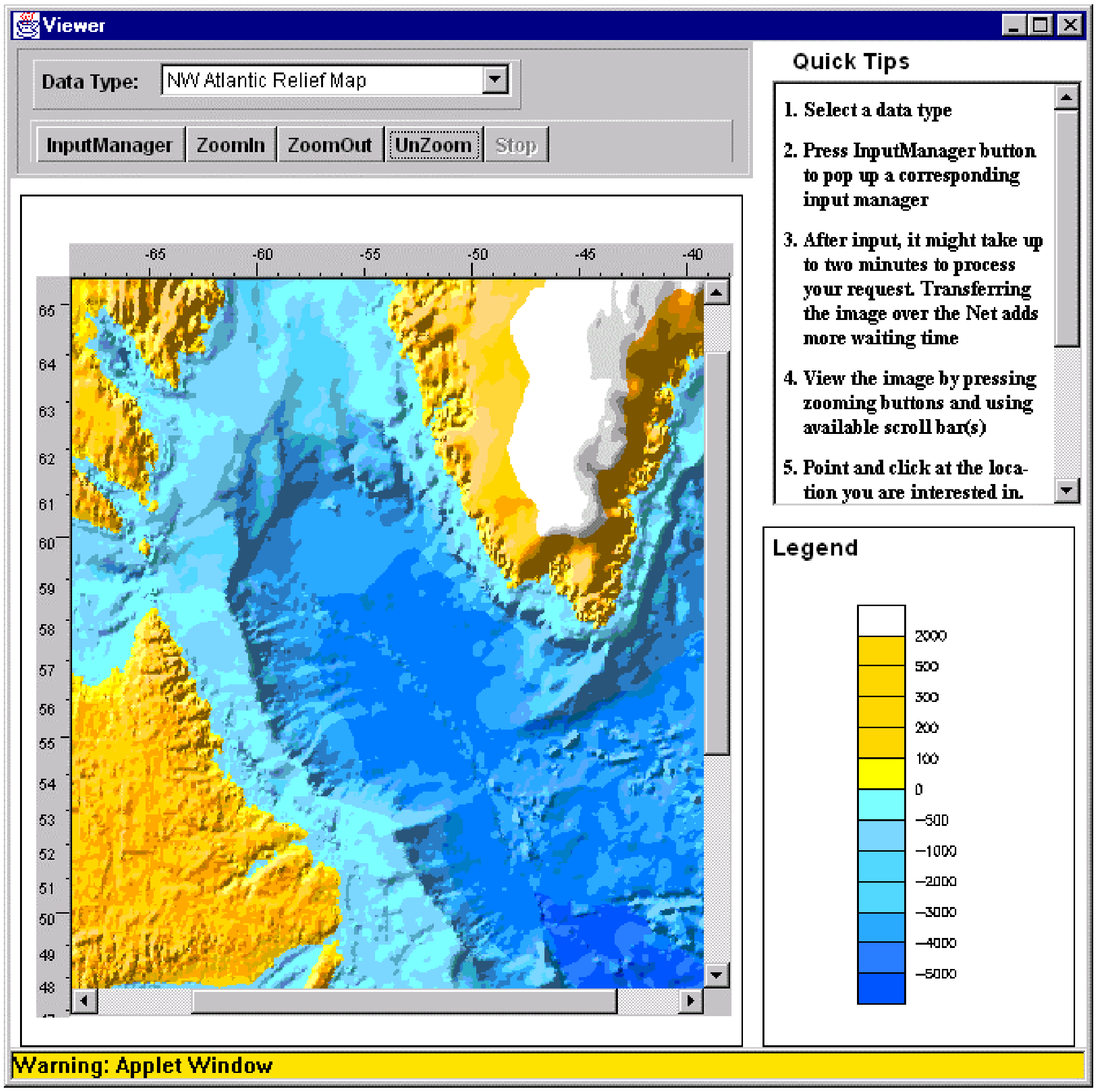
On the bottom right of the viewer window is a legend panel, which gives information related to the displayed image. For example, when we view relief maps, the legend displays the depths or heights indicated by the various colors. When we view ship tracks, the legend explains the meanings of the different colored lines.



Making a request to the server !
Please wait ......

Or press Stop button in the viewer
to cancel the request.

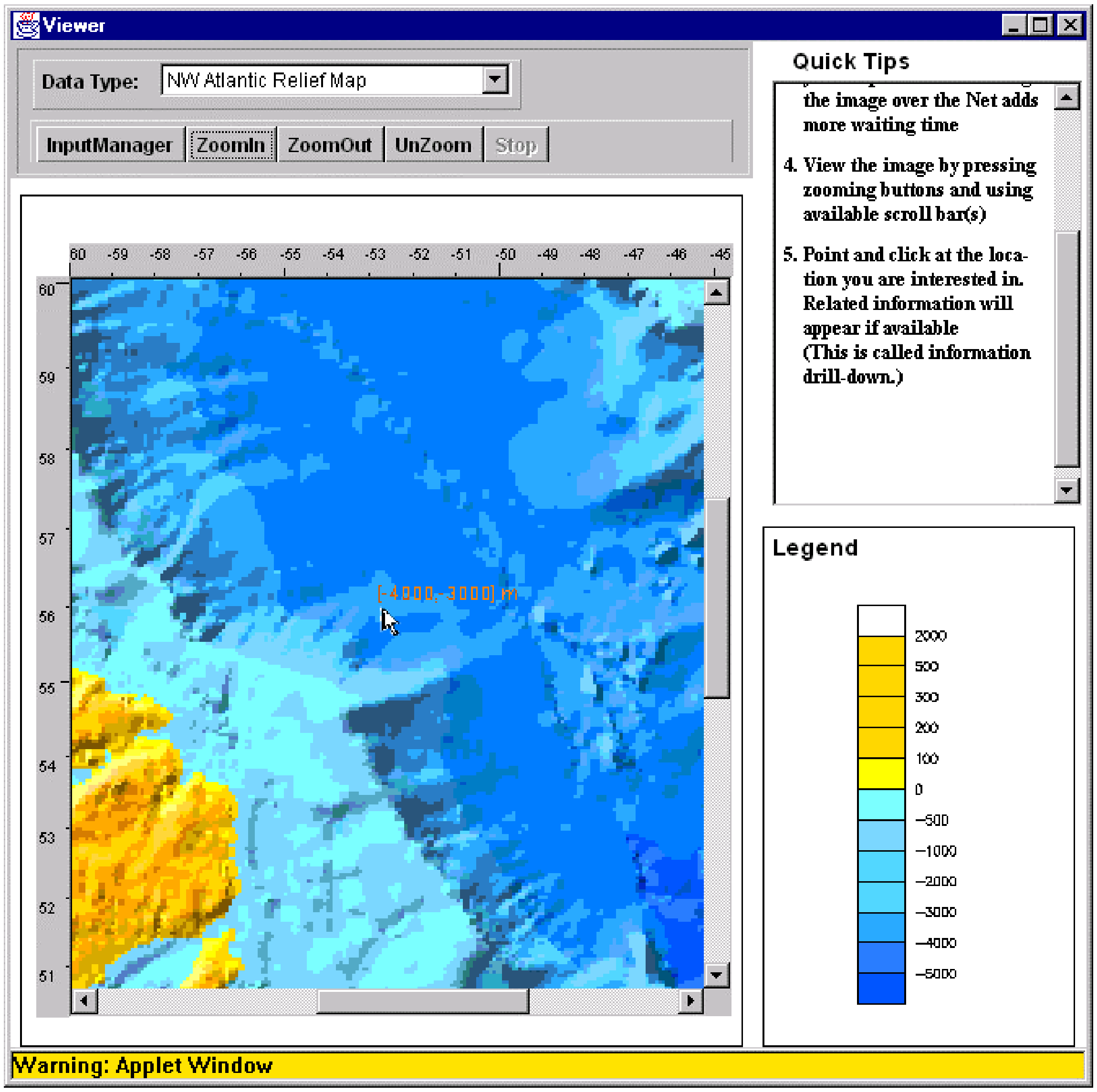
Figure 6.5 Waiting during a request to the server.



Quick Tips

1. Select a data type
2. Press InputManager button to pop up a corresponding input manager
3. After input, it might take up to two minutes to process your request. Transferring the image over the Net adds more waiting time
4. View the image by pressing zooming buttons and using available scroll bar(s)
5. Point and click at the location you are interested in.

Figure 6.6 Viewing a NW Atlantic relief map.



the image over the Net adds more waiting time

4. View the image by pressing zooming buttons and using available scroll bar(s)

5. Point and click at the location you are interested in. Related information will appear if available (This is called information drill-down.)

Figure 6.7 Depth is displayed by pressing and holding the mouse button at location of interest.

6.2.7 Quick Tips

On the right side of the viewer window and each input window is a scroll pane labeled **Quick Tips**, which provides help information appropriate to the context.

6.3 Web-Based GSC(A) Display Software

We first describe the architecture of the software, then discuss some important details.

6.3.1 Architecture

Our aim was to use the general re-engineering approach described in the chapter 5 to develop an architecture for Web-based GSC(A) Display without changing the existing non-distributed GSC(A) Display system. The architecture of the system, shown in Figure 6.8, is an expansion of the generic client server architecture in Figure 5.4.

The Harness is a Java applet, which includes a mechanism to communicate with the web server and provides the visualization environment for the end user. It was designed using object-oriented principles, and is explained in Appendix A.

The Back End Manager consists of an HTTP Web server which attends to the client Harness connections, the CGI script to drive the controller, the controller which starts the visualization engine using a shell script, and the PS2GIF converter to change the image generated by the engine from Postscript to GIF format.

The visualization framework and processing reside with a high-end Hewlett-Packard visualization server. The end user accesses the visualization facilities on the visualization server, driving the visualization from a Web browser by using the Harness

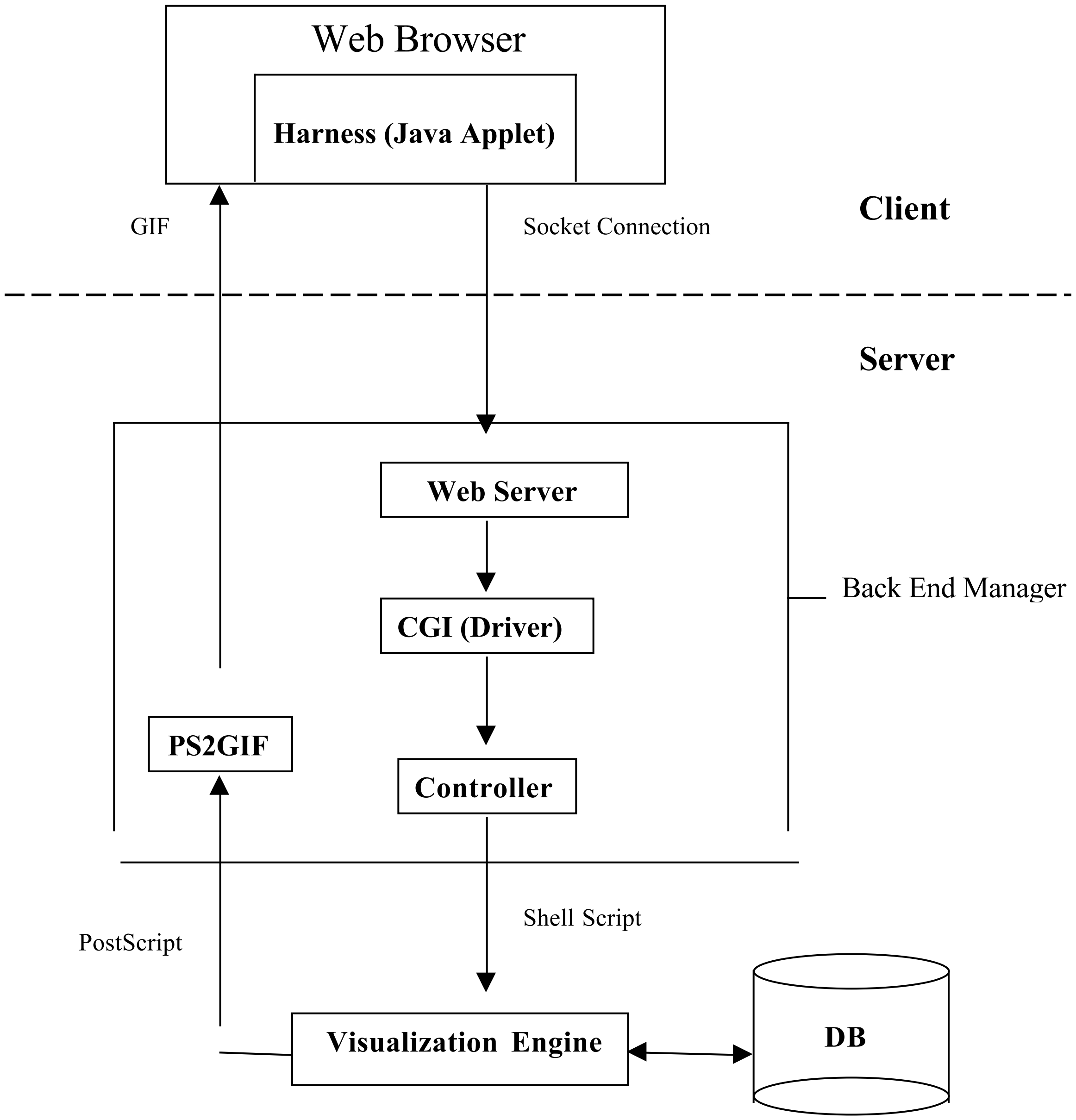


Figure 6.8 The Architecture of Web-Based GSC(A) Display.

downloaded to the browser on the fly. Visualization requests are delivered to the server which generates a GIF image and returns it to the client.

In the following, we explain some details of the software in this architecture.

6.3.2 Connection from Client Harness to Server Host

The Harness applet is downloaded from the server host to the client as required. It provides a visual user interface to the end user, collects user's inputs, sends them back to the server host for the visualization result, gets the GIF image from the server host and supplies various tools to the end user for the display.

There are three options to make a communication connection from a Java applet to the server host. The first is to make our own network connection from the Java applet to the server host. This would be done by implementing a server program, which waits for its client applets to contact it. To make the contact, this server program must continually monitor an agreed-upon port, which is not used by any of the standard services, such as email, HTTP and FTP services. The port is not a physical location, but is instead a software abstraction to identify a unique service program in a machine (mainly for bookkeeping purposes).

Although this seems like an elegant approach, it suffers from some problems, including a rather subtle one as follows. Since such a server program would be running constantly and spending most of its time waiting, it may affect CPU performance for other programs [6]. The second problem concerns firewalls in the client's organization. A firewall is a system that controls communication between the organization network and the Internet, monitoring all traffic to and from the Internet to ensure that communications conform to some standards. A firewall requires that all traffic meets these standards, and blocks any communication that does not. If a client is behind a firewall set by its own organization and the client starts connecting to the Internet using a Web browser, the firewall expects that all the client's transactions will connect to the server host using the standard HTTP port, which is 80. If the applet is trying to connect to a different port as described above, which is outside the range of the "protected" ports 0 – 1024 for standard services, the firewall does not allow the transaction to happen. As long as clients have direct connections to the Internet (for example, using a typical Internet Service Provider) without firewalls, there is no problem. But the VSP may have some important clients dwelling behind firewalls, and they will not be able to use an applet provided by the VSP to connect to its server host in this way. These situations are out of the VSP's control.

An alternative approach is to use HTTP and the CGI mechanism which provides a standard for interfacing external applications with HTTP servers. Such interfaced external applications are called *CGI programs* or *CGI scripts*, and can be invoked by Web users.

With this approach, the applet can send a CGI request to the Web server which would invoke a CGI script to process the request. This mechanism does not have the monitoring overhead required by the first option. There is no problem with firewalls either since all communications occur through an authorized port.

Obviously, there would be an advantage if there were a direct way for an applet to use the CGI mechanism that is already present in the browser. Unfortunately, the standard Java library does not provide such a communication path yet. We need to invoke a CGI script in the same way that a Web browser would by programming a socket connection through the HTTP port (port 80) of the server host and writing into the port a code that the Web server can understand. The socket is the software abstraction used to represent the "terminals" of a connection between two machines. For a given connection, there is a socket on each machine.

There is a problem with this second approach if the CGI script is written in Java. HTTP servers cannot start up a Java program because a compiled Java program (bytecode) is not executable and a Java interpreter has to be invoked to run the bytecode.

The third approach depends on a planned but not yet completely implemented technology known as *servlet servers*, which can automatically start up and run special server-side Java programs (called *servlets*) written using the Java Servlet API. With this technology, 100% pure server-side Java programs can be used to handle a client's connection and request. At present a low-cost servlet server is available from java.sun.com. In addition, Sun is encouraging other manufacturers to add servlet capabilities to their Web servers.

Because of the firewall issue and the lack of servlet servers on current Web servers, we chose the second option (Web server and CGI mechanism) for the connection from the Harness Java applet to the server host. A class called **HarnessConnection** was developed for this purpose and its Application Programming Interface (API) is shown in Appendix B. In addition, we wrote the CGI script in Java and overcame the problem that Web servers cannot start up a Java program automatically, by adding a *CGI script driver*, discussed in the next section.

6.3.3 CGI Script Driver

CGI scripts can be written in many languages that can read from standard input and write to a standard output. CGI programs are typically written in either Perl or C, and must handle everything from the client request. For consistency, we chose to use Java instead of C or Perl to write our program called *Controller*, which will be discussed in the next section.

Of course, we faced the problem described above, that a Web server cannot run a Java program automatically. To solve this problem we used a CGI script driver, software that performs an interfacing function between the Web server and our Java program by invoking a Java interpreter to run the Java bytecode. The driver is written in Unix K shell script, and can be started up automatically by the Web server through the CGI mechanism. It in turn invokes the Java interpreter to run the *Controller* in Java bytecode. The CGI script driver for world relief is shown in Appendix C.

6.3.4 Controller

The Controller receives inputs from the client, sets default values for parameters which are hidden from the users but are required by the visualization engine (refer to Table 6.1), and drives the visualization engine. The controller for world relief is shown in Appendix D. After the visualization engine generates the Postscript image, the Controller invokes the PS2GIF program to change the image format to GIF for Internet transport.

6.3.5 PS2GIF Converter

The GSC(A) Display visualization engines can generate only Postscript since the original aim of the software was to produce printed maps. Postscript is in fact a complete programming language and not a graphics-description format, so for graphics, Postscript files tend to be large and therefore expensive to transmit over the Internet.

There are two criteria for deciding whether an image format is suitable to be displayed by an applet over the Internet. One is whether the format is compact enough to be sent in reasonable time, another is whether the format is supported directly by browsers or by classes already available in the Java libraries. Postscript is excluded on both counts, while the GIF format satisfies both criteria.

The Graphic Interchange Format (GIF) is a widespread graphic format with an 8-bit (256 colors) color palette, and incorporates a compression scheme (Lempel Zev Welch, or LZW) to keep files sizes at a minimum. LZW compression squeezes out inefficiencies without causing any loss of information. GIF was introduced by the Compuserve Information Service in the 1980s as an efficient means to transmit images across data networks. In the early 1990s the designers of the Web adopted the GIF format for its efficiency and widespread familiarity. There are now several slight variants (such as "GIF87a" or "GIF89a") of the basic GIF file format that add support for transparent color, and support for the interlaced GIF graphics popularized by the Netscape Navigator Web browser.

We chose the GIF format for transmitting our images for two additional reasons. GSC(A) Display generates the map images with only a small number of colors. So the 8-bit color palette (256 colors) is suitable for these kinds of images. Secondly, Ghostscript (GS) available on most UNIX systems including the GSC(A) servers, is an interpreter for the Postscript language providing several output devices. One of them, "gif8", translates GS bitmap output into Compuserve GIF format. This provides us with the Postscript to GIF translations capability we need.

Appendix E provides details of the PS2GIF converter for world relief. The program not only runs the GS interpreter but also sets many appropriate options in order to correct results.

6.4 Summary

In this chapter we have presented a practical and concrete example of the general reengineering approach described in the chapter 5. In this example, the existing non-distributed GSC(A) Display was reengineered into the Web-based GSC(A) Display system with many useful features and functions. Details of the system architecture and implementation, as well as key software programs were discussed.

Chapter 7 Conclusions and Future Work

Over a period of forty years GSC(A) has collected large amounts of useful geographic data and more recently has developed in-house display software (GSC(A) Display) to generate color images of selected data sets for information visualization and analysis. This application is non-distributed and can only be used internally by GSC(A) staff. In order to make more widespread use of the data and visualization software, GSC(A) were interested in somehow deploying it via the Web. Addressing this practical problem provided the starting point for the research reported here. During the progress of our software development effort, we found that most legacy SciVis software has limitations similar to those of GSC(A) Display and could benefit from a similar re-engineering. The issues involved led to the systematic study reported here.

The World Wide Web provides a new mechanism for publishing scientific visualizations to a wide audience. Our work started with studying DiBiase's scientific visualization sequence which characterizes the evolution of the use of SciVis tools. This resulted in an extension to DiBiase's model. In this extended model, scientific visualizations can be categorized into four types ("I See", "They See", "We See" and "World Sees") and can be used in three realms (private, public and global). Scientific visualizations can be applied in the global realm to provide a public service: that is a visualization service to be used by individuals with a technical or professional interest in the data. An additional classification of visualization systems in a multidimensional space with three variables (purpose/task, user, and HCI) was also presented.

The work then progressed to an investigation of the structure of current SciVis software. SciVis software is implemented according to the visualization pipeline architecture [11, 39] and can be classified as "commercial" or "in-house". Commercial

SciVis software provides solutions for simple and general problems. In-house SciVis software is built using graphics packages within scientific organizations and meets specific needs in certain scientific fields. However, both commercial and in-house SciVis software has generally been designed as non-distributed applications, not in the "World Sees" category. It is expected that commercial SciVis software will be updated by its manufacturers into collaborative or Web-based versions. The remaining problem is how to re-engineer in-house visualization applications cost-effectively as multiple-user applications for public service.

The rest of our work dealt with this problem. A client/server architecture is used in the re-engineering with the Web providing the deployment medium. Taking this approach, we examined several possible scenarios for SciVis to be made available on the Web. These scenarios are:

- Provider creates the image visualization (static)
- Provider creates the 3D visualization (static)
- User creates the visualization (dynamic)
- Provider creates visualization framework and user chooses the options (dynamic)

Through the analysis of the re-engineering requirements, we made two important decisions, one of which was to choose a compromise scenario for dynamic visualization and the other was to use the Web as the deployment medium. A modified three-tier client/server configuration was proposed according to these two decisions. This configuration uses a thicker client providing some application services, and can satisfy both the needs of visualization and the partitioning of functionality that is necessary for Web deployment. An extension to the visualization pipeline architecture was also proposed to tailor the pipeline to this client/server configuration. The central parts of the re-engineering are the development of a *Back End Manager* on the server and a *Harness* on the client requiring minimal modification to the existing software.

Finally, an example of this re-engineering approach is provided, consisting of the implementation of a Web-based geographic information visualization service based on the legacy SciVis software developed by GSC(A).

7.1 Future Work

In the Harness class diagram (Appendix A), some useful tools for scientific visualization have been identified. It would be worthwhile to continue to identify more such general tools in order to develop a framework for building Web-deployed scientific visualization systems. Currently, one of the most promising technologies for implementing such a framework is *Java Beans,* which provide components rather than classes, and are therefore ideal for implementing more complex tools. Furthermore, there are many integrated development environments that support Beans.

Once a visualization has been requested, the user of Web-based GSC(A) Display must wait for a considerable time for the image to be generated and transmitted. There are two bottlenecks, which might be removed in the future. One is the visualization engine, which needs approximately one minute to generate the large Postscript file that results from a typical request using the current GSC(A) data sets. Approximately one more minute is required to convert this Postscript file to a GIF file. Although our focus so far has been on providing a service without changing the legacy software, such changes will clearly be necessary to improve performance. In the future, therefore, the visualization engine may be modified to generate the small GIF file directly. Another bottleneck is simply the need to transmit large image files, GIF files in our case. To obtain a significant improvement in cases where the data size is small and the corresponding image file is large, the data could be transmitted to the client which would perform the data manipulation and rendering. Under this policy, the whole or much of the visualization pipeline must be moved from the server to the client and implemented in the form of a Java applet or Plug-in. Note that in this case, we are abandoning the legacy system altogether.

In any future framework for building Web-based SciVis systems, we should provide a tool for integrating visualization on the server and visualization on the client. This tool should have the capability to generate visualization programs for simple visualization tasks on the client as well as the capability to allow these new programs to cooperate with the legacy system.

References

[1] Alias|Wavefront (November 1998), *Alias|Wavefront – Hom*e, http://www.aw.sgi.com/pages/home/index.html

[2] AVS Inc. (November 1998), *Advanced Visual Systems*, http://www.avs.com

[3] Booch, G. (1994)*, Object-Oriented Analysis and Design*, Addison-Wesley, New York.

[4] Buckingham Shum, S. (1996), The missing link: hypermedia usability research & the Web, *ACM SIGCHI Bulletin* 28 (4): 68-75.

[5] DiBiase, D. (1990), Visualization in the earth sciences, *Earth and Mineral Sciences, Bulletin of the College of Earth and Mineral Sciences, PSU,* 59(2):13-18.

[6] Eckel, B. (November 1998), *Thinking in Java*, http://www.eckelobjects.com/javabook.html

[7] Edmonds, E.A., Murray, B.S., Ghazikhanian, J. and Heggie, S.P. (1992), The re-use and integration of existing software: a central role for the intelligent user interface, *People and Computers VII, Proceedings of the HCI'92 Conference (York, September 1992)*: 415-427.

[8] ESRI (November 1998), *ArcView GIS*, http://www.esri.com/software/arcview/index.html

[9] Graphics, Visualization and Usability Center, Georgia Institute of Technology (February 1998), *Scientific visualization tutorial*, http://www.cc.gatech.edu/scivis/tutorial/tutorial.html

[10] Green, T.R.G. and Petre, M. (1996), Usability analysis of visual programming environments: a 'cognitive dimensions' framework, *Journal of Visual Language and computing* 7: 131-174.

[11] Haber, R.B. and McNabb, D.A. (1990), Visualization idioms: A Conceptual model for scientific visualization systems, In: *Visualization in Scientific Computing*, Shriver, B., Nielson G.M. and Rosenblum L.J. (ed.), IEEE Computer Society Press, 74-93.

[12] Haber, R.B. and McNabb, D.A. (1990), Eliminating distance scientific computing: an experiment in televisualization, *The International Journal of Supercomputer Applications*, 4(4):71-89.

[13] Hearnshaw, H.M. and Unwin D.J. (1994), *Visualization in Geographical Information Systems*, John Wiley & Sons, New York.

[14] Hornby, A. S. (ed.) (1985), *Oxford Advanced Learner's Dictionary of Current English*, Oxford University Press, Oxford.

[15] Horstmann C. S. and Cornell, G. (1997), *Core Java*, Sun Microsystem Press, California.

[16] IBM (November 1998), *IBM data visualization explorer (DX)*, http://www.almaden.ibm.com/dx/DXHome.html

[17] Jern, M. (December 1997), *Information drill-down using Web tools*, http://www.uniras.dk/info/seminars/Drilldown.htm

[18] Johnson, C. (February 1998), *The impact of retrieval delays on the value of distributed information*, http://www.dcs.gla.ac.uk/~johnson/papers/value.html

[19] Landay, J. (August 1998), *Model view controller*, http://bmrc.berkeley.edu/people/allanl/160/lectures/model-view-controller/sld001.htm

[20] Lemay, L. (1995), *Teach Yourself Web Publishing with HTML in a Week*, Sams Publishing, Indianpolis.

[21] Lynch, P. and Horton, S. (December 1997), *Yale C/AIM Web style guide*, http://info.med.yale.edu/caim/manual/index.html

[22] MacEachren, A.M. and Kraak, M. (January, 1998), *ICA commission on visualization*, http://www.geog.psu.edu/ica/ICAvis.html

[23] MacEachren, A.M., Bishop, I., Dykes, J., Dorling, D. and Gatrell, A. (1994), Introduction to advances in visualizing spatial data, In: *Visualization In Geographical Information Systems*, Hearnshaw, H.M. and Unwin D.J. (ed.), John Wiley & Sons, New York, 51-59.

[24] MacEachren, A. M. and Ganter, J. H. (1990), A pattern identification approach to cartographic visualization, *Cartographica* 27(2): 64-81.

[25] MacEachren, A. M. and Taylor, D.R.F. (1994), *Visualization in Modern Cartography*, Elsevier Science, Oxford.

[26] Maui High Performance Computing Center (November 1998), *Introduction to Khoros visualization tools*, http://www.tnt.unihannover.de/soft/imgproc/khoros/khoros1/more-info.html

[27] McCormick, B.H. , DeFanti, M.D., Brown, M.D (eds). (1987), Visualization in scientific computing, *Special Issue ACM SIGGRAPH Computer Graphics* 21(6).

[28] McKie, S. (1997), *Client/Server Accounting, Reengineering Financial Systems*, John Wiley & Sons, Inc., New York.

[29] Miller, A.I. (1996), *Insights of Genius: Imagery and creativity in science and art*, Springer-Verlag, New York.

[30] Putz, S. (1994), Interactive information services using world-wide web hypertext, *Computer networks and ISND System* 27: 273-280.

[31] Rational Software Corporation (November 1998), *Unified Modeling Language*, http://www.rational.com/uml/index.shtml

[32] Rhyne T. M. (1994), The Application Visualization System (AVS) as a tool for visualizing geographic data sets, In: *Visualization in modern cartography*, MacEachren A. M. and Taylor D. R. (ed.), Elsevier Science, Oxford, 106-112.

[33] Rhyne T. M. (1995), A WWW viewpoint on scientific visualization: an EPA case study for technology transfer, *Proceedings of 1996 IEEE Information Visualization* October 1995, Atlanta, Georgia: 112-114.

[34] Rhyne T. M. and Fowler T. (February 1998), *Examining dynamically linked geographic Visualization*, http://www.epa.gov/vislab/svc/publications/awma-gisvis.html

[35] Schmidt, V. (December 1997), *Dynamic 2D information visualization*, http://www.avs.com/products/web/pres-2d

[36] Scientific Visualization Center, USA Department of Defense (December 1997), *What is Scientific Visualization?* http://apollo.wes.army.mil/svc/scivis.html

[37] Synder, J.P. (1987), *Map projections – a working manual*, United States Government Printing Office, Washington.

[38] The Numerical Algorithms Group, Inc. (November 1998), *IRIS explorer documentation*, http://www.nag.co.uk/visual/IE/iecbb/DOC/Index.html

[39] Upson, C., Faulhaber, T., Kamis, D., Schleger, D., Laidlaw, D., Vroom, F., Gurwitz, R. and vainDam, A. (1989), The application visualization system: A computational environment for scientific visualization, *IEEE Computer Graphics and Applications* 9(4): 30-42.

[40] US Army Construction Engineering Research Laboratories (November 1998), *USACERL WWW HomePage*, http://www.cecer.army.mil

[41] Visual Numeric Inc. (November 1998)*, PV-WAVE – Analysis, Mathematics, Visualization*, http://www.vni.com/products/wave

[42] Wessel, P. and Smith, W.H.F. (December 1997), *Online map creation*, http://www.aquarius.geomar.de/omc/

[43] Wood, J. (1998), *Collaborative visualization*, PhD thesis, University of Leeds, Leeds.

[44] Wood, J., Brodlie, K. and Wright, H. (1996), Visualization over the World Wide Web and it's application to environmental data, *Proceedings of Visualization '96* October 27 – November 1, 1996. San Francisco, CA: 81- 86.

Appendices

Appendix A - Harness Class Diagram

The diagram of the major classes involved in the Harness is shown in Figure A.1, expressed in Unified Modeling Language (UML), the standard notation for describing object-oriented designs [3]. These classes might be considered in further research on the development of scientific visualization framework. Before we discuss them, we will briefly describe UML and the structure of class diagrams for the benefit of readers unfamiliar with this notation. For a full description, see UML Summary [31].

UML is a highly visual language relying mainly on graphics and symbols, and to a lesser extent on textual annotations. In objected-oriented analysis and design, the *class diagram* is of primary importance. The two essential elements of a class diagram are classes and the relationships between them. In the class diagram, a class is represented by a rectangle containing the class name at the top, optionally followed by the names of attributes and methods. Preceding the names of attributes and methods are symbols representing *visibility modifiers*. For example, a lock symbol expresses the fact that an attribute or method is 'Private'. The connections between classes indicate inheritance, aggregation or association relationships. Inheritance appears as a single line with an arrowhead indicating the superclass. Aggregation (or composition) is denoted by a single line with a diamond at the end denoting the aggregate or container. Association, a semantic connection between two classes, appears as a simple line. The cardinality of each relationship can also be shown in the diagram, but it is omitted in the Harness class diagram for the clarity because most relationships in this class diagram are one-to-one.

The Harness class diagram (Figure A.1) supplied the expressive notation and working heuristics needed for efficient development and maintenance of the visualization

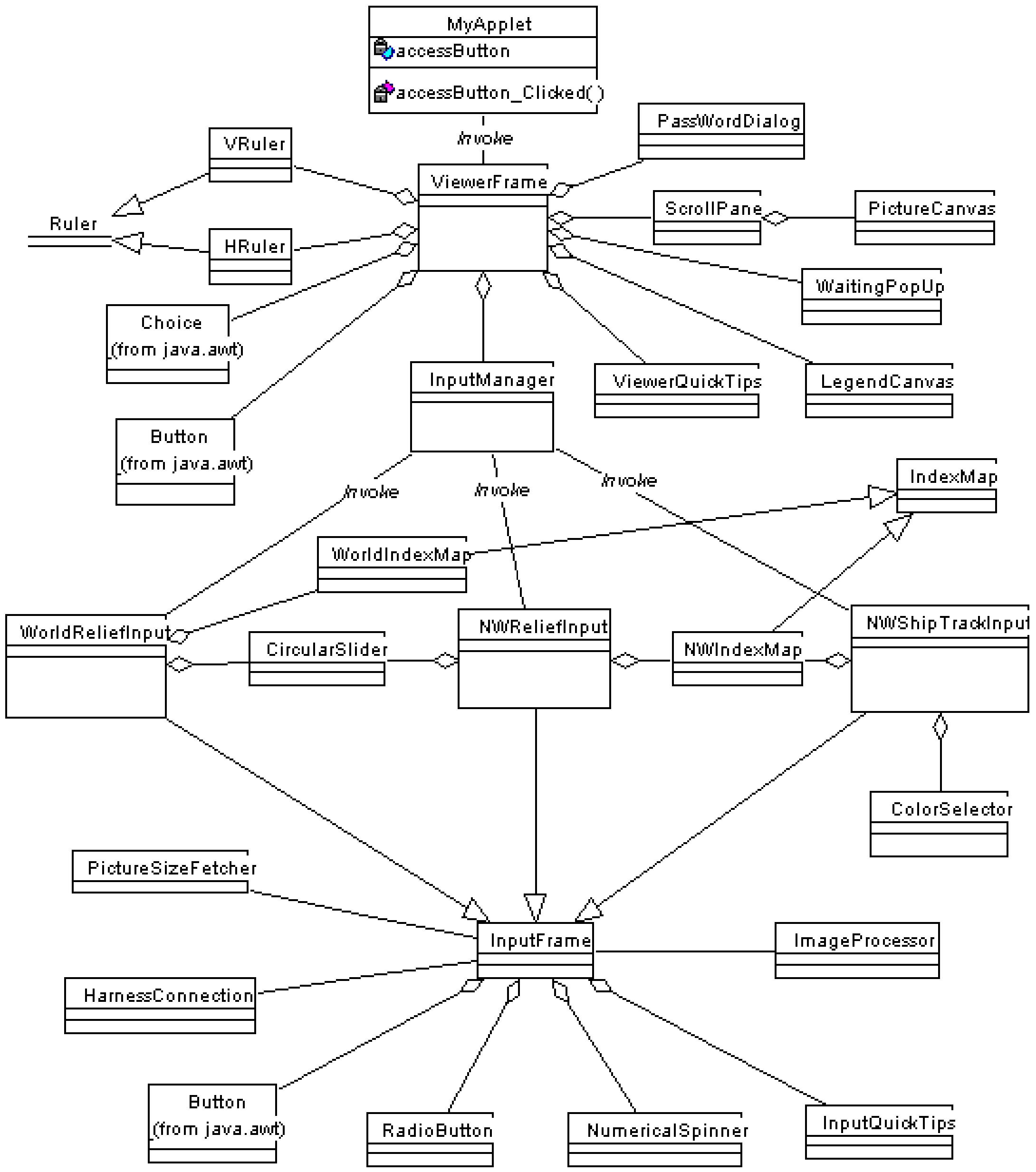


Figure A.1 The Harness Class Diagram.

environment during the design and implementation of Web-Based GSC(A) Display. The visualization environment supplies the visual user interface, which consists of concrete objects corresponding to real-world entities, such as rulers, index maps, circular sliders, and so forth.

In the class diagram, the visual user interface starts at the **MyApplet** object, which invokes the **ViewerFrame** object when a user presses the access button. The **ViewerFrame** object has an **InputManager** object, which can invoke **WoldReliefInput** object, **NWReliefInput** object, or **NWShipTrackInput** object according to the selected item in the data type **Choice** object.

There are several objects in the **ViewerFrame** object. The **Choice**, **Button** and **ScrollPane** classes are all from the Java AWT package. **VRuler** class and **HRuler** class are both inherited from **Ruler** class, which is abstract and contains several general concrete methods for a ruler. The subclasses of the **Ruler** have to implement its abstract method paint( ). The **VRuler** is a linear scale ruler. The **HRuler** is a non-linear ruler the scale of which is calculated according to the Mercator Projection [37]. They are responsible for dynamically displaying latitude and longitude coordinates for the map area shown in the scroll pane during scrolling or zooming. The **PassWordDialog** class controls the login into the **ViewerFrame** and the rest of the program. The **ViewerQuickTips** class provides on-line help for viewing while the **LegendCanvas** class dynamically supplies the corresponding legends to different visualizations. The **WaitingPopUp** class pops up a dialog frame to ~~a~~ while the user's visualization request is being processed. The **ScrollPane** class has a **PictureCanvas** class, which displays the visualization in the **ScrollPane**.

The **WorldReliefInput** class, **NWReliefInput** class and **NWShipTrackInpu**t class are all inherited from **InputFrame** class. These three subclasses aggregate

WorldIndexMap, **NWIndexMap**, **CircularSlider**, and/or **ColorSelector** classes. The **CircularSlider** is in charge of the user input for the sun angle, and the **ColorSelector** is used for choosing ship track line color. The **WorldIndexMap** and **NWIndexMap** classes are both inherited from **IndexMap** class, which provides marquee selection and displays latitude and longitude corresponding to mouse position.

The **InputFrame** class provides functionality inherited by its subclasses for inputting, making requests to the server, and processing the visualization result from the server. There are a few objects in this class. The **Button** and **RadioButton** are from Java AWT package. The **NumericalSpinner** is used for display of the input parameter value and fine adjustment of the input. The **InputQuickTips** supplies on-line help for inputting. This class also uses the **HarnessConnection** to send a request to the server and fetch the visualization result from the server, the **PictureSizeFetcher** to obtain the image size for image processing, and the **ImageProcessor** to remove the blank space in the original image.

Appendix B – Class HarnessConnection API

Class HarnessConnection

Description : The class connects to the server and fetches back the generated GIF to the client.

public Harnessconnection( )
description: the constructor.

public Image connectAndFetch( )
description: connect client to server, send the data, invoke back end programs and fetch the generated GIF file from server to client
You should call setHome( ), setScript( ) and setSendingData( ) before using this method.
return value: an image object containing the generated GIF image

public void setHome(String home)
description: set the server host name, e.g. "agc2.bio.ns.ca"
parameter home: server host name

public void setScript(String script)
description: set the cgi script name and its whole path, e.g. "/cgi-bin/cgiwrap/gao/InfoReceiver.cgi"
parameter script: the cgi script name and its whole path

public void setSendingData(String sdata)
description: set the sending data, which will be delivered to server
parameter sdata: the sending data string (a datum per line)

Appendix C – Source code of CGI Script Driver for World Relief

```
#!/usr/bin/ksh
CLASSPATH=/opt/java/lib/classes.zip/:/home/mrg/gao/public_html/cgi-bin/
export CLASSPATH
/opt/java/bin/java WorldReliefController
```

The first line invokes the Unix K shell interpreter of the Unix operating system to run the rest of this driver written in K shell script. The second and third lines set the CLASSPATH required to run Java program, and the last line brings up the Java interpreter to run *Controller* program, detailed in the following.

Appendix D – Source Code of the *Controller* for World Relief

```
import java.io.*;

/**   WorldReliefController class
 *    This is a short application program. It is invoked by
 *    a cgi scrip 'WorldReliefDriver.cgi'. Then it receives
 *    the parameter values from client end, links its database
 *    and starts its visualization engine, and invokes PS2GIF
 *    to convert postscript to gif after the visualization
 *    engine generates the postscript image.
 *    @version GSC(A) DISPLAY 1.0, 1998
 *    @author Zhihong GAO
 *    Copyright (C) 1998 GSC(A) & Daltech   All right Reserved
 */

public class WorldReliefController {
  public static void main(String[] args) {
     System.out.println("Content-type: text/plain\n");
        try {
         BufferedReader stdin = new BufferedReader (
                                  new InputStreamReader(System.in));
          //{   receive the inputs from client side
         String latL = stdin.readLine();   String latU = stdin.readLine();
         String lonL = stdin.readLine();  String lonR = stdin.readLine();
         String sunAngle = stdin.readLine();
         stdin.close();
         //}
        // {  set other parameter values required for visualization engine
          String shell = "#!/usr/bin/csh";
         String gksdir = "setenv gksdir /usr/grafpak/gks4101";   //gks = Graphical Kernel System
          String displayProg = "/home/mrg/gao/public_html/programs/"
                                   +  "sunmer_gao_world << MMM";
         String dataset = "/home/mrg/gao/database/etopo.i2";
          int typeOfOutput = 3;        int size = 1;
          String psFileName = "/home/mrg/gao/public_html/dynamic_bin/"
                                   +  "World_relief.ps";
          String title = "WORLD RELIEF";    int refLat = 0;
          String rotate = "n";   String scale = "/";
         double space_plot_frame = 0;   int typeOfFrame = 2;
          int frameColor = 1;   String legend = "n" ;
          double saturation = .75;   String factor_resolution = "/";
          int typeOfColorBar = 6;
         String nameOfColorBar = "/home/mrg/gao/ice.colors";
         String shadowGradient = "/";     String gradientInterval = "/";
          String whiteContrast = "/";          String blackContrast = "/";
          String plot_land_file = "n";        String end = "MMM";
         //}
        //{ The following is to create the  shell script
        PrintWriter out = new PrintWriter((new
            FileWriter("/home/mrg/gao/public_html/dynamic_bin/"  +
                        "World_relief.txt")), true /*autoFlush*/);
         out.println(shell);   out.println(gksdir);   out.println(displayProg);
        out.println(dataset);   out.println(typeOfOutput);   out.println(size);
        out.println(psFileName);   out.println(title);   out.println(refLat);
         out.print(latL);   out.print(" ");    out.print(latU);   out.print(" ");
        out.print(latL);   out.print(" ");    out.print(latU);   out.print(" ");
         out.print(lonL);   out.print(" ");    out.println(lonR);
        out.println(rotate);   out.println(scale);
        out.println(space_plot_frame);   out.println(typeOfFrame);
        out.println(frameColor);   out.println(legend);
        out.println(saturation);   out.println(sunAngle);
        out.println(factor_resolution);   out.println(typeOfColorBar);
        out.println(nameOfColorBar);   out.println(shadowGradient);
        out.println(gradientInterval);   out.println(whiteContrast);
        out.println(blackContrast);   out.println(plot_land_file);
        out.println(end);   out.close();
         // }
        // { To invoke Visualization engine through the shell script
         Runtime rt = Runtime.getRuntime();
        Process p = rt.exec("/home/mrg/gao/public_html/dynamic_bin/
                                 + "World_relief.txt");
        p.waitFor();
         // }
        // { To invoke a script ps2gif to change image format from PS to GIF
        Process pp = rt.exec("/home/mrg/gao/public_html/dynamic_bin/" +
          "ps2gif_WR/home/mrg/gao/public_html/dynamic_bin/World_relief.ps");
        pp.waitFor();
         // }
      }  catch (Exception e) {System.out.println("Error " + e);}
    }
}
```

Appendix E - PS2GIF Converter for World Relief

```
#!/usr/bin/csh
#
#  Author: Zhihong GAO
#  Copyright (C) GSC(A) & Daltech 1998.   All Rights Reserved.
#
#  “gs”   run Ghostscript interpreter
#  "-q”    to prevent gs from writing messages to OutputFile
#  "-dNOPAUSE"   don't pause between pages
#  "-sDEVICE"   switch the output device
#  "-sOutputFile"   switch the output file
#  "--"   when gs finishes executing the file, it exists back to the shell
#
set file = $argv[1]
/usr/contrib/bin/gs -q -dNOPAUSE -sDEVICE=gif8
-sOutputFile="/home/mrg/gao/public_html/Images/dynamic_image_WR.gif" $file --
```