CODEN:LUTFD2/(TFRT-5389)/1-047/(1988)

An Evaluation of a PHIGS Implementation for Full Graphics Control Systems

Ulf Jeppsson

Department of Automatic Control
Lund Institute of Technology
November 1988

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS REPORT

Date of issue
November 1988

Document Number
CODEN:LUTFD2/(TFRT-5389)/1-047/(1988)

Author(s)
Ulf Jeppsson

Supervisor
S. E. Mattsson, D. Brück, B. Johansson, K. Bladh

Sponsoring organisation
ABB Corporate Research and ABB Network

Title and subtitle
An Evaluation of a PHIGS Implementation for Full Graphics Control Systems

Abstract

When designing new full graphics control systems, it is important to investigate existing and coming graphics standards. A well-designed graphics support system can significantly simplify and enhance the implementation of such systems as well as providing the necessary performance to support interaction and dynamic modification of the graphics data in real-time.

The latest contribution to the graphics community is the Programmer's Hierarchical Interactive Graphics System (PHIGS). It is the first major step from Core-based graphics systems (including GKS) and provides such possibilities as grouping the graphics information in a centralized, hierarchical graphics database, traversal time-binding, a name set mechanism for detectability, highlighting and visibility control, and explicit structure body editing on-line. This project has investigated these high-level features by developing and implementing a small prototype application of the power supply system of a city. Interesting ideas in the field of man-machine interaction were also tested.

The application supports hierarchical models and different levels of representation combined with a multiple window environment. Zooming controls the amount of information displayed. When zooming in on an object, it changes from an annotated box into a representation showing its internal structure in higher levels of detail. The description of each level can be made clear and simple. Overview windows to indicate where the user is when he pans and zooms are also supported. All interaction is performed via a mouse by selecting actions from a menu.

The PHIGS implementation used was the Swedish National Microelectronics Programming (NMP) PHIGS package, with the X window system for graphical input and output. The application was implemented in C.

Key words
PHIGS, Man-Machine Interaction, Computer Graphics, User Interface

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
47

Recipient's notes

Security classification

Table of Contents

1. Introduction

During the last few years, computer graphics has been one of the most interesting and rapidly growing fields in computer science. Today, graphics is one of the most effective medium for communication between man and machine and is used in every modern control system. New software, combined with high performance workstations – including large high-resolution screens, powerful central processors and floating-point hardware – available at moderate prices, has made it practically impossible to foresee the future.

The designers of user-interfaces, window systems, and other graphical support systems have difficulties in keeping up with the latest developments. The problem of defining standards acceptable to all parties concerned is something which has always followed the quickly changing world of computers. As for graphics languages, the most common standard used today is the Graphical Kernel System (GKS). Designed as a possible 3-D extension to GKS, the proposed standard of the Programmer's Hierarchical Interactive Graphics System (PHIGS) has emerged.

1.1 Project Description

For an operator working with any kind of control system, it is totally irrelevant what processor, programming language, graphics support system, or graphics language has been used in the development of that system, as long as the result is an efficient and robust product presenting the system output in a correct and user-friendly way and with powerful tools to deal with every possible event.

For the application programmer responsible of the design and development of a system meeting the demands of the operator, the issues above are on the other hand of the utmost importance. Thorough evaluation of the numerous systems and standards available has to be performed as soon as the system to develop has been specified. Effectiveness, performance, simplicity to work with, and cost are some aspects which have to be considered.

When ABB Network decided to start the development of a new full graphics display and control system, the evaluation process had to begin in a number of different fields. In order to decide which graphics language would be the most suitable for the future system PHIGS had to be evaluated. This project, which was performed at ABB Corporate Research in Lund, is a part of that evaluation.

The main purpose of the project was to evaluate an implementation of the proposed graphics standard PHIGS from the application programmer's point of view. *"Can PHIGS with advantage be used as the graphics language for full graphics control systems? In what way does this standard differ from other graphics languages? What are the benefits of PHIGS? How effective is it in a real full scale system?"* These were some issues the project was concerned with. An application using PHIGS was implemented, showing the possibilities of dealing with hierarchical graphics data as well as displaying some interesting ideas in the field of man-machine communication.

It was also investigated how the PHIGS implementation interacted with the operative system, the application, and the X window system though in a

more overall way.

1.2 Development Conditions

When the project started, PHIGS was quite new to everybody involved. There was some basic knowledge of the overall structure, but none of us had actually used PHIGS for application purposes. Therefore it was difficult to specify in detail the project and the application to be developed. We decided to deal with the problems as they emerged, and by doing so, achieve a substantial amount of experience from PHIGS.

The PHIGS implementation used was the Swedish National Microelectronics Programming (NMP) PHIGS package (version 1.0b6), created as a part of the NMP-Base/OPEN system.

The application was developed under UNIX (HP-UX version 6.0), using a HP 9000/350 high-resolution graphics workstation with a mouse as interaction tool.

C was used as the programming language. Objective-C was also discussed but later repudiated, mainly because of the time effort needed in order to really learn using it efficiently. By the use of dispatch routines, the modularity of the program could still be accomplished.

Although completely hidden by the PHIGS implementation, the X window system (HP-X version 10.4) was used for handling the interaction with the user.

Initially, I was not familiar with any of the tools to be used for this project – UNIX, C, PHIGS and X window. Therefore the first weeks of the project had to be dedicated to the study of books and manuals concerning these items, combined with the development of smaller test programs.

When the project started on my account, my supervisor and I decided on the overall structure of the application and what the final result should be able to perform. Whenever a major problem was encountered, we had another session and decided how to solve it.

As soon as the basic programming was completed (there may still be a few details to change) I created a number of views for demonstration purposes, including an example of the power supply system of a city, to serve as a model for the possibilities of PHIGS.

Finally a small study of the performance was done. The required CPU-time for actions such as redrawing, creating, and updating pictures was investigated, in order to evaluate how the PHIGS implementation would function in a full scale system.

1.3 Outline of the Report

This report describes an application using PHIGS. It points out the advantages as well as the disadvantages of PHIGS from an application programmer's point of view.

In the next chapter a description of the developed PHIGS application and how to use it is given. In Chapter 3 the PHIGS standard will be discussed, together with a description of the actual PHIGS implementation on which this project was based. A brief comparison of PHIGS and GKS and possible future developments are also outlined. The design and implementation of our

application are described in Chapter 4. Problems encountered and alternative approaches are also discussed. Results and experiences from this work conclude the report and are summarized in Chapter 5. The Appendix contains some additional information about performance and discovered bugs.

2. A PHIGS Application

This chapter describes the application developed, using PHIGS as the graphics language for a full graphics control system. The model description concepts as well as the operation of the application are discussed. The actual design and implementation is described in Chapter 4.

2.1 Model Description Concepts

Two basic principles can be used to structure a model – abstraction and modularization. The essence of abstraction is to extract important properties while omitting insignificant details. Different levels of abstraction are defined, allowing the system to be viewed in higher detail at lower abstraction levels, though some information may disappear when changing levels. Modularization means that the information at a certain abstraction level is decomposed into smaller entities.

The main concept is that a graphical description of a model is easier to understand than a textual one. To support abstraction we use information zooming and hierarchical models. Multiple windows are used to further support multiple views of the model. These ideas are described in [Mattsson et al., 1986].

The basic ideas will now be illustrated by an example. Remember that the user at the workstation can pan, zoom, change displayed views, and update them. Naturally these dynamical aspects can not be shown in the figures of this report, which can only show snapshots of the screen.

2.2 An Example

The example below is not intended to be an optimal model of how a control and display system should look like. However, it shows the possibilities of PHIGS in a somewhat realistic manner.

Consider a model of the power supply system of a city. The power stations and power lines are displayed together with an actual map showing the rivers, lakes and main roads of the city. By doing so, the power system is immediately related to its geographical surroundings and a specific power station is not just an abstract combination of letters and numbers in the head of the operator. WorkingArea1 (the lower left window) and the overview areas (the upper left and right windows) display the picture at the lowest level of detail while WorkingArea2 (the lower right window) is zoomed in a little (Figure 2.1). At level zero only the different stations, rivers and main roads are displayed, in order to keep the overview areas from becoming overwhelmed with information. WorkingArea2 further displays the middle-sized roads and the power lines (in straight connections between the stations). The stations have now opened up and begun to show their internal description. The large rectangle in OverviewArea2 outlines the part of the city shown in WorkingArea2.

We zoom further (Figure 2.2). WorkingArea1 now displays even the smallest roads, the power lines have begun to follow the roads and the separate

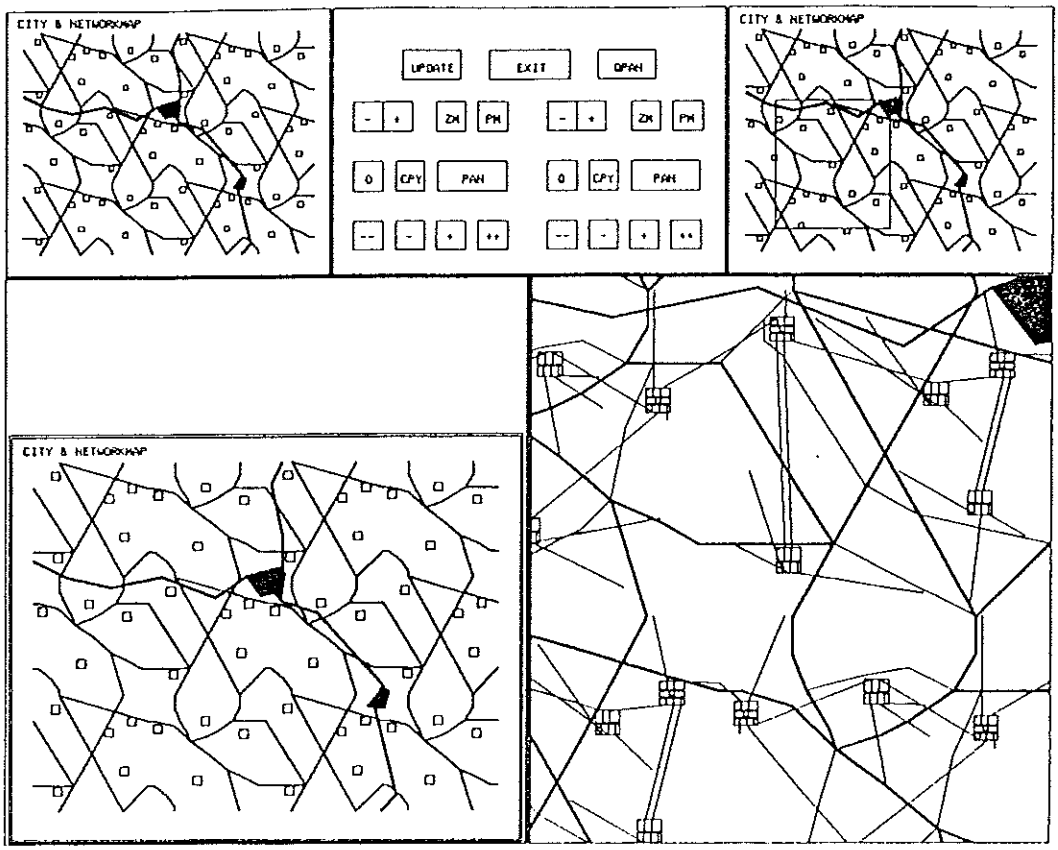

Figure 2.1 City- and networkmap in low levels of detail.

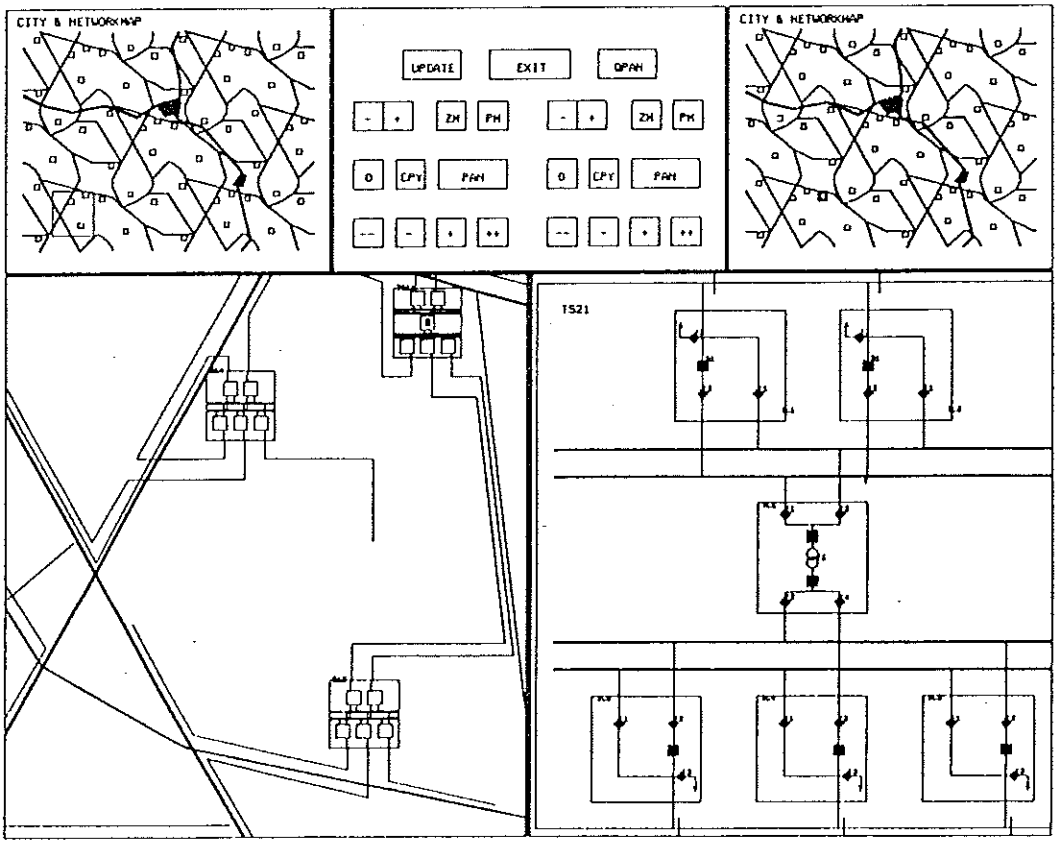

Figure 2.2 City- and networkmap zoomed in further.

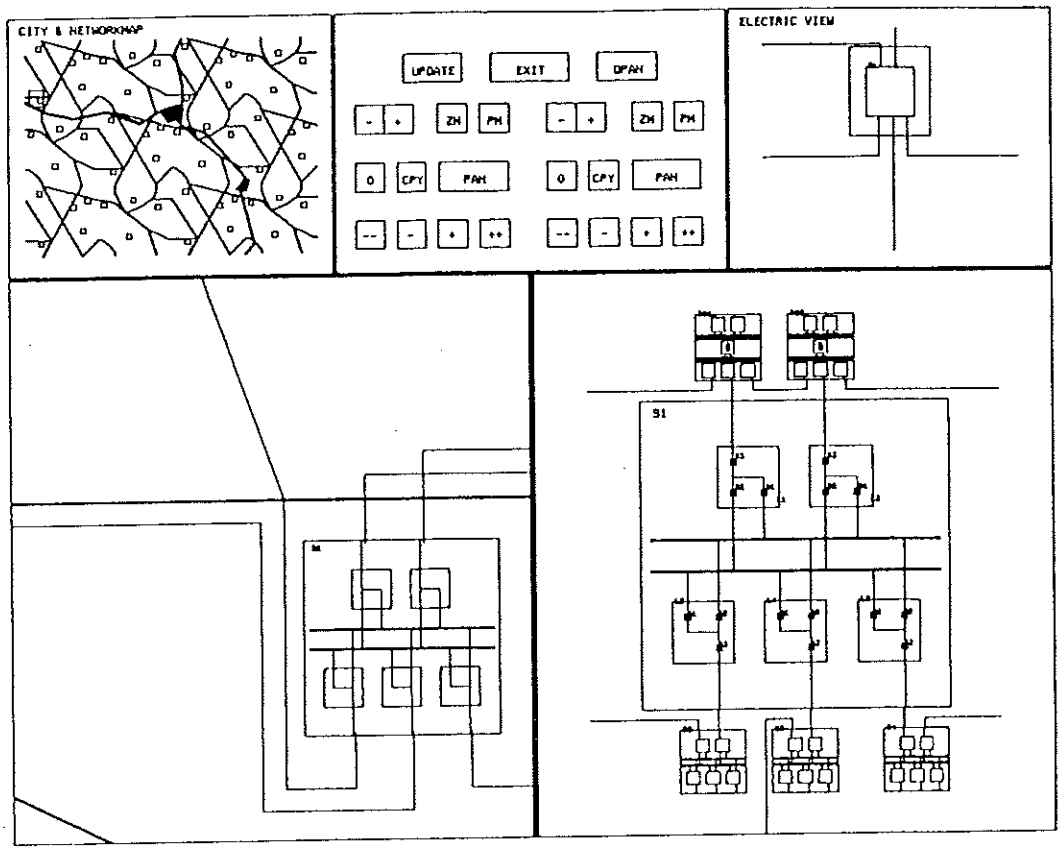

Figure 2.3 Geographical and electric point of view.

linebays inside the stations start to show. WorkingArea2 is zoomed in more and displays a transformer station showing all of its internal structures. After reaching this far in levels of detail, the geographical surroundings begin to loose their interest. The operator now has the possibility to display the power system from an electric point of view instead (it is also possible to display only the power system, the geographical map, or other special views). We choose to change the picture of WorkingArea2 in order to do this (Figure 2.3). An electric point of view means showing the station at a high level of detail and simultaneously displaying all its nearest neighbour stations somewhat smaller but still detailed right beside it, even though they may actually be kilometres apart. This is done, as an alarm in one station is very likely to affect some of the stations connected with it. The operator wants to be able to see this without having to move about in the system.

If the user wants to zoom in further, this is possible (Figure 2.4). Working Area1 now displays a transformerbay in one of the transformer stations and WorkingArea2 displays the transformer symbol in that same transformerbay. WorkingArea2 is now zoomed in as far as possible.

As seen in this example, the PHIGS structures created by this application mainly forms power stations, transformer stations, linebays, isolated linebays, transformerbays, breakers, isolators, and transformers, but also roads, rivers, power lines etc. Every single structure can be selectively updated, though for a few of the special structures, it is not advisable. The main structures appear on the screen in different levels of detail (Figure 2.5). The other used structures are more or less polylines combined to form maps and power lines.

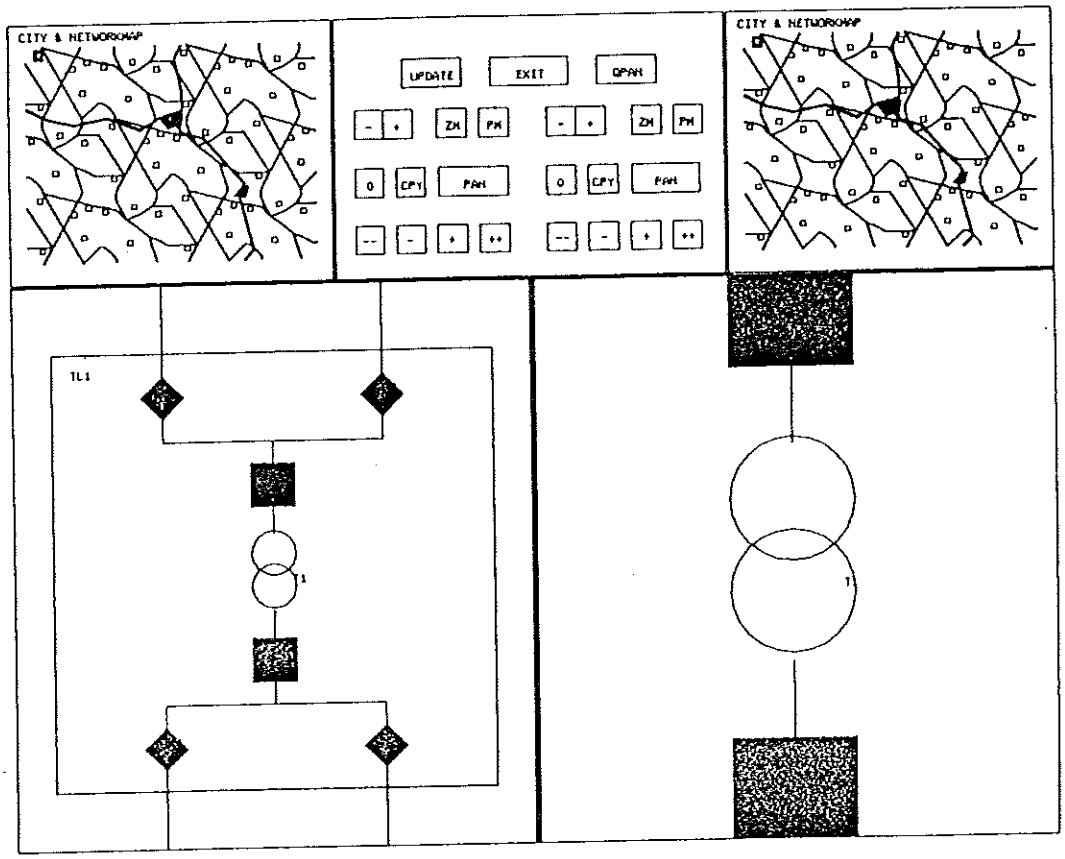

Figure 2.4 City- and networkmap zoomed in far.

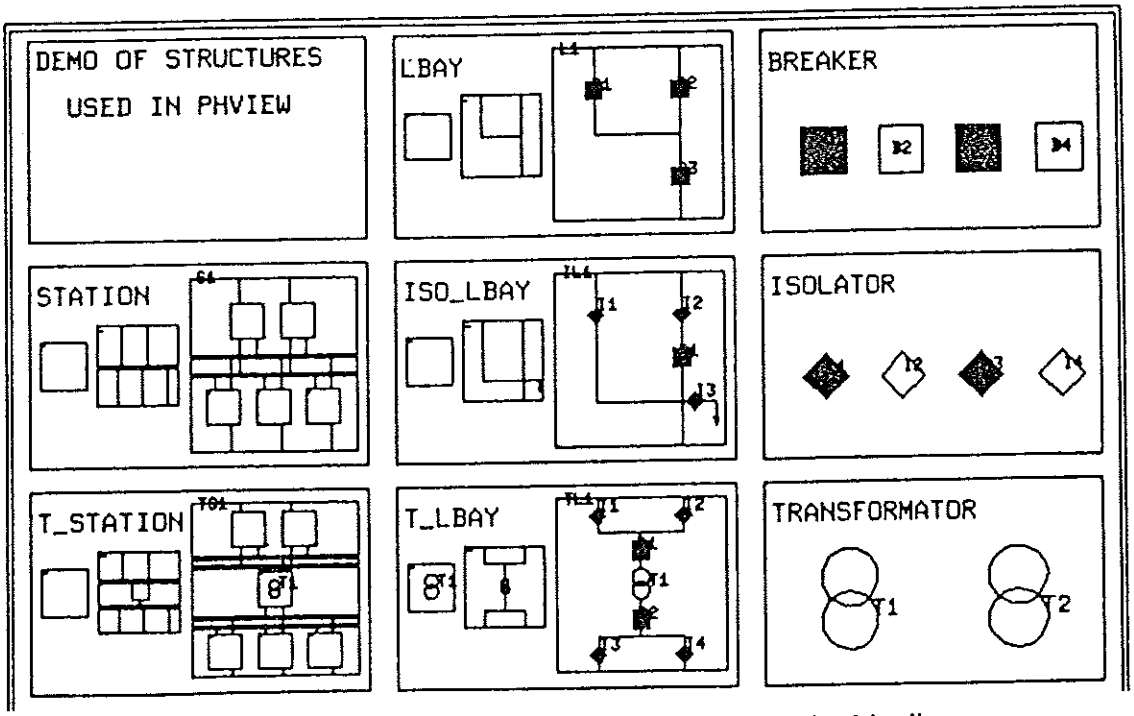

Figure 2.5 The main structures in their different levels of detail.

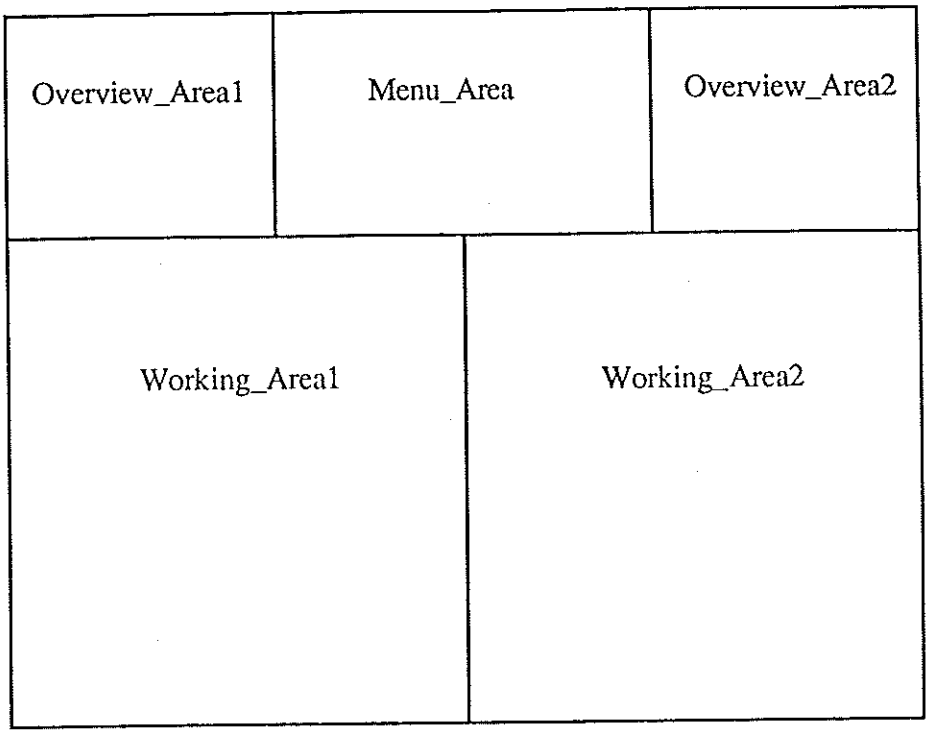

Figure 2.6 Multiple windows give different views simultaneously.

The screen is divided into five different windows (Figure 2.6) – one menu area, two working areas, and two overview areas (one for each working area). These are always visible and are not changable in size (though it would not be difficult to implement it differently).

- MenuArea – Buttons activated by positioning and clicking the mouse, giving the user possibility to pan, quickpan, update, zoom, change pictures etc. (see Figure 2.7). Many of the buttons are duplicated, since they are only connected to one of the two working areas. The menu is always visible.
- WorkingArea1 – Shows the currently displayed picture in the level of detail selected by the user. All pan and zoom actions are performed on the working areas.
- OverviewArea1 – Gives an overview of the picture currently showed in WorkingArea1, displayed in the lowest level of detail. It helps the user to keep track of where he is by outlining a rectangle, showing the current boarders of WorkingArea1. It also indicates when an alarm has been issued by coloring the affected power station red.
- WorkingArea2 – The same as WorkingArea1 but of course the user can here display a totally different picture or the same one in a different level of detail.
- OverviewArea2 – The same as OverviewArea1 but naturally it shows an overview of the picture currently displayed in WorkingArea2.

Zooming controls the amount of information displayed. When zooming in on a power station, it changes from an annotated box to a representation showing its internal structure with increasing detail. The city map and the power lines also change their appearance on the screen when zooming.

Zooming is performed in nineteen predefined steps (between zoomlevels 0 to 19), each step changing the world coordinates so that the area displayed on the working areas is changed 2:1. The effect of this is that a symbol of 1 x 1 centimetres in the lowest level is magnified to 7 x 7 metres, when displayed in the highest level of detail. The fix point is always the centre of the view. The PHIGS implementation was not powerful enough to support true continuous zooming. The application today sets the PHIGS invisibility filters to display the view to zoom in its lowest level of detail while the "continuous" zooming is performed, in order to off-load the processor. The screen is then redrawn ten times for every zoom-level. The same principles apply for continuous panning. It is to be regarded as a test of the efficiency of this implementation and not as a realistic alternative to the discrete zooming and panning.

The use of text has been extremely sparse, mainly because of the difficulty displaying text in a user-friedly and good-looking way when using this PHIGS implementation (vectortext is not supported).

2.3 How the Application is Operated

The user operates the application via a mouse only. He can inspect the model in different ways and simulate alarms, but there are no editing facilities or ways of creating new structures on-line.

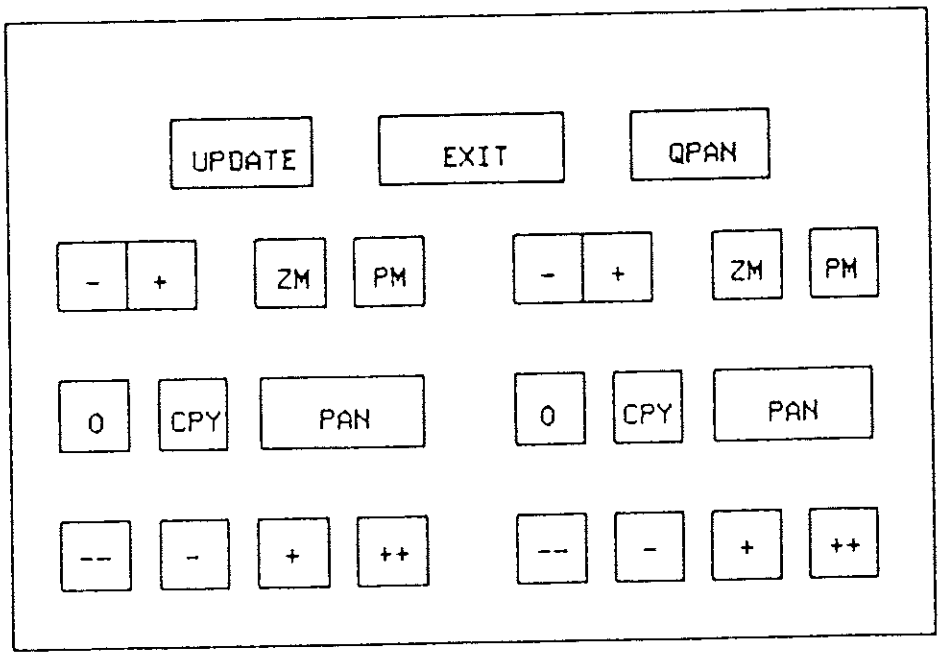

Figure 2.7 The menu.

Commands are chosen from the always visible menu (Figure 2.7). The desired command is selected by pointing at the corresponding button and clicking any of the mouse buttons. The selected button is highlighted and the action is performed. A few commands require additional input from the mouse. The eleven low-left buttons performs actions on Working- and OverviewArea1, while the low-right buttons affect Working- and OverviewArea2.

Commands

The following operations are supported:

UPDATE	Simulates an alarm from one or more stations. The station is displayed in red, and red frames are used to lead the way to the actual breaker or isolator issuing the alarm when zooming in. A second activation of UPDATE is used as an acknowledgement that the alarm has been observed and the alarm is then turned off. UPDATE affects all windows.
EXIT	Stops the application and returns to the operating system. When starting up the application it is adjusted to a 1024 x 768 pixels screen. If selecting EXIT before any other actions are taken, the application will adjust to a 1280 x 1024 pixels screen.
QPAN	Used to pan quickly in the system. The command requires a second pick in one of the overview areas where the new centre coordinates of the working area is selected. QPAN affects either Working- and OverviewArea1 or Working- and OverviewArea2.
–PIC, +PIC	Changes the picture to be displayed in steps of one. A list of seven demo pictures, labeled one to seven, is currently implemented in a circular manner. This means that selecting + when picture seven is displayed will put picture one on the screen.
ZM	Changes the zoom mode to be used. Discrete and continuous zooming are available.
PM	Changes the pan mode to be used. Discrete and continuous panning are available.
0	Used to immediately return the working area to the lowest level of detail and centralizing the view, e.g. reset the working area.
CPY	Changes the working and overview area to immediately display what is currently displayed on the other working and overview area, with the same level of detail and the same coordinate system.
PAN	Used to pan more accurate in the system than when using QPAN. The command requires a second pick in the working area, where the new centre coordinates are selected.
– –, –ZOOM	Zooms out two or one step, e.g. changes the area displayed in the working area 1:4 or 1:2. It of course also affects the level of detail.
+, ++ZOOM	Zooms in one or two steps, e.g. changes the area displayed in the working area 2:1 or 4:1. It of course also affects the level of detail.

3. About PHIGS

In this chapter we will discuss the main features of the proposed standard and its intentions [Brown, 1985]. A brief comparison of PHIGS and GKS will be given as well as a description of the proposed extensions of PHIGS, e.g. PHIGS+. Finally, the PHIGS implementation used in this project will be studied in further detail.

3.1 Introduction

PHIGS, Programmer's Hierarchical Interactive Graphics System, is a specification of the interface between an application program and its graphics support system, which controls the definition, modification, and display of hierarchical graphics data [SIS, 1985].

PHIGS is designed to let the application programmer concentrate on solving the particular application's problem, utilizing the standards powerful functionality without having to worry about graphics system development and support. This is done by providing advanced graphics tools combined with high performance workstations.

PHIGS is not intended to satisfy the requirements of all sectors of the computer graphics community. Though it will probably serve as an important complement and extension to other standards in fields such as:

- computer aided engineering (CAE)
- process control systems
- command and control systems

3.2 Description of PHIGS

Representing years of effort by ANSI technical committee X3H3, PHIGS is about to change the way application programmers think about graphics. It is the first major step from Core-based graphics systems, including the Graphics Kernel System, which generally require the application programmer to perform hierarchical modelling in his own application program.

Main Features

From the very beginning, PHIGS was developed to be a 3-D, hierarchical, interactive system. The structured organization of graphics in a centralized database allows objects to be defined in numerous ways. They are defined by a sequence of graphical elements including:

- output primitives
- attributes (colors, line styles etc.)
- modelling transformations
- view selections
- pick identifiers

- labels
- name sets (for visibility, detectability and highlighting)
- structure invocations
- application data (similar to comments in a high level language)
- ESCAPE functions

These elements can be grouped into entities, called structures, which may be related to one another geometrically, hierarchically, or by properties determined by the application. This means that PHIGS encourages programmers to define and organize graphics data in the manner most convenient for a particular application. It also separates the definition of graphics data and structural relationships from the actions required to display them.

Designed to support interactive application programs, PHIGS permits rapid modification of a graphics database. It performs general 3-D modeling transformations, 3-D viewing, and 3-D workstation transformation. Both parallel and perspective projection of oblique and non-oblique views are supported. PHIGS can also off-load host processing by taking advantage of local graphics databases and modelling capabilities. Functionally, PHIGS integrates the host processor, a centralized database and high performance workstations.

PHIGS structures (the grouped graphics data) can be created in two ways – explicitly and implicitly. The open structure function is used to create a structure explicitly and reopen one for editing. Implicitly created structures occur whenever a function references a structure's name. This feature is especially helpful in allowing programmers to implement top-down design of graphics applications, since it creates a logical tree of empty structures to be filled in later.

There are two ways to display a created structure on a workstation; one is to make it a root structure at that particular station using the post root and unpost root functions, another is to display a structure implicitly if it is a part of the hierarchy below a posted root structure. PHIGS is flexible enough that a given structure may be both posted to a workstation and displayed implicitly as part of the hierarchy.

A structure inherits the attributes (color, line style, pattern etc.) of its ancestors unless other values are defined within the current structure. The root structure inherits its attributes from the PHIGS description table, which contains system default values.

Six logical input device classes are supported by PHIGS. Each of these classes is distinguished by the type of data it is capable of returning to the application. The different classes are:

- locator – returns a position in world coordinates
- stroke – returns a sequence of positions in world coordinates
- valuator – returns a real number
- choice – returns a non-negative integer
- pick – returns a pick path to the structure
- string – returns a character string

Devices in each of these classes can be accessed in any of three modes. In request mode an application program prompts for input and then waits until the end user has signaled completion of the input operation. In sample mode an application program asks for the current value of a device, independent of any end user action. In event mode input is placed in a queue by the end user

at his convenience. Items in the queue are then retrieved at the convenience of the application program.

More than one input device may be connected to a workstation. In fact, it is feasible to have several input devices, all belonging to the same logical input class, connected to one particular workstation. The device number distinguishes different logical input devices of the same class on the same workstation.

PHIGS allow dynamic modification of graphical images. Specifically, an application can:

- edit structure contents
- perform database operations on a structure
- modify attribute values in the workstation state lists

Because bundle attribute values are stored independently of structures, the application can make dynamic changes to the bundle table without modifying the structure.

The application controls when changes of the workstation tables and database are reflected on the display surface. The modification mode governs how visual effects occur while the deferral mode governs when they are displayed. The modification modes supported by PHIGS are:

- no immediate visual effects
- update without regeneration
- perform regeneration if necessary
- use quick update methods

and the supported deferral modes are:

- as soon as possible
- before the next global interaction
- before the next local interaction
- at some time
- when the application requests it

There is considerable flexibility in viewing objects. For instance, an individual object may be viewed differently on other workstations without modifying the object in the central database. Likewise, a single object can be displayed from three plan views in three different viewports on a single graphics device simply by invoking the object three times, each time with a different view index. This feature eliminates the storage of redundant data. And since only one copy of the data is maintained, editing performed on an object from one of the plan views automatically ensures consistent updating of the object in the other two views.

Name sets are another new concept. They have two basic purposes:

- They permit the user to control the visibility, highlighting and detectability of groups of structural elements without requiring a structural edit
- They provide this control without requiring knowledge of where the elements are in the graphics database

The name sets are used during traversal of the database in conjunction with workstation-dependent filters. These filters determine whether certain structures or structure elements are visible, highlighted, and detectable by pick input devices. Each filter contains an inclusion and exclusion set.

To customize the graphics database further, PHIGS provides a structural element that enables the application program to embed non-executable data in the database. This data is analogous to comments in a high level language program and is not displayed but can be accessed by the graphics editor.

If a workstation provides more capabilities than defined by PHIGS, they may be accessed via the General Drawing Primitive or ESCAPE functions.

Two of PHIGS most powerful structure handling functionalities are its archival and retrieval capabilities. It allows the user to archive only those portions of the graphics database that correspond to the particular part of a graphics object he wishes to save and later retrieve, in random access fashion. A completely portable computer-independent format is also included for the archive file.

PHIGS is designed to be a system with few error states. This helps reduce the I/O traffic between the nodes of a distributed system. Further, the production use of applications, which have well-defined inputs and outputs, can be streamlined by minimizing error logging overhead and utilizing run-time error recovery techniques.

A PHIGS Architecture

For an efficient PHIGS implementation the PHIGS architecture needs to be modular to support a distributed environment without the overhead, that would limit its performance, response time, and resource utilization (Figure 3.1). The user level routines accept the language binding associated with a PHIGS implementation and translate them into language independent functions understood by the PHIGS kernel. The PHIGS manager is a thin shell that receives application calls and invokes the appropriate PHIGS functions. The database and state lists are maintained by the PHIGS kernel. It interprets user requests and produces workstation graphics commands. In a truly distributed environment, the code to maintain the database and state lists can be put on an intelligent workstation. Thus, a central PHIGS kernel would not be needed. The workstation manager makes sure that the correct workstation receives commands from PHIGS. Finally, the workstation controller processes graphics commands from PHIGS, implements the transformation pipeline, and outputs device dependent code for each individual workstation. It can run as a separate task from the PHIGS kernel.

3.3 A Comparison of PHIGS and GKS

GKS has recently established itself as an international standard for designing two dimensional graphics applications. It has simultaneously established a base of concepts, which greatly influences the design of subsequent standards, e.g PHIGS. This encourages the important goal of programmer portability, even in the absence of strict program portability.

PHIGS has attempted to maintain compatibility with the international GKS standard [Hopgood et al., 1983]. In some areas, however, PHIGS has deviated in order to serve the needs of its stated constituency. In particular, GKS and its extensions do not satisfy the requirements of application programs with the following characteristics:

- modification of graphical data
- definition, display, and manipulation of geometrically related objects

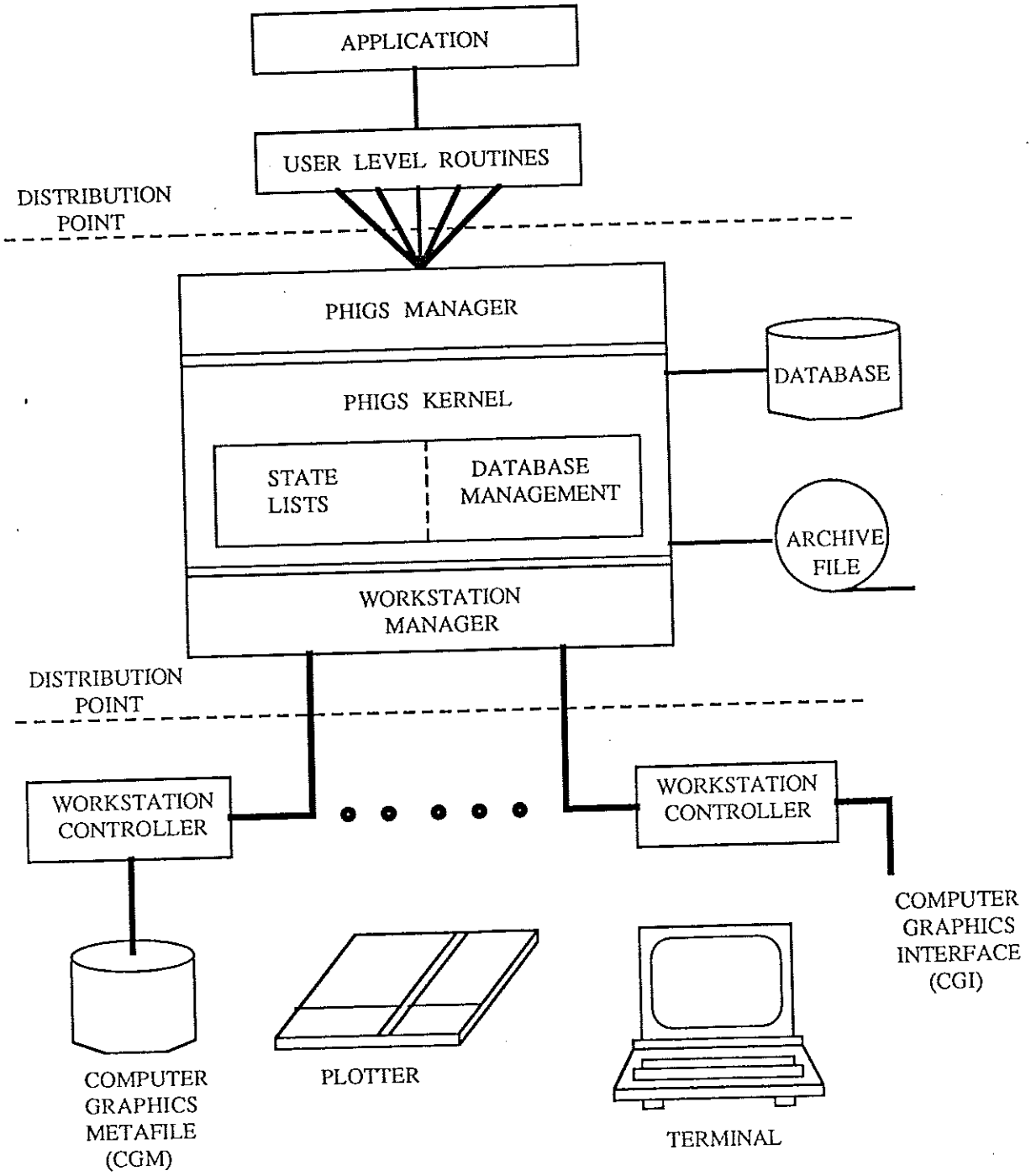

Figure 3.1 Enhanced PHIGS architecture.

- rapid dynamic articulation of graphical entities

The PHIGS output primitives have been adopted from GKS. They include polylines, polymarkers, text, fill areas, and cell arrays. As in GKS, the General Drawing Primitive may be used to implement various non-standard primitives. PHIGS has one additional output primitive "fill area set" which defines a complex polygonal area. PHIGS has the same logical input device classes as GKS. However, the pick device has been augmented to provide additional information about the location of the picked primitive in the hierarchical

database. In addition, name sets allow applications to selectively define groups of items from which an end user can pick. PHIGS supports the input, output, and input/output workstation models of GKS.

The new, more powerful concepts provided by PHIGS include structures, structure body editing, traversal-time binding, and the name set mechanism for detectability, highlighting, and visibility control.

PHIGS groups graphics information into structures which reside in a centralized, hierarchical graphics database. An important capability of PHIGS, lacking in GKS, is the ability to edit structures by inserting and deleting structure elements. GKS requires the application program to supply hierarchical modelling capabilities, limits dynamic operations to groups of primitives stored after the viewing operation, and groups the primitives into entities called segments, which reside in a decentralized mono-level database. In GKS, segments cannot invoke other segments and cannot be extended or edited.

In PHIGS, creation and modification of graphics data are independent of how and when the data is displayed. In GKS, the data has to be sent to a particular workstation or the workstation independent segment storage (WISS) in order to be defined.

PHIGS assigns attributes to primitives during the actual database traversal. Traversal-time binding permits both inheritance and dynamic modification of primitive attributes through structure editing. In GKS, primitives are assigned attributes at definition time.

In PHIGS, dynamic capabilities – visibility, highlighting, detectability, transformations – can be assigned to output primitives, which are stored as structure elements. In GKS, these capabilities can only be assigned to segments. This refinement in the control of the representation of primitives is known as granularity.

Finally, PHIGS defines two conforming implementation levels; one contain extensions to capabilities existing in the other. GKS has twelve levels. Therefore PHIGS avoids the confusion and lack of portability introduced by the existence of many conformance levels.

3.4 Future Developments

Even though PHIGS has not yet been formally approved as a standard, a working draft of proposed extensions called PHIGS+ to cover the area of lighting and shading models and advanced rendering primitives has been issued [PHIGS+, 1987]. The PHIGS+ specification addresses the capabilities of the new generations of ultrahigh-performance workstations. These workstations will have the performance necessary to support such tasks as hierarchy traversal, lighting, and shading of 3-D geometric data in real-time.

In order to provide such rendering effects as Gouraud and Phong shading, PHIGS+ extends PHIGS by adding geometric information like vertex normals to primitives, additional attributes, e.g. direct colour, surface primitives, and light sources. New primitives like quadrilateral meshes, triangle strips, and non-uniform rational B-spline surfaces are also proposed. Of course, PHIGS+ also supports all standard PHIGS functions in addition to those mentioned.

PHIGS+ implementations are already available on the open market. A graphics system from Raster Technologies provides both PHIGS and PHIGS+ implemented in the hardware, though at a considerable cost.

3.5 NMP–PHIGS

The PHIGS implementation used in this project was the Swedish National Microelectronics Program PHIGS (version 1.0b6) developed as a part of a graphical layout editor (YALE) for VLSI-CAD purposes [Base/OPEN, 1988]. This version is a prerelease and discovered bugs are discussed in Appendix B.

Implementation and Interfaces

The NMP-PHIGS package is a subset of the PHIGS standard proposal SIS TR-306. It is implemented in objective-C (25 000 lines) and has a C procedural interface. As there are no standard C-language binding, the language binding is modeled on the C binding of GKS. It has drivers to the X window system and to CGI (though restricted to output). By exploiting the graphics capability of X, it is portable between many different workstations.

The package consists of three major parts well separated from eachother (Figure 3.2). Those are the graphical model called DataStorage, the workstation dependent parts called Workstations, and the PHIGS archive functionality called Archives.

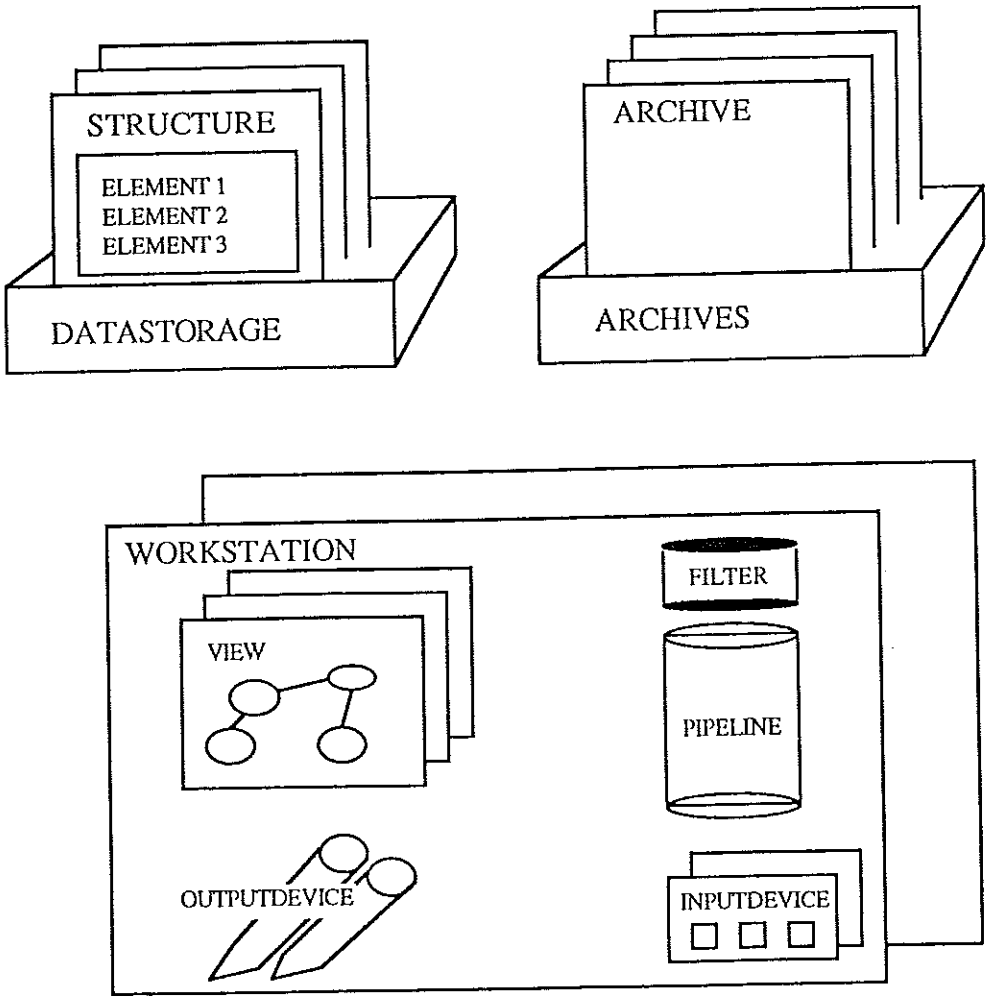

Figure 3.2 NMP-PHIGS object overview.

DataStorage is implemented as a "Set" of PHIGS structures. Structure is implemented as an "OrdCltn" (a predefined class in objective-C) of Structure elements.

Workstations is implemented as an "OrdCltn" of objects of the class Workstation. Workstation is a "SortCltn" (another predefined class in objective-C) of "Views" where the sort criterion shall be view priority. The Workstation owns a pipeline object that does all transformations. It owns output devices needed to perform all drawing primitives and if it has input capability also the input devices available. A "View" is a "OrdCltn" of Structures posted to that view, a window and a viewport definition.

Archives is implemented as an "OrdCltn" of Archive objects. An Archive object is an object responsible for an archive file. It is implemented as a "Set" of StructureFile objects. An archive file is a file created with the UNIX command "ar". Finally, a StructureFile object is an object that has methods to read or write from or to the archive file. This is the object the Structure elements use when archiving themselves.

The NMP package uses the communicating and operative system, CaOS, interface. The CaOS interface has been developed to guarantee portability between modern UNIX workstations such as Apollo, Hewlett Packard, GPX and SUN. CaOS is a subset of the X/OPEN standard, which is a joint initiative by several of the world's major information system suppliers.

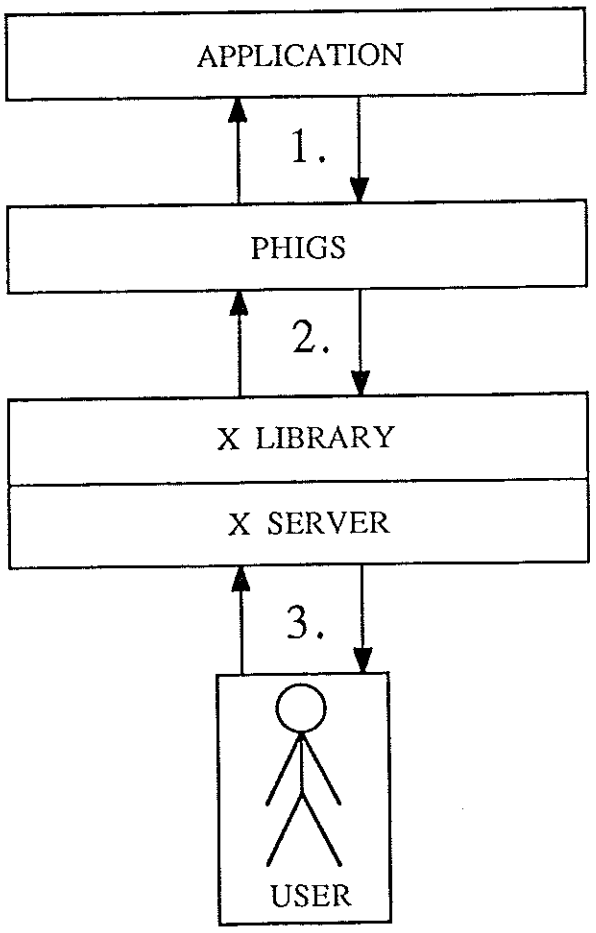

Figure 3.3 Overview of interfaces.

An overview of the different interfaces are given (Figure 3.3). The type of data passing these interfaces are:

1. ↓ Graphical output
   ↑ Various data inquired by the application
2. ↓ Graphical output
   ↑ Graphical input
3. ↓ Graphical output
   ↑ Input from keyboard and mouse

Every PHIGS function used in the application invokes a procedure in the PHIGS library through the procedural interface. The procedures in the library create and manage the PHIGS database, make the transformations, and perform actions on the PHIGS structures. All this is done in normal objective-C manner, without using any special hardware or fast assembler routines. For every PHIGS input and output function there exist special interface files to the X library, which is the low-level interface to the X-server protocol. Therefore, the PHIGS implementation can be regarded as a shell for the X window system, where all the actual graphics display actions are performed, making every application a specific client from the X point of view. The X interface is not to be used directly by the application since the appropriate functionality is provided by PHIGS. The use of special graphics hardware and graphics functions in the operative system is depending on which X system is running on your machine, as the PHIGS library does not use any of these specialities.

The error handling function of PHIGS also serves as an interface between PHIGS and the application program. For each PHIGS function, a finite number of error situations exists. Any of these situations will cause the error handling functions of PHIGS to be called. The standard PHIGS error logger prints a message to the error file and then returns control to the calling routine.

NMP–PHIGS versus the PHIGS Standard Proposal

NMP-PHIGS is a subset of PHIGS. The functionality has been greatly reduced. The number of functions in the proposed standard are close to 300, while there are only about 100 in NMP-PHIGS. One major reason for this reduction is that NMP-PHIGS is only implemented to 2-D graphics and does not need the double functionality of PHIGS in order to cover both 2-D and 3-D graphics. But there are also several explicit reductions in the functionality, especially in the fields of attribute and input functions as well as for inquire functions.

The input devices implemented are:

- locator request and sample mode
- pick request and sample mode
- string request mode

For a layout editor the request and sample modes are the most useful ones, but when wanting to develop an application with interaction and input in real time, the event mode is a very useful tool.

Only bundled attributes are available, except for a few text attributes. As they work quite well this is no major limitation. However, the number

of available text attributes are inadequate. It is impossible to display text on the screen in a desirable manner, especially when combined with rotations and scaling. Another drawback is that rotations are limited to right angles or multiples of right angles.

Due to the lack of 3-D, the viewing pipeline is simplified. The world coordinate system maps itself on a virtual screen described in normalized device coordinates (NDC) and is then transformed to device coordinates. Though called NDC, they must not be normalized but is definable. Default will be normalized with the aspect ratio of the device screen maintained.

The means for PHIGS to perform update in a selective manner appear to be good also for the NMP package, but do not work as they are supposed to.

The NMP-PHIGS error handler is implemented to perform the functionality of the standard proposal with the additional function "check error", that gives the application possibility to find out the error condition of a function call, immediately after the call.

The view handling differs. Instead of having the "set view index" as a structure element changing at traversal time, NMP-PHIGS posts the structures to views.

A "shell" is laid upon the execute structure elements and the transformation elements, and those are not available in the interface. Instead transformations are restricted to instantiations and a new structure element is introduced. This is an execute structure element with an instantiation transformation matrix called "instance structure", which helps to improve traversal performance.

Finally, only retained structures are implemented. This means that the possibility of displaying structures on the screen, without storing them in the PHIGS database, is lost.

Apart from these modifications and limitations, the NMP-PHIGS follows the standard PHIGS proposal and for many applications the reduced functionality will probably prove quite sufficient, especially for CAD and similar applications. Though for applications requiring high performance and real-time updating, it will not serve the purposes and intentions of the proposed PHIGS standard.

4. Design and Implementation

In Chapter 2 the conceptual ideas of the developed PHIGS application were outlined. This chapter deals with the actual design and implementation of that application with emphasis on the three most interesting issues; how the PHIGS structure hierarchy was used, the dispatch routines, and the combination of PHIGS invisibility filters and information zooming. Problems encountered during the development will also be described as well as some alternative methods of design.

4.1 Program Overview

The main purpose of this program was to investigate how PHIGS could be used in a somewhat realistic application. The efficiency, developed tools and structure of the application itself was not the main issue. The design is to be regarded as an example of the possibilities when using PHIGS.

The code for the application consists of 5000 lines of C and the executable code is 450 kbytes. The software is quite modular although not object-oriented in its true sense. The entire application is run as one task, however, other processes are running simultaneously under the control of X and PHIGS.

It is possible to divide the application into a number of modules (Figure 4.1):

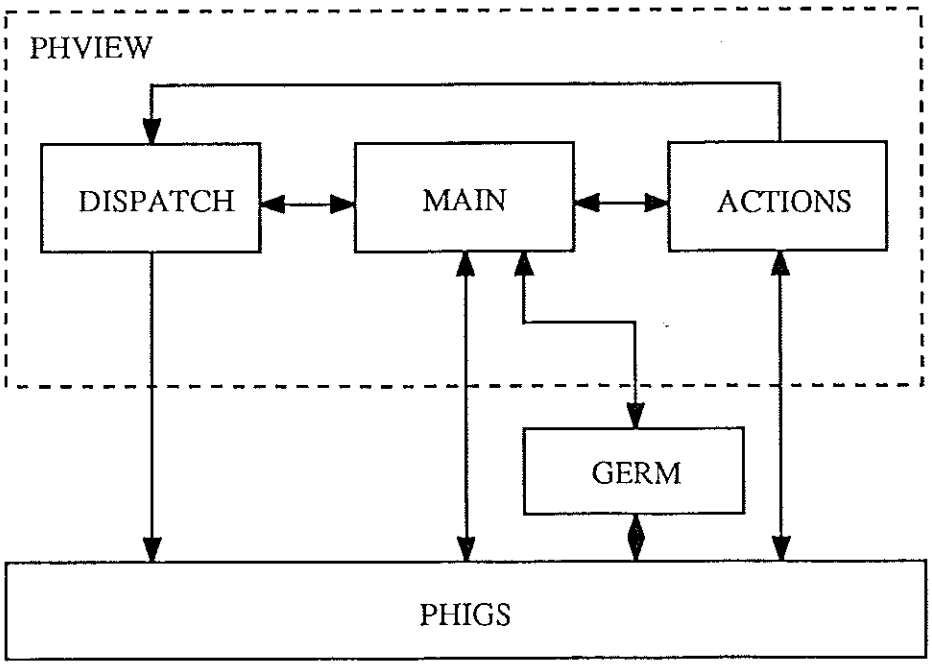

Figure 4.1 The program structure.

- Dispatch – Special routines for creating, displaying and updating the PHIGS structures in a powerful and modular manner
- Main – Initiates and coordinates the other modules.
- Actions – All routines for detecting and performing commands from the end user, either affecting the views (pan, zoom etc.) or the PHIGS structures (update)
- Germ – Routines for error detection and error handling included in the NMP-Base/OPEN package (not an application module)

The modularity makes it easy to extend the program and develop new PHIGS structures if needed. Since all objects displayed are created explicitly in code, they should not be too complicated. A graphical editor has to be developed in order to create more complex objects in an easy and less time-consuming way.

An overview of how the program works is given (Figure 4.2). The PHIGS database is first created and posted to the appropriate views. Thereafter the application waits for input from the user and performs the actions selected. If no input is received the program is idle.

A local coordinate system is assigned to every object when it is created such that the lower left corner has the coordinates (-500, -500) and the upper right corner has the coordinates (500, 500). This hierarchy of coordinate systems makes it easy to scale and to display the substructures properly.

4.2 The PHIGS Structure Hierarchy

As discussed in Chapter 3, PHIGS groups the graphics information into structures which reside in a centralized, hierarchical graphics database. One of the main issues when working with PHIGS is to determine what this hierarchy should look like, in order to achieve a logic structure of the graphics data. Quick access to specific structures, suitability for using filters, and simple inheritance rules for the PHIGS attributes are other aspects to be considered when deciding on the hierarchy.

The main concern of this project was to define a hierarchy which is easy to combine with information zooming and also easy to extend by adding new PHIGS structures. The number of structures was fairly large (about 3000), though the individual structures were rather small (10 to 100 structure elements). The reason for using so many structures was that every single part of the created power supply system should be possible to change and update individually without affecting any other structure. This implies that every power station, linebay, breaker etc. has to be implemented as a specific structure with its own individual PHIGS attributes assigned. As almost every structure has its own set of attributes as an explicit structure element, the concept of letting substructures inherit attributes from their rootstructure was not very useful. The large number of structures was due to the fact that each object should be possible to display using different levels of detail. We chose to use one PHIGS structure for each level and implement three levels of detail for every station, linebay etc. All this combined, makes an ordinary power station when created at all levels of detail consist of 33 separate PHIGS structures and a transformer station of 48 structures.

Let us take a closer look at the hierarchy for a normal power station. The complete hierarchy is formed by three station structures (three levels of detail),

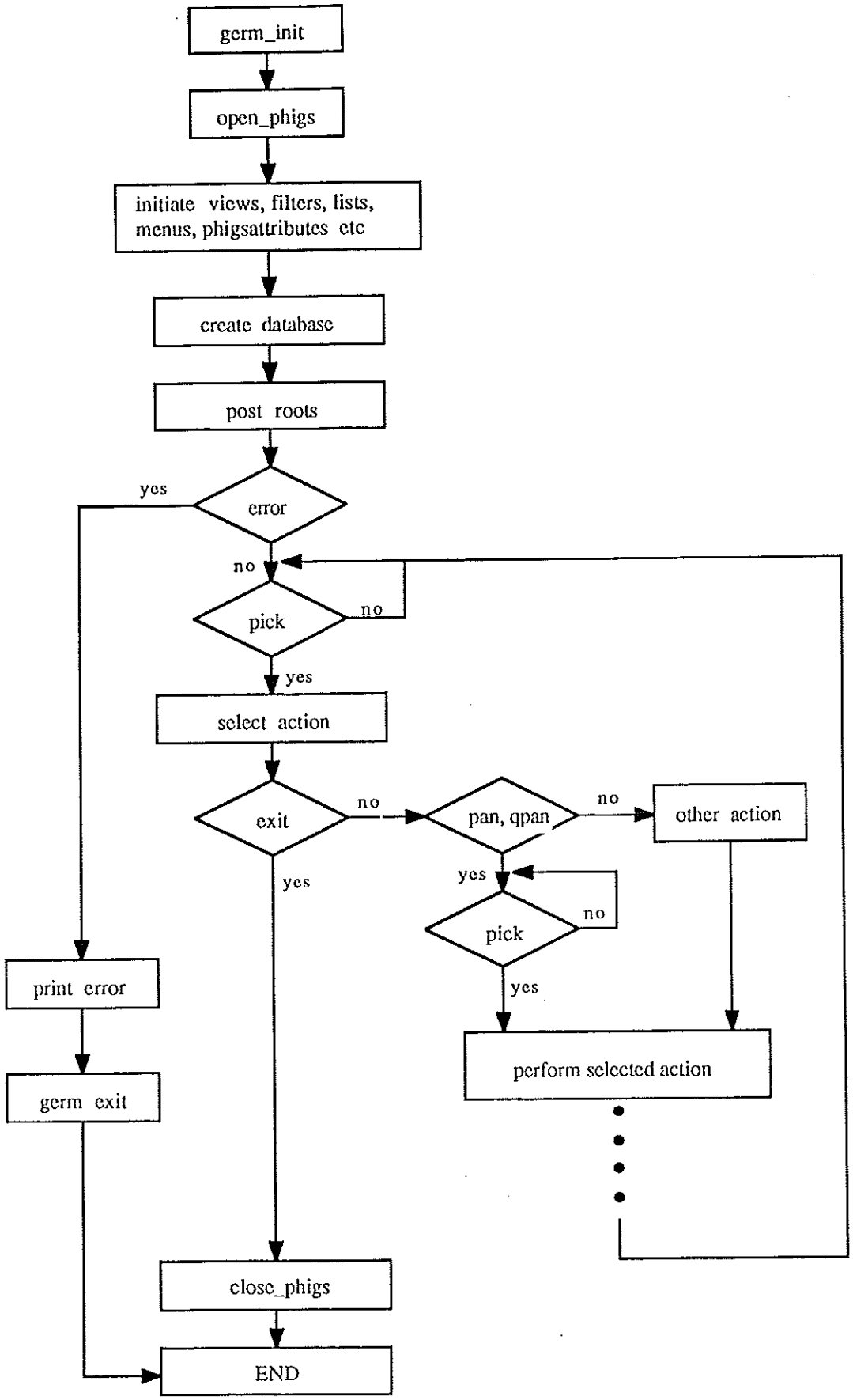

Figure 4.2 Simplified flowchart of the program.

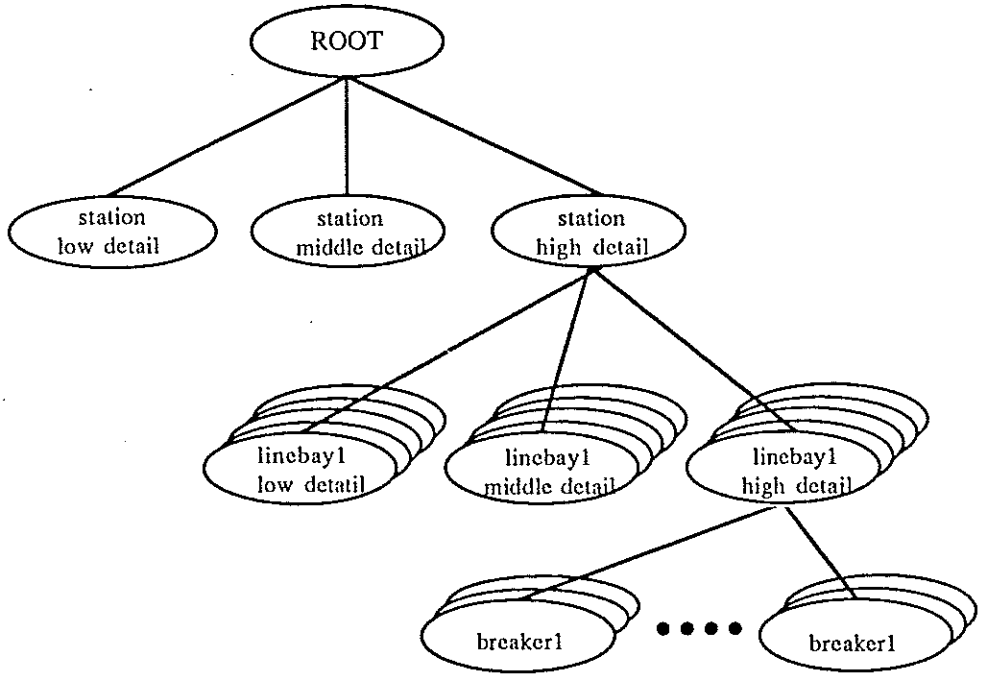

Figure 4.3 The PHIGS hierarchy for one power station.

fifteen linebay structures (five different linebays in three levels of detail), and fifteen breaker structures (three breakers for every linebay in its highest level of detail) – a total of 33 structures (Figure 4.3). This tree structure describes a power station in an understandable and logic way. Filters are also assigned in a natural way in order to select what part of the tree is to be visible on the screen. The main disadvantage is the number of created structures. The more structures, the more CPU-time is required to traverse and perform searches on the tree. A compromise between logical and easy understandable hierarchies on one hand and the performance and efficiency demands on the other, has to be done. In this application we have chosen to create hierarchies that are quite simple to understand and logical to work with. When examining Figures 4.3, 4.4 and 4.5 it is useful to bear in mind Figures 2.1 – 2.5 in order to relate the PHIGS hierarchies to what is actually displayed on the screen.

The attributes of the PHIGS structures are always stored as the first structure elements which makes it easy for the application to locate them when an edit on the stuctures is performed.

The hierarchy of a transformer station is created in the same principle manner though somewhat more complicated (Figure 4.4).

The described hierarchies are not difficult to extend. By creating different levels of detail for the breaker structures as well and develop and connect new substructures to them – the same way as for stations and linebays – all the used concepts of the application would still function. In this way a totally integrated system can be developed, from the power supply system of an entire city down to the smallest electrical component of that same system.

The structures for the maps, power lines, menues etc. use a very simple hierarchy. Since these structures are not dynamically edited by the application and only change appearance on the screen when the invisibility filters are changed, they can be created as large, mono-level structures with filters

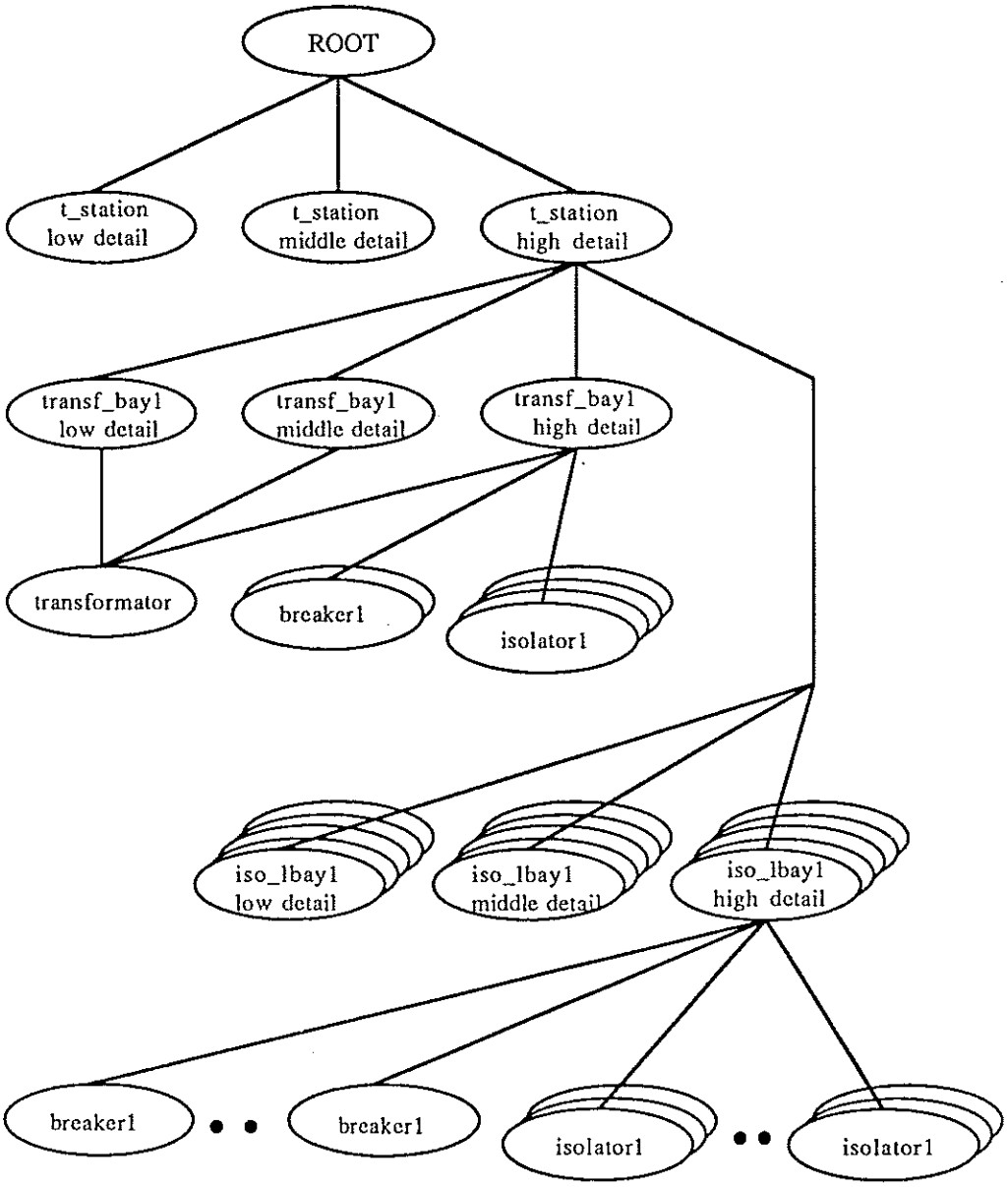

Figure 4.4 The PHIGS hierachy for one transformer station.

assigned to specific structure elements, not to whole structures.

When all structure trees have been created, they are connected to appropriate root structures to form the different pictures. The root structures are then posted to a specific view and displayed or stored in the PHIGS database for later use. A slightly simplified hierarchy of what is actually posted to the working areas when both the citymap and power system are displayed is shown in Figure 4.5.

The structures for the frames in the working areas and for the rectangles outlining the operators position in the overview areas (see Chapter 2) are defined as root structures of their own and are not connected to any of the other PHIGS structure trees. These special structures are always to be present on the screen whatever picture is currently displayed on the views (they can be regarded as background objects). To have several root structures posted

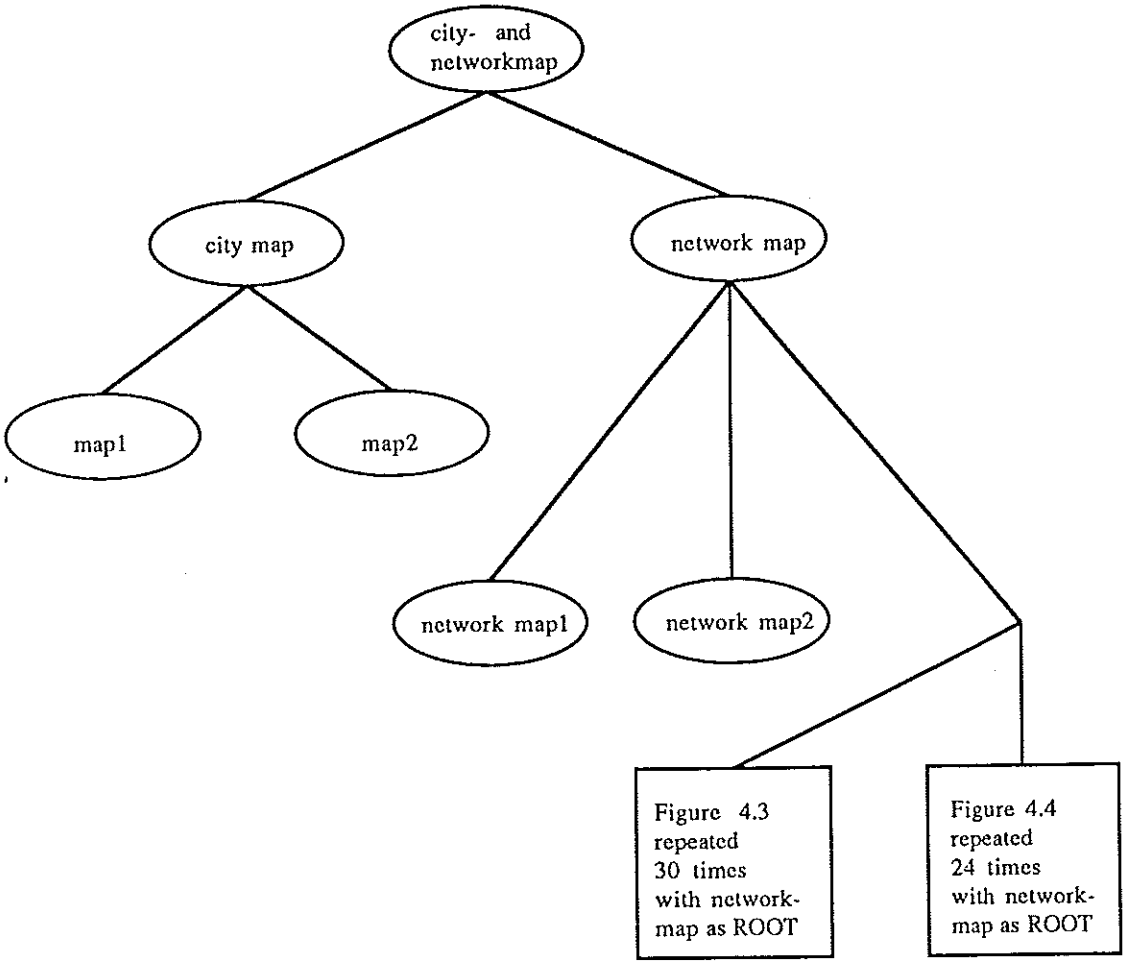

Figure 4.5 The PHIGS hierarchy for the city- and networkmap.

to the same view at the same time is no problem and it makes it possible to change only a selected part of the displayed objects by unposting one of the displayed root structures.

Parallel structure

Although the PHIGS structures form a hierarchical tree, some kind of parallel structure is needed by the application to find the appropriate PHIGS structure, when an action is to be performed. This parallel structure can be implemented in numerous ways. We chose the simplest possible solution and used an array in which we stored the necessary data.

When a PHIGS stucture is created a unique integer is assigned as an identifier to that specific structure and stored in the database. At the same time the identifier is stored by the application together with some other useful data in the parallel structure. Every element of the array consists of:

- ID – a string of maximum twenty characters assigning the PHIGS structures to an ASCII name
- length – the number of characters in the ID variable, used to improve the performance of the search routines on the array

- phindex – an array of three integers where the actual PHIGS index used by the database are stored
- classname – an enumeration variable used by the dispatch routines (see Section 4.3)

The ID provides a PHIGS structure with a specific name which is easier for the user to understand and for the application programmer to work with than an integer. We used a certain combination of letters and figures to form the names. These names are also the ones seen on the screen when the objects are displayed. The reason for storing three PHIGS indexes for every name is because of the three levels of detail used for the stations, linebays etc. Since every such level is created as a specific structure it is natural to give them the same name yet still being able to seperate them – `phindex[0]` is always the reference to the PHIGS structure in the lowest level of detail, `phindex[1]` to the middle level structure, and `phindex[2]` to the high level of detail structure. For structures with only one level of detail (breaker, isolator, transformator) the reference is always stored in `phindex[0]`. The classname is used as a reference by the dispatch routines to determine what type of PHIGS structure the phindex in this element is referring to – station, breaker, linebay etc.

All this is best explained by a few examples (Figure 4.6).

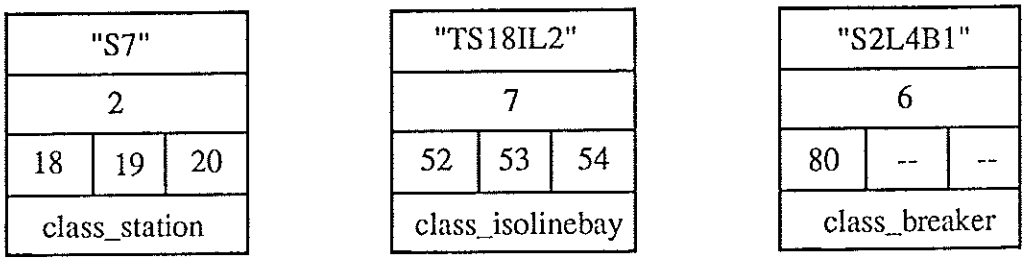

Figure 4.6 Examples of the parallel structure.

The first example informs the programmer that the object – station number seven – is stored in the PHIGS database as structures number 18, 19, 20 with increasing level of detail, the second that the isolated linebay number two in the transformer station number eighteen is stored in the database as structures 52, 53, 54 with increasing level of detail and the third example says that breaker number one in linebay number four in station number two is stored as PHIGS structure number 80 (only one level of detail).

Since all objects inside a power station or a transformator station are named and numbered in a certain way, the programmer can relate the names to the screen and immediately locate them.

The number of actions possible to perform on the parallel structure is limited to a minimum. They are:

- `obj new node(string)` – locates the first empty element of the array and inserts string as ID
- `obj index(string)` – locates the element where ID is equal to string and returns the array index of that node
- `prev obj index(index)` – locates the father of the structures in array element number index by removing the last figures and then the last letters of its ID (for example "S2L4B1" would become "S2L4"), locates this new

element and returns its array index. This is used to traverse the PHIGS structure trees upwards and for spreading alarms to all structures affected.

If the application wants to access a certain PHIGS structure this is simply done with the command

```
phopenstructure(objlist[objindex("S2L3")].phindex[1])
```

This would make the PHIGS structure for the third linebay in power station number two in its middle level of detail, ready for structure editing.

The parallel structure is only used for the PHIGS structures of the actual power system. For the maps, power lines, menu etc., a number of constants are defined assigning a specific name to a special structure, making them easy to recognize for the user as well as for the programmer.

4.3 The Dispatch Routines

Dispatch routines are used to modularize a program in an object-oriented way without using an object-oriented programming language. This application currently has nine "classes" defined:

- class common
- class station
- class transformer station
- class linebay
- class isolated linebay
- class transformerbay
- class breaker
- class isolator
- class transformer

Every "class" consists of a number of C-routines for creating, displaying and updating the PHIGS structures. The class common is used to initiate and coordinate the other classes as well as to check for errors. In order to create this modularity some special lists are used (Figure 4.7). The classname variable mentioned in Section 4.2 is actually a reference to the class name list. This list holds nine arrays of references to the class function lists from where the calls to the specific C-routines for performing the actual actions are issued.

Consider an example where we want to create a tree structure for a power station like the one in Figure 4.3. One single command is needed – `obj create("S1", class station)` – everything else is taken care of by the dispatch routines. This command is a call to the class common to create a station. The essence of what then happens is shown in Figure 4.8. The routines create the parallel structure used by the application as well as the PHIGS hierarchy.

The same principles are used by the routines for displaying a hierarchy. The previously created structures are then connected to a root structure and filters are assigned to the specific structures. When updating is performed the routines work the other way around, starting at the bottom of the structure tree and move upwards.

By using the dispatch routines properly it is possible to create a number of different structure hierarchies and displaying them in almost every conceivable combination. The routines are somewhat difficult to implement, especially to get all the necessary pointer references right. But when they function properly,

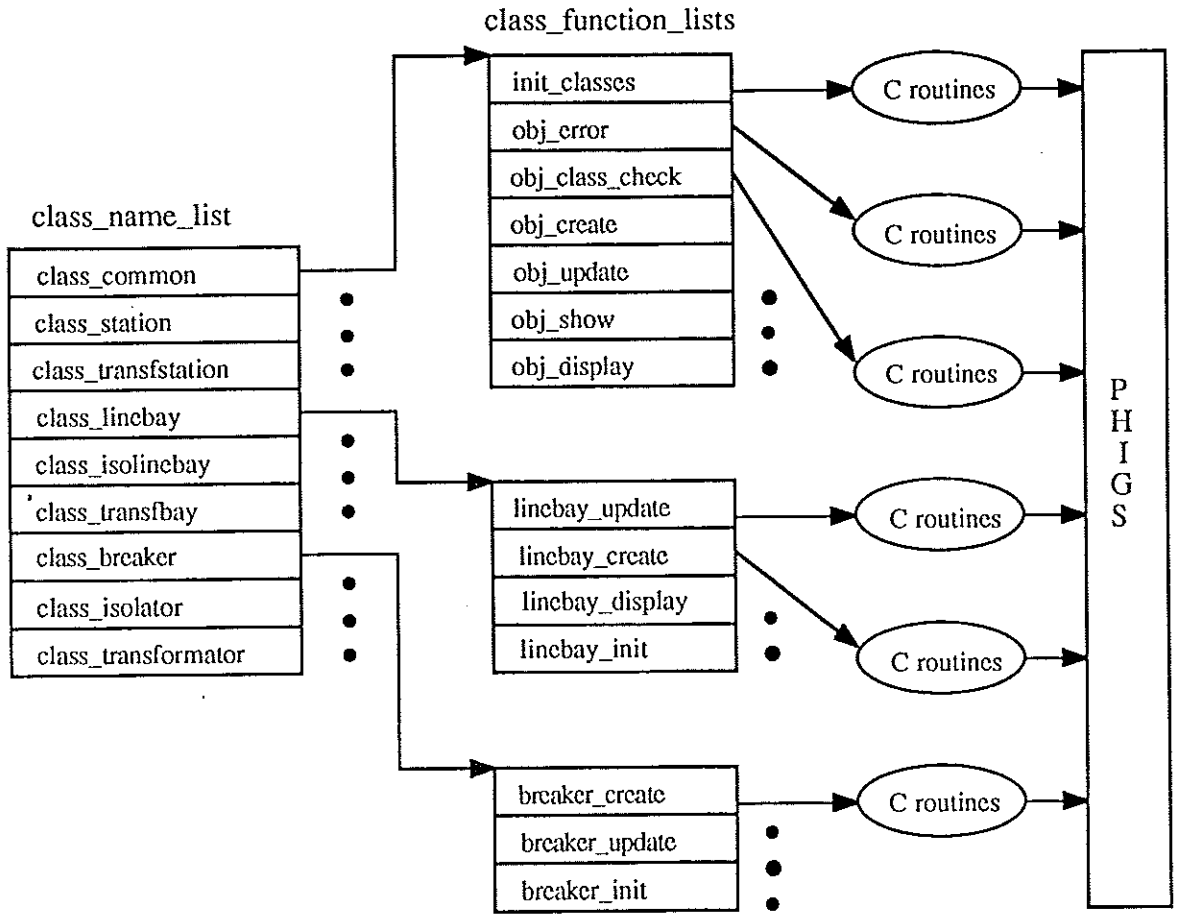

Figure 4.7 The design of the dispatch routines.

the dispatch routines are a well structured and powerful tool. The modularity also makes it elementary to implement new classes and create new PHIGS hierarchies.

All actions on the power system structures are performed by the dispatch routines. Actions on other types of structures – maps, power lines etc. – are handled in an easier, more direct manner, as these types of structures are much less complex.

4.4 The Invisibility Filters

The combination of information zooming and the PHIGS invisibility filters proved to be a very simple and elegant solution to an otherwise rather difficult problem. With a few simple instructions, a set of integers is assigned to every PHIGS structure as they are connected to form the hierarchical tree structures. As earlier mentioned, the application uses twenty different zoom levels and therefore the number of filters were also chosen to twenty. By simply changing the filter to be visible one step, whenever the zoom level is changed one step, the objects on the screen will automatically change their level of detail when a zoom operation is performed. This implies that the predefined zoom levels must be related to the size (in world coordinates) of the window where the zooming is performed. Some extra difficulties appear for nested structures

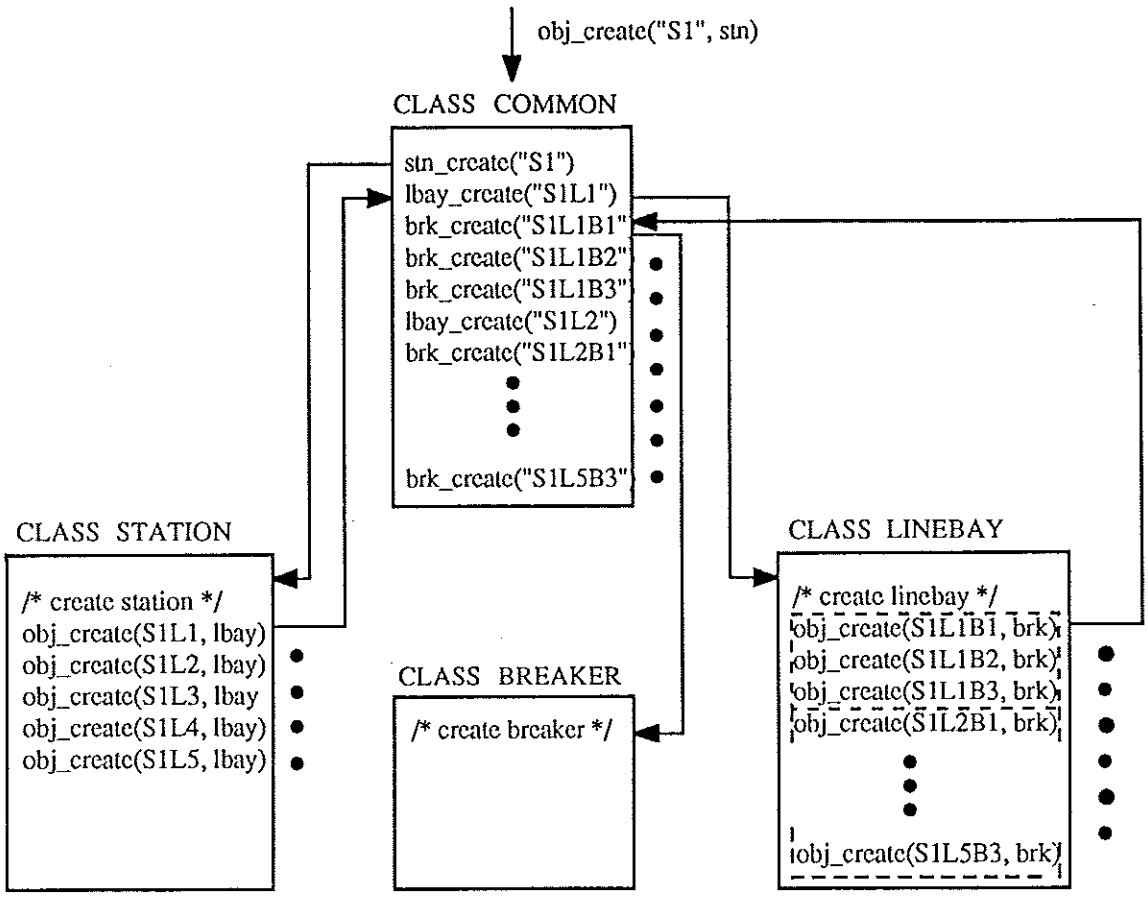

Figure 4.8 The dispatch calls for creating a power station.

(the same structure visible several times but at different zoom levels).

Let us as an example regard the PHIGS tree for a power station and this time include a possible setting of the filters to be visible (Figure 4.9). For every third zoom level the displayed power station will change its level of detail. For example, at zoom level 12, the station will be displayed at its highest level of detail and its internal linebays at their middle level of detail. When zooming in one level further the station will display all its internal objects in their highest level of detail including the breakers inside the linebays. This illustrates how the zoom function in a very simple way governs the actual size of the objects as well as in what level of detail they should appear on the screen.

Since all filters in PHIGS are workstation dependent, the five views created by the application has to be implemented as different workstations in order to display different levels of detail. However, both overview areas are defined as the same workstation because they are both always set to display filter level zero (the lowest level of detail). The MenuArea is defined as a separate workstation, although no filters are used here. A possible extension of the program it would be to have different menus visible at different times depending on what action the user wants to perform. The current design makes this extension easy to implement.

Today the application is implemented to give the programmer a considerable degree of freedom when setting the filters. By proper use of the dispatch routines it is possible to display every single structure at a special filter level. It is also possible to display the same structure tree several times somewhat apart but with different filters assigned, as the invisibility filters are not set

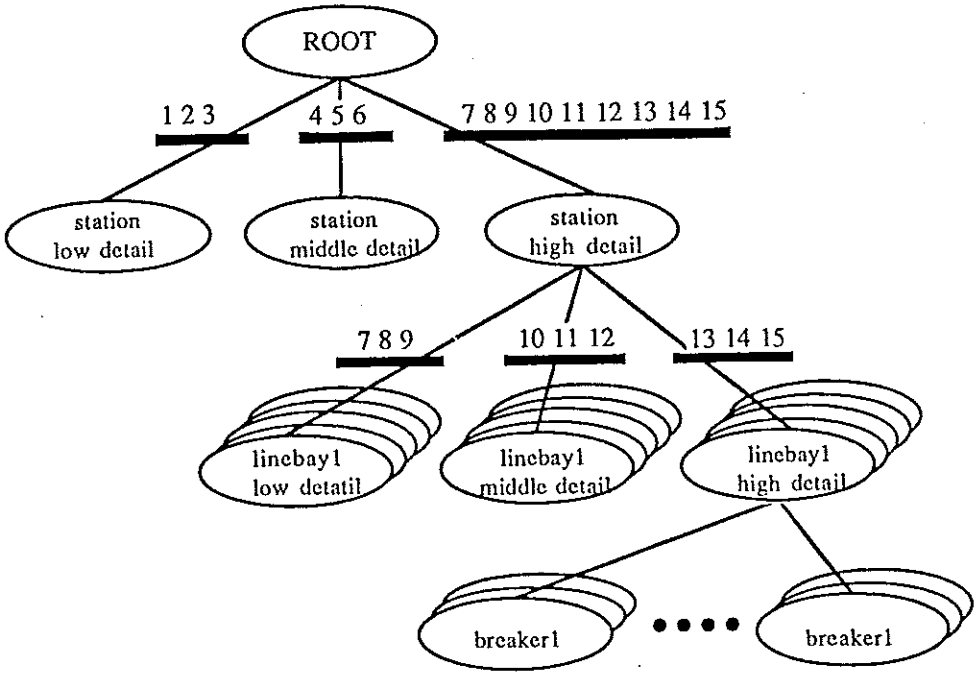

Figure 4.9 The PHIGS hierarchy for one power station combined with the visibility filters.

until a specific structure is executed from its root structure and not when it is created.

Altogether, the use of the invisibility filters is an efficient, easy to implement, and elegant way of solving the implementation of information zooming which is one argument for using PHIGS.

4.5 Problems

Most of the encountered problems when developing this application were caused by the NMP-PHIGS package. A number of bugs were discovered as well as some important lack in the functionality.

The major drawback of NMP-PHIGS is its performance and efficency. It is much too slow for any larger applications, especially when the number of PHIGS structures are great and some kind of dynamic updating is needed in real time. This is discussed further in Appendix A.

The absence of the event mode for interaction via the mouse was a disappointment. Today the application uses the request mode for input from the user. This means that when no input is given the program is idle. The event mode allows input to be stored in a queue which the application can flush when it is appropriate. Polling between the user's input and input from the outside world (alarms and other signals from the power system) could then be easily implemented. This can be solved by using a separately running process, handling all input from the power system and creating some link of communication between this process and the application, however this has not been done.

In order to use text representation in combination with zooming, it is

necessary to display characters of many different sizes. This PHIGS implementation supports only three fonts, which is insufficient. It is possible to develop a support program to generate new fonts, preferably using both raster and graphical fonts (graphical fonts are needed for really large text), however, this is not the purpose of this project.

The possibility of storing application data in the PHIGS structures is not implemented (it is possibly a useful feature, though it has not been evaluated how it ought to be used).

4.6 Alternative Methods

The design and implementation described in this chapter is probably far from beeing the best possible solution. A variety of alternatives exists and another application programmer would no doubt choose a different design, fitting his experiences and preferences. This is actually one important strength of PHIGS, allowing every programmer to define and organize the graphics data in the manner most convenient for his own needs.

The parallel structure (see Section 4.2) is obviously one part which could have been implemented differently. By using linked lists or tree structures similar to the PHIGS hierarchy, the data in this structure could be organized in a better, more logical way and allow faster access to specific structures. This application uses a kind of linear search on the parallel structure, which is of course not satisfying when fast access is wanted. If the application had been implemented in an object-oriented language, it would have been very convenient to have one PHIGS structure associated with every object in the application (ono-to-one correspondence), in order to take the maximum advantage of the hierarchical structures in PHIGS.

Also, the PHIGS hierarchy could have been used in a number of different ways. However, the concept of many small, well-defined structures, as in this project, is usually easier to understand than a hierarchy consisting of a few large structures for the same amount of graphics data. Though by using labels inside the PHIGS structures it would be possible to create a kind of "substructure environment" inside a large ordinary structure. PHIGS labels have not been used at all in this project, but they could be used to reduce the number of structures and still make it easy for the application to access specific parts of the structures. This could be accomplished by not only storing the PHIGS index in the parallel structure but also the label name where the specific part was stored. It would also be an advantage if the PHIGS structure trees were more balanced, e.g. have about the same number of childstructures whichever branch of the tree was examined. This allows search routines on the trees to work more effectively. If this application was extended with new types of substructures, only the right-most part of the hierarchy would continue to grow if the original concept was adapted.

A possibility not used in this application is to change the values of a certain PHIGS attribute in the workstation state lists, when a dynamic updating is performed on a structure. Instead we have used the more powerful but also more time-consuming method of explicit structure editing (deleting the structure element that assigns the specific structure with its attributes and inserting a new one). This is not really needed, since only the attributes are changed and no new instructions are inserted. By assigning a specific set of attributes to every single structure and simply change these when for example

an alarm occurs, the objects would change their appearance on the screen – without having to perform a search in the PHIGS database and edit the structure in question – when the next traversal of the PHIGS hierarchy was done. This would have a positive effect on the performance of an update operation, but it is not known if there is a limit, how many sets of attributes that can be stored in the workstation state lists. Today the application includes about twenty different PHIGS attributes (bundled attributes) for every workstation but this alternative method would require between 100 and 1000 times more attributes. A combination of both methods for performing dynamic updating would perhaps be the most efficient solution.

5. Conclusions

This project was inspired by the new graphics standard PHIGS, offering new possibilities for man-machine interaction and giving the application programmer a higher degree of freedom when designing a system. The objective was to implement a small prototype of a full graphics control system so that ideas could be tested and experiences gained. In this chapter these experiences will be summarized.

5.1 The User Interface

One of the main problems when designing a user interface for a modern control system is how to limit the amount of information displayed. A large system of today is such a complex one, with thousands of messages and signals being received and handled every second, that the operator is most likely to be overwhelmed with information and is not able to determine what is essential. The user interface must be designed to control this, as well as supporting concepts and operations that are natural to the user.

The representation of the system itself is always a key issue. There are several levels of representations and ways of viewing a system [Mattsson et al., 1986]:

- physical representation
- stylized representations
- conceptual representations, explaining the behaviour of a system
- mathematical descriptions
- graphical representations of different characteristics

To handle models for large and complex systems, the model developer must be able to decompose the model into submodels. Modularization simplifies modelling and makes the model more flexible and easier to adapt and manage. Since technical systems are often built in a modular way this decomposing is often natural.

In this project information zooming and different levels of representation combined with a multiple window environment were used to create a natural and easily understandable representation of the system. The concept of using menus, buttons, and the mouse for selecting operations, makes the interaction with the user simple and highly self-explanatory. Though the principle of seeing and pointing was not used in this application – because event driven input was not supported – it would be preferable to allow the user just to point at an object on the screen and have it zoomed in without first having to select the action in the menu.

Hierarchical models and changable levels of representation are recommended as a general tool for describing these kinds of systems. There are still many open questions concerning their looks and the means for creating and editing them [Mattsson et al., 1986]. Should symbols (icons) be used to denote different parts instead of annotated boxes? How should keyboard and mouse be used when editing and inspecting models? There is also a need

for other complementary structuring concepts for handling models of different complexities.

Continuous panning and zooming demand very fast graphics. It is not necessary to have these features for this kind of control system. Actually it is more natural to zoom in a step-like fashion if the display is regarded to consist of layers in different levels of detail.

5.2 Graphics

The application developed uses graphics to describe the system. The graphical objects themselves as seen on the screen are simple 2-D objects in the form of lines, rectangles, and characters. Due to the size and complexity of the PHIGS database, fast graphics is needed to make it look nice when actions are performed on the objects. The user also requires rapid feedback.

The concept of PHIGS as the graphics standard for this project worked very well, except the used implementation was unacceptably slow and had a number of irritating bugs (see Appendix B). The high-level features of PHIGS, in particular the hierarchical structure for the graphics data, makes it easy and logical for the application programmer to work with, as well as allowing the designer to define and organize the graphics data as best suited for the specific application. Another very useful feature was the combination of the PHIGS invisibility filters and information zooming. The filters allow a large degree of freedom for the programmer, it is easy to implement, alterable on-line etc. The possibility of performing specific structure editing on the PHIGS structures on-line is also a powerful concept which allows all kinds of updating on the objects displayed, in a direct and simple way.

The correspondence between the application and the PHIGS structure hierarchy is still a problem. The concept of implementing the application in an object-oriented language and associating every object with a specific structure in PHIGS appears quite promising in theory but has to be evaluated in practice as well. It would probably require some fancy programming, but should not be impossible to implement in an efficient and user-friendly way.

Full graphics display systems they have to be regarded as a necessity in the following years. For many types of system full graphics will allow significant improvements, as well as higher performance but for others they would work quite as well using some kind of semigraphical system. To be able to compete on the open market, full graphics will be a must, as most buyers will not accept anything else. It does not matter whether there is an actual need for full graphics or not.

In the years to come there will be a strong need for further graphics standards, which include the most recent work in this area, e.g. object-oriented graphics and window managers. But until such a standard is developed (still a number of years away) PHIGS is recommended as a possible graphics standard for the development of full graphics control systems for the following reasons:

- easy to work with and yet gives the programmer a high degree of freedom for designing solutions for specific applications
- the PHIGS structure hierarchy allow the programmer to arrange the graphics data in a logical and easy understandable way
- the PHIGS invisibility filters provide new possibilities for information zooming

- explicit structure editing is possible to perform on-line

It should also be noted that PHIGS is strongly promoted by IBM. A considerably faster and more complete PHIGS implementation than the one used for this project must be used. An implementation which really fulfills the quite extensive intentions and promises of the PHIGS standard proposal without sacrificing the efficiency and performance aspects is needed.

5.3 Performance

The NMP-PHIGS package was primary developed for CAD applications. In such applications the performance and efficiency demands are considerably less important than for a control system performing dynamic modifications of the displayed objects in real time. This project has made that clear.

The PHIGS implementation is written in objective-C. This is not the most suitable programming language if high performance is a priority. Due to the fact that compatibility between a number of different workstation types has been one important issue when designing the system, there is no special use of fast graphics hardware, fast assembler routines or other special methods to improve performance. All this combined make this PHIGS implementation extremely slow and not at all suited for the purposes of this project [see Appendix A]. However, it is not unrealistic to imagine a PHIGS implementation with an increased efficiency of about 100 times. This would really make PHIGS a highly interesting and powerful alternative to other possible graphics standards for any kind of commercial control system.

Acknowledgements

The work described in this report is the documentation of a master thesis, done as a part of the physical engineering education at Lunds Institute of Technology. The author is most greatful to the staff at the Department KLL, ASEA Brown Boveri in Lund, who made this project possible. A special thanks to Bo Johansson, who was most helpful in sorting out the major problems in the early stages of this work and to Kent Bladh for his assistance while writing this report. The author also wishes to thank Sven Erik Mattsson and Dag Brück at the Department of Automatic Control in Lund for reviewing a draft copy of this document. Their constructive comments were greatly appreciated.

References

BAKER, M. P. and D. HEARN (1986): *Computer Graphics*, Prentice-Hall International, New Jersey, USA.

BANAHAN, M. and A. RUTTER (1982): *UNIX the Book*, Sigma Technical Press, Cheshire, United Kingdom.

BROWN, M. D. (1985): *Understanding PHIGS*, Template, The Software Division of Megatek Corporation, San Diego, USA.

BRUNS, R. L. and M. HECK (1985): "Draft PHIGS Standard Makes Major Departure from Both Core and GKS," *Computer Technology Review*, USA.

HECK, M. and M. PLAEHN (1985): "Standard Defines Complex Graphics," *Computer-Aided Engineering*, USA.

HEWLETT-PACKARD (1987): *HP-UX Reference Manual, volume I, II and III*, Hewlett-Packard Company, Fort Collins, Colorado, USA.

HEWLETT-PACKARD (1987): *Programming With the X Window System*, Hewlett-Packard Company, Corvallis, Oregon, USA.

HOPGOOD, F. R. A., D. A. DUCE, J. R. GALLOP, and D. C. SUTCLIFFE (1983): *Introduction to the Graphical Kernel Standard (GKS)*, Academic Press, USA.

ISO (1987): "Programmer's Hierarchical Interactive Graphics System (PHIGS)," *Draft Proposal no.9592*, ISO – the International Organization for Standardization.

ISO (1988): "Computer Graphics," *Draft Proposal no.9593*, ISO – the International Organization for Standardization.

KELLEY, A. and I. POHL (1984): *A Book on C*, The Benjamin/Cummings Publishing Company, Menlo Park, CA, USA.

MATTSSON, S. E., H. ELMQVIST, and D. BRÜCK (1986): "New Forms of Man-Machine Interaction," Report CODEN: LUTFD2/TFRT-3181, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MORGAN, R. and H. MCGILTON (1987): *Introducing UNIX System V*, McGraw-Hill Book Company, New York, USA.

PHIGS+ (1987): "PHIGS+, Functional Description," *PHIGS+/87-5R1*.

SIS (1985): "Datorgrafi – PHIGS," *Technical Report no.306*, Minab/Gotab, Kungaelv, Sweden, SIS – Standardiseringskommissionen i Sverige.

STALLMAN, R. M. (1986): *GNU Emacs Manual*, Free Software Foundation, Cambridge, MA, USA.

TELELOGIC (1988): *Base/OPEN, volume I and II*, TeleLOGIC AB, Sundsvall, Sweden.

Appendix

A. More about Performance

"A chain is not stronger than its weakest link." In the case of NMP-PHIGS this weak link is no doubt its performance and efficiency. However good the concept of PHIGS may be, an implementation must be efficient enough to allow the programmer to use the possibilities provided without achieving a result which can hardly be used at all because of its slow performance.

If we regard the developed application, the first problems occur when the PHIGS database is created. More than fifteen minutes CPU-time is required to create the 2700 structures and combining them into hierarchical structure trees (see Chapter 4). However, the most serious problem is not the required time itself but the way the CPU-time increases as the database gets larger (Figures 1 and 2). To create the first complete transformer station (this means creating 48 new PHIGS structures) the system requires about 0.8 seconds, but to create the twentyfourth and last transformer station more than 14 seconds CPU-time is needed. This means that it takes 18 times longer to create a transformer station when the database holds about 2000 structures than when it is almost empty. The operation of combining the structures to hierarchical trees shows the same increase of required CPU-time as the database gets larger. This implies that the routines for handling the database are not created for dealing with a large number of structures and probably performs its actions and searches on the database in a linear way, which is totally unacceptable. Most performance problems which appear later are a result of the implementation of these routines. Obviously, major improvements can be done here.

When examining these results closer it is discovered that the operations responsible for this unfortunate development are

`phopenstructure(idendifier)`

`phinststructure(identifier)`

Both these operations perform an explicit search in the PHIGS database and sets a pointer reference at a specific structure when found. Figure 3 shows how the CPU-time required for "phopenstructure" increases as the number of structures in the database gets larger (approximately the same applies for "phinststructure"). Since these two operations are very often used, their performance affect the whole implementation in a negative way.

When the database is finally fully created the interactive part of the application begins. For example, a pick in the menu to select an operation often requires several seconds of CPU-time to be acknowledged and approved, though sometimes the pick is immediately approved. The reason for this difference is not understood. The approximate time needed for the possible actions is given in Figure 4 but these times do not include the redraw of the views concerned. The update operation requires about 3.4 seconds of CPU-time. This is quite reasonable since five "phopenstructure" are performed during an update (5 * 0.6 = 3.0 seconds). The actual structure editing is performed in

a split part of a second. The performance of the redraw operation is of course highly dependent on how many objects and in what level of detail they are to be drawn on the screen. The required CPU-time varies from 0.2 seconds up to 3 seconds.

All operations and searches on the parallel structure as well as producing the transformation matrixes needed are totally neglectible in comparison with the ones described.

What needs to be improved the most are the search routines for locating a specific PHIGS structure in the database and the operations for redrawing a view on the screen. However it may prove difficult to increase the performance and efficiency as much as is required. When Satt Control evaluated PHIGS (as well as GKS and Core) for their new full graphics display and control system – SattGraph 1000 – it was rejected, mainly because of efficiency and performance problems. Satt Control then developed a special designed graphics implementation which was extremely effective but of course non-portable. If the efficiency problems can be solved, PHIGS will be a powerful tool not only for CAD applications but also for full graphics control systems.

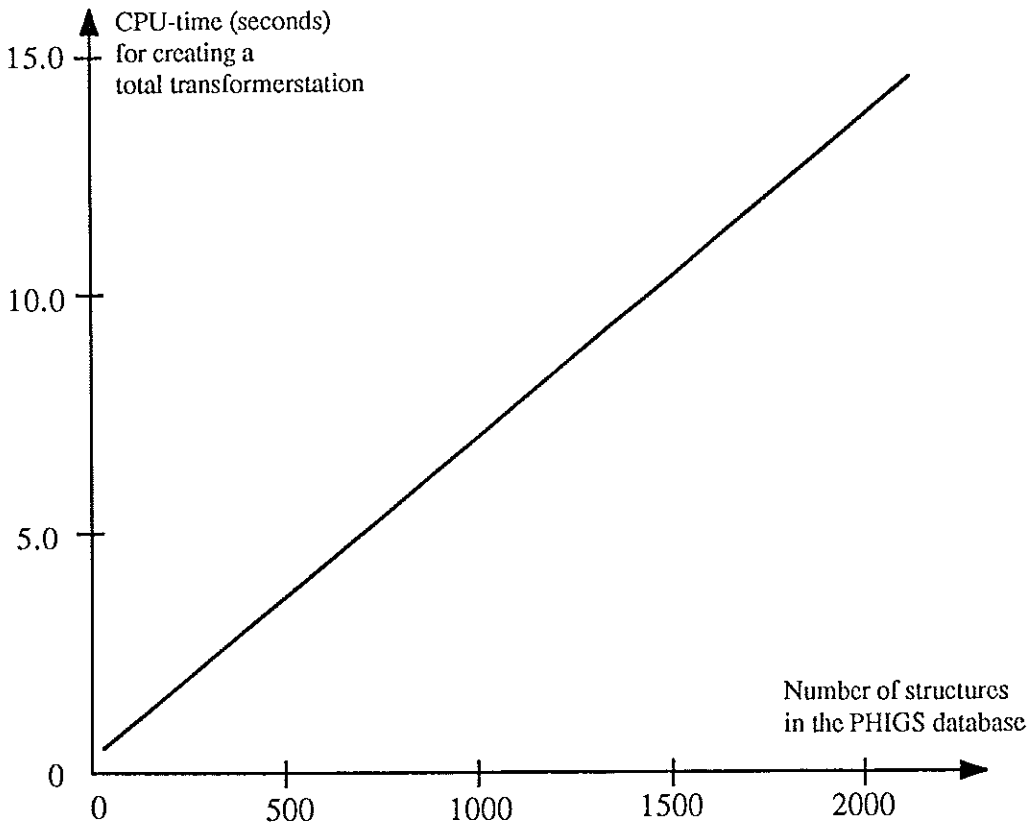

Figure 1. Required CPU-time for creating a transformer station as a function of the number of structures already in the database.

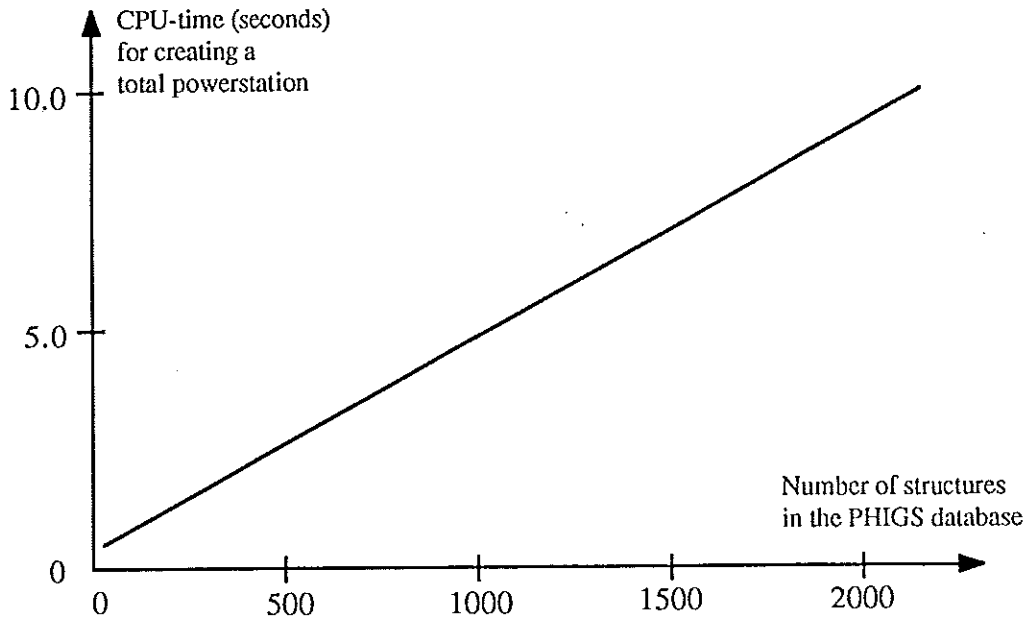

Figure 2. Required CPU-time for creating a power station as a function of the number of structures already in the database.

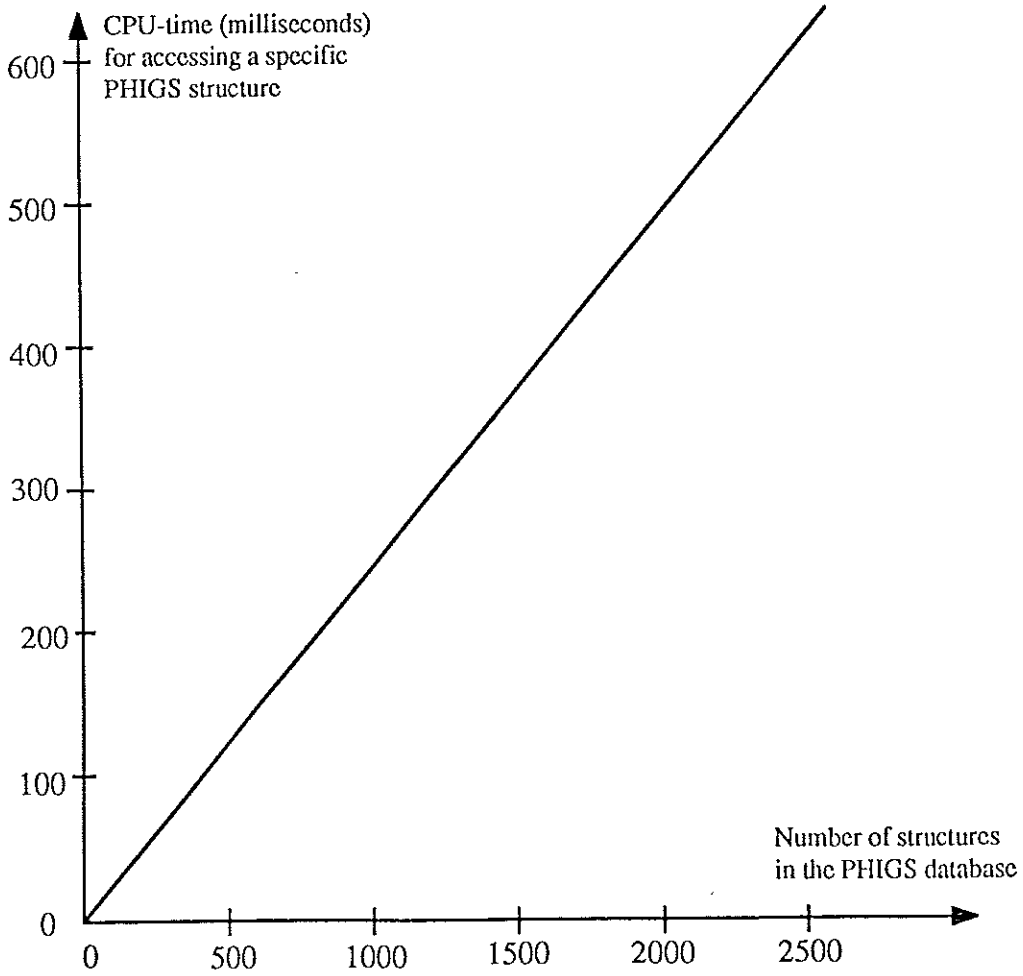

Figure 3. Required CPU-time for accessing a specific PHIGS structure as a function of the number of structures in the database.

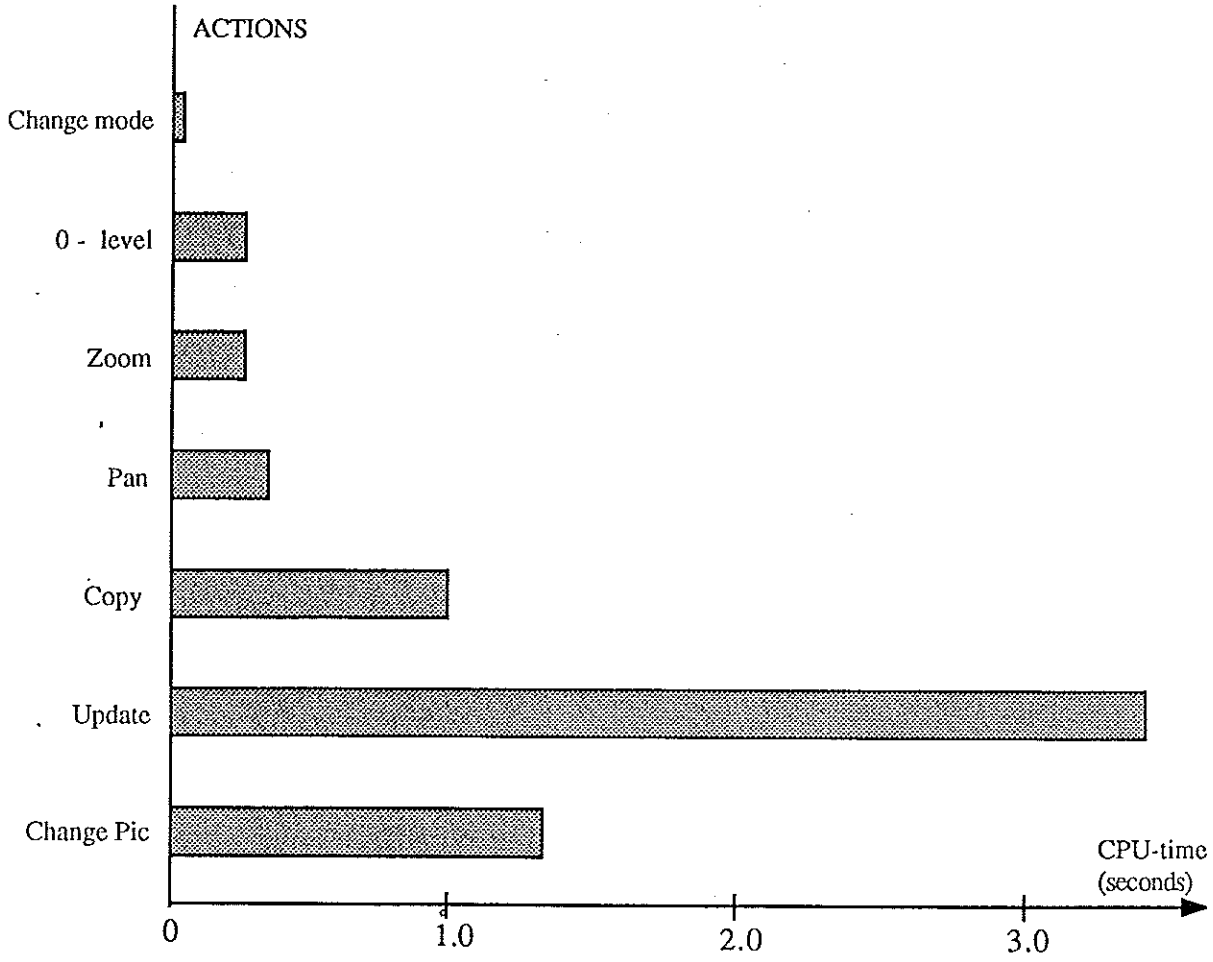

Figure 4. Required CPU-time for the actions supported by the application.

B. Discovered Bugs

One of the more noticable bugs in this PHIGS implementation concerns the different update functions. A number of modes exist but they do not perform what they are intended to. In order to be sure of getting what is wanted on the screen the operation "redraw all structures" has to be used. This makes the screen twinkle since the CPU-time required for a total redraw is not neglectable and the PHIGS implementation only works with a single picture memory for every defined view. By using a double-buffered memory, the updating and redrawing of the views would look much better, especially if it was syncronized with the electron sweep of the workstation screen. This means putting the new picture to be displayed in the currently not used memory and change the buffers when the sweep is moved from the down right corner to the upper left corner of the screen, and then redraw the screen. This would make sure that the display did not twinkle when a redraw was performed. It should not be necessary to redraw all structures, but only those which have been edited or changed.

A similar problem exists for the "post root" function. It is supposed to perform a complete redraw of the view to which it has been posted but when there are many substructures to the posted root only a part of the view is redrawn. Therefore, an explicit redraw command has to be given after every post root command, which makes the screen twinkle twice.

Even when the redraw command is given, it is not unusual that smaller parts of the screen is not properly regenerated. A polyline here and there is often left out.

Another bug exists in the clipping functions. When a PHIGS structure, which has been rotated is displayed with a part slightly outside the defined view area, the whole object is deleted from the screen. This makes the created demonstration views somewhat unrealistic since the number of rotated structures had to be reduced to a minimum.

A more serious error exists in the PHIGS implementation. When the number of structures in the PHIGS database is too large (approximately more than 3000) something goes very wrong. The program is suspended and the execution has to be interrupted. It is probably caused by some kind of memory assignment error, as the program starts to execute procedures it is never supposed to use before it is suspended. This is why the number of structures used by this application is limited to 3000.

Another error occurs when there is a need for two consectutive picks on two different workstations, e.g. when panning is performed. After picking the pan button in the MenuArea has been accepted by PHIGS, a new initiation of the pick device is done, this time for the centre coordinates in one of the working areas. The result of the initiation is stored in the workstation state lists. This action takes some time, because if a delay is not performed by the application between the initiation and the actual pick, the program will again be suspended. The pick will be recognized by PHIGS but not approved as a pick in the proper workstation and as request mode is used, the operator can keep on picking forever without getting by. Therefore, a sleep operation which suspends the application for one second is used. When the user wants

to pick a filled area another bug appears. If the pick is not performed on the borderline of that area, it will not be recognized and the pick aperature will automatically increase its size and try again. This means that the pick will finally be approved, but it might take some time if the pickidentifier lists have to be searched several times. It also implies that a pick performed outside any defined pickable area will cause the pick aperature to grow and finally find something pickable.

When leaving the program by using the exit function, the objective-C runtime error system will type an error message, which says that the system can not find the right structures to delete. This occurs when the function "close phigs" is executed and the database shall be emptied. It does not affect the application during its execution and is probably an error in the way PHIGS deletes substructures, which are referenced by several rootstructures.

For some unknown reason the UNIX p-option when compiling the program, in order to produce a code that counts the number of times each routine is called, does not function. When executing the program after such a compilation, a bus error and a core dump will be the only result. This is why a number of simple time-measurement routines was implemented along with the rest of the application.