CONF-890555--2

# AN INTELLIGENT SIMULATION ENVIRONMENT FOR CONTROL SYSTEM DESIGN*

James T. Robinson

Engineering Physics and Mathematics Division
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, Tennessee 37831-6364

CONF-890555--2

DE89 008744



Invited Presentation: 7th Power Plant Dynamics Control and Testing Symposium, Knoxville, Tennessee, May 15–17, 1989

* Research sponsored by the Advanced Controls Program of the Office of Reactor Technologies Development, of the U.S. Department of Energy, under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

# AN INTELLIGENT SIMULATION ENVIRONMENT FOR CONTROL SYSTEM DESIGN

James T. Robinson
Oak Ridge National Laboratory
Oak Ridge, TN 37831-6364

## ABSTRACT

The Oak Ridge National Laboratory is currently assisting in the development of advanced control systems for the next generation of nuclear power plants. This paper presents a prototype interactive and intelligent simulation environment being developed to support this effort. The environment combines tools from the field of Artificial Intelligence; in particular object-oriented programming, a LISP programming environment, and a direct manipulation user interface; with traditional numerical methods for simulating combined continuous/discrete processes. The resulting environment is highly interactive and easy to use. Models may be created and modified quickly through a window oriented direct manipulation interface. Models may be modified at any time, even as the simulation is running, and the results observed immediately via real-time graphics.

## INTRODUCTION

The capabilities and limitations of current computer aided control system design and simulation software are well documented in the literature (see for example Rimvall 1988). For the most part, current software emphasizes linear analysis as automated methods for linear systems have been available for some time. For non-linear systems and rule-based systems (such as fuzzy control), general automated design methods are not available and development depends heavily on simulation. However, the potential benefit of simulation as a design tool has not been fully realized in the past due to the high costs of developing and maintaining models. This paper describes a prototype simulation environment which has been designed explicitly for exploratory modeling

of power plant processes and control systems. The ease of use, flexibility, and interactive nature of the environment should allow simulation to be incorporated more completely into the design process.

## DESCRIPTION

The environment was originally developed on a LMI Lisp Machine using the FLAVORS object-oriented language (Stallman, Weinreb, & Moon 1984) and has recently been installed on a Macintosh-II based Texas Instruments microExplorer. The advantages of the LISP programming environment for developing advanced simulation systems have been noted by other researchers (Stairmon and Kreutzer 1988, Pliske and Halley 1988). LISP environments were developed for exploratory programming and most feature an interpreter (and/or incremental compiler) and many built in tools for advanced user interfaces. The use of an interpreted language such as LISP frees the user from the tedious compile-link-load cycle. This is especially important during the design process, when many alternative plant and control system configurations may be explored.

Object-oriented programming has been successfully applied to discrete-event simulations by a number of researchers (for example Khlar 1986; Ziegler 1987; Stairmong and Kreutzer 1988; Ghaznavi-Collins and Thelen 1988). However, its extension to a predominantly continuous process, such as a power plant, is not straightforward due to the tightly coupled nature of such systems. In particular, simulation by direct message passing between component level objects, as in a round-robin scheme, leads inevitably to numerical instabilities. Our approach to deal with this problem has been to introduce special classes which implement traditional numerical integration algorithms for simulating groups of interconnected objects. This is described below.

### Class library

The core of the simulation environment consists of a library of class definitions. These classes may be categorized into the following levels of abstraction:

(1) component-level classes,

(2) physical-system classes, and

(3) numerical-methods classes.

At the lowest level are component-level classes. These are the basic building blocks from which simulation models are constructed. They may represent either actual plant components (such as pumps, pipes, and valves) or abstractions such as heat conductors, flow sources, and time delays. FLOW-CONTROL-VALVE is a good example of a component level class. This class is a member of the VALVE family and inherits the following instance variables from the GENERIC-VALVE class:

*inlet-connection*
*outlet-connection*
*valve-position*
*flow-rate*
*pressure-drop*

The variables *inlet-connection* and *outlet-connection* specify the classes' ports and are used to record the names of connected objects. The last three instance variables are dynamic in nature and specify the instantaneous state of the valve. In addition to these inherited instance variables, FLOW-CONTROL-VALVE includes a parameter related to the flow capacity of the valve, *Cvmax*. The methods for FLOW-CONTROL-VALVE include accessor functions for all instance variables as well as *:compute-flow-rate* and *:compute-pressure-drop*.

During an actual simulation, interconnected component-level objects are automatically collected and treated as a group by objects of the higher level *physical-system* classes. This group includes THERMAL-NETWORK, which solves the heat conduction/convection equations for temperature, and HYDRAULIC-LOOP, which solves a loop momentum equation for flow rate. These classes inherit integration methods from *numerical-methods* classes such as LINEAR-SYSTEM and NON-LINEAR-ODE.

THERMAL-NETWORK is an example of a *physical-system* class. Objects of this class solves the energy conservation equations (heat

conduction and convection) for groups of thermally connected objects. These equations are represented in the form

$$\frac{d\vec{T}}{dt} = A(t)\,\vec{T} + \vec{f}(t)$$

where $\vec{T}$ represents the vector of temperatures, A(t) is a time-varying coefficient matrix, and f(t) is a time varying forcing function. The instance variables of class THERMAL-NETWORK include:

*components*
*A-matrix*
*f-vector,* and
*T-vector.*

At instantiation an object of class THERMAL-NETWORK automatically creates and initializes the matrix A and vectors $\vec{f}$ and $\vec{T}$ of the appropriate dimensions and assigns a row index number to each component. At each time step the THERMAL-NETWORK object sends messages to all components instructing them to update their slot(s) in the coefficient matrix A or forcing vector $\vec{f}$, and then advances the resulting equation system one time step according to the integration method inherited from the class LINEAR-SYSTEM.

It should be noted that the user does not have to deal directly with objects of the physical-system or numerical-methods classes. They are automatically constructed prior to a simulation run based on the connections between component-level objects and remain transparent unless the user chooses to modify them.

## User Interface

The user-interface is highly interactive and makes use of multiple windows, pull-down menus, icons, etc. A model is constructed by placing icons on the screen and specifying interconnections with the aid of a mouse. Connections are made by specifying a series of horizontal and vertical line segments between ports by dragging and clicking the mouse. Ports and connecting lines are automatically aligned. Connections between ports of unlike type are protected against

Parameters for individual objects are usually entered and/or changed through forms accessed by choosing edit-object from a pull down menu and selecting an objects icon with the mouse. A form for a controller of type SISO-CONTROLLER is illustrated in Figure 1. The forms provide current access to an objects instance variables and may be used to inspect or modify an object during a simulation run. The effects of the change are reflected immediately in the simulation.

Hierarchial zooming. To help manage system complexity the interface implements a hierarchial zooming concept similar to that presented by Elmqvist and Mattson (1986) which allows individual objects to be arranged into a hierarchy of systems and subsystems. A system is simply a collection of objects (which may include other systems). When collapsed it is represented by a rectangular box which may be manipulated as any other object. However, the edit-object operation opens a window into the system rather than a form for editing its attributes. System objects may (and usually do) have ports. These ports are visible on the system icon but in reality belong to one of its components. They are created by connecting a component of the system to a special SYSTEM-PORT object. This causes a port of an appropriate type to be created automatically and made visible upon viewing the system from the next higher level. This port possesses all the characteristics of the original object's port to which it is connected and will accept appropriate connections from other systems or objects. This allows SYSTEM objects to be interconnected freely with component-level objects.

The use of hierarchial zooming is illustrated in Figure 2. Figure 2(a) contains three system boxes, MODULE-A, MODULE-B, and BOP (these correspond to two power blocks of a modular liquid metal reactor and the balance of plant). Editing MODULE-A would open a window into that system as illustrated in Figure 2(b). This system is composed of a number of interconnected component-level objects and yet another system, IHX-A. Note that the icon at the lower right, which represents the steam generator SG-A, is connected via input and output ports to a SYSTEM-PORT object. Thus these ports are visible on the system icon for MODULE-A.

The purpose of the hierarchial organization scheme is to allow the concentration of modeling effort on individual systems. To aid in this, object names are made relative to the system in which they are defined

This permits the duplication of names from system to system without conflict.

Scripts. A simulation scenario is prescribed through the use of scripts, accessed through the SIMULATE pull-down menu. The INIT script contains a series of messages which are sent upon simulation initialization. The RUN script is a series of messages which are repeated each time step. This script may be used to prescribe the default behavior of the plant or an entire simulation scenario. The run script is analogous to the main program of a traditional simulation, with the important difference that it may be changed at any time (even during a simulation run) without recompilation.

Output. Simulation results are observed through gauges or strip charts. These objects may be connected to the interior of any object and may display any of its parameters. During a simulation, they are expanded and displayed on a special output window. The user interface as it appears during a typical simulation is illustrated in Figure 3. This is a typical layout with the menubar at the top, a process schematic window on the left, and an output window on the right.

## File System Interface

The permanent storage of models is achieved through the file system interface. The options available are SAVE-MODEL, SAVE-SYSTEM, LOAD-MODEL, and LOAD-SYSTEM. The SAVE-MODEL command saves all systems of the current model along with the INIT and RUN scripts. The SAVE-SYSTEM command saves the current system (the system whose window is currently active) and all its subsystems. The SAVE-MODEL and SAVE-SYSTEM options may be executed at any time, and saves each object of the model or system in its current state. This allows simulations to be frozen in their current state and later recreated with the LOAD-MODEL or LOAD-SYSTEM commands.

## CONCLUSIONS

This paper describes a prototype interactive and intelligent simulation environment being developed to support the development

through exploratory modeling of advanced control systems for power plants. The environment combines tools from the field of Artificial Intelligence; in particular object-oriented programming, LISP programming environments, and direct manipulation user interfaces; with traditional numerical methods for the simulation of continuous systems. This synergistic approach appears to us to be a promising alternative to traditional programming techniques for developing advanced simulation environments.

## REFERENCES


Elmqvist, H. and Mattsson, S.E. 1986, "A Simulator for Dynamical Systems Using Graphics and Equations for Modeling," Proceedings of the Third Symposium on Computer-Aided Control System Design, Arlington, VA, September 24-26, 134-140.

Ghaznavi-Collins, I. and Thelen, D. 1988, "An object oriented approach toward system architecture simulation," AI PAPERS,1988 (R.J. Uttamsingh ed.) *Simulation Series* (20) 4. SCS, San Diego, CA. 103-107.

Khlar, P. 1986. "Expressibility in ROSS, an Object-Oriented Simulation System," Artificial Intelligence in Simulation , (G.C. Vansteenkiste; E.J.H. Kerchoffs; B.P. Zeigler eds.). SCS, San Diego, CA, 147-156.

Pliske, D.B, and Halley, M.R. 1988, "Queing Lab - A Workflow Modeling System for Decision Support," AI PAPERS,1988 (R.J. Uttamsingh ed.) *Simulation Series* (20) 4. SCS, San Diego, CA, 42-46.

Rimvall, M. 1988, "Computer-Aided Control Systems: Techniques and Tools," in Systems Modeling and Computer Simulation (N.A. Kheir, ed.), Marcel Dekker, Inc., New York, 631-679.

Stairmong, M.C. and Kreutzer, W. 1988, "POSE: a Process-Oriented Simulation Environment embedded in SCHEME," Simulation (50)4, 143-153.

Stallman, R., Weinreb, D. and Moon, D. 1984, Lisp Machine Manual. Lisp Machine Incorporated, Los Angeles, CA.

Zeigler, B.P. 1987, "Hierarchical, modular discrete-event modelling in an object-oriented environment," Simulation (49)5. 219-230.

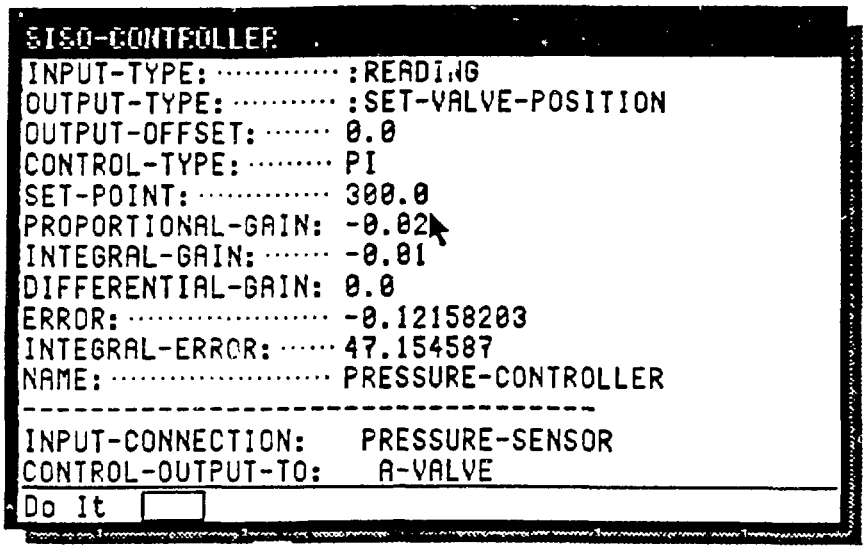
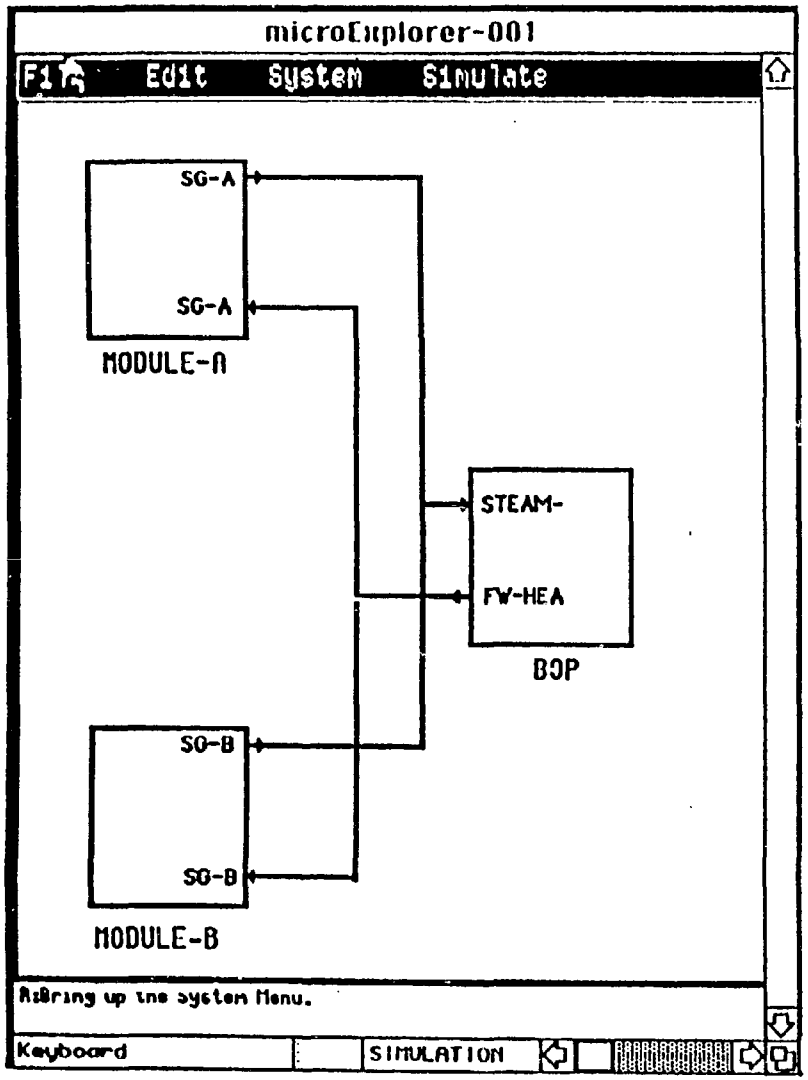
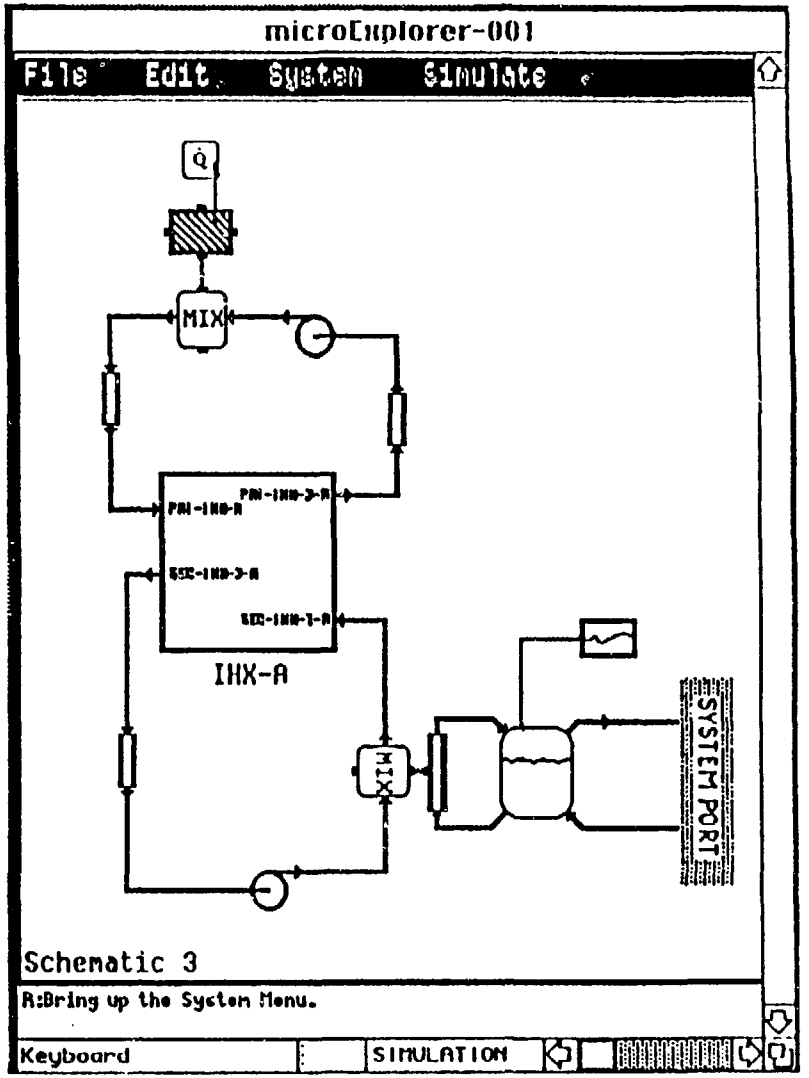



Figure 1. Example form editor for a SISO-CONTROLLER.




(a) Window for top level system.




(b) Window for MODULE-A.

Figure 2. Example ot hierarchial zooming.

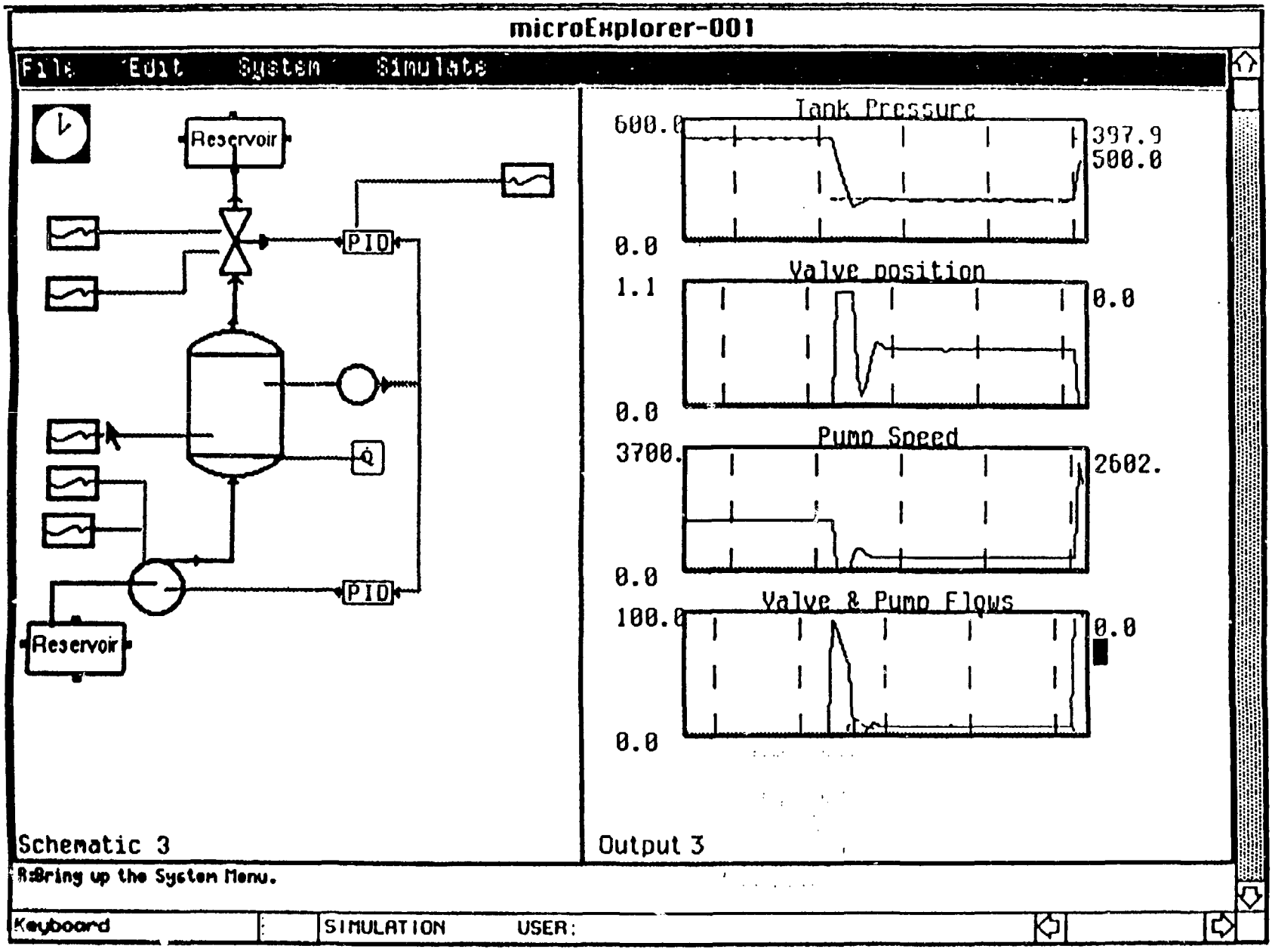



Figure 3. Typical run-time user interface.