CODEN: LUTFD2/(TFRT-3181)/1-025/(1986)

New Forms of Man-Machine Interaction

Sven Erik Mattsson
Hilding Elmqvist
Dag Brück

STU project 84-5069

Department of Automatic Control
Lund Institute of Technology
September 1986

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

Document name
Report

Date of issue
1986-09-29

Document Number
CODEN:LUTFD2/(TFRT-3181)/1-025/(1986)

Author(s)
Sven Erik Mattsson
Hilding Elmqvist
Dag Brück

Supervisor

Sponsoring organisation
The National Swedish Board of Technical Development (STU contract 84-5069)

Title and subtitle
New Forms of Man-Machine Interaction

Abstract

The user interface is a very important part of interactive software. A good user interface can significantly enhance the software's apparent functionality. Unfortunately, there is no theory of interface design. Vaguely you can say that they should support the user's mental model of the problem and consequently they are application dependent.

The new workstations with high performance graphics offer new possibilities for visualizing and giving a more concrete form to important concepts. In this project, some of these possibilities have been investigated by developing and implementing a prototype simulator for dynamical systems. The attention has been focused on the user interface for model development. In automatic control the notion of system is important and its representation is a key issue.

The simulator supports hierarchical block diagrams to describe the model decomposition and the interconnection structure. At the highest level the user sees a block diagram of annotated boxes connected with lines. The user can scroll, pan and zoom the block diagram continuously in real-time. Zooming controls the amount of information displayed. When zooming in on a block, it changes from an annotated box into a representation showing internal structure with increasing detail. Since the block diagrams can be hierarchical, it is possible to make the description at each level simple and clear. The simulator is called Hibliz which stands for HIerarchical BLock diagrams with Information Zooming. The user creates and edits his block diagrams in a Macintosh-like fashion. He can also create overview windows which indicates where he is when he pans and zooms.

Hibliz also simplifies model development by allowing submodels in the form of ordinary differential and algebraic equations rather than assignment statements for derivatives and algebraic variables.

Key words
Computer Aided Control Engineering, Man-Machine Interaction, Computer Graphics, Simulation, Differential/Algebraic Systems

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
25

Recipient's notes

Security classification

Contents

1. Introduction

The user interface is a very important part of interactive software. A good user interface supporting the user's mental model of the problem can significantly enhance the software's apparent functionality.

It is natural for a control engineer to use block diagrams when describing a process or a model. A graphical description of the decomposition of a model into submodels and their interaction structure is more apprehendable than a textual description. Unfortunately, interactive software for Computer Aided Control Engineering (CACE) leaves a great deal to be desired in this respect. The use of graphics in CACE has been hampered by lack of feasible hardware for a long time. Most commonly available packages for analysis and design of control systems have just simple graphics facilities. For an overview of a number of packages see Jamshidi and Herget (1985). The packages can typically draw trend curves, phase planes, Bode plots, Nyquist plots and root locus plots. You can find fancy graphics systems for animation in special purpose training simulators. Unfortunately, such systems are very expensive, typically costing more than a million dollars. However, the situation is changing drastically. The new workstations with high performance, real-time graphics now appearing on the market offer new possibilities for man-machine interaction. See the survey paper Machover and Myers (1984). For less than 50 000 dollars you can today buy a workstation that in real-time can animate for example a robot in shaded 3-D color graphics. For example Silicon Graphics offers such systems (Machover and Dill, 1986).

The main purpose of the project "New forms of man-machine interactions" was to set up and implement a prototype system exploring some of the possibilities of high performance, real-time graphics so that ideas could be demonstrated and tested, and experiences of using graphics for man-machine interaction could be gained.

The prototype system was chosen to be a simulator for dynamical systems. There were several reasons for this choice. First, a simulator is a very important and useful tool of a CACE system. Second, the design and implementation of a simulator touch upon many different aspects of man-machine interaction. An important issue is the representation and visualization of systems. It is a key issue in CACE, since the notion of system is an essential element of control theory. Third, the development and the use of the simulation package Simnon (Elmqvist, 1975 and Åström, 1983) and the work by Elmqvist (1978, 1983 and 1985) were good sources of inspiration. Fourth, software in Pascal from the Dymola package (Elmqvist, 1978) and from the LICS project (Elmqvist, 1985) could be used to give a quick start.

The attention has been focused on the possibilities to use graphics to

represent the model. The prototype simulator which is called Hibliz (HIerarchical BLock diagrams with Information Zooming) supports hierarchical block diagrams to describe the model decomposition and interconnection structure. The user can at the terminal pan and zoom the block diagram continuously in real-time. Zooming controls the amount of information displayed. When zooming in on a block, the block changes from an annotated box to a representation showing internal structure with increasing detail. Since the block diagrams can be made hierarchical, it is possible to make the description at each level simple and clear.

Hibliz accepts models that can be described by sets of ordinary differential equations and algebraic equations and allows the submodels at the lowest hierarchical level to be described in the form of equations rather than assignment statements. In most simulators of today, the user must solve for the derivatives from the equations and input explicit expressions for calculation of the derivatives to the simulator. Hibliz accepts the equations as they are, thus simplifying the user's work and giving a more readable and better documented model.

A description of Hibliz is given in Chapter 2. The model description concepts and the ideas behind them are discussed. The use of Hibliz is also illustrated by means of an example. A short description of the implementation is also given. The conclusions are given in Chapter 3.

2. Hibliz

This chapter describes the prototype simulator Hibliz. The chapter is organized as follows. The model description concepts of Hibliz are discussed in Section 2.1. The operation of Hibliz is described in Section 2.2 and the implementation in Section 2.3.

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 with increasing detail. The first abstraction level for a model might be just its name or icon. The next level might contain a description of usage and external behaviour, and a third level details about its internal behaviour. The amount of information increases at lower abstraction levels. Modularization can be used in order to maintain useful views with a limited number of related concepts. Modularization means that the information at a certain abstraction level is decomposed into smaller entities.

Our concept is that a graphical description of the structure is easier to understand than a textual description. Modularization is achieved by use of block diagrams. To support abstraction we propose information zooming and hierarchical block diagrams. Multiple windows are used to further support multiple views of the model.

The use of hierarchical block diagrams and information zooming will be illustrated by an example. Please remember that when sitting at the terminal you can scroll, pan and zoom in the windows, but that the dynamical aspects are lost in a paper which only can show snapshots of the screen.

An Example

As an example consider a model of a thermal power plant. The block diagram in Figure 1 shows the major components and their interaction. The annotated boxes represent submodels and the lines between the boxes indicate interaction between the submodels. To the left in Figure 1, we find the model for the combustion chamber. It delivers energy to the boiler, the heaters and the reheaters in the turbine part. The boiler produces steam, which is heated in the superheaters. The steam then goes to the turbines via the steam valve. From the turbine system the steam enters the feed water system. Extract steam from the turbines is used to preheat the feed water. The feed water goes to the boiler, but also to sprayers in the heater. The equations describing

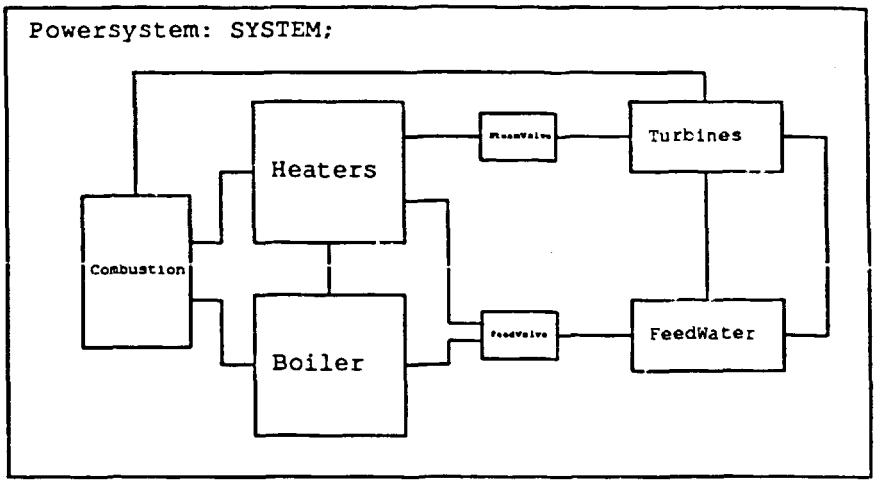

Figure 2.1 Power system model.

the system are typically mass and energy balances. External functions for interpolating in steam tables are also required. The model has 470 variables and in its textual form it is 1200 lines long, including layout information.

By pressing the right mouse button when moving the mouse, we scroll and pan the block diagram continuously. If we press both the right and the middle button, we zoom. Let us do that (Figures 2 and 3).

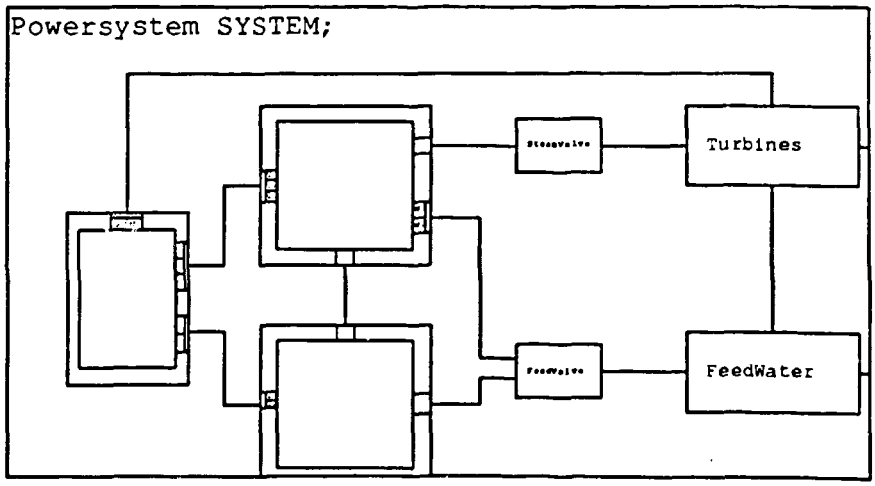

Figure 2.2 The power system model zoomed-in a little.

When we zoom in on the diagram, the blocks open up and show their internal details.

Let us zoom further (Figure 4). Now we can see the internal description of the boiler. It is a new block diagram.

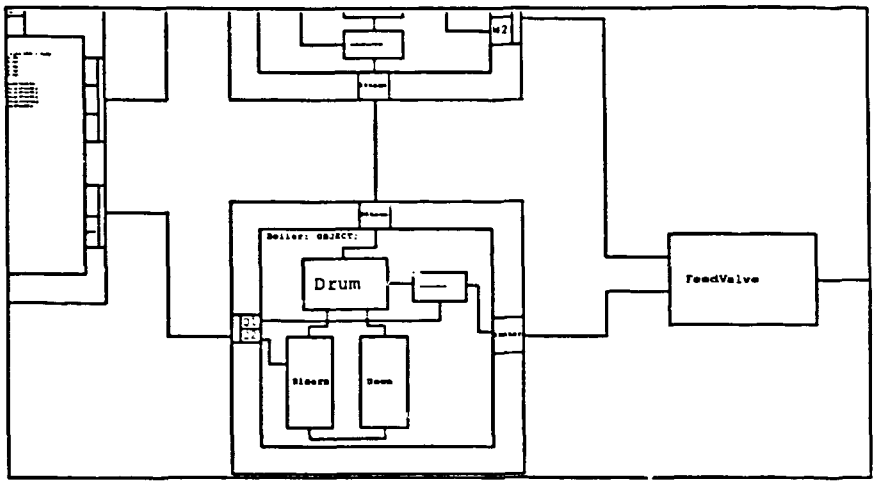

Figure 2.3 The power system model zoomed-in further.

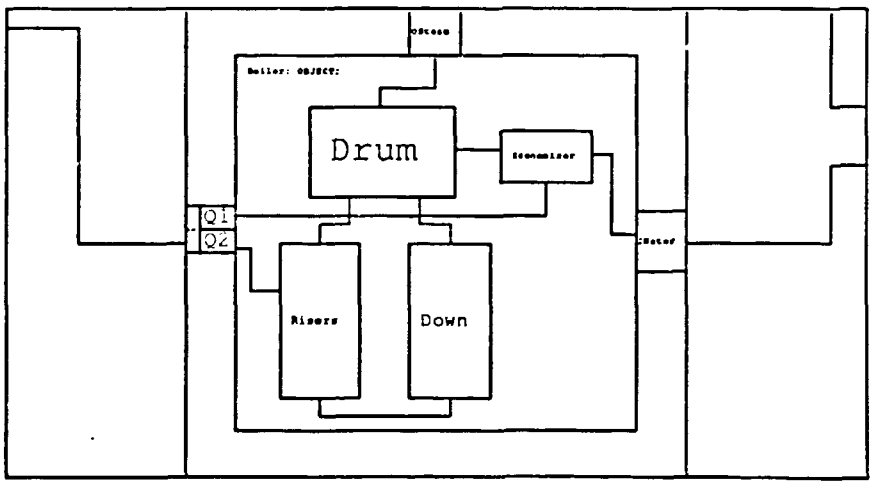

Figure 2.4 The power system model zoomed-in at Boiler.

Model Decomposition

Hibliz supports the use of hierarchical block diagrams as a tool to handle complexity. A basic rule is that a block diagram should be simple and contain only a small number of blocks. The selection of module boundaries is guided by our perception of the problem space. If our first attempt of structuring results in too many blocks, it is advisable to introduce a new hierarchy. There is almost always interaction (coupling) between modules. In order to be useful, a decomposition must be chosen in such a way that the external interaction complexity is small compared with the internal complexity. There should not be criss cross lines between the blocks. Furthermore, the entities in a module should be related (cohesion).

Modularization gives many advantages. It simplifies the modelling, it makes the model more flexible and easier to adapt and manage. We can also build and use libraries of models. Technical systems are often built in a modular way composed of standard components. Their behaviour may be well-known. Good, generally accepted models may already exist.

Multiple Windows

The user can create new windows for viewing a model. One of these windows is the current interaction window for scrolling, panning and zooming. To help the user keep track of where he is, rectangles outline the parts of a window that are also shown in other windows (See Figure 5, dashed rectangles).

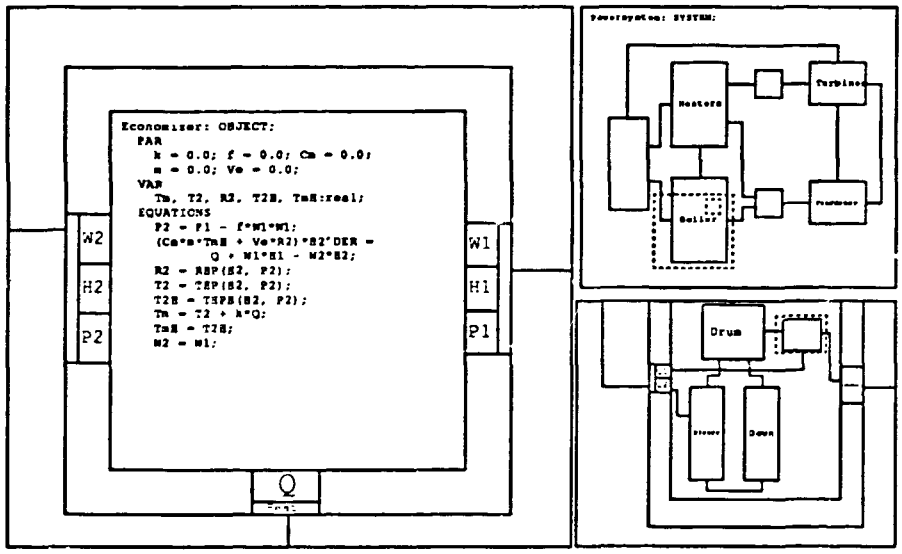

Figure 2.5 Multiple windows give different views simultaneously.

If the user wants to move fast he can point at an object in any window and ask for automatic scrolling, panning and zooming to this object in the interaction window.

Interaction Structure

Let us consider the interaction between models. The first level, the block diagram (Figure 1) shows which models interact. The next level (Figure 4) is concerned with how models interact, i.e. which variables that are involved. The graphical representation for a submodel consists of two rectangles; one inside the other. The descriptions of interfaces are placed at the border of the submodel between the inner and outer rectangle. At the most detailed level, the effect of interaction can be seen in the equations containing interaction variables.

A model is an encapsulated entity and the interaction variables are the only visible from outside. The interaction variables are associated with submodels. They cannot be associated with connections, because it should be

possible to develop a submodel without knowing in what environment it will be used. This is necessary in order to allow for model types and model libraries. A model often interacts with several other models, implying that the formal interaction variables should be grouped corresponding to the different possible connections. Such groups are called interfaces. Interfaces may have a hierarchical structure.

Consider Figure 6.

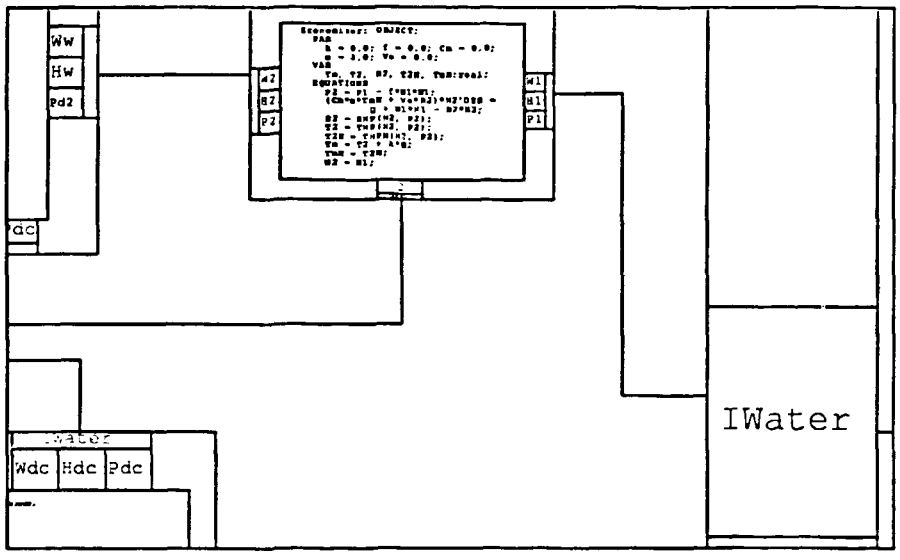

Figure 2.6 Zoomed-in picture of Boiler model.

It shows a zoomed-in picture of the model Boiler. To the right, the interface Iwater of Boiler can be seen. It models the incoming feed water to the boiler. This interface is connected to the right interface of Economizer to model that the feed water goes to the economizer. The feed water is heated in the economizer. It then leaves the economizer and flows into the drum. This is modelled by the connection between the left interface of Economizer and the right interface of Drum. A connection between two structured interfaces means that their corresponding components are connected. The number of components must be the same in the two interfaces. Primitive interface components may also be used to pass through a structured connection to submodels. The user need only specify the interfaces of non-primitive models to the degree of detail that is necessary to draw the block diagram within the actual block. The connection between the interface Iwater of Boiler and the right interface of Economizer illustrates this.

Behaviour

Before discussing the semantics of connected variables, let us have a look at a primitive submodel. The model Economizer is one. We can as before zoom manually, but there is also a facility for quick automatic zooming. If we point with the cursor at the outer rectangle of a model and press the middle button, Hibliz automatically zooms in on the model (Figure 7).

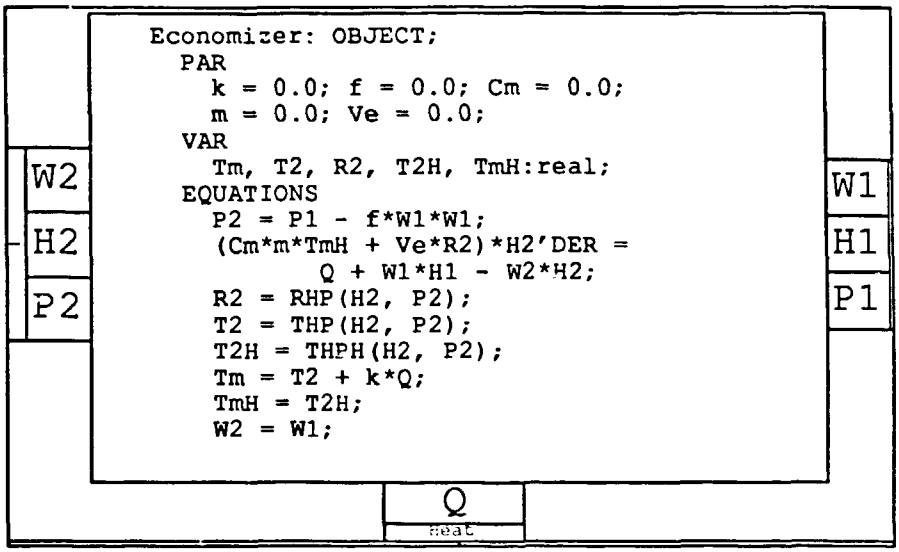

Figure 2.7 Economizer model.

The behaviour of models at the lowest hierarchical level is described by equations in textual form. An equation should have the form

$$expression = expression$$

We will not describe the syntax and semantics of expressions in full detail (see Elmqvist 1985). Expressions have the usual syntax with arithmetic, relational and boolean operators. Conditional parts (`if-then-else` expressions) as in for example Algol 60 are also allowed. Common mathematical functions like sin, cos and exp are available. The language supports simple integer, real and boolean variables. Hibliz also provides a mechanism for incorporating additional functions written in Pascal. These functions can be used directly in the model. For example `RHP`, `THP` and `THPH` used in the model Economizer (see Figure 7) are such external functions implementing steam tables.

There are four kinds of variables; constants, parameters, interface variables and internal variables. Constants and parameters are considered to be constant during a simulation, but the user can change the values of parameters on-line between simulations. The values of interface variables and internal variables may of course vary with time. The time derivative $\dot{x}$ of a variable x is written as `x'der`. The scope rules for variables are very simple because of the powerful connection concept. Variables can only be referenced from equations in the submodel where they are declared.

It should be noted that the basic concept is not assignment statements but general equations. A model can thus be represented as $f(t, \dot{x}, x, p) = 0$. Many integration algorithms and simulation packages require that the derivatives are solved explicitly by the user; $\dot{x} = F(t, x, p)$. The support of general equations simplifies the model development and the documentation becomes better, since equations are closer to first principles. When developing a model for a physical system one uses fundamental laws such as mass balances, energy balances and

phenomenological equations. These are either algebraic equations or ordinary differential equations which relates certain variables. Compared with the assignment form, it is easier to check that the model is entered correctly and the risk of introducing errors during manual transformation into assignment statements is reduced.

Furthermore, as thoroughly motivated by Elmqvist (1978), the equation form is the only reasonable representation for model libraries. With models on assignment form, it must be decided for each submodel which of its variables that are inputs (in other words are known) and which of its variables that are outputs (defined by the model). As a simple example consider a resistor. Ohm's law states $V_1 - V_2 = RI$, where V_1 and V_2 are the voltages at the ends of the resistor, I is the current through the resistor and R is the resistance. The model has three variables V_1, V_2 and I. The resistance R is in this model a given parameter. If we should write the model on assignment form there are three possibilities

$$I := (V_1 - V_2)/R$$
$$V_1 := V_2 + RI$$
$$V_2 := V_1 - RI$$

The first variant assumes that V_1 and V_2 are inputs and defines I. This model is appropriate if for example one end of the resistor is connected to a voltage source and the other to ground. The second and third variants assume that the current and the voltage at one end is known. These models are appropriate if the resistor is connected to a current source and ground. Consequently, for models on assignment form we need several different models for a resistor, depending on how it is connected to the environment. This makes both use and maintenance of a model library messy. Furthermore, other environments may result in algebraic loops so that equation systems with equations from several submodels must be solved to transform the model into assignment form. Two resistors connected in series between a voltage source and ground is a simple example of this. Submodels cannot be transformed into assignment form individually, but the transformation is a global problem.

A more sophisticated connection mechanism can be introduced when equations are allowed. Hibliz supports two kinds of connection semantics depending on the character of the interaction variable. Consider, for example, three connected electrical wires. Each wire is represented by a voltage and a current. The constraints at the junctions are that the voltages are equal and the sum of the currents is zero. These two types of cut variables are sometimes called across variables and through variables. Other examples of across variables are pressure and temperature. Mass and energy flow, thrust and torque are examples of through variables. It is natural to associate a direction with a through variable. The directions are important when putting up the equation for connected variables so that the terms in the "zero sum equation" are given correct signs. The interfaces are presently declared textually, for example, the interfaces of a resistor may be declared as

```
End1:  (V1:  real, I1:  IN real)
End2:  (V2:  real, I2:  OUT real)
```

An interface variable is defined to be a through variable by the keyword IN or OUT as indicated above for I1 and I2.

Hibliz interprets the connections drawn by the user and generates appropriate equations automatically.

2.2 How Hibliz is Operated

The user operates Hibliz via a keyboard and a mouse with three buttons. He can create and edit hierarchical block diagrams, inspect the model and simulate.

Scrolling, Panning and Zooming

The mouse is used for pointing on the screen; a cursor follows the movements of the mouse. We indicated earlier how the mouse is used when scrolling, panning and zooming. In order to scroll and pan, press the right button and move the mouse. To zoom, press first the right button and then the middle button. Now moving the cursor up means zooming in and down zooming out. An object (model, interface, curve or text) can be zoomed-in by pointing at it and pressing the middle button only. Zooming is done smoothly and the final size of the object is chosen as large as possible while still being contained in the window. Text objects are treated specially. Their final sizes are chosen so that the longest line of the text matches the width of the zoom window. The text line pointed at is scrolled to the center of the zoom window. This allows convenient scrolling within a window. A connection is considered to belong to the enclosing model. Pointing at a connection is therefore a convenient way of zooming out to a higher hierarchical level. If the middle button is pressed without pointing at anything, a zoom-out to 70% is performed. Note that it is possible to point at an object in a window other than the current zoom window. Pointing in an overview window thus allows rapid inspection of different objects. A new window is selected to be the current zoom window by pointing at it and pressing the left and right buttons.

Two-Button-Stretching

To layout objects like windows, models and interfaces on the screen is a common operation. The layout with respect to position and size is done with "two-button-stretching". The lower left corner and the upper right corner of the objects follows the mouse in different ways depending on which of the left and middle buttons that are pressed. If the left button is pressed, the cursor points at the lower left corner which follows the movement of the mouse. The same thing applies for the upper right corner when the middle button is pressed. If both buttons are pressed the cursor points at the center of the object and the whole object moves. The stretching is finished when both buttons are released. It should be noted that the objects are completely redrawn, even in the two-button-stretching mode.

Commands

Commands are chosen from a pop-up menu. The menu is shown when the left button is pressed. The desired command is selected by pointing at the corresponding menu entry. Hibliz highlights the selected entry and performs

the command when the button is released. If no entry is selected when releasing the button, the menu just disappears. Command actions sometimes require additional input from keyboard or mouse. Hibliz then prompts in the command area at the bottom of the screen.

As an example we will describe the Model command in more detail. This command creates a model and its graphical representation (two rectangles and the name). Hibliz prompts for name, which the user should type on the keyboard, and the enclosing model, which the user should select by pointing at it with the mouse and pressing the left button. The layout is done with "two-button-stretching" as described above. The Interface command is used to declare and position an interface. An interface is presently declared textually and its graphical layout is done automatically. The user positions and stretches it using two-button-stretching.

The Connect command makes it possible to draw connections between interfaces by using the mouse and the left button to input a sequence of line segments. The start interface is first selected by pressing the left button. Break points are given by releasing and pressing the button. The last line segment is refreshed while the mouse is moved (rubber-band drawing) and the interface structure is searched for the destination interface.

The Text command is used to edit the textual parts of models. The editing is performed using a simple screen oriented editor. When the user leaves the editor the text description is parsed. Error messages are currently given as text in the command window. Here there are lots of possibilities to use graphics and color to explain the error to the user.

The Remove command allows deletion of models, interfaces and connections. The Copy command copies a model and all its submodels, interfaces, connections and the text. The Save command stores the current model hierarchy as a text file. Such a file can be read by the Get command to recreate the model. If a model has been defined earlier, the hierarchical position and layout is given as for the Model command. The Copy, Save and Get commands make it possible to build simple model libraries.

The Layout command is used for changing the position and size of windows and models. There is no facility to change layout of interfaces and connections. The View command creates a new window for viewing the model.

The Compile command analyses the model and prepares for simulation. The Simulate command prompts for start and stop time and then starts the simulation.

The Display command creates a display for presentation simulation results of any variable. The presentation of simulation results is very primitive at present. Only simple trend curves are implemented.

The Hardcopy command creates a description of the current content of the screen in the form of a PostScript program (Adobe Systems Incorporated, 1985). The program can then be used to create hard copies on for example Apple's LaserWriter. The figures in this paper have been made in this way.

The Exit command stops Hibliz and returns to the operating system.

2.3 Implementation

The code for Hibliz consists of some 28 000 lines of Pascal. The software is highly modular. Related types, variables and procedures are grouped together. Machine dependent parts like file and string handling are isolated to improve portability. A preprocessor which we call Packman is used to produce a standard Pascal program from module files, since Pascal does not allow mixed declarations of global constants, types, variables and procedures.

Packman

Packman accepts files consisting of sections of code preceded by headings such as `.PROGRAM`, `.LABEL`, `.CONST`, `.TYPE`, `.VAR`, `.FORWARD`, `.PROCEDURE`, `.INIT` and `.MAIN`. Packman outputs the contents of all sections labeled `.CONST` on a file named `CONST.SEC` and so on.

Typically the file issued to Packman is a short file containing a number of commands to include files. Inclusions may be nested. Hibliz is built like a transparent onion consisting of six layers. Outer layers can use elements of inner layers. Packman is designed to promote separate compilation. By the commands `.DEFINITION` or `.IMPLEMENTATION` it is possible to specify what should be visible outside the compilation unit and what should be hidden. Portability with respect to separate compilation facilities of different Pascal compilers can be handled by modifying Packman.

Data Structure

A model is represented as a record containing lists of submodels, variable declarations, connections and equations. A general list package for doubly linked lists with headers is used. It has operations such as `NewList`, `Into`, `First` and `SuccElem`. A node is a Pascal record with variants. The common area contains information like forward and backwards pointers for list manipulation, and the variant part contains a pointer to a record describing models, interfaces, connections etc. Note that this way of implementing lists makes it possible for a model description etc. to be a member of several lists at the same time. The list package makes it easy to handle and manipulate lists.

Compiler

The model descriptions at lowest level are parsed at exit from the editor or when read from a file by the Get command. The parser builds a syntax tree for each equation. The Compile command links the identifiers to their declarations. Interface variables which are connected to each other are put into a list. These lists of connected interface variables are then analyzed to generate the proper equations, their syntax trees and the links to declarations. It also checks that across and through variables are not connected to each other. The type consistency of all expressions is checked.

When Hibliz has collected all equations, it has a differential/algebraic system of the form

$$g(t, \dot{x}, x, v, p, c) = 0$$

where t is the time, x and v are vectors of unknown variables, p is a vector of known parameters and c is a vector of known constants. The vector v

contains those unknown variables which do not appear differentiated in the equations. Hibliz uses the differential/algebraic system solver DASSL (Petzold, 1982). DASSL has a reputation of being one of the best and most robust numerical solvers for differential/algebraic systems. DASSL accepts problems on this form if it is provided with a routine for calculating the residual $\Delta = g(t, \dot{x}, x, v, p, c)$ when the arguments are known. However, to decrease the order and the complexity of the problem, simple symbolic formula manipulation is performed as follows. Connections of across variables lead to simple identities of the form $A = B$. It is easy to explore these entities and eliminate variables. The record describing a variable has an element called `alias` with an initial value implying that it is its own alias. When simple equations are found, Hibliz modifies the alias elements accordingly and removes the equation from the list of interesting equations.

After the elimination of simple equations, Hibliz assumes that $\dot{x}$ and v are unknown and sorts the equations and the variables so that the problem becomes block lower triangular with minimal diagonal blocks. If a block is scalar and the variable assigned to the equation is a component of the vector v, and the equation is of the form $v_i = \langle$expression independent of $v_i\rangle$ the variable v_i is eliminated from the vector of unknown variables passed to DASSL. The routine for evaluating the residual Δ can in this case calculate v_i itself. The partition to lower block triangular form may fail. An error message is then given listing unassigned variables and redundant equations. The problem is then either singular or it has algebraic relations between the components of the vector x.

To make the calculation of the residual vector Δ more efficient, Hibliz generates code for a virtual stack machine. The code is interpreted by a Pascal procedure. The values of constants, parameters and variables are stored in a global array.

The Run command sets the initial values of states as given by the initial section of the models and then uses DASSL to solve the system.

Graphics

Routines for handling graphics are an essential part of Hibliz. A local coordinate system is assigned to each model such that the lower left corner of the rectangle has the coordinates (0,0) and the upper right corner has the coordinates (1,1). The positions of its interfaces, submodels, equations etc. are expressed in this coordinate system. When moving and scaling a model, this hierarchy of coordinate systems makes it almost trivial to scale and move its submodels properly. All coordinates are stored as real numbers since continuous zooming requires high resolution coordinates.

Hibliz currently runs on an IRIS 2400 from Silicon Graphics, Inc. (Clark and Davis, 1983). The IRIS is a high performance engineering workstation designed for interactive color graphics and computing applications. The program interface to the graphics is the IRIS Graphics Library. It has routines for definition and manipulation of objects in (local) world coordinate systems and projection of these objects onto the screen. The IRIS has special graphics hardware for transformations from local world coordinates into screen coordinates. Clipping and scan conversion are also done in hardware.

Fonts

A block diagram contains text, and to make continuous zooming possible it is necessary to display characters of different sizes. The IRIS Graphics Library supports one fixed pitch raster font of height 16 and width 9. New raster fonts can be defined, but, due to memory constraints, large character has to be viewed as graphical objects consisting of straight and curved lines.

We have developed a support program to generate new fonts. This program is based on ideas given in Knuth (1979). The user defines the shape of a character as a number of line segments and the size and form (rectangular, circular, oval etc.) of the pen to be used. A line segment is defined by its start and end point and its tangents in these points. Intermediate points on the line segment are defined by the cubic spline function given in Knuth (1979, pp. 24-26).

Hibliz uses both raster and graphical fonts; for characters up to 16×20 pixels, raster fonts are used. The use of both raster and graphical fonts makes outputting of text somewhat more complex. There are, however, two good reasons for using raster fonts and not only graphical fonts. First, when drawing a small character the quantization may deform the character so that it looks ugly and is difficult to recognize. If a character is moved over the screen, its form changes due to the quantization and for example an 'o' looks like an amoeba. Second, it is important to make the graphics as fast as possible. When the characters are small there may be many of them on the screen. If the characters are defined as graphical objects this implies drawing of many line segments. For example, to draw a nice looking O, at least 20 line segments are needed. The IRIS can at maximum draw 65 000 line segments per second whereas it can display up to 150 000 raster characters per second.

3. Conclusions

The project "New Forms of Man-Machine Interactions" was inspired by the new workstations with fast graphics offering new possibilities for man-machine interaction. The objective of the project was to implement a prototype simulator so that ideas could be tested and experiences of using graphics could be gained. Such a prototype simulator has been implemented according to the project plans. The simulator is called Hibliz and is described in Chapter 2. The attention was focused on the user interface for model development. We will in this chapter try to summarize our experiences gained in the project.

3.1 The User Interface

How should the user interface of software supporting model development and simulation be designed? Hollan et al. (1986, page 25) summarize the state of the art of interface design as

> "Interface design is currently very much more of an art than a science. There is a tremendous need for better theories of interface design and for much more powerful tools to assist in their design and implementation. Currently there is virtually no theory of interface design. We do not even understand what contributes to the effectiveness of the most successful interfaces. We are in a state similar to when bridges were built by copying existing bridges without knowing in advance what would result from even the most minor variation. We need a more principled base for design of interfaces, one that characterizes the dimensions of the space of interfaces. Such theoretical characterization is the only way to be able to understand and intelligently make the myriad of tradeoff decisions inherent in interface design. Hopefully it is clear from this chapter that we think a theory needs to be erected from an understanding of interfaces as representational languages and based on an appreciation of the cognitive tasks that people are employing such representational systems to solve."

This answer implies that user interfaces are application dependent. It should support concepts and operations that are natural to the users (and are powerful). The notion of system is important. Its representation is a key issue in both simulators and other CACE programs. There are several levels of representations and ways of viewing a system:

1. The physical representation; the plant itself.
2. Stylized representations, which can be used for animated cartoons.
3. Conceptual (metamorphical) representations helping you to think about and understand the behaviour of systems.

4. Mathematical high level descriptions like "this system is described by three energy balance equations."
5. Graphical representations of characteristics like time responses and frequency responses.
6. Pure mathematical descriptions.

Hibliz supports pure mathematical descriptions.

Hierarchical Block Diagrams

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. He can also build libraries of models. Technical systems are often built in a modular way composed of standard components. Their behaviour may be well-known. Good, generally accepted models may already exist.

In Hibliz, hierarchical block diagrams are used to describe the model decomposition. Many demonstrations for people from university and industry indicate that the hierarchical block diagram is a natural and easily understood concept for describing model decomposition. Hierarchical block diagrams also make the model more concrete. The user can view the model as an object. The concept of seeing and pointing is important. For example, to inspect a model the user can just point at it and have it zoomed-in.

We recommend hierarchical block diagrams as a general tool for describing model structure. However, we would also like to stress that there are many open questions concerning their look and the means for creating and editing them. Should symbols (icons) be used to denote different parts instead of annotated boxes? How should keyboard and mouse be used when creating, editing and inspecting models? There is also a need for other complementary structuring concepts for organization of model libraries and to handle models of different complexities.

Continuous scrolling, panning and zooming demand fast graphics. It is not necessary to have these features to implement hierarchical block diagrams. The zooming up of a block can be implemented by creating a new window and displaying the block in this window.

Differential/Algebraic Systems

It is important that the user can give the mathematical description of a submodel on a natural form. Many models for physical systems are naturally given as sets of ordinary differential and algebraic equations (DAE). Furthermore, as motivated in Chapter 2, the equation form is the only reasonable for model libraries. Hibliz accepts mathematical descriptions given as DAE systems. However, most simulation packages of today do not allow models given as DAE systems, but require assignment statements for derivatives and algebraic variables.

When simulating, Hibliz has to solve a DAE system of the form

$$g(t, \dot{x}, x, v, p, c) = 0$$

where t is the time, x and v vectors of unknown variables, p a vector of known parameters and c a vector of known constants. The vector v contains

the unknown variables which do not appear differentiated in the equations whereas the components of x appear differentiated.

There are numerical solvers for DAE systems, but they are not as reliable and robust as numerical solvers for ordinary differential equations on state space form. Unfortunately, DAE systems may exhibit bad properties, which make them difficult to solve numerically today. In Mattsson (1986) it is illustrated that even small, simple DAE systems can exhibit bad properties. The examples are by no means pathological. They arise naturally during model development.

We would like to encourage research in this area by stating that it is of a vital interest that software for model development and simulation supports models given as DAE systems. There are many open questions to answer. The questions cover a broad range from theoretical to practical problems. Interesting issues are

1. When can we decide whether a problem is well-posed, i.e. having a unique solution? Can we put any not too restrictive demands on $g(t, \dot{x}, x, v, p, c)$, which can be checked, to guarantee that the problem is well-posed? For special applications the problem will be well-posed if special application dependent rules are followed.

2. In simple cases it is possible to solve $\dot{x}$ and v from $g(t, \dot{x}, x, v, p, c) = 0$ directly. In more complex cases differentiations of equations are needed. It turns out that then the numerical DAE solvers of today are also in trouble. There are then algebraic constraints on the components of the x-vector. Can the constraints be detected and eliminated by symbolic formula manipulation? They may put restrictions on how the starting vector $x(t_0)$ can be chosen. Can these restrictions be determined by the simulator?

3. Discontinuities cause also difficulties, especially if they depend on x and v. Can they be detected and can changes be flagged for the numerical solver? Discontinuities may imply that the solution could contain Dirac delta pulses. In many cases they are results of idealizations made by the model developer to make the model simpler. How can such be detected and eliminated for the numerical solver? Since the model developer often is aware of these effects, it may be of interest to introduce new language constructs so he can handle and eliminate Dirac delta pulses explicitly. You may take the stand point that idealizations implies that the model developer is not really interested in those variables containing Dirac delta pulses, so they should not appear in the model. The user should not be allowed to pose questions about their behaviour.

4. Besides dealing with algebraic constraints on the vector x and discontinuities, how could symbolic formula manipulation be used? An interesting question is "How should we partition the problem to make numerical solution more efficient and reliable?".

5. One possible way to reduce the complexity of the problem by formula manipulation is to eliminate components of the vector v from the vector of unknowns passed to the numerical solver. However, the elimination must be made with care not to destroy the structure or make the problem ill-posed. For example, division by zero must not be introduced.

Connection of submodels gives simple equations of the type $A = B$ which are simple to eliminate. When defining models it is also natural to introduce sequences of auxiliary variables to improve the readability. If we can find those sequences, it is easy to eliminate the associated variables from the vector of unknown passed to the numerical solver. It may, however, be an good idea to keep them as internal variables in the routine for calculating the residual. Note, also that when searching for such sequences of auxiliary variables it can be assumed that $\dot{x}$ and x are known since they are calculated by the numerical DAE solver. Unfortunately, there are no good algorithms available for this kind of problem, so it is an interesting research topic.

6. The man-machine interface is important. The system must be able to give clear error messages, explaining the errors in terms of the original equations not in transformed ones unrecognizable for the user.

   The user also probably has more information available about the problem than what he has stated. It may for example be very useful for the system to know that certain combinations of parameter values will never or should never be tried. It is desirable that the simulator could be able to pose such questions. When modelling, it is often not obvious what is critical.

7. How can the numerical DAE solvers themselves be improved?

3.2 Graphics

Hibliz uses graphics to describe the model. The graphical objects themselves as seen on the screen are simple 2 D objects in the form of lines, rectangles and characters. However, to perform continuous scrolling, panning and zooming, fast graphics is needed, not only to make it look nice but to make it easy for the user to scroll, pan and zoom. The user needs fast feedback. Slow graphics introduces phase lag and delay which makes it difficult for him to move with precision. Graphics capabilities of the power of an IRIS 2400 are actually needed to make continuous scrolling, panning and zooming attractive. As indicated above, it is probably not necessary to have continuous zooming and panning to implement hierarchical block diagrams. The zooming up of a block can be implemented by creating a new window and displaying the block in this window.

Implementation of Information Zooming

Information zooming means that the representation of a submodel depends on its size on the screen. Consequently, Hibliz needs to know the size of a submodel in pixels so it can draw the proper representation. Hibliz associates a normalized, local coordinate system to each submodel and the transformations into screen coordinates are done by hardware. However, there is no simple, direct way of finding out the size of an object in screen coordinates from a program using the IRIS Graphics Library. Ways to do it are described in Brück (1986).

This problem has an aspect that is of general interest, when designing graphics hardware to solve it. The basic difficulty of making an efficient implementation is that the graphics hardware must return a result until the

application program can continue. In normal drawing of a picture, the application program and the graphics processor can work rather independently. But to read information from the graphics processor, the two processors must synchronize, which leads to less parallelism and decreased throughput.

Graphics Standards

Graphics standards is an important issue. There is today one generally accepted graphics standard: GKS (Enderle et al. 1984, Hopgood et al. 1983). An extended and improved standard, PHIGS (SIS 1986, Shuey et al. 1986), is coming.

GKS is rather low-level, but all high-level operations in an integrated engineering environment can probably be implemented on-top of GKS. Regrettably, many of today's implementations are unacceptably slow; interactive "workstation-type" graphics cannot be based on GKS (today).

PHIGS promises to have valuable high-level features, in particular a hierarchical structure for graphics. It should also be noted that PHIGS is very strongly promoted by IBM.

One problem area (which may eventually solve itself) is the lack of experience of using GKS or PHIGS. We do not yet know what "programming style" best explores the strengths and features of e.g. GKS. Neither is the relationship to object-oriented programming understood.

Windowing

Another maybe more difficult problem is windowing. Standardization cannot be expected before 1989. One aspect is the interaction between window manager and graphics package. The windowing facilities are normally not accessible from the graphics packages, and user input is handled differently. It is extremely frustrating to have to use two vastly different ways of interaction simultaneously.

Another aspect is that a user does not like to learn and use several window managers. The advantages of having a standard window manager for all CACE programs are quite obvious and uncontroversial, but it should be noted that CACE programs are not the only use of a workstation. The user will use the vendor-supplied window manager, and would therefore prefer that one also in CACE programs. The same applies also for text editors.

3.3 Implementation Languages

In order to design an effective user interface, a very flexible and interactive environment is required. It should be possible to investigate the different possibilities offered by the hardware and software technology in a fast and convenient way. To evaluate various designs and ideas, they must be implemented. Consequently, it must be possible to do fast prototyping.

The experiences from this project are that Pascal is too cumbersome for prototyping. It takes too long time to implement an idea and the turn around time for compilation is too long. The the experiences from the projects "Experiments with Expert System Interfaces" (STU contract 85-3042) and "Expert Control" (STU contract 85-3084) indicate that Lisp offers a better environment.

Our current recommendation is to use Common Lisp together with a object-oriented package like Flavors as the high level implementation language during the design of a user interface.

3.4 Demonstrations and Presentations

Many demonstrations at the department for people from university and industry have been met with very positive response.

Hibliz was demonstrated for the participants of the 2nd IFAC Workshop on Adaptive Systems in Control and Signal Processing, 1-3 July, 1986, Lund, Sweden. The workshop had about 200 participants coming from all over the world.

For presentations outside the department, when we cannot give a live demonstration, we use a set of color slides and a video film. The color slides present Hibliz in a similar way as done in Chapter 2. The slides include color photos of the screen corresponding to the pictures in Chapter 2. Color slides have good resolution but cannot illustrate the dynamical aspects when scrolling, panning and zooming continuously. The video film which is seven minutes long illustrates these aspects, but the resolution is lower although still acceptable.

A presentation was given at the Joint SERC/STU Workshop on Graphical Front Ends for CACE, 14-18 July, 1986, UMIST, Manchester, U.K.

Hibliz was also presented at the IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design (CACSD), Arlington, Virginia, September 24-26, 1986. See Elmqvist and Mattsson (1986).

Acknowledgements

This project "New Forms of Man-Machine Interaction" has been a part of the project Computer Aided Control Engineering (CACE) at Lund Institute of Technology. We are grateful to the National Swedish Board of Technical Development (STU) who has supported this project under contract 84-5069.

The authors would like to thank Professor Karl Johan Åström for many useful discussions and for his strong support.

References

ADOBE SYSTEMS INCORPORATED (1985): *PostScript Language, Reference Manual*, Addison-Wesley Publishing Company.

BRÜCK, D.M. (1986): "Implementation of Graphics for Hibliz," Report CODEN: LUTFD2/(TFRT-7328). Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

CLARK, J.H. and T. DAVIS (1983): "Work station unites real-time graphics with Unix, Ethernet," *Electronics*, Oct. 20, 1983, pp. 113–119.

ELMQVIST, H. (1978): *A Structured Model Language for Large Continuous Systems*, Ph.D-thesis, Report CODEN: LUTFD2/(TFRT-1015). Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1985): *LICS - Language for Implementation of Control Systems*, Report CODEN: LUTFD2/(TFRT-3179). Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. and S.E. MATTSSON (1986): "A Simulator for Dynamical Systems Using Graphics and Equations for Modelling," Proceedings of the IEEE Control Systems Society Third Symposium on Computer-Aided Control System Design (CACSD), Arlington, Virginia, September 24-26, 1986.

ENDERLE, G., K. KANSY and G. PFAFF (1984): *Computer Graphics Programming (GKS—The Graphics Standard)*, Springer-Verlag.

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

Press.

JAMSHIDI M. and C.J. HERGET (Eds.) (1985): *Computer-Aided Control Systems Engineering*, North-Holland.

KNUTH, D.E. (1979): *TEX and Metafont - New Directions in Typesetting*, American Mathematical Society and Digital Press.

MACHOVER, C. and J.C. DILL (Eds.) (1985): "New Products," *IEEE Computer Graphics and Applications*, Vol. 6, No. 5, May 1986, 72.

MACHOVER, C. and W. MYERS (1984): "Interactive Computer Graphics," *Computer*, Vol. 17, No. 10, Oct 1984, pp. 145–161.

MATTSSON, S.E. (1986): *On Differential/Algebraic Systems*, Report CODEN: LUTFD2/(TFRT-7327). Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

PETZOLD, L.R. (1982): "A Description of DASSL: A Differential/Algebraic System Solver," Proceedings of IMACS World Congress, Montreal, Canada, 1982.

SHUEY, D., D. BAILEY and T.P. MORRISSEY (1986): "PHIGS: A Standard, Dynamic, Interactive Graphics Interface," *IEEE Computer Graphics and Applications*, Vol. 6, No. 8, August 1986, pp. 50–57.

SIS (1985): "Datorgrafi—PHIGS, Programmers Hierarchical Interactive Graphics Standard," Technical report no. 306, SIS—Standardiseringskommisionen i Sverige.