This is to certify that the

thesis entitled

A DESIGN ENVIRONMENT FOR THE GRAPHICAL REPRESENTATION OF HIERARCHICAL ENGINEERING MODELS

presented by

Michael Keith Hales

has been accepted towards fulfillment
of the requirements for

M.S. degree in **Mechanical Engineering**

RC Rosenberg
Major professor

Date Nov. 17, 1995

A DESIGN ENVIRONMENT FOR THE GRAPHICAL REPRESENTATION OF HIERARCHICAL ENGINEERING MODELS

By

Michael Keith Hales

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Mechanical Engineering

1995

ABSTRACT

A DESIGN ENVIRONMENT FOR THE GRAPHICAL REPRESENTATION OF HIERARCHICAL ENGINEERING MODELS

By

Michael Keith Hales

In response to growing demand for robust, flexible solutions, engineering modeling design efforts have become increasingly complex. This ever-increasing complexity necessitates modeling software which provides tools to effectively manage complex systems. Tools for building hierarchical models to support their proper (re)use must be created, thus reducing model development demands on engineers.

This thesis presents a design environment created to support component modeling based on the hierarchical representation of large systems as interacting subsystems. A prototype software package was produced, incorporating a graphical interface that provides simple, intuitive tools for the creation and manipulation of hierarchical models. A generalized multiport node was defined to ensure consistent implementation of model components in system design. The software design provides a strong foundation for future extensions. An example-based tutorial illustrates the features of the software.

To Julie

ACKNOWLEDGEMENTS

I would like to express my appreciation to my advising professor, Dr. Rosenberg, for his insight and guidance. His knowledge and experience helped me understand fundamental principles and concepts.

Dr. Somerton and Dr. Radcliffe deserve my gratitude for serving as members of my thesis committee and for providing useful suggestions and comments.

My friends and colleagues of the Computational Design Laboratory also provided valuable assistance in discussing with me issues that affected design and implementation issues.

My thanks go to my parents, Keith and LaRae Hales, and the rest of my family. My success has been built from the foundation of strength that they have given me.

Finally, my wife, Julie Schramm Hales, deserves great appreciation for the support and encouragement she has freely given me.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 Problem Definition

1.1.1 Challenges of System Design

Engineers today are confronted with increasingly diverse and complex design problems. This situation arises, in part, from the demands for higher quality products that perform a wider variety of tasks. It is not satisfactory to meet a limited set of functionality requirements. To be competitive in a global market, a broad spectrum of objectives must be met with high standards of durability, efficiency, reliability, and safety. In short, products are expected to do more for less.

The achievement of high quality, cost-effective products comes at the price of greater complexity and sophistication. Systems may involve interacting electrical, hydraulic, mechanical, and thermal components. Physical and mathematical understanding of each area is becoming a standard requirement for the successful engineer. Indeed, an emerging discipline, Mechatronics, was conceived to confront the challenges involved in the congruous interaction of various technologies. The International Federation for the Theory of Machines and Mechanisms adopted the following definition of this term:

"Mechatronics is the synergistic combination of precision mechanical engineering, electronic control and systems thinking in the design of products and manufacturing processes." [1]

As demands for quality increase, interdependencies between subsystems often develop. Consider the relative complexities of the conventional passive braking system compared to the anti-lock braking system popular today. Both systems were designed with the primary objective of decelerating a car. However, the action of the anti-lock braking system is much more complex, due in part to the *active* dependance on other subsystems such as the current vehicle speed, angular velocity of the wheel, and feedback control schemes. Thus the engineer must not only individually understand the dynamics of the vehicle, the dynamics of the wheel, the interaction of the tire with the road, the electronic measurement of system performance, and the feedback control scheme, but she must also understand how these subsystems interact with each other.

1.1.2 Design Tools

To help meet design objectives, engineers have often turned to modeling. A model is a simplified, abstracted representation of an actual system that aids in the analysis of a problem [2], [3]. Physical models have been extensively exploited in the past. The wind tunnel, in wide use today, is a scaled physical model that simulates the forces an airplane would encounter while in flight. A mathematical model is a precise description of physical behavior, such as a set of equations. A well-known mathematical model is Newton's

Second Law of Motion ($F=ma$) which describes the acceleration of a mass under the influence of an applied force. One aspect of both physical and mathematical models is that some details of the actual system are not considered. The wind tunnel model may or may not account for air temperature. An application of Newton's Second Law may or may not account for mass deformations. The level of detail included in a model is dependent on several factors including the desired accuracy of the solution, the relative significance of a particular effect, and the skill and insight of the person building the model. A proper model will have sufficient complexity to accurately predict system behavior [4].

Obtaining a mathematical model of a physical system is often a daunting task. The difficulty increases when many different energy domains are present, because classical methods apply only to parts of the system. The bond graph modeling technique has proven to be an effective, systematic method for structuring governing equations for systems built of multiple energy domains [2], [5]. Once the system equations are obtained, determining a solution is another difficult task. For many linear systems and most non-linear systems, it is not possible to determine an analytical solution to the governing equations. Numerical solution of the mathematical equations is often the only recourse. These concepts are generally implemented using computer simulations. A number of simulation software packages have been developed that use bond graphs for system modeling and numerical solution generation. ENPORT, CAMAS, and CAMBUS are examples of such software [6], [7], [8].

An increasing need for model information management tools develops as systems

grow in complexity. One effective tool in the system design process is macro representation. Macro representation is built on the premise that a complex system can be broken down into a hierarchical structure of less-complex, interacting subsystems. Each of the subsystems can, in turn, be considered a system that is composed of subsystems. This procedure continues until a subsystem becomes simple enough to understand or describe with well-known modeling tools.

Consider the objective of describing the motion of a car. Figure 1 shows a macro representation of a car and a subsequent division into subsystems. At the most general level the model can simply be represented by the label "Car". While the model of the car on this level is very simple, only a limited amount of knowledge is available about the car's motion. If the car was treated as a rigid body, the translational and rotational characteristics could be determined. However, it is impossible to determine the detailed motion of specific parts of the car, such as the right front fender.

The basic car model can be divided into subsystems, namely a rigid chassis and four quarter-car models. The input to the system on this level is given the generic label "Ground" . Interaction between the submodels occurs between submodel ports and is indicated by an arrow from one submodel to another. The direction of the arrows indicates positive power flow. Notice that the rigid chassis model is affected by each of the quarter-car models and each quarter-car model is affected by the movement of the chassis and the ground. This level of abstraction aids in the understanding of the system

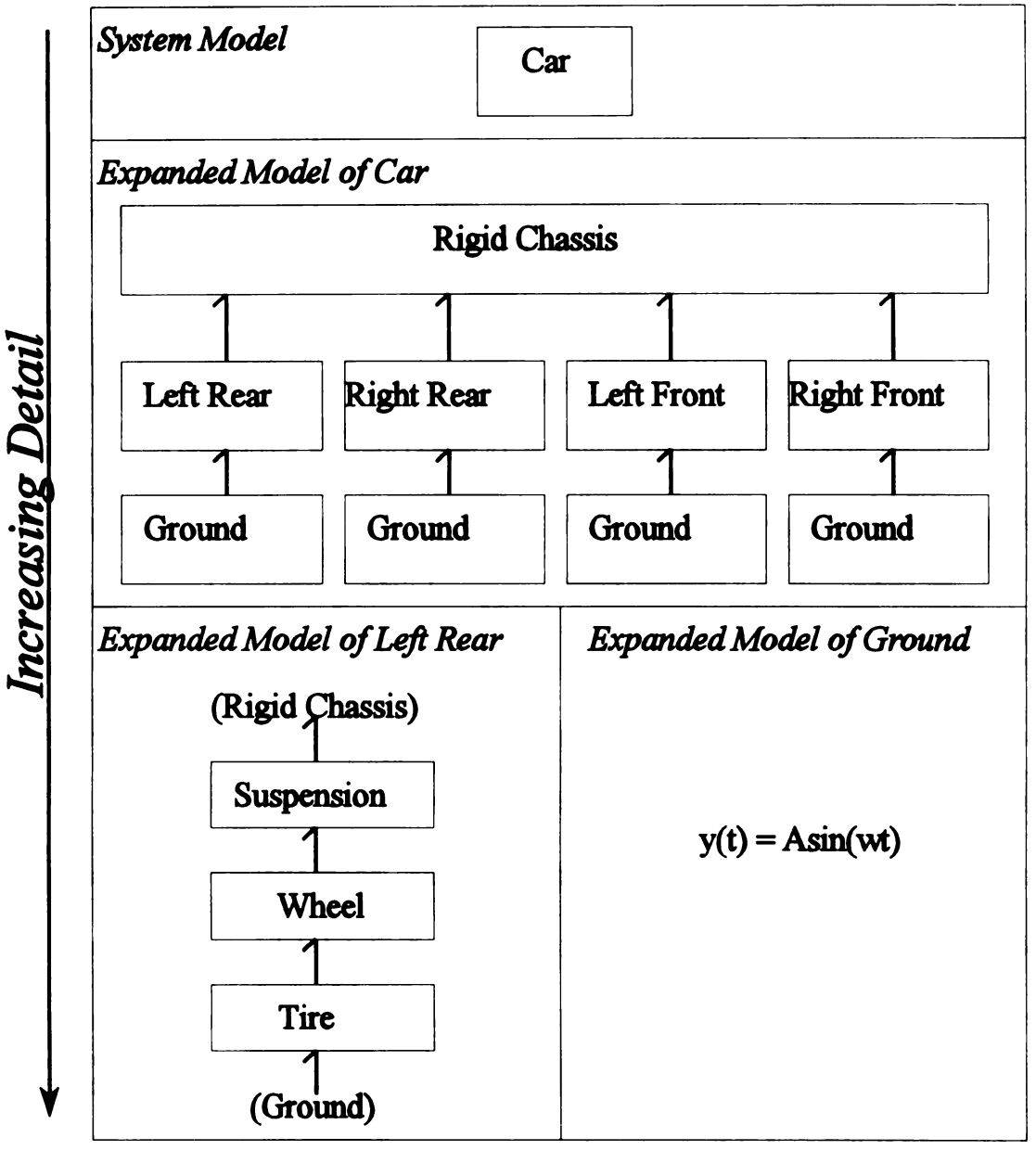


Figure 1. A Macro Representation of a Car

as a whole. For example, each of the quarter-car models is affected by all of the other quarter-car models. This valuable system information is obvious since the system model detail is limited. However, at this level of detail, it is still not possible to obtain specific information about the car's motion.

One of the quarter-car models can now broken down into three subsystems: the suspension, the wheel, and the tire. In addition to the interaction between subsystems on this level (arrows between subsystems), the interaction from systems outside this model are shown as inputs and outputs (arrows not from or to another system). Specifically, the interaction between the tire and the road and the interaction between the chassis and the suspension are shown as inputs and outputs. At this level, the source and destination of the inputs and outputs are not important to the model; the useful information is simply that an input and output port exist. At this point, equations describing the car's motion still can not be obtained. The systematic breakdown of systems into subsystems can continue until a concrete model can be obtained. For example, a simple representation of the road could be modeled with the function $Y(t) = A\sin(\omega t)$. This mathematical model is completely defined and yields direct numerical results that can be used in determining the motion of the car.

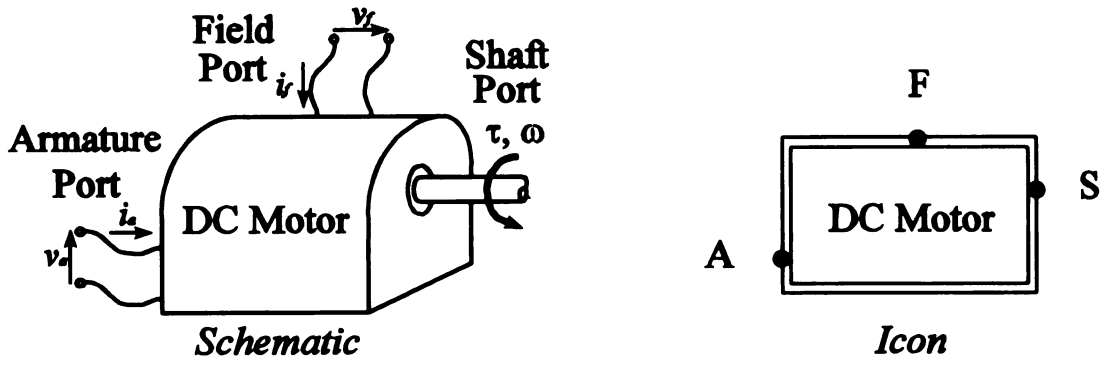
This modeling procedure produces a model that closely approximates the structure of the physical system. This is helpful in designing system models. However, some engineering judgement is required to justify inconsistencies between the physical system and the model. For example, it is convenient to represent the ground as separate inputs to

the systems even though the ground is a continuous surface. If the ground surface is asphalt, this modeling decision may be justified. If the ground surface is soft dirt, then the ground's shape may be affected by the tires and the assumption is not justified.

Another benefit obtained from using a macro representation of a model is the ability to use and reuse component models. In the example above, substantial effort could be exerted in generating a general quarter-car model. If done correctly, the same model could be used four times for each part of the car without having to repeat the initial work of building and verifying the model. If the model takes the form of a FORTRAN subroutine, using the model is simply a matter of calling the routine with the correct argument list.

Additional power of component models comes from the possibility of using a model constructed by someone else without explicit knowledge of model details. The difficulty of using predefined "black box" models is ensuring their correct use. Consider three representations of a "black box" model of a DC Motor shown in Figure 2. There are three power ports shown on this model, corresponding to the field, armature and shaft. If this model is to be used correctly, several issues, including the following, must be addressed:

1) *What information is expected at each power port?* Each port requires very specific information. Providing incorrect information would invalidate the model. For example, confusing the armature and field ports could result if the ports are not explicitly labeled.



Subroutine DCMOTOR(va,ia,vf,if,t,w)
Subroutine

Figure 2. Three Black Box DC Motor Models

2) *What is the input variable at each power port?* If the armature port requires voltage as input then the user of the model must comply. Further, that requirement should be explicitly understood without knowledge of the DC motor model. The name of the model implies that the shaft port will provide a torque as an output. However, DC Motors can be run in reverse, to function as generators. The model of the DC motor may or may not account for this physical effect.

3) *What are the reference power directions?* The reference power directions are shown in the schematic model with arrows. This information must be properly communicated in order to ensure proper model use.

4) *What limiting assumptions were made when the model was constructed?* If a model does not specify assumptions, then a second party may use the model in a way that would violate the assumptions made when it was built.

In summary, the proper (re)use of a component model depends on knowledge of the required interface characteristics and model limitation assumptions [9].

1.1.3 Engineering Design Software

A design engineer faced with large, complex problems often turns to a computer simulation package. Computer programs have long since migrated away from pure textual descriptions, such FORTRAN and C subroutines, to graphical representations. Graphical models provide an intuitive description of a system and better emulate the physical entities they represent. A set of differential equations that represent the motion of a car suspension is a textual model. A diagram of the parts of the car, from which equations of motion can be generated, is a graphical model. Graphical models have proven useful because conceptualization of the model parallels conceptualization of the physical system.

The usefulness of graphical models has spurred the growth of software with graphical user interface tools for engineering design. Engineers have come to expect software packages equipped with simple, intuitive tools for model generation, representation, and manipulation. Model elements become "objects" upon which operations can be performed. Objects in a computer program are analogous to objects in the real world. They can be *selected* and their *attributes* examined or modified. The parallels between the graphical modelling and graphical interface tools help the designer transfer abstract ideas into concrete realities.

The ideas associated with a graphical interface have long been familiar to users of the Apple Macintosh computer system and have more recently become popular among PC users with the increasing use of Microsoft Windows. The mouse is used as a tool to point at objects such as files on a disk, insertion points in a word processor, and items on a menu. Pressing the mouse button indicates that the object pointed to is to be selected. Depending on the selected object type, subsequent actions with the mouse and/or the keyboard define the operation to perform on the object. Hitting the delete key after selecting a file instructs the operating software that it should be deleted from the disk. Selecting the insertion point in a word processor instructs the program to move the cursor to that point. Selecting an item on a menu may cause another menu to drop down with a new list of choices.

Programming tools and philosophies have changed in concert with the demands on software performance. One widely used approach is object-oriented programming. On an abstract level, object-oriented programming is a set of rules, principles, practices and procedures used to create a program to achieve a given objective [10]. Although specific definitions of object-oriented programming vary, there are some concepts that are generally accepted. One of these concepts is the focus on system *objects* and the *procedures* performed on them. A programming object combines both data structure and behavior in a single entity [11]. This is opposed to traditional programming views where the focus is in some way reversed; procedures are developed that manipulate data, and data structure and behavior are only loosely connected. Properly implemented object-oriented ideas produces more robust, readable, and reusable code.

1.2 Research Objectives

The principal objective of this research is to develop a design environment providing graphical tools for the macro building, representation, and manipulation of models based on Bond Graph and Block Diagram elements. The object-oriented programming philosophy will be adopted. The completed software will lay a foundation for which future enhancements can easily be implemented. The new design approach is to be integrated into the existing ENPORT environment. ENPORT provides key tools and algorithms, reducing development overhead. Specific implementation is a DOS-based program written in a hybrid of the FORTRAN, C, and ASSEMBLY programming languages.

To meet the stated objectives the project was divided into four phases, as follows.

1) *Develop a hierarchial database and database tools.* The data storage scheme must support the hierarchical display characteristics. Also, the data should be stored to optimize various programming procedures. For example, screen refreshing should be rapid to minimize blinking effects; however, deleting an object from the database does not require the same speed. As with any database, tools must be developed to create, modify, access and remove objects.
2) *Develop a Graphical User Interface (GUI).* Extensive use of the mouse should be incorporated so that users can select objects and perform operations on them. Whenever possible, information should be represented graphically.

3) *Explicitly define a subsystem's ports and port attributes.* By defining the attributes of node ports, it becomes possible to help insure proper use of subsystems as black boxes (i.e. as locally defined entities).

4) *Develop Macro representation, display, and interaction tools.* Development of these tools was the main objective of this work. Successful implementation of this feature depends on the completion of the first three phases.

Throughout the project, two objectives were kept in mind. First, object-oriented programming techniques were to be used as much as possible. Second, the developed tools should be written to simplify future software enhancements.

1.3 Description of Thesis Organization

This thesis is presented in four chapters. Chapter 1 has presented some challenges in the modeling of large systems and some concepts useful in dealing with them. The objectives of this work in addressing the concerns associated with large models are defined. Chapter 2 gives an overview of the work which was accomplished. Terms and concepts used in defining and solving the problem are presented. The philosophy and perspective of the approach are discussed. Chapter 3 presents an in-depth tutorial for the use of the prototype program. The tutorial can be used in conjunction with the developed software. Chapter 4 provides a summary of the results obtained and recommends areas for further research. Appendix A is a technical listing of programming objects and their definition. Source code for database tools given in Appendix B. Additional Source code,

used for key features is presented in Appendix C. The thesis concludes with a list of references which aided in the completion of this research.

Chapter 2

Software Development and Design

2.1 Design Approach

Developing the tools for a design environment that allows for graphical creation and edition of a hierarchical system is no small task. In order to facilitate successful completion of the project goals, the system was divided into four phases: database development, Graphical User Interface (GUI) development, port attribute definition, and macro representation and manipulation tools development. The program language used for a majority of this work was FORTRAN. Code written in C and ASSEMBLY was written to accomplish tasks that were not available from FORTRAN. Whenever possible, the concepts of object-oriented programming were employed. However, the ANSI standard of the FORTRAN language is not structured for full implementation of object-oriented programming concepts. The compiler used (Microsoft FORTRAN 5.1) provided additional language extensions to facilitate object-oriented programming, but the supply of tools was limited. This limitation of FORTRAN proved to be a major obstacle in using structured object-oriented techniques.

The first part of this chapter introduces some key terms and concepts used in the software design and implementation. The second part of this chapter explains some of the

challenges encountered and decisions made in completing each phase of this project.

2.2 Key Concepts

A fundamental construct in object-oriented programming is the *object class*. "An *object class* describes a group of objects with similar properties (attributes), common behavior (operations), common relationships to other objects, and common semantics."[11] The object class is the general set of properties that make an object unique. *People*, *mechanical engineer*, and *node* are examples of object classes. A specific item that belongs to a class is referred to as an *object instance*. *George Washington* is an object instance of the object class *People*. Philosophically, an object instance is unique simply by its existence. Object attributes do not define uniqueness; i.e. two object instances may have identical attributes. Unfortunately, the general usage of the terms object and class is not always consistent. The particular meaning can usually be discerned from the context.

To be functional in a computer program, the distinction of two objects is made by associating a unique identification (ID) to each object [11]. The object ID is not one of the attributes of the class but rather a tool used to locate the object. For example, consider the object class *names*. If a list of specific names (object instances) was created, the information could be stored in an array. Each instance is made distinct by its location in the array. The array index could be used as the object instance ID, but the array index value is not an attribute of the class *names*.

In order to understand the structure of the software and the design decisions made during development, a partial list of class descriptions will be given. A complete list of the class objects used in the program can be found in Appendix A. Due to the complex nature of these objects, the description of some graph objects depends on the description of other graph objects. Therefore, the descriptions below are somewhat recursive. As the various objects are defined, it will be helpful to make frequent reference to Figure 3.

Subgraph - A subgraph object can contain other graph objects such as nodes and connectors. Subsystems of a model are defined by the contents of a subgraph. A (parent) subgraph may spawn other (child) subgraphs. A child subgraph is said to be contained in the parent subgraph. This concept is also referred to as nesting. Some attributes of a subgraph are the number of nodes contained in the graph and a text description of the submodel.

System Graph - This object class is a special type of subgraph. All the attributes of a subgraph are inherited by a system graph. A system graph represents the entire system model. There is one system graph for each model. The system graph contains exactly one node: a Macro Node. Some attributes of a system graph are its file name, the total number of subgraphs, and the number of levels.

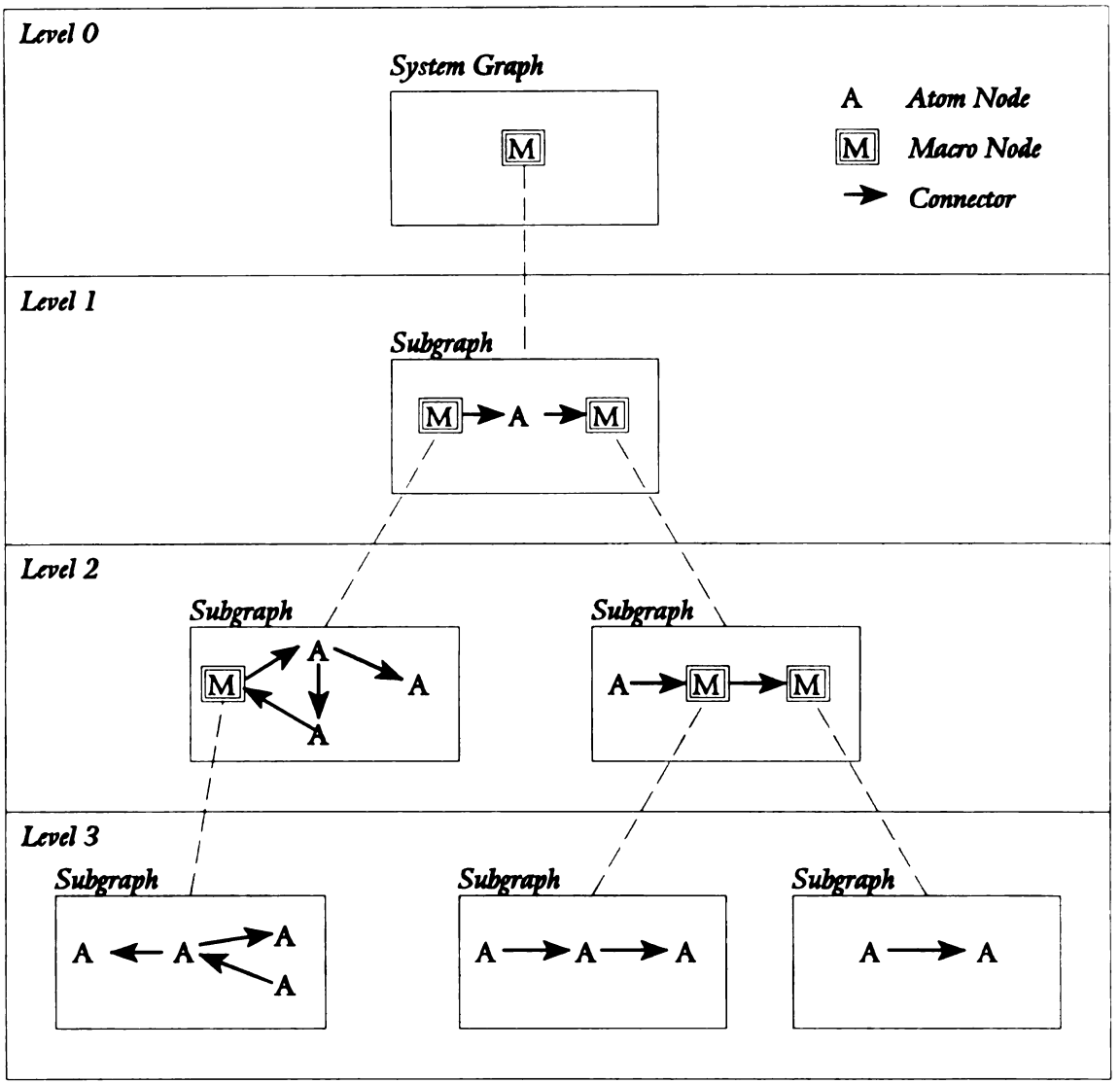


Figure 3. Objects in a Hierarchical Graph Structure

Level - A level is defined by the amount of subgraph nesting. Each level contains subgraphs that have the same amount of nesting. Thus the first subgraph created, the system graph, has no nesting and is contained in Level 0. When a new subgraph is created from the first subgraph, it is placed in the level directly beneath the current Level. An attribute of a level is the number of subgraphs it contains. Note that only one subgraph may be contained in Level 0 and Level 1.

Node - A node represents a model of a physical component, an effect, or a system. Some attributes of a node are its label, its location on the screen, its icon, and its ports. The node class contains two subclasses: Atom Node and Macro Node. A brief description of these node classes follows.

Atom - An Atom Node is the fundamental building block of a model. It is the only node that contains *explicit* information used in determining governing equations. In general, the governing equations of an Atom Node may be represented by equations (e.g. y = sin(t)), a FORTRAN or C subroutine, or an implied set of dynamic equations (e.g. a transfer function) [14].

Macro - A Macro Node is a single node representation of a subgraph. The Macro Node is referred to as a parent to the subgraph, which is called its child. There are no equations directly associated with a Macro Node. The Macro Node serves as a place holder and helps manage complex model information.

Port - A port is always associated with a node; i.e. a port cannot exist without a Node. The ports on a node indicate locations where information is supplied to the node. Ports on a node are analogous to the formal argument list variables in a computer program subroutine. The port passes the information from the connector to the Node and/or vice versa. A port on a Macro Node passes the information to or receives information from a port contained in the child subgraph. The governing equations for a system graph cannot be determined until the ports on all nodes are connected. Some attributes of a port are its power regime and its input/output characteristic.

Connector - A connector is a directional line between two ports. Each end of a connector is associated with exactly one port. Conversely, a port is associated with exactly one end of a connector. A connector indicates that there is some interaction between nodes and represents variables in the system. Some attributes of a connector include its label, type, and color.

2.3 Database Structure

2.3.1 General Features

One of the principles of object-oriented programming requires a separation of program functions that act on objects from the storage of the information associated with the object. In practice, this means that the manner in which the main body of the program manipulates data must be unaffected by the manner in which data is stored. Additionally,

data should not be thought of as information stacked in an array, even if this is the way in which the data is stored. Data is always associated with a specific object and access to the data is made available by the object ID. For a dynamic database, four general database interface functions are required:

Add - Create an instance of an object class and store initial attribute values,

Set - Assign an attribute value to an existing class instance,

Get - Retrieve an attribute value from an existing class instance, and

Delete - Remove an instance from the database.

These interface routines are "taught" how to access and manipulate objects in the database. Whenever the database is accessed, it is accessed through one of these routines.

The two major drawbacks of this arrangement are that data access is slowed and the programming overhead is increased. The payback comes in the main program where

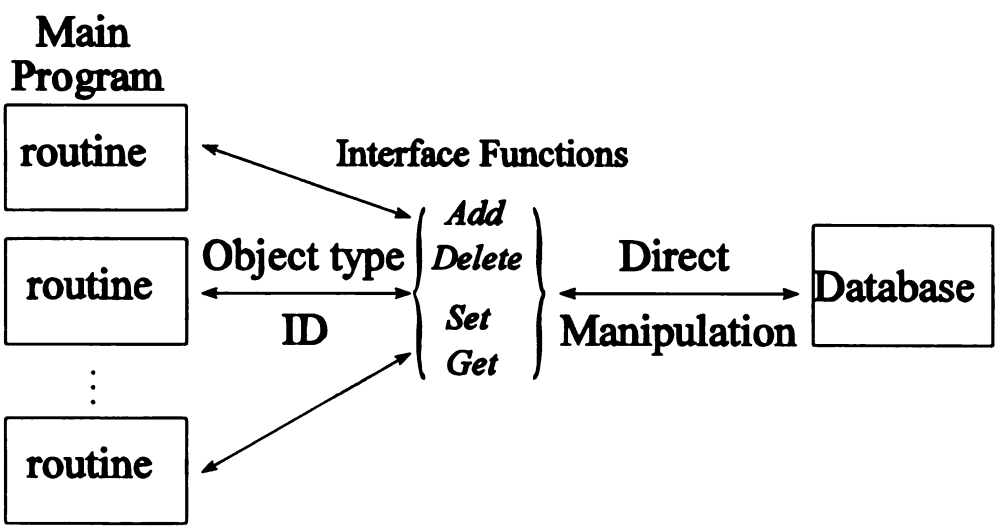


Figure 4. Separation of the Main Program from the Database

access to data is very explicit, leading to clearer code. In addition, the procedures used by the interface functions or the database structure itself can be changed without affecting the main program code. In the long run, the code is self-contained and enhancements are easier to install. Figure 4 is a graphical representation of these concepts.

2.3.2. The Object Instance Database

Two different databases were established in the prototype program: an object instance database and an object template database. Both of these databases are associated with the defined object classes (i.e. node, connector, ...) . The attribute database with its accompanying interface functions was completed for the object classes described. The template database was only established for the object class *node* and the interface tool *get*. However, the concepts were developed and the foundation laid for future completion of this task. The purpose of these two databases is explained below.

The object instance database stores the attribute information associated with an object instance. When an object instance is created, a unique identification (ID) is also created and is associated with this object. Figure 5 displays instances of the object classes level, subgraph, and node and their associated IDs. The object class *level* is identified with a single integer, the instance number. The ID of the first instance of the class *level* is 0. *Subgraphs* are identified using a data structure of two integers. The first integer is the ID of the level where the subgraph exists; the second integer is the instance number of the subgraph within that level. The first subgraph created in Level 2 has the ID 2.1.

Objects within a subgraph are identified with three integers. The first two numbers are the ID of the subgraph where the object exists and the third number is the object instance number within the subgraph. The second node in the second subgraph in the second level has the ID 2.2.2.

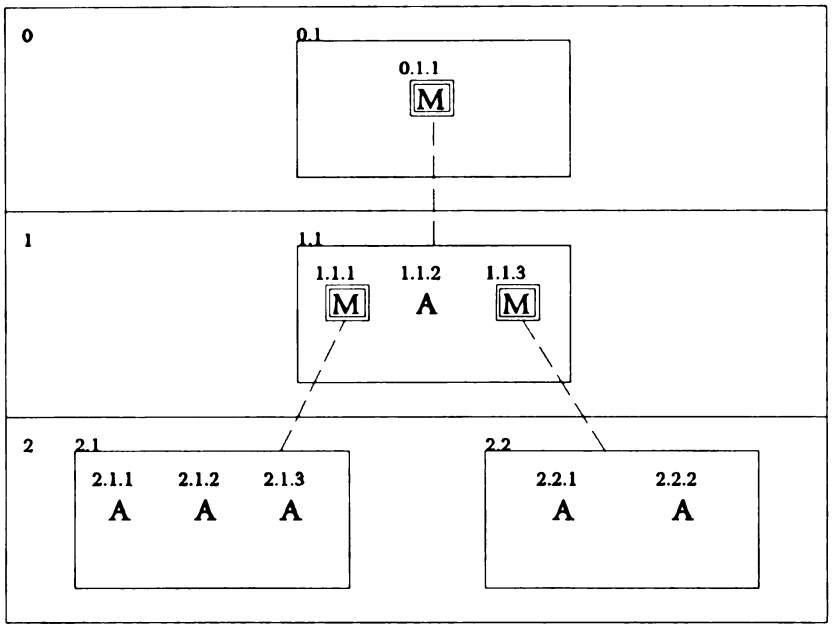


Figure 5. Object Identification System

This numbering system was used to store attribute information in sorted arrays. The location of an object attribute in an array is calculated using the ID. Storing information in this manner requires more time to perform the *add* and *delete* operations, but the *set* and *get* tasks are quicker. Furthermore, the time taken to reach any item in the list is independent of its position in the list; i.e., data is stored in a random access data structure [12]. This scheme was chosen so that operations which are time critical, such as refreshing the screen, are performed quicker. Note that the ID of an object instance

changes as other instances are added and deleted. This is not a problem since an ID is not an object attribute.

In order to ensure correct implementation of these ideas in a programming environment which did not directly support them (i.e. FORTRAN), a relatively small database program was developed. This limited database was text based. The storage, access, and deleting tools were developed for a general object class. The object class Node, with only two attributes, was used to test the database interface tools. The database tools were modified for the prototype system. Source code for the database tools can be found in Appendix B.

2.3.3 The Object Template Database

An object template is a set of rules or limits that define the legal behavior and valid attributes of an object class. Consider the attribute *age* of the object class *people*. The object template for this class stipulates that an object instance must specify positive values. A upper bound may also be assigned which is somewhat artificial, because the actual upper bound on the attribute *age* is unknown. Attempts to assign values outside of the template definition are not allowed. The object template would also be able to identify valid operations. The operation to increase the age attribute as a function of time is valid, but the opposite operation is not.

The object template is a very powerful tool. To illustrate its power some template

ideas were implemented for the class *node*. Different node *types* were defined by specifying a set of rules and limits. This information is stored in the template database. In the prototype software, a set of Bond Graph node types, a set of Block Diagram node types, and the Macro Node type were defined. Whenever a decision was to be made about a legal attribute assignment or valid node operation, the template database was queried.

As an example, consider the operation of adding a port to a node. The Bond Graph node type *Source of Effort* (Se) is defined as having exactly one port. (An applied force or a voltage supply are sources of effort.) The operation to add a port to this node type is not valid. The bond graph node type *Capacitor* (C) may have any number of ports (greater than zero). The operation to add a port to this node type is valid. The template database stores information about the minimum and maximum number of ports allowed for each of the node types.

The *add port* subroutine can handle this restriction in several ways. One way to deal with the issue is to predefine the node types Se and C. An IF THEN ELSE or SELECT CASE construct could determine if the operation is legal. However, the construct becomes larger as new node types are defined. Even more problematic, addition of a new node type requires modification of the code wherever logic based on predefined node types is used. A better way to handle this issue is for the subroutine to query the template database. The subroutine passes the object *type* to the template database through the interface routine. The database returns information about the object

which is used to determine if the operation is valid. This method not only simplifies the code, but localizes future code enhancements and changes. If a new node type is to be defined, then the template database is the only structure that requires modification. The difference in computer code between a program without and with a template database is illustrated in Figure 6.

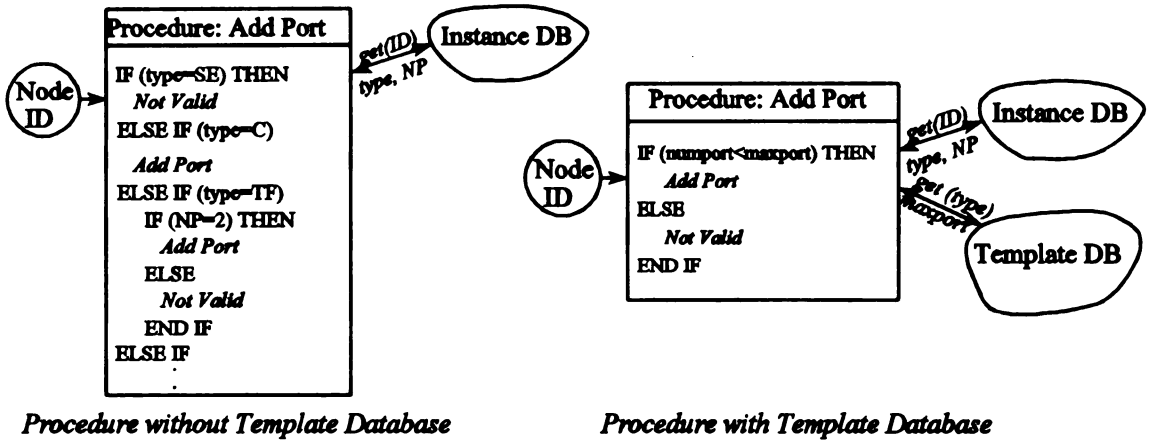


Figure 6. Comparison of Code Without and With the Template Database

The present work was intended to demonstrate feasibility of certain design principles. A project of great interest is to complete the design and implement these concepts. The current template database only allows for queries of attributes (the *get* interface function). Additional node types must be defined by modifying the code. A full implementation of the database would provide the *add*, *set*, and *delete* interface functions. Several violations of the design in respect to the template database currently exist; e.g. explicit reference to node types is made. These incongruities arise because a complete set of node template definitions have not yet been established.

2.4 Graphical User Interface

One of the goals of this project was to simplify user interaction with the modeling software through development of a Graphical User Interface. A recent survey among users of simulation software indicated that the most important feature of such software is a friendly interface with program interaction taking place mainly with a mouse [13]. In the developed software, interaction with objects stored in the database is made, as much as possible, with a mouse, object instance details and information are graphically conveyed, and results of operations are graphically displayed. Objects are represented on the screen with *icons*. An icon is a graphical picture associated with an object class. Identification of an instance in the database is made through its unique ID. Identification of a graph object in the GUI is made unique through its screen location. Therefore, any two objects of the same class that exist in the same subgraph must have unique graphical coordinates.

One design philosophy of the GUI was to present the user with only valid options. It was felt that less confusion would result from this design. The alternative is to present a full list of options and signal the user when an invalid operation is attempted. Consider again the *add port* operation for a node. When a node is selected and the edit button pressed, a menu appears that displays the valid operations for that particular node type. The edit menu is different for the various node types. A menu associated with Se (source of effort) node does not present the option of adding a power port. A menu associated with a C (capacitance) node does present this option.

During the model building process, the GUI operates in one of three modes. Selection of the current mode is made by pressing a button on a tool bar at the top of the screen. The button associated with current mode is highlighted and a status message at the bottom of the screen informs the user which mode the program is in. Instructional messages also appear at the bottom of the screen to assist the user in understanding the available choices and expected inputs. A description of each of the modes follows.

Add Node mode - A list of the available node types is printed on the left of the screen. One of the types is highlighted. Clicking the mouse in the space to the right of the list causes the highlighted node type to be added to the database. The node is defined with a default setting. If desired, an option can be chosen that will prompt the user for a node label. Otherwise, the label field is left blank.

Add Connector mode - There are two connector *types*, defined by the type of information carried by the connector: *Power* and *Signal*. The following rules apply to connectors:

1) Connections are always made between two ports. The ports must be on separate nodes.
2) A connection must be between ports of the same *type*. Port have the same distinction in *type* as connectors.
3) A connector's *type* is derived from its associated ports' *type*. Consequently, the connector type does not need to be chosen.
4) A connector's direction is explicitly understood by the input/output characteristic of the ports to which it is attached.

A connector is created by successively clicking the mouse on two ports. When the first port is chosen, it is highlighted in red. After selection of the first port, interior points to the connector can be created by clicking the mouse anywhere in the graph. An incomplete connector is drawn as a dotted line until the second port is chosen.

Select Object mode - Once objects are created, then various operations can be performed on them. Clicking the mouse on a object causes the object to be highlighted in red. Two additional buttons on the tool bar at the top of the screen appear for object operations. Currently, the only objects that can be selected are nodes, ports, and connectors. Additional objects, such as labels and interior points, could be made selectable with some effort. There are three general operations that can be performed on an object: move, edit, and delete. To move an object, the mouse button is held down and the object is dragged around the screen. To edit an object, the object is selected, and the edit button is pressed. Pressing the edit button while an object is selected causes a menu to be presented that lists various edit options for that particular object. Some edit options for a node include *edit attributes*, *edit equations*, and *add port*. To delete an object, the delete button, delete key, or backspace key is pressed after an object is selected.

Other features of the GUI include tools for viewing the work space (e.g. tools for moving all object in the screen left, right, up, or down). In the top right corner of the screen is the icon representing the current parent Macro Node that represents the displayed subgraph.

To aid in the building of large models, a graph checking utility has been provided. This tool can be accessed by clicking the Check Graph button. This button is represented with a check mark. The options in this menu are: *Check Setup*, *Check Causality*, *Check Equations*, and *Sort Equations*. The menu items are listed in order from the most general to the most specific. A successful check of any item in the list depends on the successful check of all the previous items. Once an item has been successfully verified, a check mark will appear next to it in the menu.

The *Check Setup* option determines if the layout of nodes and connectors is topologically complete. In practice, a graph is topologically complete if every port on a node has been connected. No analysis is performed at this time on the feasibility of the model. A poor model that would never lead to solvable equations could have correct topology.

The *Check Causality* option attempts to assign the bond graph property of causality to the system graph. Causality defines the input/output relationship at a power port. For a more complete explanation of bond graph causality, see [2] and [5]. The algorithm for assigning graph causality was previously developed [6].

The *Check Equations* option determines if the user has defined a valid set of equations for the system. Macro Nodes are not directly associated with equations. Some node types such as the 0-Junction, 1-Junction, Sink, and Distributor Nodes do not have user-defined equation. The equations for these node types are defined by their use in the

graph. Other node types, such as the Inertia and Capacitance Nodes have a combination of user-defined and system-defined equations.

The *Sort Equations* option produces the information that is required to numerically solve the equations associated with the graph. The sorting procedure will determine if an infeasible graph has been designed, such as a graph with not enough constraints. This algorithm was previously developed [6]. Equations are defined in terms of the names of the ports on the node. Port names are locally defined; i.e. two different nodes may ports with identical names. The sorting of equations depends on a unique labeling system. Rather than burden the user with defining a set of unique variable names, an algorithm was developed which generates a set of unique, global variables. This algorithm is transparent to the user.

2.5 Node Ports

The ports on a node are an explicit representation of information being passed to and from the node. Ports on a node are typed as [Power or Signal] and [Input or Output]. Power ports are drawn as circles and signal ports are drawn as squares. Input ports are drawn as an outline (where information needs to be placed) and output ports are drawn filled (where information has been placed). Ports can be specified to be of a particular power regime, i.e. mechanical, electrical, thermal, etc. Different power regimes are represented with different colors. Power ports carry two variables that, when multiplied, represent the power. For example, electrical power connectors carry the variable pair

voltage and current and mechanical translation power connectors carry the pair force and velocity. Bond Graph nodes use one of the variables on a power connector as an input and the other variable as an output. The input/output characteristic of a port is termed the port causality. An attribute of a port can be specified which defines required causality (in the form of the output variable).

The attributes of a port explained above assist in the building of consistent models. Connection of ports representing different power regimes is prohibited, as is connection of ports with the same input/output attribute. Currently, the causality attribute is not exploited. Graph causality is assigned using standard Bond Graph rules [2],[5]. In the future, the port causality attribute should be used in the causality assignment process. This would further promote the implementation of reusable models.

2.6 Macro Representation and Manipulation Tools

When a Macro Node is created, a child subgraph is also created. The Macro Node initially has no ports and the child subgraph is empty. For a Macro Node to be functional, it must have at least one port. If a model of a DC Motor was being built , then three power ports would be added: one port each for the armature, the field, and the shaft.

To build a subgraph model represented by a Macro Node, the Macro Node must be opened. This is one of the options in the edit menu for a Macro Node. Opening the Macro Node causes its child subgraph to be displayed. The icon of the Macro Node just

opened, with its accompanying ports, is displayed in the top right corner of the screen. A model is built in this subgraph the same way a model is built in any subgraph.

To connect a model in a child subgraph to the model in the parent subgraph, an association must be made between the ports on the Parent Macro and a Port in the child subgraph. This association is made while in the *add connector* mode. The mouse is clicked on a Port on the Parent Macro Icon and then on an empty Port in the child subgraph. The Icon for the Parent Macro Node is drawn in gray before all the required association are made, and drawn in bright white when all its ports have been associated to Ports in the child subgraph. When editing of a subgraph in Level 2 or greater is completed, a button at the top of the screen labeled ".." (similar to the DOS directory command "cd .." to move up one directory) is pressed to move to the parent subgraph.

From the user's point of view, a Macro node can be deleted in the same manner as any other node, even if the Macro Node is not empty. Functionally, however, the database deletion tool requires that a Macro Node be empty in order for it to be deleted. An algorithm was developed that systematically removes all graph objects from a Macro Node before deleting it. Since a Macro Node may itself contain other Macro Nodes, this algorithm is somewhat complex.

Chapter 3

Using the Macro Model Building Software

3.1 Software Overview

To a large degree, the successful implementation of the design discussed in Chapter 2 can only be evaluated by using the software. A tutorial is an effective way to illustrate the capabilities of the software package and demonstrate successful implementation of the design. A prototype software package was developed that would provide tools to deal with the issues raised in Chapter 1 using the design from Chapter 2. A copy of the software is available from the author or from Michigan State University. In this chapter the modeling tools provided by the software will be demonstrated through a tutorial. Primary focus will be placed on how to use the available tools.

Consider the schematic of a radar pedestal positioning system shown in Figure 7 [2]. The output of interest in the system is the angular position of the pedestal. An input command signal is compared to the output signal producing an error signal. The error is conditioned by a controller to produce an input to the plant. The control signal is input to the field port of a DC Motor. The voltage supply to the armature port is constant. The motor generates a torque which, after stepping through some gears, drives the pedestal. The system can be modeled with various levels of detail depending on the desired level of

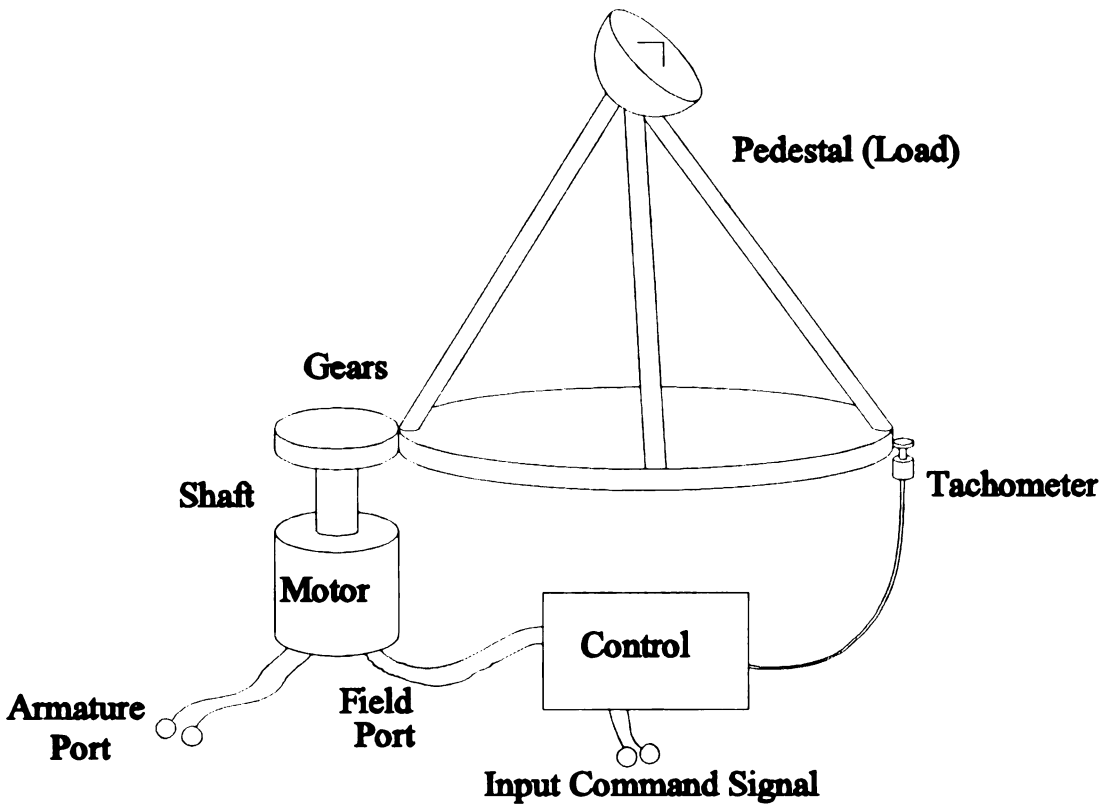


Figure 7. Schematic of Radar Pedestal Positioning System

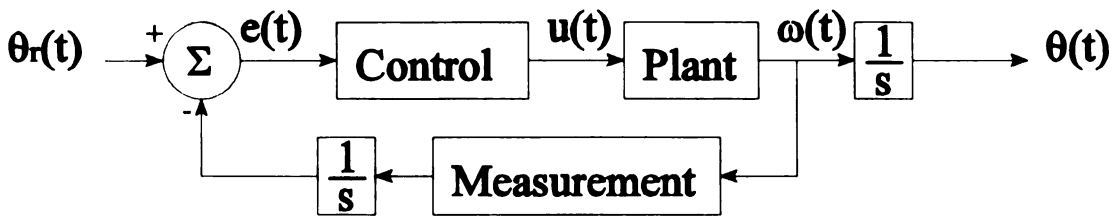


Figure 8. Macro Representation of a Radar Pedestal Positioning System

accuracy and the effects being investigated.

The radar pedestal positioning system can be divided into three components or subsystems: the control, the plant (motor, shaft, gears, pedestal), and the measurement. The macro representation of these subsystems, shown in Figure 8, should be familiar.

This representation clearly illustrates how the subsystems interact. The details of each of these subsystems could also be divided. For example, the plant could be divided into the subcomponents of the motor, shaft, gears, and pedestal. Clearly, if the models for each of these systems is simple enough, then it is possible to model the whole system without defining explicit subsystems. In the tutorial that follows, a model will be built of the radar pedestal positioning system. The system will be divided into subsystems as shown in Figure 8. The Plant will be divided into four submodels. A simple model of the motor will be shown. The purpose of the extensive submodeling is to demonstrate the features of the software.

3.2 A Software Tutorial

The macro model building software is a DOS program. It must be run from the DOS command line. Typing the name of the program, MB, causes it to execute. The main screen displays a top line menu. The left mouse button is used to select a menu item. Alternatively, the arrow keys on the keyboard can be used to position the menu pointer on the desired selection. Hitting return selects the item. Under the top menu option *Graph*,

the *Edit* option is listed. Selecting *Edit* brings up the edit screen and allows for changes to be made to the graph.

3.2.1 General Graph Editing

The edit screen operates in one of three modes: *Add Node*, *Add Connector*, and *Select Object*. The current mode is selected by clicking a button on the top of the screen that represents that mode. Pressing the tab key also changes the current mode. The current mode is indicated by highlighting its button and displaying a message at the bottom of the screen. The default mode is *Add Node*. A scroll list on the left of the screen displays a list of nodes that have been predefined. A basic set of Bond Graph, Block Diagram, and a Macro Node are available. One of the node types is always active. The current active node type is highlighted in the list and displayed at the top of the list. While in the *Add Node* mode, clicking the mouse anywhere in the graphics screen causes an instance of the current active node type to be added to the database.

The radar pedestal model will be built from the top down. The first element in the top level is an input signal. The Block Diagram node SRC (source) will provide this function. The SRC node is chosen be the active node. When the word SRC is highlighted on the list, the node can be added to the database by clicking the mouse in the graphics screen. Figure 9 shows the edit screen in the Add Node mode with the SRC node added. The node is represented by the icon SRC with a box around it. Since a source node provides an output signal, a small square box is drawn on the right side of the node

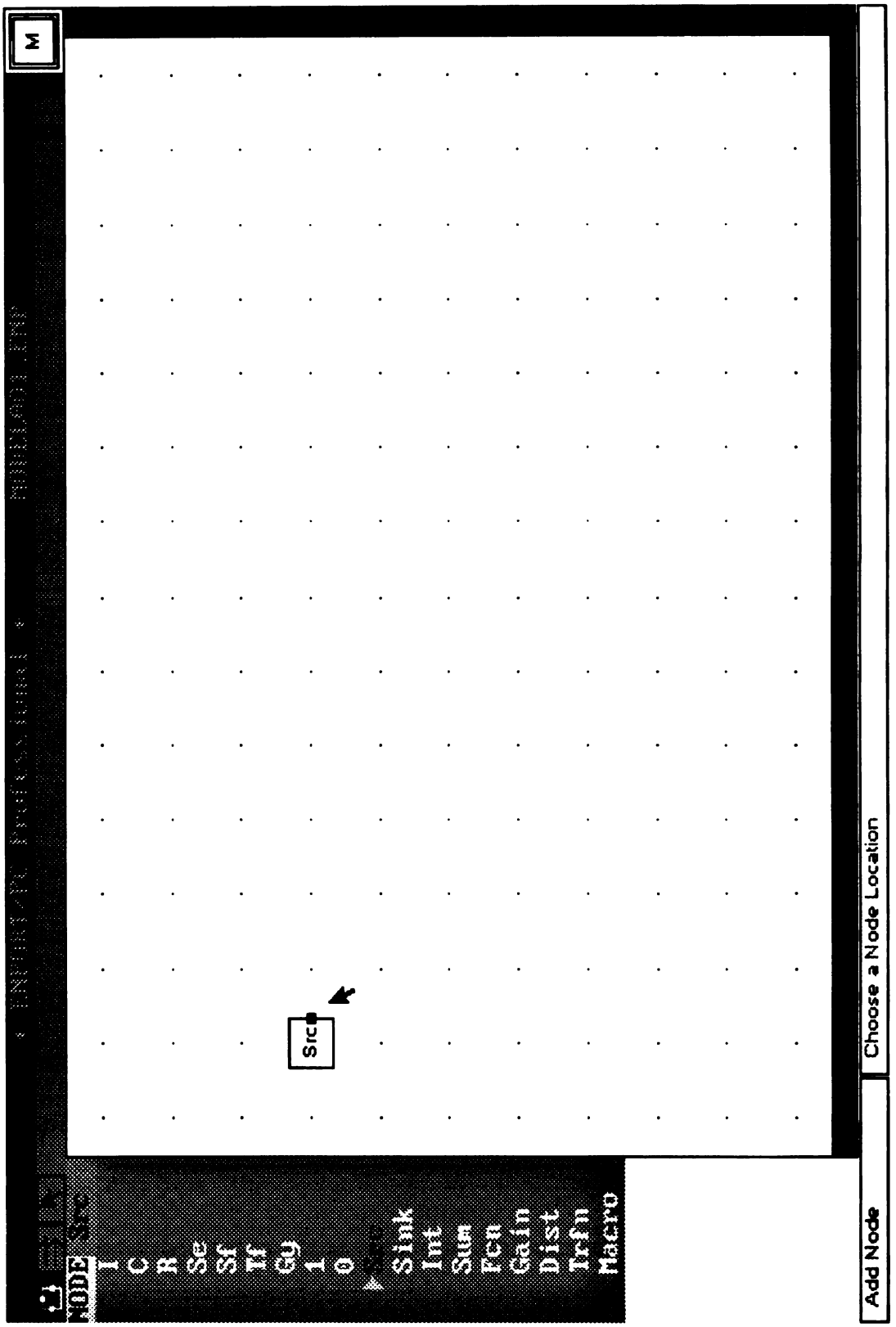


Figure 9. Adding a Source Node to the System

representing the signal port. The filled in square represents an output signal. (An empty square represents an input signal.)

Using the same procedures, a Block Diagram SUM node and a Macro Node are added to the graph and placed to the right of the SRC node. The Macro node is created with no ports. This first Macro Node represents the controller. It must have two ports: an input signal port (for the error signal) and an output signal port (for the conditioned error signal). To add a port to the Macro Node, the node must be selected. To select a node, the *Select Object* mode must be chosen. The *Select Object* mode button is drawn with an arrow pointer similar to the mouse arrow pointer. Alternatively, pushing SHIFT-TAB will change to *Select Object* mode.

The Macro Node is selected with the mouse. When an item is selected, it is highlighted in red. Selecting the Macro Node causes two additional buttons to appear at the top of the screen: the *Edit* and *Delete* buttons. These operations are performed on a object and therefore do not appear until an object is selected. Pressing the *Edit* button causes a menu to appear which lists the valid edit operations for a Macro Node. One of the options is *Add Signal In Port*. Figure 10 shows the graphics screen with the Macro *Edit Menu*. Selecting the Add Signal In Port option causes an input signal port to be added to the data base. This change is reflected on the graph with an empty rectangle. The procedure is repeated to add an output signal to the node. The *Edit Menu* for all node types contains the option of *Attributes*. Two attributes of a node may be edited: the *label* and the *label visibility*.

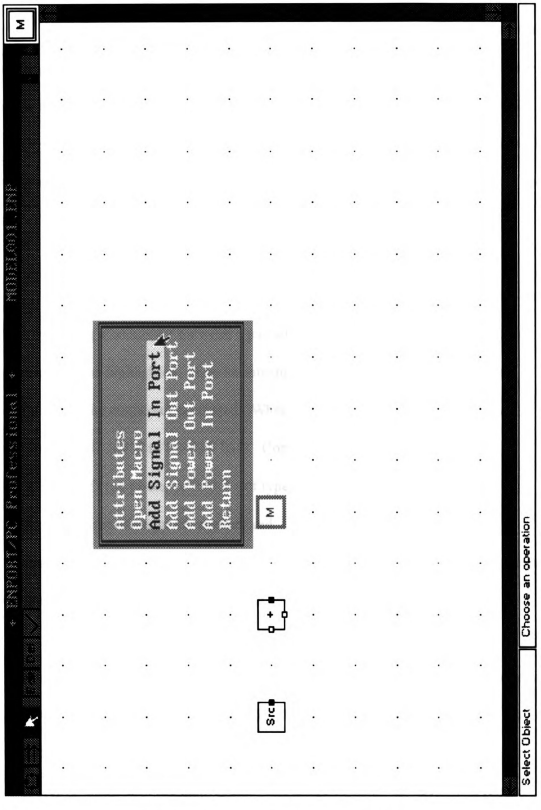


Figure 10. The Macro Node Edit Menu

If the default locations of the ports are not convenient, they can easily be moved. It is a long-term objective for this software that all graph objects be positionable by the user. At this point, the only objects that can be positioned are nodes and ports. The *Select Object* mode must be chosen to position an object. Positioning is accomplished by pressing and holding down the mouse button while the arrow is positioned on top of the object. The selected object tracks the position of the mouse.

The *Add Connector* mode is used to connect nodes. Nodes are connected to one another by adding a connector between a pair of ports. Clicking the mouse on the first port causes it to be highlighted. If an intermediate point on the graph is selected an interior point of the connector is created. When a compatible second port is chosen, a connector object is added to the data base. Compatible ports must

1) both be of type power or both be of type signal,
2) belong to the same power regime, and
3) have complementary input/output characteristics. Output ports are filled circles and squares and input ports are open circles and squares.

Using the three modes described above, the top level of the graph is drawn to look like Figure 11 . Notice the similarity to Figure 8. A distinction in the software is the use of explicit notation for the Distributor Node and the Sink Node. At this point, only a graphical representation of the control and plant models exists. More detail is required to obtain a graph that will yield a mathematical model.

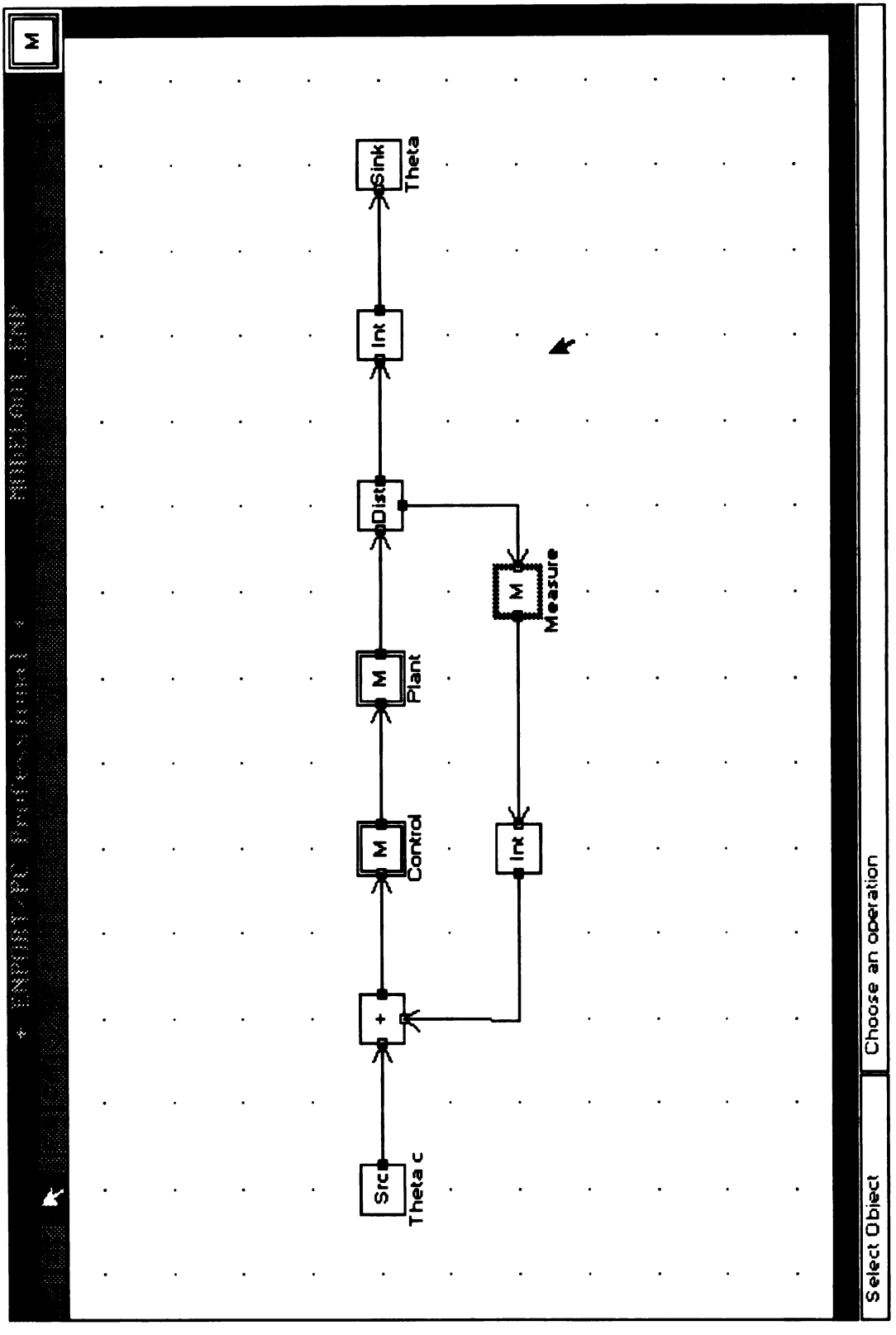


Figure 11. Top Level Model of the Radar Positioning System

3.2.2 Defining a Macro Node

When a Macro Node is added to the database, its child subgraph is also created. The contents of the Macro Node *Plant* may be viewed by opening it. *Open Macro* is one of the options in the edit menu for a macro node. Initially, the subgraph is empty.

The Parent Macro to this subgraph is displayed in the top right corner of the screen and is drawn in gray. The gray color indicates that one or more ports on the Parent Macro Node are currently not associated with ports in the child subgraph. Currently, association between ports on the Parent Macro and ports in the child subgraph are made explicit through use of a special node type not previously discussed: the Port Node. The use of a Port Node is temporary and will not be used in later versions of the software. The left node on the Parent Macro Node is a Signal In Port. A Signal Output Port Node will explicitly represent this port in the graph.

Once the Signal Output Port Node is added, it must be assigned to a port on the Parent Macro Node. Until a Port Node is assigned to a port on the Parent Macro Node, both the Port Node and the port on the Parent Macro are drawn in gray. To assign a Port Node to a port on the Parent Macro Node, the system must be in Add Connector mode. The Port Node, not the port on the Port Node is now highlighted. The assignment process is completed by selecting a port on the Parent Macro.

Connecting a Port Node and a port on the Parent Macro Node causes these

objects to be drawn in bright white. Clicking on a Port Node that is already selected will cause the port that it is assigned to it to be highlighted. Also, a message at the bottom of the screen appears to inform user that the Port Node has already been assigned.

The rest of the Plant model can now be built by adding a Macro Node for the DC motor, the shaft, the gears, the pedestal, and a Signal In Port Node. Figure 12 shows the completed macro model of the plant. Notice that this representation is very general. Modeling details that would lead to governing system equations still have not been developed. The Macro Node representation has provided a way to organize the model into smaller, more tractable subsystems. If the subdivision of the system at this point is adequate, the Bond Graph and Block Diagram Nodes can be used to build a system model that will yield system equations.

Figure 13 shows one possible model for the DC Motor. This model is created by opening the DC Motor Macro Node, adding and assigning the proper Port Nodes, adding the desired Bond Graph Nodes that represent the system, and making connections. Once the bond graph model for the DC Motor is completed, the parent subgraph can be seen by clicking on the *Go Up One Level* button on the top right hand of the screen. The button is represented by two dots ". . ", similar to the DOS command to move up one directory.

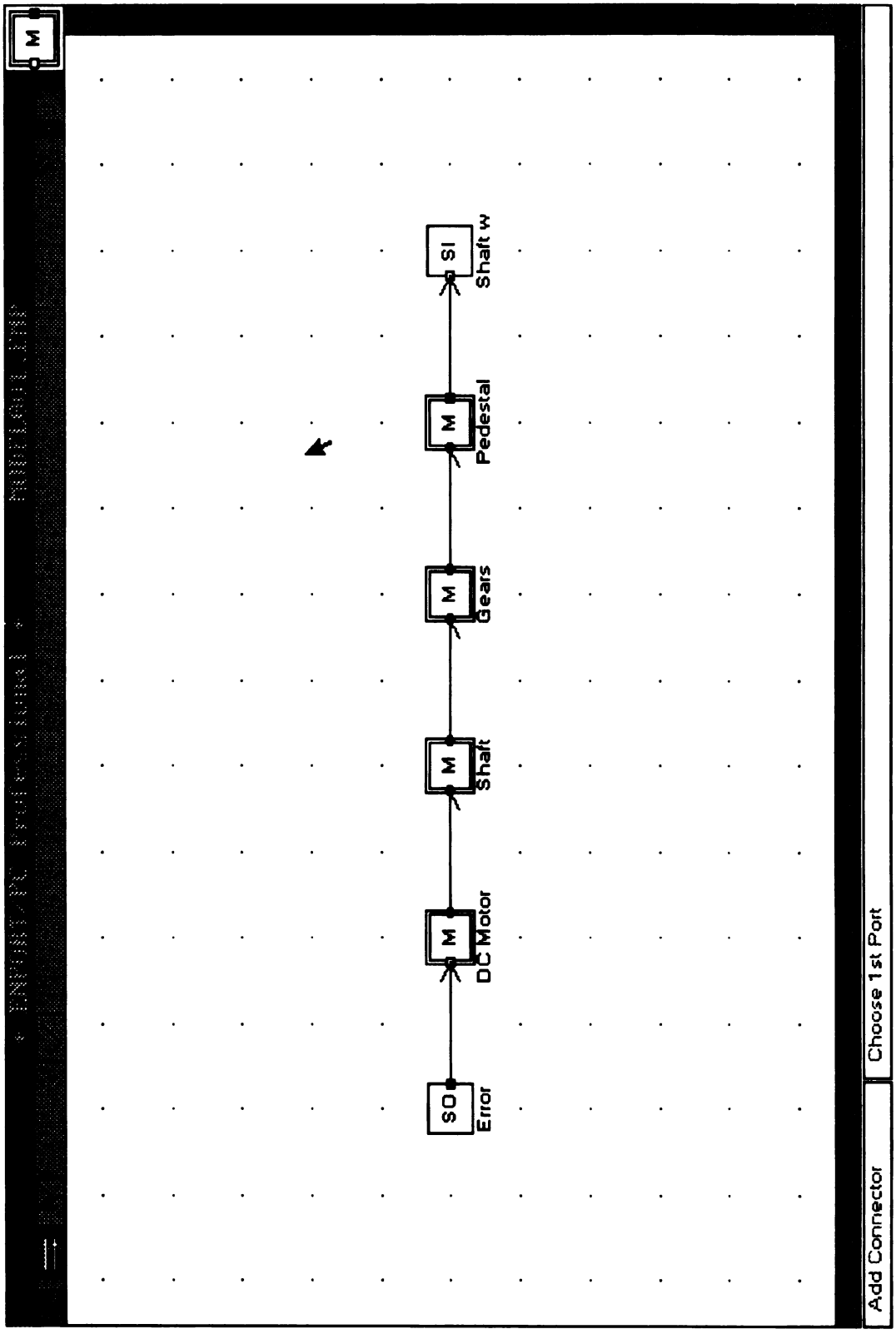


Figure 12. Macro Model of Radar Positioning System Plan

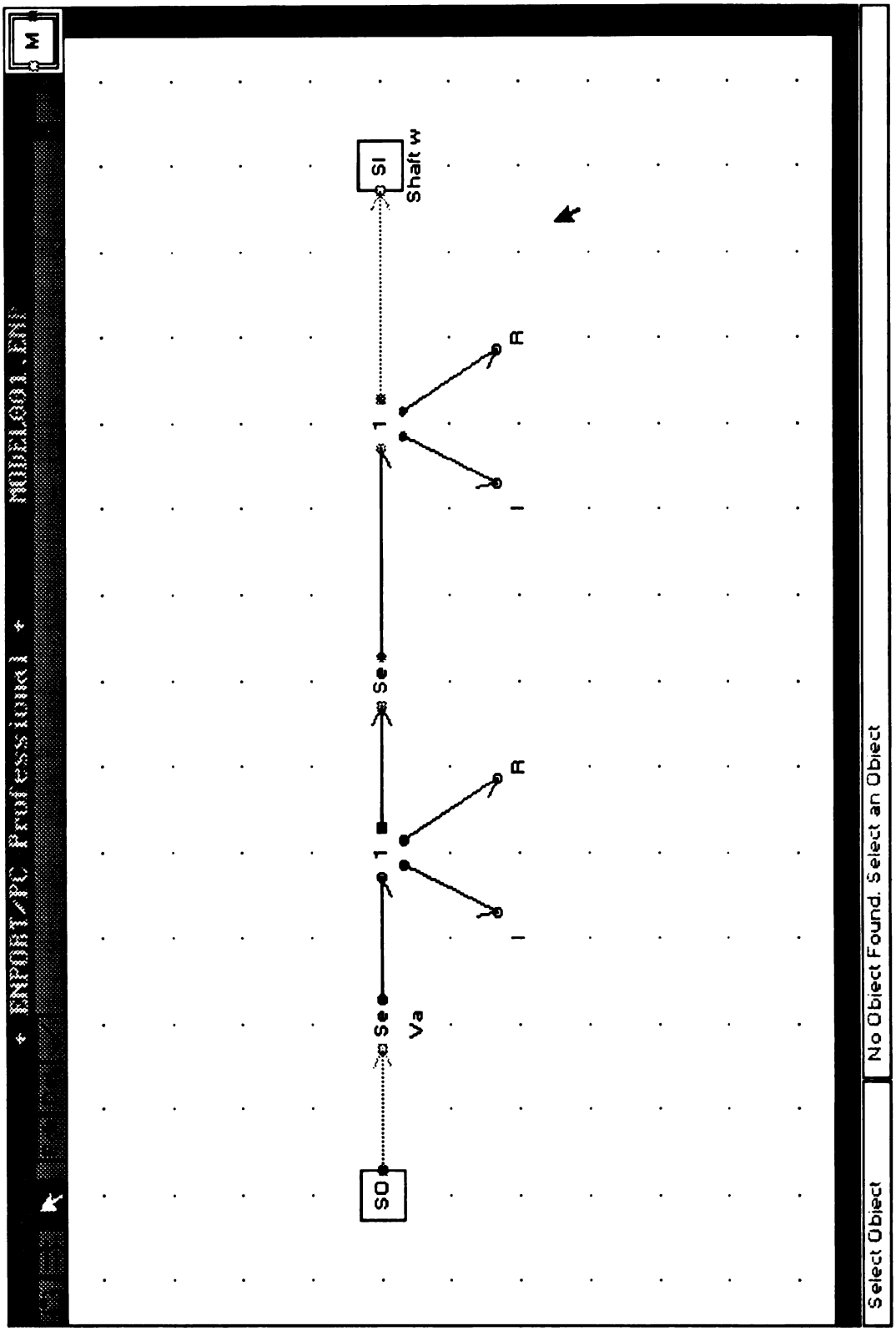


Figure 13. Bond Graph Model of a Motor

3.2.3 Checking the Graph

To demonstrate the graph checking feature, a previously completed model of the radar pedestal will be used. This model has been created with some flaws that prevent generation of a valid set of system equations. The name of the file is RADPED.BMF and can be retrieved by using the *Open* command from the *File* menu.

Once the file is loaded, clicking the *Graph Check* button reveals the *Graph Check* menu. The graph topology is not complete in this mode because a Port Node has not been associated with a port on a Parent Macro. This is an easy error to make and a difficult error to locate, especially in a large system. Selecting the *Check Setup* menu item causes the system to search for incomplete connections and assignments. As shown in Figure 14, a message box appears that reports the results of the check. Clicking on the message indicating a Port Node was not assigned or clicking the *Find* option at the bottom of the screen when this option is highlighted will located the offending Node. The contents of the subgraph containing this node are displayed and the node highlighted. It is now a simple matter to assign the Port Node to the Port on the Parent Macro.

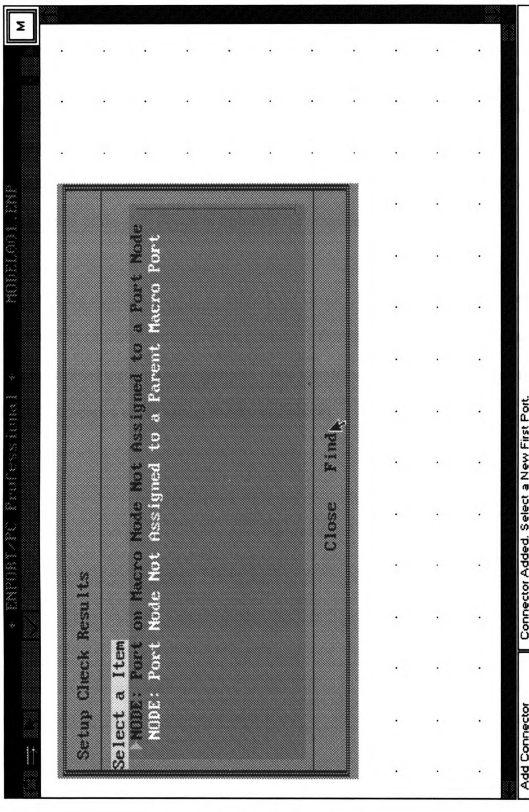


Figure 14. The Graph Check Utility

3.2.4 Defining and Sorting Equations

Bond Graph and Block Diagram Nodes contain explicit information about the mathematical equations for the system. To define an equation, select a node and from the edit menu choose the *Edit Equations* option. The procedure used in this system for defining the equations associated with a node has not been changed from the previous implementation of the *Edit Equation* feature in the program ENPORT/PC [6].

Once the equations have been defined, the Graph Check option *Sort Equations* can be chosen. The equation sorting process produces information required for numerically solving the defined mathematical equations. This algorithm was taken directly from the ENPORT/PC version of the Model Builder module [6].

Chapter 4

Conclusions

4.1 Contributions

The intent of this work was to produce an enhanced environment for the development of the macro representation of models. The overall result is a prototype software package that implements the following: a database built on object-oriented principles, an intuitive, easy to use Graphical User Interface, explicit representation of node ports, and macro representation and manipulation tools. These features are summarized below.

Improved Database - A prototype database system was developed which implemented the ideas of a hierarchical graph structure which is separable from the main code. Tools to interface easily with the database in an explicit, consistent manner were developed; namely, the *Add*, *Get*, *Set*, and *Delete* functions were implemented. This prototype database served as the foundation for the more comprehensive modeling system. It is good practice in the design of large programming projects to make a clean separation between various parts of the program. The development of the database tools was successful at providing another level of program separation. This style of programming may initially take more time and energy, but future enhancements are less intrusive, i.e.,

fewer areas of the code need modification.

Graphical User Interface (GUI) - An intuitive, easy to use, graphical user interface for building models was developed. Interaction with graph objects and operations are initiated through a graphical interaction with the user. Information is graphically displayed where possible and the mouse is used extensively.

Explicit Node Ports - Through the use of an explicit representation of a node's ports, restrictions can be placed on the use of a node. This tool is helpful in clearly identifying the proper use of a Macro Node without explicit knowledge of its contents. Since an equation associated with a node is only a function of the names of its ports, the graph appearance is consistent with its equation structure.

Macro Representation/Manipulation Tools - Successful implementation of this objective depended heavily on the success of the previous objectives. Introduction of the concepts of Level, Subgraph, Macro Node, and Parent Macro Node helped in a clearer representation of these ideas.

An important result of this work is that a strong foundation has been laid for future enhancements. While the tools that have been implemented are helpful and important, the design ideas and concepts outlined in this thesis (e.g. the Subgraph, Macro Node, and Ports) are equally helpful and important. The current work has focused the definition of graph objects and operations and has provided a rich environment for the implementation

of new tools.

4.2 Future Work

The present work has laid a clear foundation for the ready implementation of many additional tools. Several possibilities are presented.

Library Tools - A considerable effort is often made in creating a model. When models are structured such that the system is divided into subsystems, certain components become very general. Tools for importing and exporting submodels to create a library of component models are needed.

Functionally, this goal could be achieved with relative ease by enhancing this software. In the past, an obstacle that prevented a robust use of libraries was the requirement that each node and connector had to be uniquely named. The only way for a component library model to be loaded more than once into a system was to rename each of the nodes and connectors. This was accomplished implicitly by the computer (which would explicitly affect the model) or carried out explicitly by the user; a burdensome task. The system developed does not require the user to provide unique labeling of nodes and connectors, since the labeling process is automatic.

Additionally, careful thought is required to ensure proper (re)use of library models. The introduction of explicit node ports is a step in the right direction, but many other

issues arise. Limiting assumptions that were made in the development of a submodel must be made explicit. Perhaps a set of rules and attributes could be established that allows for automatic verification of the suitability of a particular submodel for a specific use.

Advanced Graphical User Interface Tools - There are many software packages today which provide a rich graphical editing environment. In such an environment, one helpful function is the *Multiple Selection* tool. It would be beneficial to select several like objects and perform an operation on them all at once. Currently, this software allows only one object to be selected at a time. An extension should be pursued.

Another tool should provide for the creation of a submodel by grouping a set of existing nodes and replacing them with a Macro Node. This is often the way in which models progress. A user may have spent considerable time developing a detailed model in one graph. It may only later occur to him that the model would be conveniently represented as a set of submodels. The tools to accommodate for this type of design progression should be available.

Another common set of functions - *Undo*, *Cut*, *Paste*, and *Copy* would be helpful. The *Undo* command would require some deep consideration into the storage of superseded information. The *Cut*, *Paste*, and *Copy* utilities may be implemented with relatively little effort as the object-oriented data structure would not make these function difficult to build.

Definition of Atom Node Templates - There are two tasks associated with the broader use of node templates. First, the current system does not fully exploit the node template ideas. There are many instances where the explicit node type is used to guide a procedure. This violation of the template concept arises from the difficulty in defining a comprehensive set of attributes that is capable of completely specifying any type of node. Identification of this complete attribute set would be invaluable. Second, only the node template database tool *get* was used. A complete database interface would allow a user to dynamically create and modify new node types by defining template structures. The first task listed above would obviously have to be completed before the second. Definition of new node types in this manner would provide a rich, flexible system for the user.

Programming Language - It has become obvious that FORTRAN is not the programming language best-suited for object-oriented programming. The ideas of object-oriented programming could only be partially utilized since FORTRAN simply does not provide the correct tools. One motivation for using FORTRAN was the existence of code to perform a large array of computationally difficult and intensive tasks, such as causality assignment and equation sorting. However, most computer languages provide for interfaces with other computer languages. Indeed, this project was successful only because FORTRAN allowed for interfaces to C. Future programming projects that include the implementation of a graphical interface should be based on other programming languages that provide object-oriented tools.

APPENDICES

Appendix A

Graph Object Definitions

The Software developed has been defined with structures that define attributes of graph objects. Following is the code that was used to formally define the graph objects.

```
#<DSPFILE:DSDEFN.FD>-------------------------Last Change: 31-Aug-95 mkh
;
;---- Purpose:  Define max values and mnemonics for accessing the
;               database using the database utilities.
;
;---- Note:     An 'Object' is defined as such if it is defined by
;               a subset of information or attributes.  An 'Attribute'
;               is a specific piece of information.
;
;---- Variables:
;       MAXLV           Maximum number of levels
;       MAXSG           Maximum number of subgraphs
;       MAXNP           Maximum number of nodes per subgraph
;       MAXND           Maximum number of nodes
;       MAXPRT          Maximum number of ports
;       MAXCTOR         Maximum number of connectors per subgraph
;       MAXCTOR         Maximum number of connectors
;       MAXCPT          Maximum number of points for a connector
;       MAXSEG          Maximum number of segments
;
;---- Objects:
;       SG              Sub Graph
;       ND              Node
;       PRT             Port
;       PT              Point
;       CTOR            Connector
;       IPT             Interior point
;       SEG             Segment
;
;---- Types of Attributes:
;       TEXT            Text
;       VALUE           Numeric value
;       NUMPTS          Number of connection points
;       NUMIV           Number of input variables
;       NUMOV           Number of output variables
;       NUMIOV          Number of input/output variables
;       INVAR           Input Variable
;       OUTVAR          Output Variable
;       PARVAR          Parameter Variable
;       STAT            Status
;       NUMSUBG         Number of subgraphs
;       VISSUBG         Visible subgraphs
;       ENGINE          Engine
;       VIS             Visibility
;       NUMPRT          Number of power ports
;       NUMSIP          Number of signal ports
;       ORIENT          Angle/Orient
;       SIGN            Sign
;       NUMPT           Number of interior points
;       FMNODE          From Node Index
;       FMPRTNO         From Node Absolute Index
;       TONODE          To Node Index
;       TOPRTNO         To Node Local Index
;       TOPRTNO         To Node Absolute Index
;       TOPORT          To Port Index
;       PRTNO           Port Index
;       PRTTYP          Power/SIP Number        [For ports on a Macro]
;       PRTPTR          parent port number      [For Port Nodes]
;       NUMSUBG         Spawned subgraph number [For MACRO Nodes]
;
;---- Attribute Value Codes:
;       POWER/SIGNAL    Port/Connector types
;       W/S/I/A/O/N     Node groups
;       W/S/I/A/O/N/P/C Port types
;       M/A/F/B/C/S/P/C
;       L/R/T/B         Positions
;       IN/OUT          "
;       H/V             "
;       TOP/S/PLANE     Depth
;       ON/OFF          Engine
;       ON/OFF/ERROR    Connector Status
;       VIS/NV          "
;       SHORAL/MECHAN/MOTATE/ELECTR/HYDRAL/THERMO/MAGNET  power regimes
;       SHORAL/MECHAN/MOTATE/ELECTR/HYDRAL/THERMO/MAGNET/
;       SIG/ELECTR/ELECTRSIG/MECHSIG/HYDRAL/HYDRAULIC/
;       SIG/ELECTR/ELECTRSIG/MECHSIG/HYDRAL/HYDRAULIC/
;       SUPPORT/DATABUS/SYSTEM/TEPSEC/MACRO/PSPORT/SIPORT/
;                                                       node types
;
;---- Declarations:
;---- Maximum values ***** put some where else
    INTEGER MAXLV, MAXSG, MAXNP, MAXND, MAXCTOR, MAXSEGS, MAXCPT
    INTEGER MAXPRT, MAXPRT
    PARAMETER ( MAXLV=10, MAXSG=100, MAXNP=4, MAXND=4*100 )
    PARAMETER ( MAXPRT=16, MAXCTOR=4 )
    PARAMETER ( MAXPRT=MAXND*MAXCTOR, MAXCTOR=MAXPRT*MAXCTOR )
    PARAMETER ( MAXSEGS=MAXCPT*MAXCPT )  !Define more generally
C
C---- Define mnemonics for object types
    INTEGER NODE, ICON, PORT, LABEL, CONN, IMPT, PGP
    PARAMETER ( NODE=1, ICON=2, PORT=3, LABEL=4, CONN=5, IMPT=6 )
    PARAMETER (PGP=7)
C
C---- Define mnemonics for types of attributes
C
    INTEGER TYPE, STAT, VISUBG, ENGINE, COLOR, VIS, NPWR
    INTEGER NUMPTS, NUMIV, NUMOV, NUMIOV, NUMSUBG
    INTEGER FMNODE, TONODE, TOMODE, TOPORT, LABEL, SIGN
    INTEGER ORIENT, PRTNO, PRTTYP, PRTPTR, VPOS, PARVAR, NPGP
    INTEGER NUMPT, NUMSEG, NUMCPT
    PARAMETER (TYPE=1, STAT=2, VISUBG=3, ENGINE=4, COLOR=5)
    PARAMETER (VIS=6, NPWR=7, NUMPTS=8, NUMIV=9, NUMOV=10, NUMIOV=11)
    PARAMETER (NUMSUBG=12, FMNODE=13, FMPRTNO=14, TONODE=15)
    PARAMETER (TOMODE=16, TOPORT=17, LABEL=18, SIGN=19)
    PARAMETER (ORIENT=20, PRTNO=21, PRTTYP=22, PRTPTR=23)
    PARAMETER (NUMPT=24, VPOS=25, PARVAR=26, NPGP=27)
    PARAMETER (NUMSEG=28, NUMCPT=29)
C
C---- Define mnemonics of Attribute value codes
C
    INTEGER POWER, SIGNAL, MULTI, EACH, IN, OUT, W, SUPPORT
    INTEGER TWIN, PLANE, TOP, ON, ERROR, VNV, NV, SUPPORT
    PARAMETER (POWER=1, SIGNAL=2, MULTI=3)
    PARAMETER (W=1, S=2, I=3, A=4, O=5, N=6, P=7, C=8)
    PARAMETER (EACH=1, OUT=2, TWIN=3, PLANE=4, IN=5, SUPPORT=6)
    PARAMETER (L=1, R=2, T=3, B=4, M=5, F=6, B=7, C=8, S=9, P=10)
    PARAMETER (ON=1, OFF=2, ERROR=3, VIS=4, NV=5)
C
C---- Node types
    INTEGER INVAR, IMPICA, CAPACT, RESIST, ENCPST, ENCPTP, INMETR
    INTEGER OUTVAR, SUMPIC, JUNCTI, RELAY, RELAYC, INVLIF, DIMELP
    INTEGER PIMELP, VALVES, DIVTER, TEPSEC, MACRO, PSPORT, SIPORT
C
    PARAMETER (OUTVAR=1, INMETR=2, CAPACT=3, RESIST=4, ENCPST=5)
    PARAMETER (ENCPTP=6, TEPSEC=7, SUMAIR=8, JUNCTI=9, JUNCTS=10)
    PARAMETER (RELAY=11, RELAYC=12, INVLIF=13, DIMELP=14, PIMELP=15)
    PARAMETER (VALVES=16, DIVTER=17, TEPSEC=18, MACRO=19, PSPORT=20)
    PARAMETER (SIPORT=21, SUPPORT=31
C
C---- Power Regime  Variable value is also color index.
```

```
    INTEGER SHORAL, MECHAN, ROTATE, ELECTR, HYDRAL, THERMO, MAGNET
    PARAMETER (SHORAL=16, MECHAN=32, ROTATE=32, ELECTR=33, HYDRAL=35)
    PARAMETER (THERMO=36, MAGNET=14)
C
C---- Define identification key
      TYPE IDENTIFY_KEY
        INTEGER  LV
        INTEGER  SG
        INTEGER  ND
      END STRUCTURE
C
C#<DSPFILE:CNDEFN.FD>-------------------------------------------------*
C
C#<CONNECTOR:CNDEFN.FD>-----------Last Change: 31-Aug-95 mkh
C
C---- Purpose:  Define Graph information
C
C---- Variables:
C     $$$ PARTNO  Local Subgraph number where Parent Node resides
C     $$$ PARNDX  Node Index of Parent Node in Parent
C     $$$ NUMSG   Number of Subgraphs in Subgraph
C     $$$ MAXND   Number of Macro nodes in a subgraph
C     $$$ NUMND   Number of nodes in a subgraph
C     $$$ VPOS    Viewport Left  X-Coordinate
C     $$$ VPOS    Viewport Left  Y-Coordinate
C     $$$ VPOS    Viewport Right X-Coordinate
C     $$$ VPOS    Viewport Right Y-Coordinate
C     $$$ VPOS    Viewport Top   Y-Coordinate
C
C---- Declarations:
      TYPE IDENTIFY_KEY
        INTEGER  PARTNO
        INTEGER  PARNDX
        INTEGER  NUMSG
        INTEGER  MAXND
        INTEGER  NUMND
        REAL     VPOS
        REAL     VPOS
        REAL     VPOS
        REAL     VPOS
      END STRUCTURE
C
C#<MEMBER:CNDEFN.FD>--------------------------------------------------*
```

```
C#<DSPFILE:DSNODE>----------------------Last Change: 24-Aug-95 mkh
C
C---- Purpose:  Define attributes of Nodes
C
C---- Note:     The last Node stored in this list is the "System Macro"
C               that is the parent of all nodes in the first subgraph.
C               First subgraph on level one (first visible SG).  This
C               Node is created automatically when a new graph is built.
C               The number of nodes is NOT reflected in the total number
C                       of nodes.
C
C---- Variables:
C     $$$$ SUBG   Spawned Subgraph NoMber (if a Macro node; 0 if none
C     $$$$ PRT    Parent Port Number(Port Nodes) 0 if none
C     $$$$ NPP    Number of power ports on the node
C     $$$$ NSP    Number of signal ports on the node
C     $$$$ TYPE   The node type (see list of NODE  TYPEs defined DSDEFN.FD
C     $$$$ ICN.X  X-coord of center ID icon
C     $$$$ ICN.Y  Y-coord of center ID icon
C     $$$$ LBL.NUM  Label regime of node (determined by icon's color)
C     $$$$ LBL.X  X-coord of center of label
C     $$$$ LBL.ROT  Rotation of label counterclockwise relative to the level
C     $$$$ LBL.X  X-coord of center of label (relative to the icon)
C     $$$$ LBL.Y  Y-coord of center of label (relative to the icon)
C     $$$$ LBL.VIS  Label is visibility: ON or OFF
C     $$$$ LBL.TXT  Label text
C     $$$$ PORT(I).PRT  Port Node number associated with port(Macro nodes)
C     $$$$ PORT(I).ANG  Angle between port. node center. & horiz.(-180 to
C                                                                   180)
C     $$$$ PORT(I).NUM  Power regime of port
C     $$$$ PORT(I).TYP  Type of port: POWER or SIGNAL
C     $$$$ PORT(I).SIGN  Sign of port: +1 or -1 (for signal ports)
C     $$$$ PORT(I).SIGNAL  Port Signal variable (name) or signal variable
C     $$$$ PORT(I).POWER  Port Power Variable (name) or power variable
C     $$$$ PORT(I).NUMBIT  Port Bit Variable (bound:NUMPTR, DISPLY)
C     $$$$ PORT(I).VPAR  Input variable type
C
C---- Declarations:
      STRUCTURE /NODE_DEF/
        INTEGER  SUBG
        INTEGER  PRT
        INTEGER  NPP
        INTEGER  NSP
        REAL     X
        REAL     Y
      END STRUCTURE
C
      STRUCTURE /LABEL_DEF/
        INTEGER  NUMREGIME
        INTEGER  ICN
        INTEGER  X
        INTEGER  Y
        REAL     ROT
      END STRUCTURE
C
      STRUCTURE /PORT_DEF/
        INTEGER  PRT
        INTEGER  NUM
        INTEGER  TYP
        INTEGER  SIGN
        INTEGER  ANG
        CHARACTER SIGNAL(1)*16
      END STRUCTURE
C
      STRUCTURE /NODE_DEF/
        INTEGER  NODE
        INTEGER  TYPE
        INTEGER  STAT
        INTEGER  VIS
        INTEGER  NUMPTS
        INTEGER  NUMIV
        INTEGER  NUMOV
        INTEGER  NUMIOV
        INTEGER  NUMSUBG
        RECORD /NODE_DEF/  ICN
        RECORD /LABEL_DEF/ LBL
        RECORD /PORT_DEF/  PORT(MAXPRT)
      END STRUCTURE
C#<DSPFILE:DSNODE>----------------------------------------------------*
```

```
C#<DSPFILE:DSCONN>----------------------Last Change: 31-Aug-95 mkh
C
C---- Purpose:  Define attributes of Connectors
C
C---- Variables:
C     $$$$ CONN   The ending Node number (in the subgraph)
C     $$$$ CONN   The ending port number
C     $$$$ CONN   The ending connector (in the subgraph)
C     $$$$ CONN   The starting port number
C     $$$$ CONN   The starting connector (visibility stroke or 0
C     $$$$ CONN   Number of interior points
C     $$$$ CONN   Visibility: ON or OFF
C     $$$$ CONN   Connector Status
C                   OK    -> Regular connection
C                   ERROR -> Compatibility conflict between ports
C                   VIS   -> On 1 Port Node (no direction or
C                            destination)
C
C---- Declarations:
      STRUCTURE /IMPT_DEF/
        INTEGER  X
        INTEGER  Y
      END STRUCTURE
C
      STRUCTURE /CONN_DEF/
        INTEGER  ENDND
        INTEGER  ENDPRT
        INTEGER  ENDCTOR
        INTEGER  STRPRT
        INTEGER  STRCTOR
        INTEGER  NUMIPT
        INTEGER  VISIB
        INTEGER  STAT
        INTEGER  VIS
        RECORD /IMPT_DEF/ IPT(MAXCPT)
      END STRUCTURE
C
C#<DSPFILE:DSCONN>----------------------------------------------------*
```

Appendix B

Database Source Code

The source code for the separable database was previously developed in a limited environment that stored node objects using a text interface. The database tools were enhanced and expanded for use in the current software. Two files, MBDBU1.FOR and MBDBU2.FOR, define the database tools.

```
C
      IF ( (ID1.LV.EQ.ID2.LV) .AND. (ID1.SO.EQ.ID2.SO) .AND.
     *     (ID1.SN.EQ.ID2.SN) ) THEN
        IDSQID = .TRUE.
      ELSE
        IDSQID = .FALSE.
      END IF
C
      RETURN
      END
C
C
CSUB:IDSQID*****************************************************
C
$MESSAGE:   ID21L function
CID21L**************************************** Last Change: 00-May-95 mkh
      INTEGER FUNCTION ID21L(ID)
      IMPLICIT NONE
C
C---- Purpose:   Convert an objects ID Key to the local index value
C               in the data base.  Current Subgraph.  Although this
C               function may be simple  it allows for a cleaner
C               separation between the development code and the database.
C
C---- Inputs:    ID.      Object identification
C
C---- Outputs:   ID21L    Object Local index value
C
      INCLUDE 'DBDEFS.FD'
C
      RECORD /ID_KEY/ ID
C
C***ID21L*******************************************************
C
      ID21L = ID.SN
      RETURN
      END
C
C
CSUB:ID21L******************************************************
C
$MESSAGE:   ID2LV function
CID2LV**************************************** Last Change: 27-Jul-95 mkh
      INTEGER FUNCTION ID2LV(ID)
      IMPLICIT NONE
C
C---- Purpose:   Determine from an objects ID Key the Level where the
C               object is located in the data base.  Although this
C               function may be simple, it allows for a cleaner
C               separation between the development code and the database.
C
C---- Inputs:    ID.      Object identification
C
C---- Outputs:   ID2LV    Object's Level
C
      INCLUDE 'DBDEFS.FD'
C
      RECORD /ID_KEY/ ID
C
C***ID2LV*******************************************************
C
      ID2LV = ID.LV
      RETURN
      END
C
C
CSUB:ID2LV******************************************************
C
$MESSAGE:   ID2SO function
CID2SO**************************************** Last Change: 27-Jul-95 mkh
      INTEGER FUNCTION ID2SO(ID)
      IMPLICIT NONE
C
C---- Purpose:   Determine from an objects ID Key the Subraph where
C               the object is located in the data base. Although this
C               function may be simple, it allows for a cleaner
C               separation between the development code and the database.
C
C---- Inputs:    ID.      Object identification
C
C---- Outputs:   ID2SO    Object Local index value
C
      INCLUDE 'DBDEFS.FD'
C
      RECORD /ID_KEY/ ID
C
C***ID2SO*******************************************************
C
      ID2SO = ID.SO
      RETURN
      END
C
C
CSUB:ID2SO******************************************************
C
$MESSAGE:   ID2IA function
CID2IA**************************************** Last Change: 00-May-95 mkh
      INTEGER FUNCTION ID2IA(OBJ,ID)
      IMPLICIT NONE
C
C---- Purpose:   Convert an objects ID Key to the absolute index value
C               in the data base
C
C---- Inputs:    OBJ.     Object type
C               ID.      Object identification
C
C---- Outputs:   ID2IA    Object absolute index value
C
      INCLUDE 'DBCEFS.FD'
      INCLUDE 'DB3DEV.FD'
      INCLUDE 'DB3DEV.CBK'
C
      INTEGER OBJ
      INTEGER N, I, HUMROD, HUMCON, SI
      INTEGER SID2SI
      RECORD /ID_KEY/ ID
C
      EXTERNAL SID2SI
C
C***ID2IA*******************************************************
C
      N= ID.LV
      SELECT CASE(OBJ)
      CASE(NODE):
        IF (N.LE.1) THEN          ! In the top Subgraph
          ID2IA= ID.SN
        ELSE
C
C---- Get the Subgraph index
          SI = SID2SI(ID)
C
C---- Calculate the number of nodes in the "above" Subgraphs
          HUMNOD= 0
          DO I= 1, SI-1)
            HUMNOD= HUMNOD + SGN(I).NODES
          END DO
C
C---- Calculate the node index
          ID2IA= HUMNOD + ID.SN
        END IF
C
      CASE(CONN)
        IF (N.LE.1) THEN          ! In the top Subgraph
          ID2IA = ID.SN
        ELSE
C
C---- Get the Subgraph index
          SI = SID2SI(ID)
C
C---- Calculate the number of Connectors in the "above" Subgraphs
          HUMCON= 0
          DO I= 1, (SI-1)
            HUMCON= HUMCON + SGN(I).CONNS
          END DO
C
C---- Calculate the connector index
          ID2IA= HUMCON + ID.SN
        END IF
      END SELECT
      RETURN
      END
C
C
CSUB:ID2IA******************************************************
C
$MESSAGE:   I2IDL
CI2IDL**************************************** Last Change: 27-Jul-95 mkh
      SUBROUTINE I2IDL(INDEX,OBJ,ID)
      IMPLICIT NONE
C
C---- Purpose:   Convert an object index value to an ID value.
C               Based on current level and subgraph.
C
C---- Inputs:    INDEX.   The node number in the current subgraph
C               OBJ      The object type
C
C---- Output:    ID.      The ID number of the object.
C
      INCLUDE 'DBCEFS.FD'
      INCLUDE 'DB3DEV.FD'
      INCLUDE 'DB3DEV.CBK'
C
      INTEGER INDEX, OBJ
      RECORD /ID_KEY/ ID
C
C***I2IDL*******************************************************
C
      SELECT CASE (OBJ)
      CASE('PCSG'):
        IF (INDEX.GT.SGTOT) Return          ! Bad index request
        ID.LV = CURLV
        ID.SO = CURSG
        ID.SN = INDEX
      CASE ('CONN'):
        IF (INDEX.GT.CONTOT) Return         ! Bad index request
        ID.LV = CURLV
        ID.SO = CURSG
        ID.SN = INDEX
      END SELECT
C
      RETURN
      END
C
C
CSUB:I2IDL******************************************************
C
$MESSAGE:   I2IDL3
CI2IDL3**************************************** Last Change: 30-Aug-95 mkh
      SUBROUTINE I2IDL3(SID,SUN,ID)
      IMPLICIT NONE
C
C---- Purpose:   Convert an object index value to an ID value
C               based on the specified Subgraph.  This routine should
C               be used to hide the ID details from other parts of the
C               program.
C
C---- Inputs:    SID.     Subgraph Id or Object ID in SID
C               SUN.     The Object Number of Local Index.
C
C---- Output:    ID.      The ID number of the object.
C
      INCLUDE 'DBDEFS.FD'
C
      INTEGER SUN
      RECORD /ID_KEY/ SID,ID
C
C***I2IDL3******************************************************
C
      ID = SID
      ID.SN = SUN
      RETURN
      END
CSUB:I3IDL******************************************************
C
$MESSAGE:   I3IDA
CI3IDA **************************************** Last Change: 00-May-95 mkh
      SUBROUTINE I3IDA(INDEX,OBJ,ID)
      IMPLICIT NONE
C
C---- Purpose:   Convert an Object index value to an ID value
C               A methodical guess at the ID key is made which is
C               in turn converted to a test index.  If INDEX matches
C               the test index then you got it!
C
C---- Inputs:    INDEX.   The Object number in the current subgraph
C               OBJ.     The Object type
C
C---- Output:    ID.      The ID number of the Object.
C
      INCLUDE 'DBCEFS.FD'
      INCLUDE 'DB3DEV.FD'
      INCLUDE 'DB3DEV.CBK'
C
      INTEGER INDEX, OBJ
      INTEGER I, J, K, TEMPI
      INTEGER ID2IA
      RECORD /ID_KEY/ ID
C
      EXTERNAL ID2IA
C
C***I3IDA*******************************************************
C
      $SELECT CASE(OBJ)
      CASE('NODE'):
        DO I = 1, LVTOT
          DO J = 1, SGPLV(I)
            DO K = 1, SGN(J).NODES
              ID.LV = I
              ID.SO = J
              ID.SN = K
              TEMPI = ID2IA
              IF (TEMPI.EQ.INDEX) RETURN
            END DO
          END DO
        END DO
      END SELECT
C
      RETURN
      END
CSUB:I2IDA******************************************************
C
$MESSAGE:   P2PSID
CP2PSID *************************************** Last Change: 31-Aug-95 mkh
C
      SUBROUTINE P2PSID(SID,PNUM,PSID,ERR)
      IMPLICIT NONE
C
C---- PURPOSE:   Determine the PortNode ID associated with
C               a port on a MACRO node
C
C---- INPUT:     SID      MACRO node ID
C               PNUM     Port number
C
C---- OUTPUT:    PSID     Port-Node ID
C               ERR.     -1  Port on Macro is not assigned to a PortNode
C                         0  successful
C
      INCLUDE 'DBCEFS.FD'
      INCLUDE 'DB3DEV.FD'
      INCLUDE 'DB3DEV.CBK'
C
      INTEGER PNUM, ERR
      INTEGER I, NTYP, OLV, OBG, OBI, PPS
      INTEGER DB3IFT, ID2LV, SID2SI
      RECORD /ID_KEY/ SID, PSID, SID
C
      EXTERNAL DB3IFT, ID2IL, ID2LV, MESGLS, SID2SI
C
C***P2PSID******************************************************
C
C---- Error Protection
      NTYP = DB3IFT(SCDB,SID,TYPE,0,0,0)
      IF (.NTYP.EQ.MACRO) THEN
        CALL MESGLS('Fatal Error in P2PSID: Illegal Call',999)
      END IF
CC
      PSID.LV = SID.LV+1
      PSID.SO = DB3IFT(SCDB,SID,SOGMSN,0,0,0)
      PSID.SN = DB3IFT(SCDB,SID,PORT,PNUM,PSNUM,0)
      ERR = 0
      IF (PSID.SN.EQ.0) ERR=-1
      RETURN
CC
C
C---- Set up for the search of nodes in subgraph
      OLV = CURLV
      OBG = CURSG
      OBI = CBI
      CURLV = ID2LV(SID) + 1
      CURSG = DB3IFT(SCDB,SID,SOGMSN,0,0,0)
      SIC.LV = CURLV
      SID.SO = CURSG
      OBI = SID2SI(SID)
C
      DO I = 1,SGN(OBI).NODES
        CALL I2IDL(I,SCDB,PSID)
        PPS = DB3IFT(SCDB,PSID,PRETPS,0,0,0)
        IF (PPS.EQ.PNUM) GOTO 800
      END DO
C
C---- Bad News - Couldn't Find Match between port and port node
      CALL MESGLS('Fatal error in P2PSID: Futile search',999)
C
C---- Successful Return
800   CONTINUE
      CURSG = OBG
      CURLV = OLV
      CBI   = OBI
      RETURN
      END
CSUB:P2PSID*****************************************************
C
$MESSAGE:   PN2EQN
CPN2EQN *************************************** Last Change: 07-Sep-95 mkh
C
      SUBROUTINE PN2EQN(SID,PNUM,QNUM)
      IMPLICIT NONE
C
C---- PURPOSE:   Determine the equation number associated with the
C               port PNUM.  These may not be equal due to power
C               and signal ports on a single node.
C               a port on a MACRO node.
C
C---- INPUT:     SID      Node ID
C               PNUM     Port number
C
C---- OUTPUT:    QNUM.    Equation number
C
      INCLUDE 'DBDEFS.FD'
C
      INTEGER PNUM, QNUM
      INTEGER P, PTYP, PID
      INTEGER DB3IFT
      RECORD /ID_KEY/ SID
C
      EXTERNAL DB3IFT
C
C***PN2EQN******************************************************
C
      QNUM = 0
C
C---- Count the ports with output information
      DO P = 1 PNUM
        PTYP = DB3IFT(SCDB,SID,PORT,P,TYPE,0)
        PID  = DB3IFT(SCDB,SID,PORT,P,IBOUT,0)
        IF (.PTYP.EQ.POWER).OR.((PTYP.EQ.SIGNAL).AND.(PID.EQ.OUT))) THEN
          QNUM = QNUM + 1
        END IF
      END DO
C
      RETURN
      END
CSUB:PN2EQN*****************************************************
C
$MESSAGE:   EQN2PS
CEQN2PS *************************************** Last Change: 07-Sep-95 mkh
C
      SUBROUTINE EQN2PS(SID,QNUM,PNUM)
      IMPLICIT NONE
C
C---- PURPOSE:   Determine the Port number associated with the
C               equation QNUM.  These may not be equal due to power
C               and signal ports on a single node.
C               a port on a MACRO node.
C
C---- INPUT:     SID      Node ID
C               QNUM     Equation number
C
C---- OUTPUT:    PNUM.    Port number
C
      INCLUDE 'DBDEFS.FD'
C
      INTEGER QNUM, PNUM
      INTEGER P, NP, QCPT, PTYP, PIO
      INTEGER DB3IFT
      RECORD /ID_KEY/ SID
C
      EXTERNAL DB3IFT
C
C***EQN2PS******************************************************
C
      PNUM = 0
      QCPT = 0
      NP=DB3IFT(SCDB,SID,NPORTS,0,0,0)+DB3IFT(SCDB,SID,NBIOP,0,0,0)
C
C---- Count the ports with output information
      DO P = 1 NP
        PTYP = DB3IFT(SCDB,SID,PORT,P,TYPE,0)
        PIO  = DB3IFT(SCDB,SID,PORT,P,IBOUT,0)
        PNUM = P
        IF ( (PTYP.EQ.POWER).OR.((PTYP.EQ.POWER).AND.(PIO.EQ.OUT)) )THEN
          QCPT = QCPT + 1
          IF (QCPT.EQ.QNUM) RETURN
        END IF
      END DO
C
      PNUM = 0  ! Not enough ports
      RETURN
      END
CSUB:EQN2PS*****************************************************
C
$MESSAGE:   PRETN
CPRETN **************************************** Last Change: 30-Aug-95 mkh
      SUBROUTINE PRETN(SID,PNID)
      IMPLICIT NONE
C
C---- Purpose:   Determine the Parent Macro Node ID from the subgraph
C               or object in subgraph.
C
C---- Inputs:    SID.     Subgraph ID or object in Subgraph ID
C
C---- Output:    PNID,    Parent Macro ID
C
      INCLUDE 'DBCEFS.FD'
      INCLUDE 'DB3DEV.FD'
      INCLUDE 'DB3DEV.CBK'
C
      INTEGER SI
      INTEGER SID2SI
      RECORD /ID_KEY/ SID, PNID
C
      EXTERNAL SID2SI
C
C***PRETN*******************************************************
C
      SI = SID2SI(SID)
      PNID.LV = SID.LV - 1
      PNID.SO = SGN(SI).PRETSO
      PNID.SN = SGN(SI).PRETSN
      RETURN
      END
CSUB:PRETN******************************************************
C
$MESSAGE:   SGMOVE
CSGMOVE*************************************Last Change: 23-Aug-95 mkh
      SUBROUTINE SGMOVE(SID,DIR)
      IMPLICIT NONE
C
C---- Purpose:   Set the proper Graph environment variables such that
C               the current subgraph is set.  Used to open (move down)
C               or close (move up) a subgraph.  A system draw is
C               performed by the calling routine.
C
C---- Input:     SID.     Macro Node's identification key if move down
C                         or dummy.
C               DIR.     Direction of move, UP or DOWN
C
      INCLUDE 'DBCEFS.FD'
      INCLUDE 'DB3DEV.FD'
      INCLUDE 'SGFISH.CBK'
      INCLUDE 'DB3DEV.CBK'
C
      INTEGER DIR
      INTEGER NTYP
      INTEGER DB3IFT, SID2SI
      RECORD /ID_KEY/ SID,IC,KEY/ SID
      RECORD /ID_KEY/ SID
C
      EXTERNAL DB3IFT, SID2SI, I2IDL, MESGLS, SETCSO, SID2SI
C
CSGMOVE*********************************************************
C
C
C---- Error Checking
      IF (.(DIR.EQ.UP).AND.(CURLV.EQ.1) ) THEN
        CALL MESGLS('Fatal Error: Tried to move UP at top level',999)
      END IF
C
C---- Do the move
      SELECT CASE(DIR)
      CASE(DOWN):                                   ! Move down
        NTYP = DB3IFT(SCDB,SID,TYPE,0,0,0)
        IF (.NTYP.EQ.MACRO) THEN
          CALL MESGLS('Fatal Error: Tried to open Non-MACRO node',999)
        END IF
        SIC.LV = CURLV + 1
        SIC.SO = DB3IFT(SCDB,SID,SOGMSN,0,0,0)
      CASE(UP):                                     ! Move Up
        SIC.LV  = CURLV - 1
        SID.SO  = SGN(CBI).PRETSO
      CASE DEFAULT
        CALL MESGLS('Fatal Error: Illegal Call to SGMOVE',999)
      END SELECT
C
C---- Update
C
      CALL SETCSO(SID)
      CALL I2IDL(1,SCDB,CURID)                      ! Update current object
      CUROBJ = 0
      CUP:1  = 0
C
      RETURN
      END
CSUB:SGMOVE*****************************************************
```

```
      EXTERNAL DBGINT, DBGSTR, DBSINT, DBSSTR, ID2IA, I2IDL, SID2SI
C
C***SFTSGS************************************************************
C
C---- Calculate variables for shifting
      SID.LV = NID.LV + 1
      SID.SG = DBGINT(NODE,NID,SSGNUM,0,0,0)
      SI = SID2SI(SID)
C
C---- Determine if an SG shift is legal
      ERR = 0
      IF (DIR.EQ.DOWN) THEN
        IF (SGTOT.EQ.MAXSG) THEN          ! Too many Subgraphs
          ERR = -1
          RETURN
        END IF
      ELSE IF (DIR.EQ.UP) THEN
        IF (SGS(SI).NODES.GT.0) THEN        ! Subgraph is not empty
          ERR = -2
          RETURN
        END IF
      END IF
C
      IF (DIR.EQ.UP) THEN                  ! Shift SG up
C
        DO I= SI,SGTOT-1
          SGS(I) = SGS(I+1)
        END DO
C
        DO I= CURSG,SGPLV(CURLV)          ! Update Child info
          ID2.LV = CURLV
          ID2.SG = I
          NN1 = 1
          IF (I.EQ.CURSG) NN1 = NID.NN + 1
          DO J= NN1,SGS(SID2SI(ID2)).NODES
            ID2.NN = J
            NTYP = DBGINT(NODE,ID2,TYPE,0,0,0)
            IF (NTYP.EQ.MACRO) THEN
              SSNM = DBGINT(NODE,ID2,SSGNUM,0,0,0)
              SSNM = SSNM - 1
              CALL DBSINT(NODE,ID2,SSGNUM,0,0,0,SSNM)
            END IF
          END DO
        END DO
C
C---- Adjust enviroment varibles
        SGTOT= SGTOT- 1
        SGPLV(CURLV+1) = SGPLV(CURLV+1) - 1
        SGS(CSI).MACND = SGS(CSI).MACND - 1
        IF (SGPLV(CURLV).EQ.0) LVTOT = LVTOT-1
      ELSE                                 ! Shift SG down
        DO I=SGTOT+1,SI+1,-1               ! Swap Attributes
          SGS(I) = SGS(I-1)
        END DO
C
C---- Set the default subgraph information. Parent/Child info set in SFTOBJ
        SGS(SI).NODES  =  0
        SGS(SI).MACND  =  0
        SGS(SI).GRID   =  100.0
        SGS(SI).VPX1   = -750.0
        SGS(SI).VPX2   =  750.0
        SGS(SI).VPY1   = -550.0
        SGS(SI).VPY2   =  550.0
C
C---- Adjust enviroment varibles
        SGTOT= SGTOT + 1
        SGPLV(CURLV+1) = SGPLV(CURLV+1) + 1
        SGS(CSI).MACND = SGS(CSI).MACND + 1
        IF (SGPLV(CURLV+1).EQ.1) LVTOT = LVTOT+1
      END IF
C
      RETURN
      END
C
CEND:SFTSGS<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

C
$MESSAGE:'  SID2SI function'
CSID2SI>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>Last Change: 22-Nov-94 mkh
      INTEGER FUNCTION SID2SI(SID)
      IMPLICIT NONE
C
C---- Purpose:  Convert a Subgraph identification (SID) to the
C               index value
C
      INCLUDE 'DBDEFS.FD'
      INCLUDE 'DBGENV.FD'
      INCLUDE 'DBGENV.CBK'
C
      INTEGER N, I, NUMLVL
      RECORD /ID_KEY/ SID
C
C***SID2SI************************************************************
C
      N= SID.LV
C
      IF (N.LE.1) THEN              ! In the top Level
        SID2SI= SID.SG
      ELSE
C
C---- Calculate the number of Subgraphs in the "above" Levels
        NUMLVL= 0
        DO I= 1, N-1
          NUMLVL= NUMLVL + SGPLV(I)
        END DO
C
C---- Calculate the Subgraph index
        SID2SI= NUMLVL + SID.SG
C
      END IF
C
      RETURN
      END
C
CEND:SID2SI<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
C
$MESSAGE:'  SI2SID'
CSI2SID>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>Last Change: 22-Nov-94 mkh
      SUBROUTINE SI2SID(INDEX,SID,ERR)
      IMPLICIT NONE
C
C---- Purpose:  Convert a Subgraph index value to an SID key.
C               A methodical guess at the SID key is made which is
C               in turn converted to a test index.  If INDEX matches
C               the test index, then you got it!
C
C---- Inputs:   INDEX,  The Subgraph index.
C
C---- Output:   SID,    The ID number of the Subgraph.
C               ERR,    0 if successfull, -1 if not
C
      INCLUDE 'DBDEFS.FD'
      INCLUDE 'DBGENV.FD'
      INCLUDE 'DBGENV.CBK'
C
      INTEGER INDEX, ERR, I, J, TESTSI
      INTEGER SID2SI
      RECORD /ID_KEY/ SID
C
      EXTERNAL SID2SI
C
C***SI2SID************************************************************
C
      ERR = -1
      IF (INDEX.GT.SGTOT) RETURN         ! Bad subgraph request
C
      ERR = 0
      DO I = 1, LVTOT
        DO J = 1, SGPLV(I)
          SID.LV = I
          SID.SG = J
          TESTSI = SID2SI(SID)
          IF (TESTSI.EQ.INDEX) RETURN
        END DO
      END DO
C
      RETURN
      END
C
CEND:SI2SID<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

Appendix C

Source Code for Key Features

Several features were introduced to provide a simple intuitive graphical interface. Source code for graph editing is included here. The files of interest are MBEDT1.FOR, MBEDT2.FOR, and MBEDT5.FOR. Two additional files MBSUPP.FOR and NEWCODE.FOR supply miscellaneous function.

```
C
C***MENU***************************************************
C
C--- Don't do this routine if no mouse is present
C    IF (.NOT.MOUSE) RETURN
C
C    Determine mouse position in pixel (Physical Coordinates)
      CALL GETMOUSEPOS(IPX,IPY,MOUSE,BUTTON)
      CALL GETMOUSEPOS(IPX,IPY,MOUSE,MOUSE1,BUTTON)
C
C--- Convert physical coordinates to local viewport coordinates
      CALL GETVIEWCORD(IPX,IPY,MOUSEX,MOUSEY,KK)
      IF (KK .EQ. 0) GOTO 999
      MX = XV SCONG
C
C
C    IF ((MOUSEX.GT.SMENPX1).AND.(MOUSEX.LT.SMENPX2) .AND. ! On viewport
     +   (MOUSEY.GT.SMENPY1).AND.(MOUSEY.LT.SMENPY2)) THEN
C
C--- Process the on graph mouse event
      MPRESS = FALSE.
      IF ((BUTTON.EQ.1).AND.(OPT(1).EQ.1) ) THEN
        LFTSET = TRUE.
        LFTCLR = FALSE.
        IF ((LFTSET.EQV.EQ.1).AND.(OPT(1).EQ.2) ) THEN
          BUTSET = TRUE.
          BUTCLR = FALSE.
          IF ((BUTSET.EQ.1).AND.(OPT(1).EQ.3) ) THEN
            TWOSET = TRUE.
            TWOCLR = FALSE.
          END IF
          IF ((BUTSET.EQV.TRUE).AND.(OPT(1).EQ.4) ) THEN
            MOVSET = TRUE.
            MOVCLR = FALSE.
            MPRESS = .TRUE.
          END IF
        END IF
      END IF
      IF ((BUTSET.EQV.TRUE).AND.(OPT(1).EQ.4) ) THEN
        IF (OPT(1).EQ.1) THEN
          LFTSET = FALSE.
          LFTCLR = TRUE.
        END IF
        IF (OPT(1).EQ.2) THEN
          BUTSET = FALSE.
          BUTCLR = TRUE.
        END IF
        IF ((LFTSET.EQV.TRUE).AND.(OPT(1).EQ.3) ) THEN
          TWOSET = FALSE.
          TWOCLR = TRUE.
        END IF
        MOVSET = FALSE.
        MOVCLR = TRUE.
      END IF
      IF ((LFTSET.EQV.TRUE).AND.(OPT(1).EQ.1) ) THEN
        LFTSET = FALSE.
        LFTCLR = TRUE.
        MPRESS = .TRUE.
      END IF
      IF ((BUTSET.EQV.TRUE).AND.(OPT(1).EQ.2) ) THEN
        BUTSET = FALSE.
        BUTCLR = TRUE.
        MPRESS = .TRUE.
      END IF
      IF ((TWOSET.EQV.TRUE).AND.(OPT(1).EQ.3) ) THEN
        TWOSET = FALSE.
        TWOCLR = TRUE.
        MPRESS = .TRUE.
      END IF
C
C--- Process the off field mouse event
      MPRESS = FALSE.
      IF ((BUTTON.EQ.1).AND.(OPT(1).EQ.1) ) THEN
        LFTSET = TRUE.
        LFTCLR = FALSE.
        IF ((LFTSET.EQV.EQ.1).AND.(OPT(1).EQ.2) ) THEN
          BUTSET = TRUE.
          BUTCLR = FALSE.
          IF ((BUTSET.EQV.EQ.2).AND.(OPT(1).EQ.3) ) THEN
            TWOSET = TRUE.
            TWOCLR = FALSE.
          END IF
          IF ((BUTSET.EQV.TRUE).AND.(OPT(1).EQ.4) ) THEN
            MOVSET = TRUE.
            MOVCLR = FALSE.
            MPRESS = .TRUE.
          END IF
        END IF
      END IF
      IF ((BUTSET.EQV.TRUE).AND.(OPT(1).EQ.4) ) THEN
        IF (OPT(1).EQ.1) THEN
          LFTSET = FALSE.
          LFTCLR = TRUE.
        END IF
        IF (OPT(1).EQ.2) THEN
          BUTSET = FALSE.
          BUTCLR = TRUE.
        END IF
        IF ((LFTSET.EQV.TRUE).AND.(OPT(1).EQ.3) ) THEN
          TWOSET = FALSE.
          TWOCLR = TRUE.
        END IF
        MOVSET = FALSE.
        MOVCLR = TRUE.
      END IF
      IF ((LFTSET.EQV.TRUE).AND.(OPT(1).EQ.1) ) THEN
        LFTSET = FALSE.
        LFTCLR = TRUE.
        MPRESS = .TRUE.
      END IF
      IF ((BUTSET.EQV.TRUE).AND.(OPT(1).EQ.2) ) THEN
        BUTSET = FALSE.
        BUTCLR = TRUE.
        MPRESS = .TRUE.
      END IF
      IF ((TWOSET.EQV.TRUE).AND.(OPT(1).EQ.3) ) THEN
        TWOSET = FALSE.
        TWOCLR = TRUE.
        MPRESS = .TRUE.
      END IF
C
C--- Process the off mouse event
      IF ((.NOT.MOUSE).AND.(.NOT.MPRESS)) THEN
        IF ((OPT(1).EQ.1).AND.(LFTSET) ) THEN ! From last subroutine call
          LFTSET = FALSE.
          LFTCLR = TRUE.
          IF ((OPT(1).EQ.2).AND.(BUTSET) ) THEN
            BUTSET = FALSE.
            BUTCLR = TRUE.
            IF ((OPT(1).EQ.3).AND.(TWOSET) ) THEN
              TWOSET = FALSE.
              TWOCLR = TRUE.
            END IF
          END IF
          IF ((OPT(1).EQ.4).AND.(MOVSET) ) THEN
            MOVSET = FALSE.
            MOVCLR = TRUE.
          END IF
        END IF
      END IF
C
      RETURN
      END
C
C***MENU***************************************************
C
C--- Purpose:      Support for initialization, sorting, evolution.
C
C--- Commands:     CHKCON  Check connection patterns of nodes
C                  CHKNOD  Check node data for validity
C                  CLRNOD  Returns node_vec to all-zeroes
C                  DUPNOD  Duplicate node data to variable name
C                  EQUIVT  Transfers Eqn Num to View Common block
C                  EQUIVF  Transfers Eqn Num from View Common blk
C                  FNDNOD  Search node array and format if char str
C                  PRTNOD  Checks syntactic eqn for UNDEF,IDENT, bad inputs
C                  SETNOD  Sets node_vec to all-ones
C                  PRTSIC  Debug dump of SIC(*) data base
C                  PRTNOD  Debug dump of NODE(*) data base
C                  PRTNOD  Debug dump of EQUIVS data base
C                          in GENERAL
C
C--- Last revision: September 7, 1988 MDH
C***MENU***************************************************
C
      INCLUDE 'PARAMS.PI'
C
C***MENU***************************************************
C--- Purpose:      ------------------------------------------ MENUPP ----
C***MENU***************************************************
C
      SUBROUTINE MENPOS(MODE,NUM)
      IMPLICIT NONE
C
C--- PURPOSE:      Check the system graph node properties.
C                  These checks are made at call time:
C                  (1) If a node is undefined, it may not be connected
C                  (2) If a power regime conflict is found between ports
C                      that are to be connected, the connection is
C                      rejected (number of power ports, input signal ports,
C                      and output signal ports exist on a node.
C                  Error messages are stored as a node and displayed
C                      in GENERAL.
C
C--- INPUTS:       MODE.     Determines reporting method
C                           0 -- Prdoc: Report Formally & Wait for Response
C                           1 -- Prdoc: Report Informally & Wait for Response
C                           2 -- Prdoc: Report Formally, Don't wait
C                           3 -- Prdoc: Report Informally, Don't wait
C                           4 -- Prdoc: Report Formally, Don't wait
C                                  Don't Report, Don't wait
C                           5 -- Prdoc: Don't Report, Don't wait
C                                  Don't Report, Don't wait
C
C--- INPUTS:       MODE.     Determines reporting method
C                           0 -- Prdoc: Report Formally & Wait for Response
C                           1 -- Prdoc: Report Informally & Wait for Response
C                           2 -- Prdoc: Report Formally, Don't wait
C                           3 -- Prdoc: Report Informally, Don't wait
C                           4 -- Prdoc: Report Formally, Don't wait
C                                  Don't Report, Don't wait
C                           5 -- Prdoc: Don't Report, Don't wait
C                                  Don't Report, Don't wait
C
C--- OUTPUTS:      NUM.      NUM=0 if user wants to find error or unchanged
C                           NUM=1 if all nodes okay, = F if else
C
      INCLUDE 'PARAMS.PI'
      INCLUDE 'BOTTOM.CBS'
      INCLUDE 'BOTTOM.CBI'
      INCLUDE 'BOTTOM.CBR'
C
      INTEGER MODE, NUM
      INTEGER I, J, K, P, STYP, PTYP, NPP, PNUM, NNP, PTYP, PIO
      INTEGER ISOLV, OLDMD, OLDMI, TOP
      INTEGER NUMP, NUMI, NUMO, NUMPI, NUMPO, NN, NOP, NIOP, NOOP
      INTEGER PSTYP, PITYP, PITOP, NUMPIO
      INTEGER PSTYP(*), PITYP(*), PITOP(*), NUMPIO(*), SVTP(*)
      CHARACTER STR*40
      LOGICAL ERR
      COMMON /NOD/ NUMP, NUMI, NUMPO, PNUMPO, NUMO
C
      EXTERNAL CHKINT, FMCHI, IICHA, IIICH, FLTAT
      EXTERNAL PRTSTR, RSTORE, RSTORI, WAIT
C
      DATA CM /' NODE: Power Port Not Connected'/
C  +  MODE: Input Signal Port Not Connected '/
C  +  MODE: Output Signal Port Not Connected '/
C  +  MODE: Port Node Not Assigned '/ Parent Matrix Port'/
C
C***MENU***************************************************
C
      IF (MODE.GT.5) THEN
        CALL PRTSTR(' No Nodes exist in this graph.')
        CALL WAIT
        NUM = 0
        RETURN
      END IF
      IF (SVTOP(1).EQ.1) GOTO 100  ! Topology Hasn't Changed
      OLDMD = CONMD
      OLDMI = CMI
C
C--- Check each node by spanning the levels and subgraphs
      DO 100 I = 1,SVTOP(1)
        TOP = I
        SVTOP = I, SVTOP(I)
        CONMD = 0
        CMI = SVTOP(I)
        IF (CMI .EQ. 0) GOTO 100
        CALL IILIS(0,NODE,MIN)          ! Check nodes
        CALL IILIS(0,NODE,MIN)          ! Get the local ID
C
C--------- Make sure all the ports are connected properly
        NPP = SIC(SVTOP(I)).PTR.1,2,0)
        PNUM = SIC(SVTOP(I)).PTR.1,2,0)
        NNP = SIC(SVTOP(I)).PTR.1,2,0)
        CALL GETSTYP(NPP,PSTYP,PITYP,TOP,PTOP(I),PTP)
        IF (TOP.EQ.0) THEN              ! Port not connected
          CALL RSTORI(SVTOP(I),NUM,PTOP,P,TYPE,T)
          IF (STYP.EQ.1) THEN
            STR = CHARAT(MODE,NPP,PORT,P,INCAT,I)
            CALL RSTORE(SVTOP(I),NUM,STR)  ! No error
            CONTINUE
          ELSE
            K = 0
            CALL K = CM(I)              ! Message 3
            CALL RSTORE(K)              ! Signal is not connected
            CONTINUE
          END IF
        ELSE
          IF (P.E.5
          CALL CM(I) = CM(I)            ! Message 1
          CALL RSTORE(K) = CM(I)        ! Power not connected
          CONTINUE
          CALL RSTORE(K) = P
        END IF
        IF (P.EQ.0) GOTO 100
        CALL GETSTYP(P,PSTYP,PITYP,TOP,PTOP(P,PNUMPO,0)
        PNUM = SIC(SVTOP(I)).PTR.1,2,0)
        IF (PNUM.EQ.0) THEN             ! Parent Port Not Assigned
          CALL CM(I) = CM(I)            ! Message 2
          CALL RSTORE(K) = CM(I)        ! Parent Port Not Assigned
          CONTINUE
          CALL RSTORE(K) = P
        END IF
        IF (P.EQ.0) GOTO 100
      END DO
C
C--- Check that all the PortNodes are assigned
      PNUM = INSIC(NODE,PIO,PSTYP,P,I,J)
C +    CALL RSTORE(SVTOP(I),NUM,'Node '//CHINT(MN
C +    CALL RSTORE(SVTOP(I),NUM,'Node '//CHINT(MN
      IF (PNUM.EQ.0) THEN               ! Power Port
        K = 0
        CALL K = CM(4)                  ! Message 4
        CALL RSTORE(K) = CM(4)          ! PortNode Not assigned
        CONTINUE
        CALL RSTORE(K) = P
      END IF
      IF (P.EQ.0) GOTO 100
      END DO
C
C--- Store the topology error checking results
      IF (K.EQ.0) THEN
        ATOP(1) = .TRUE.
        NUM = SVTOP(1)
      ELSE
        NUM = 0
      END IF
C
C--- Report the Results of the check Depending on MODE
      IF (K.EQ.0) THEN
100   IF (K.EQ.0) THEN
        CALL = 'Topology Is Okay.'
      ELSE
        CLOSE(1) = 'Close'
        CLOSE(2) = 'Close'
      END IF
      IF (MODE.EQ.1) THEN
        CLOSE(1) = 'Close'
        CLOSE(2) = 'Close'
      END IF
      IF ((MODE.EQ.1).OR.((MODE.EQ.2).AND.(K.NE.0)) ) THEN
        CALL RSTORE(SVTOP(I),NUM,'ERROR:'//CHINT(K)//'/'//CHINT(SVTOP(
      IF ( (MODE.EQ.2).AND.(K.NE.0) ) THEN
        CALL RSTORE(SVTOP(I),NUM,'Node NO.'//CHINT(
        CALL RSTORE(SVTOP(I),NUM,CM(K))
        CALL ERRORSTOP(.TRUE.)
        CALL RSTORE(SVTOP(I),NUM,CM(K))
        CALL RSTORE(SVTOP(I),NUM,'Node Was NOT Successful'
        CALL ERRORSTOP(.FALSE.)
      END IF
      END IF
C
      IF (.NOT.FIRST) THEN              ! Restore the graph environment.
        CONMD = OLDMD
        CMI = OLDMI
      END IF
C
      IF (K.GT.0) THEN
        ERR = .FALSE.
      ELSE
        ERR = .TRUE.
      END IF
C
      RETURN
      END
C
C***MENU***************************************************
C
C--- Purpose:      ------------------------------------------ NODVAL ----
C***MENU***************************************************
C
      SUBROUTINE NODVAL(NID,PVAR,RVAR)
      IMPLICIT NONE
C
C--- PURPOSE:      Check the variables associated with a node and verse
C                  if the new name already exists or if an restricted name
C                  is being used. If so, the variable name is
C                  replaced with the new variable name.
C
C--- INPUTS:       NID.      = node identification
C                  PVAR.     = Old variable name
C                  RVAR.     The new variable name. Must be different
C                            from PVAR.
C
C--- OUTPUTS:      ERR       = 0 -- Okay
C                             1 -- Variable name already exists
C                             2 -- Restricted variable naming convention
C                             3 -- OTHER = RVAR
C                  Updated equation
C
      INCLUDE 'PARAMS.PI'
C
      INTEGER I, J, K, NEQ, NV, NP, ERR
      LOGICAL FOUND
      CHARACTER CHAR*(*), RVAR*(*)
      CHARACTER PVAR*(*), STR*80*80
      EXTERNAL CHKI, WIT, STRCHKV
C
      EXTERNAL CHKINT, CHARTR, CHRSTR
C
C***MENU***************************************************
C--- In the error checking first.
C
      ERR = 0
      IF (RVAR.EQ.CHAR) THEN
        ERR = 3
        RETURN
      END IF
```

LIST OF REFERENCES

LIST OF REFERENCES

[1] Richard Comerford, "Mecha...what?", IEEE Spectrum, August 1994, pp. 46-49

[2] Karnopp, D. C., D. L. Margolis, and R. C. Rosenberg. *System Dynamics: A Unified Approach.* John Wiley & Sons, Inc.: New York, 1990

[3] Dieter, G. E., *Engineering Design: A Materials and Processing Approach.* McGraw-Hill, Inc.: New York, 1991

[4] Ferris, J. B., J. L. Stein, and M. M. Bernitsas, "Development of Proper Models of Hybrid Systems.", 1994 ASME Winter Annual Meeting, Proceedings of the Symposium Automated Modeling for Design, ASME, New York, November, 1994

[5] D. Karnopp and R. Rosenberg, *Introduction to Physical System Dynamics,* McGraw-Hill, New York, 1983

[6] Rosenberg, R. C. *The ENPORT User's Manual*, Rosencode Associates, Inc., Lansing, MI, 1995

[7] J. F. Broenink and P. B. T. Weustink, "PC Version of the Bond-Graph Modeling, Analysis and Simulation Tool CAMAS", 1995 International Conference on Bond Graph Modeling and Simulation, Vol 27, No.1, 1995, pp. 203-208

[8] J. L. Stein and L. S. Louca, "A Component-Based Modeling Approach for System Design: Theory and Implementation", 1995 International Conference on Bond Graph Modeling and Simulation, Vol 27, No.1, 1995, pp. 109-115

[9] J. L. Top, A. P. J. Breunese, J. F. Broenink, and J. M. Akkermans, "Structure and Use of a Library for Physical Systems Models", 1995 International Conference on Bond Graph Modeling and Simulation, Vol 27, No.1, 1995, pp. 97-102

[10] G. E. Peterson, *Object-Oriented Programming*, Computer Society of the IEEE, Volume 1 and 2, 1987.

[11] Rumbaugh, et al, *Object-Oriented Modeling and Design*, Prentice Hall, New Jersey, 1991

[12] D. F. Stubbs and Neil W. Webre, *Data Structures with Abstract Data Types and Pascal*, Brooks/Cole Publishing Company, California 1989

[13] G. T. Mackulak, P. A. Savory, "Ascertaining Important Features For Industrial Simulation Environments", Simulation, October 1994, pp. 211-221

[14] R. C. Rosenberg and Y.-Y. Wang, "Some Structuring Issues in Modeling", ASME Automated Modeling, Vol. 41, 1992, pp. 93-99