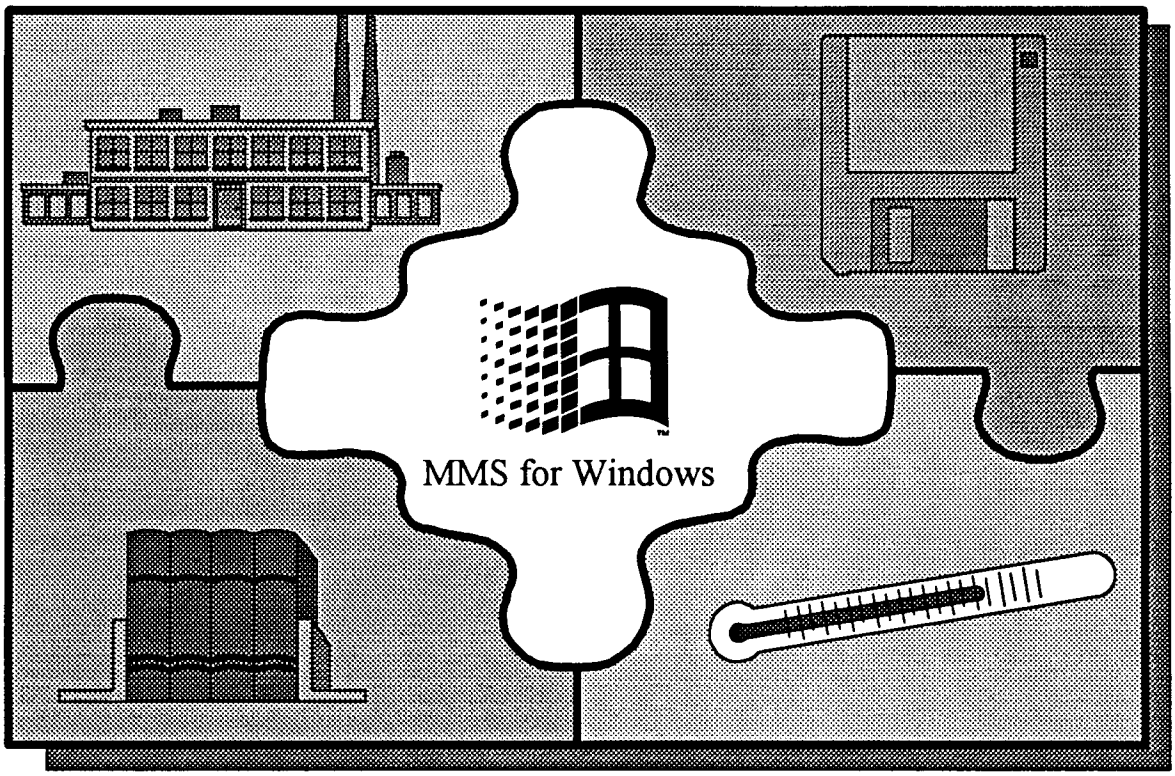
An Object-Oriented Approach to Grahpics Interaction With The Modular Modeling System

Charlie Alan Jones

B&W Nuclear Service Company



An Object-Oriented Approach to Graphical Interaction with the Modular Modeling System

By

Charlie Alan Jones

B&W Nuclear Service Company
Engineering Services
Lynchburg, Virginia, 24506

March 1992

An Object-Oriented Approach to Graphical Interaction with the Modular Modeling System

Charlie Alan Jones
B&W Nuclear Service Company
Engineering Services
Lynchburg, Virginia, 24506

ABSTRACT

This paper presents the construction of a graphical interface for the Modular Modeling System, a pre-existing modular library used for modeling power plant dynamics. Application of object-oriented programming methods has produced a superior end-user simulation development environment, reduced program development time, and provided a paradigm for data abstraction in the interface. Use of these methods, at the graphical level, demonstrates the use of data encapsulation and complexity hiding that can be applied to many programming problems. Similar methods show how object-oriented programming is applied in a nontraditional sense to object generalization. Code maintenance, automation, and program operation are discussed.

INTRODUCTION

Object-oriented technology has an inherent relationship with graphical representation and physical models. Today the trend in program development is to make use of object-oriented technology to reduce development time and improve maintainability. This technology has been applied to the Modular Modeling System[1] to provide a more useful and productive power plant simulation environment.

The Modular Modeling System (MMS) includes a library of algebraic and differential equations representing over 160 power plant components that are programmed as macros for use in a simulation language. The MMS permits rapid construction and modification of dynamic models for a wide variety of power plant systems by the invocation of these macros from the library while following certain connection conventions. Model construction is accomplished by creating a text file using the desired MMS macros. Adherence to the connection rules and macro conventions is essential to the construction of working models. Parameters required for the macros must be calculated manually based worksheets provided in the MMS documentation.

Recently available products, designed for use in the personal computer market, have provided the development tools and operating systems necessary to equip MMS with a comprehensive graphical user interface. Evaluation of current technology resulted in the selection of Microsoft Windows[2] as the graphical user interface platform. A Windows version of the Advanced Continuous Simulation Language[3] (ACSL), the simulation language used by MMS, recently became available. The source coding for *MMS for Windows* is written in a combination of the C++ programming language for the graphical interface and FORTRAN for engineering calculations. The new MMS interface will be called *MMS for Windows*.

The major advantages of Microsoft Windows are multi-tasking, device-independent graphics, and advanced memory management. Having the ability to run models concurrently with the MMS

graphical interface allows a close association between development and model execution.

The original concept for rapid and flexible model construction by using a library of component modules has been extended by providing an easy-to-use, object-oriented graphical user interface. Connection and macro conventions are now handled by the graphical user interface instead of by the user; thereby eliminating many common user errors such as mismatched identifiers. Parameters are entered through input data forms, and worksheet calculations are performed automatically.

The MMS library is ideally suited for graphical representation because of its modularity. Each macro in the library is represented by a unique graphical icon having ports that allow for icon interconnection. The result is a schematic graphical representation that is true to both the physical system being modeled and the MMS macro library being used.

Figure 1 shows a model of a typical plant system. Each icon has a one-to-one relationship with a macro in the MMS library. As modules are placed on the screen, operating point parameters are entered by the user. Once the model is built and parameterized, it can be translated into FORTRAN by the ACSL translator. The FORTRAN file can be compiled and linked to create an executable file. Execution of the model is controlled by a runtime executive with a command line interface supplied with the ACSL tools.

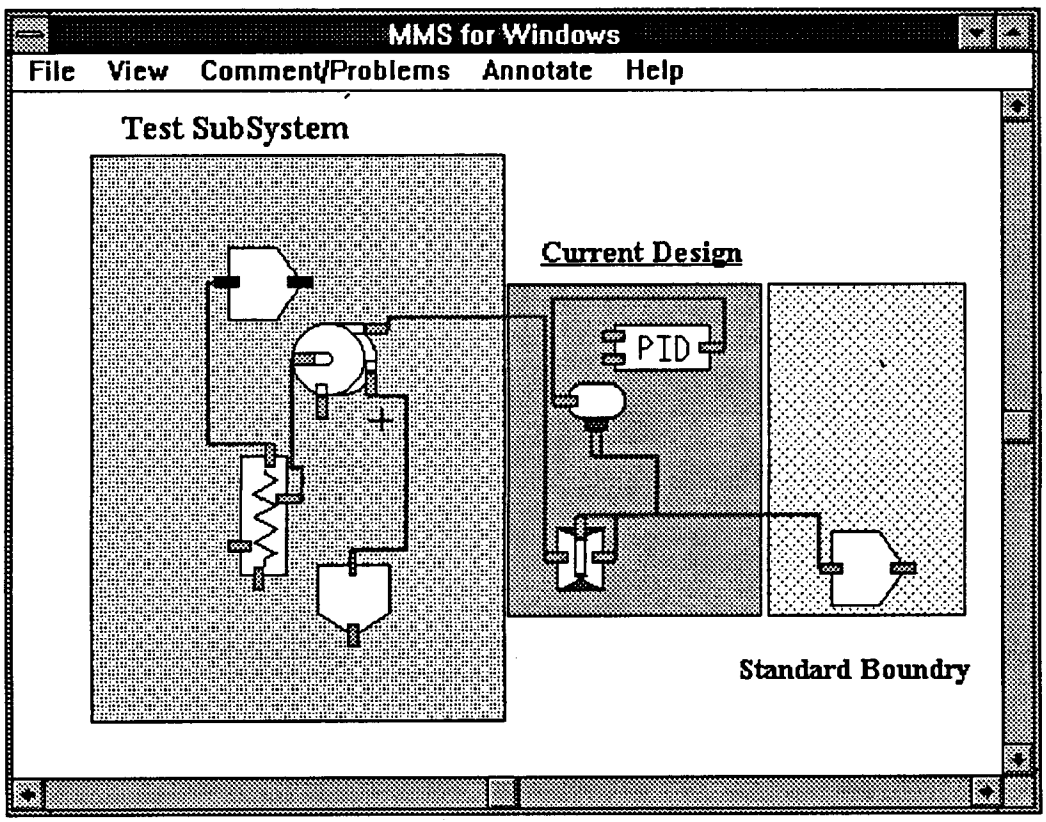


Figure 1. Typical "MMS for Windows" Model Representation

PROGRAM CONSTRUCTION

The programming methodology used in the development of *MMS for Windows* takes advantage of the inherent object-orientation of the MMS library. Several of the major features that define object-oriented programming and how they are used to advantage in the *MMS for Windows* are described below. For a definition of terms, see the *Table of Definitions* after the references.

Encapsulation

Encapsulation is a process in which data and executable code are packaged in the same class. The advantage of encapsulation is that each object created from that class operates on, and protects data in, that object. When changes to data structures or code are required, only the object is affected. The main program does not need to be changed. Encapsulation of data and executable functions in classes, as allowed by C++, permits the creation of a consistent set of functions that operate on objects with different characteristics. Behavior of functions on the objects is based on each object's own knowledge of itself. As an example, each object on the screen is responsible for drawing itself based on its own knowledge of its location, size and type.

MMS for Windows takes maximum advantage of the data manipulation abilities of C++. Each graphical object is based on one of several classes. Three types of objects are currently defined in the program: module icons, connectors, and annotation text. All classes share common object maintenance functions, including linked lists, constructors, destructors, and abstract functions[4].

Complexity Hiding

Benefits of advanced memory management features, as found in Windows, can be offset by the complexity of their use. Complexity hiding in the member functions of the object classes greatly simplifies the use of these memory management features[5].

An example of complexity hiding, as used in the *MMS for Windows*, is the conservation of local heap space. Local heap space is a limited resource (64K Bytes) that is maintained by each Windows program instance. By default, data inside the classes uses local heap space. A more plentiful resource is the global heap (limited only by machine configuration). Use of the global heap requires allocation, retention of a memory handle, locking, unlocking, and freeing of the global object. Placing these functions inside the class definitions removes this complexity from the main program. Use of the global heap permits each graphical object to consume a smaller part of the local heap. This provides significantly improved memory management and program execution.

The Windows operating environment provides its own opportunities for complexity hiding[6]. Custom controls is one type of complexity hiding used by *MMS for Windows*. Many of the MMS macros require tables of data, such as pump head curves, to model components. Figure 2 shows the custom control in a dialog box for the graphical editing of table data. The control is handled by the operating system in the same manner as standard Windows controls such as buttons, list boxes and check boxes. In this way, the control complexity is completely separated from the main program. Another advantage of custom controls is that any application can use the control and, in effect, provide an extension to the Windows operating environment.

The table edit control provides a easy-to-use graphical method of entering table data. The shape of the curve can be editing by clicking on the control with the mouse. Exact values can be entered

through a pop-up dialog box (not shown).

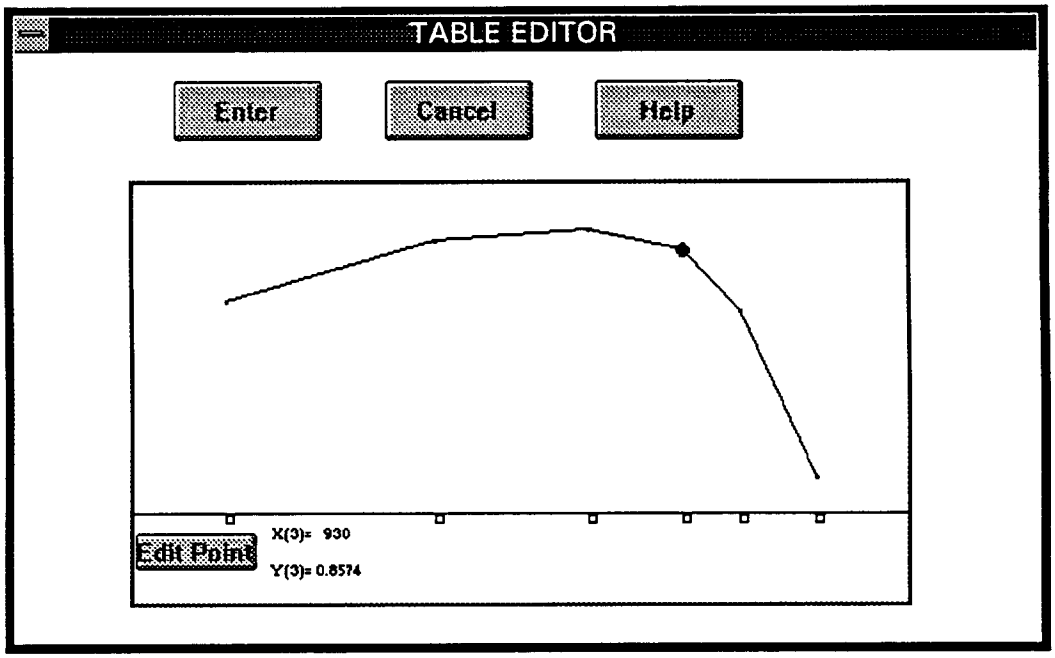


Figure 2. Custom Control for Table Edit

Object Generalization

When completed, *MMS for Windows* will contain more than 100 module icons, including power plant component modules to support both fossil and nuclear plant simulations from different sub-libraries with numerous connection conventions. Specific code for the functions of each of these modules would require considerable effort. This effort has been greatly reduced through abstraction.

Abstraction is the ability to manipulate objects in a uniform manner when the objects are of different types or possess unique data. Abstraction leads to general methods for the control and display of all objects in the program. Generalization of the objects has been made more effective with the use of behavior dictionaries and rule databases. These dictionaries influence every major function of *MMS for Windows*, including object creation, display, writing to model files, reading from model files, form control, ASCII import, ASCII export, and ACSL source code generation. A graphical feature of Windows is the ability to save graphical instructions (Metafile) as a set of rules that can be played back when needed. This feature not only aids in generalization, but also allows for rotation of the icons. Other dictionaries supply data on data structure, connection conventions, ACSL source code generation rules, and context-sensitive help information.

OVERALL INTEGRATION

Figure 3 shows the interaction of each of the parts of *MMS for Windows*, several of the features used, and their relation to the basic philosophical approach. The following section discusses these components.

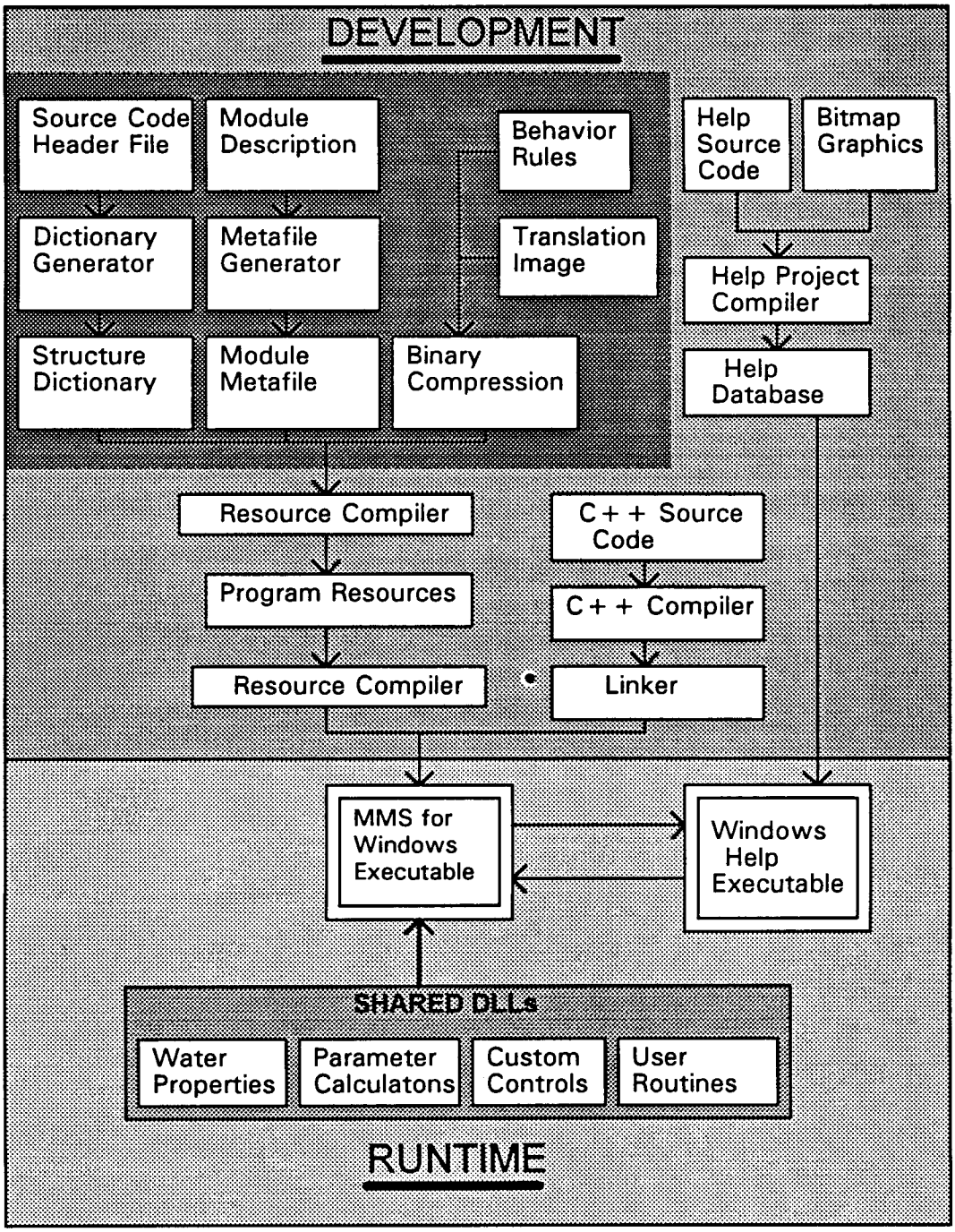


Figure 3. Development-Runtime Organization

Dynamic Link Libraries

Dynamic Link Libraries (DLL) provide a way to share code among several different programs and multiple instances of the same program. *MMS for Windows* uses DLLs for complexity hiding of the action for the table edit custom control.

Water property subroutines need to be shared by the parameterization functions in the *MMS for Windows* and by the executable programs generated by ACSL using the MMS macro library of component modules. A significant amount of memory overhead, both on the hard disk and in computer RAM, is conserved by this method. Whenever possible, DLL technology has been employed.

Program Resources

Fixed dictionaries and data files, known as program resources, are stored in a special format in the executable file. Use of resources allows for reduction in the number of files required and provides for closer integration of the application with the operating system. Memory management complexity is hidden through the use of resource features.

Graphics

Windows provides a set of device-independent graphics routines that have numerous benefits, such as a reduction of the total number of rules needed to describe each module. The graphics system also supports a rule-based play and record system that is compatible with the program resource feature. Rapid recall, rotation, and display of modules are made possible with this system. Included in the graphical functionality is the ability to resize and overlap windows, zoom, pan, and scale. Device-independent graphics has the additional benefit of no software maintenance of *MMS for Windows* to support advances in graphics hardware.

Code Maintenance

Encapsulation and generalization are already recognized as methods of reducing software maintenance efforts. As an example, if a change is made to the way in which the global heap is managed in future versions of Windows, changes would have to be made only to member functions in the object class. All changes are localized to one part of the source code.

Figure 3 shows the development area that builds the behavior and data dictionaries. Input for the programs is based on the C++ source code for the main program whenever possible. This automates the maintenance of the structure definition dictionaries whenever the structure source code is modified. The reading of the source code by the dictionary generator eliminates the need for manual entry of the data structures in the dictionaries.

User Configuration

User configuration is possible through the use of Dynamic Link Library and dictionary modification. Since the *MMS for Windows* permits the user to define his own module behavior, the user can provide a replacement for the default parameterization calculations. Any compiler compatible with the DLL format can produce a parameterization DLL.

Choice of color and style for objects is user selectable. Annotation font, size, color, and box attributes (if any) are controlled by a dialog box selection.

PROGRAM OPERATION

Users with Windows experience will be comfortable with the user interface of *MMS for Windows*. Because the *MMS for Windows* adheres to the Microsoft Windows Style Guide standards set forth for Windows programs, ease of use and training are simplified. The *MMS for Windows* includes a complete, Windows-compliant, on-line help facility using hypertext and graphics.

Module Placement and Connection

Placement of a new module on the graphical worksheet starts with a double-click anywhere on the *MMS for Windows* worksheet. Figure 4 shows the dialog box for module selection. An identification name can be entered and the module orientation can be selected. The enter button is selected when the user has made his selection.

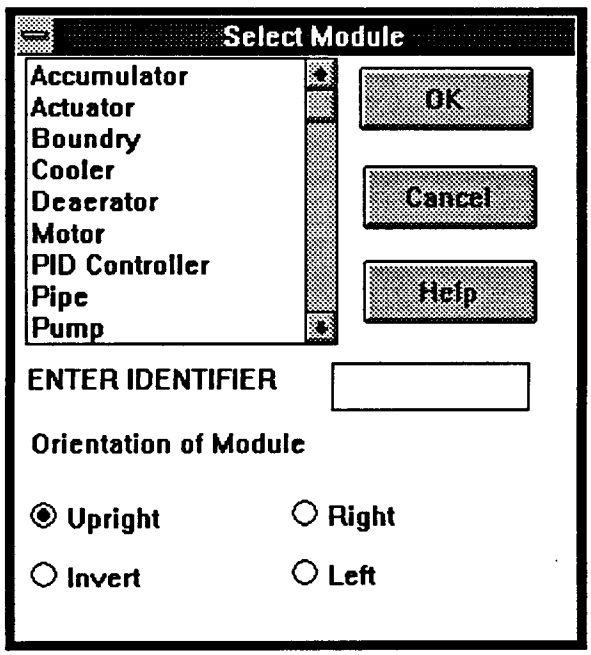


Figure 4. Module Selection Dialog Box

Connections begin when the user double-clicks on the port of a module icon. Figure 5 shows the "rubber band" line that remains connected to the initial port. Double-clicking on another module icon port completes the connection. When the initial port is selected, ports having the proper connection convention are highlighted. Connector lines are placed using an automatic routing algorithm.

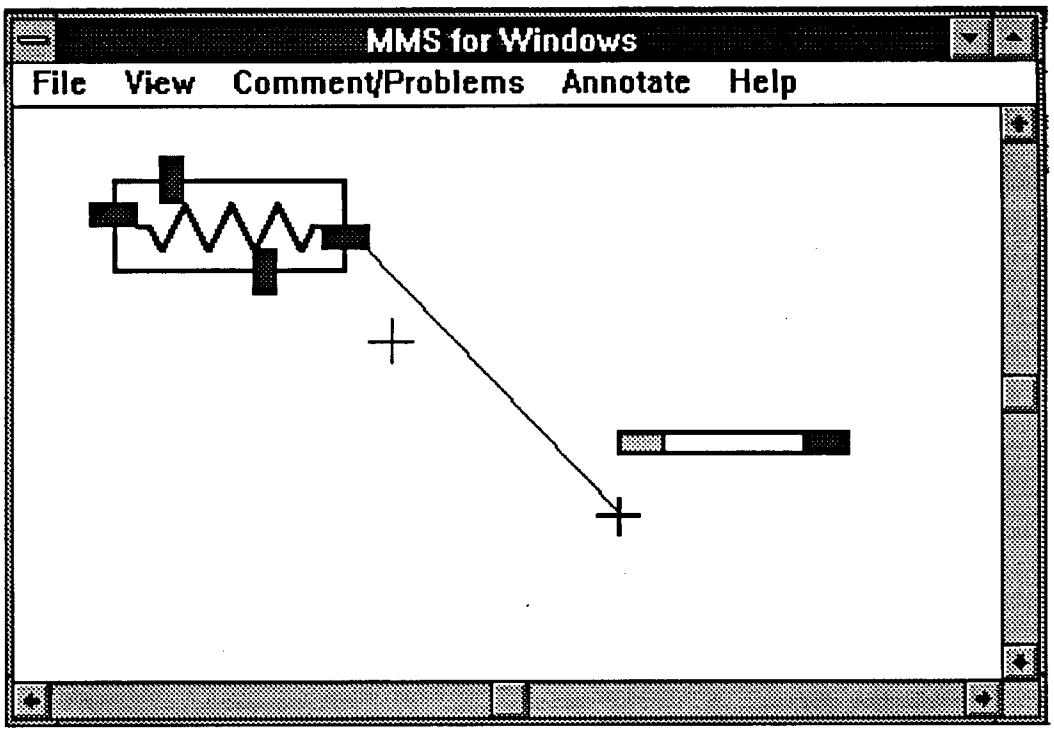


Figure 5. Rubber Band Connect Mode

Parameter Data Input and Calculations

Entry of parameter data is a significant part of the effort in creating a model for simulation. Use of the rule and behavior dictionaries makes it possible to provide a uniform data entry method of every module. Figure 6 shows the module input data dialog box that is a generic input interface depending solely on the encapsulated object data and the dictionary rules for the specific module type.

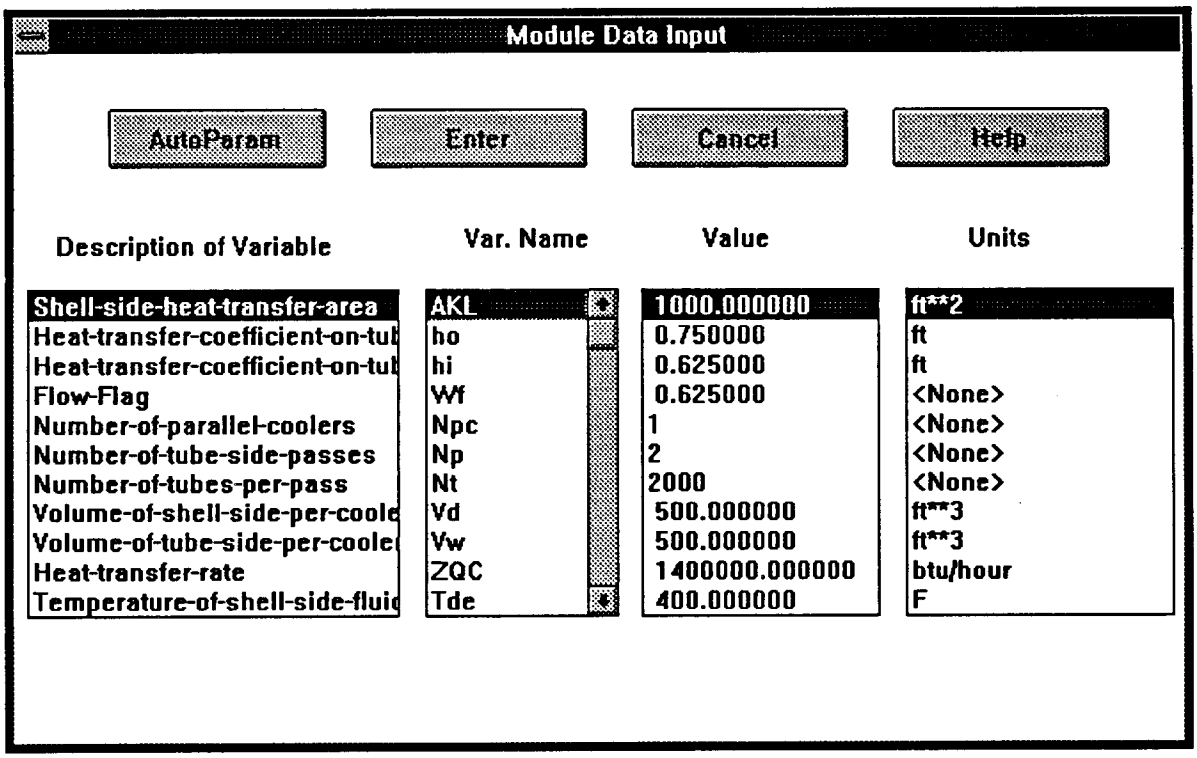


Figure 6. Module Input Data Dialog Box

The descriptions and variable names in the dialog box match the name used in the MMS macros and user worksheets. When a user selects a variable name from the scrollable list box, a dialog box based on the variable type appears. A check box appears for a logical value, an edit line for a floating point or integer value, or whatever type of dialog box is appropriate for the variable type. Figure 7 shows a multi-state variable that is controlled through the standard Window "Radio Button" controls.

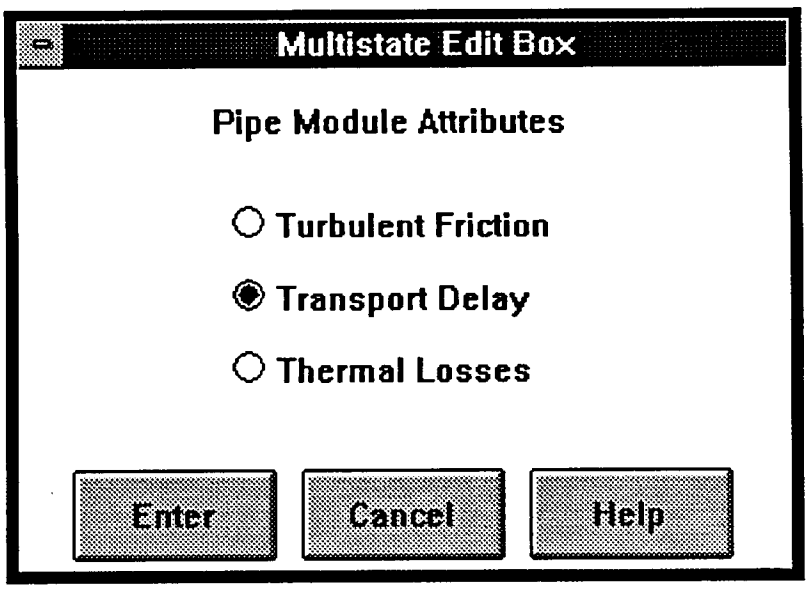


Figure 7. Multi-stage Edit Dialog Box

Creation of ACSL Source Code and Model Execution

Parameter calculations for any module can be made from the input dialog box as parameters are entered. This allows the user to review results of the calculations. The user can make a selection from the main menu to perform the calculations for all modules.

Data in the objects and data dictionaries is converted into ACSL source code using information in the rule and behavior dictionaries. These rules are based primarily on variable names within the objects. Module-specific and special purpose rules can also be defined.

The ACSL translation program converts the ACSL source code into FORTRAN. Compilation and linking of the FORTRAN file produces the final executable file. Under the Windows environment, the executable program can be run concurrently with the *MMS for Windows* in the model building mode.

SUMMARY

The combination of a graphical operating system and object-oriented tools has produced a profound improvement in the way MMS models are built. The *MMS for Windows* provides a much improved method of generating ACSL source code compatible with the MMS macro library. Module representation in the Windows environment is based on each library macro being represented by a graphical icon. Object-oriented programming techniques have provided a significant reduction in development time. Object generalization is enhanced by the use of behavior and data dictionaries.

REFERENCES

1. B&W Nuclear Service Company, MMS Reference Manual, P.O. Box 10935, Lynchburg, VA, USA 24506-0935, 1988.

2. Microsoft is a registered trademark and Windows is a trademark of Microsoft Corp.

3. Mitchell & Gauthier Associates, *ACSL Reference Manual*, 200 Baker Avenue, Concord, MA, USA 01742-2100, 1991.

4. Stroustrup B., *The C++ Programming Language - Second Edition*, Addison-Westly, 1991, pp. 391-439.

5. Smith J.D., *Reusability & Software Construction*, Wiley, 1990, p. 8.

6. Van Camp D.,"Inherit the Win," BYTE , McCraw-Hill , September 1991, pp. 325-389.

Table of Definitions

Abstraction - A common method for dealing with complex systems is to try and break the system into manageable subsystems. Constructing a program with components that have been designed to be similar, having common traits, yet can contain unique information, is abstraction. In this way, maximum advantage is made of coding that is designed to operate only on data common to many objects.

Class - A C++ term for the base definition of objects in a program. New classes can be built on all or part of the definition of lower classes. This is known as inheritance.

Complexity Hiding - The ability to hide the details of a complex action from the main program. Subroutines are a similar concept in non-object-oriented programming.

Constructor - When a new object is created based on a class, a constructor function is called. This allows the programmer to assign default data to the object.

Destructor - When an object is deleted, a destructor function is called. Usually, this is to return any additional memory the object may be holding to the main program.

Encapsulation - An object-oriented programming term for defining data and functions inside of a object definition. In C++, encapsulation is the design of the class structure.

Heap - Memory is set aside in programs for the storage of program data on a dynamic basis during program execution. As an example, sorting may use a large amount of heap space. How heaps are organized for a program is based on the hardware, operating system, and the compiler technology.

Member Functions - Functions that are placed inside (Encapsulated) class definitions are know as member functions.

Program Instance - The Windows operating system permits more than one copy of a program to be executed concurrently. Each copy of the program is an instance.