SIMULATION IN A DYNAMIC PROTOTYPING ENVIRONMENT: PETRI NETS OR RULES?

Loretta A. Moore and Shannon W. Price
Computer Science and Engineering
Auburn University
Auburn, AL 36849
(205) 844 - 6330
moore@eng.auburn.edu

Joseph P. Hale
Mission Operations Laboratory
NASA Marshall Space Flight Center
MSFC, AL 35812
(205) 544-2193
joe.hale@msfc.nasa.gov

ABSTRACT

An evaluation of a prototyped user interface is best supported by a simulation of the system. A simulation allows for dynamic evaluation of the interface rather than just a static evaluation of the screen's appearance. This allows potential users to evaluate both the look (in terms of the screen layout, color, objects, etc.) and feel (in terms of operations and actions which need to be performed) of a system's interface. Because of the need to provide dynamic evaluation of an interface, there must be support for producing active simulations. The high-fidelity training simulators are normally delivered too late to be effectively used in prototyping the displays. Therefore, it is important to build a low fidelity simulator, so that the iterative cycle of refining the human computer interface based upon a user's interactions can proceed early in software development.

INTRODUCTION

The Crew Systems Engineering Branch of the Mission Operations Laboratory of NASA Marshall Space Flight Center was interested in a dynamic Human Computer Interface Prototyping Environment for the International Space Station Alpha's on-board payload displays. On the Space Station, new payloads will be added to the on-board complement of payloads in ninety day increments. Although a payload starts its development and integration processes from two to four years before launch, a set of new payloads' displays are due every ninety days. Thus, this drives the need for an efficient and effective prototyping process. The functional components of a dynamic prototyping environment in which the process of rapid prototyping can be carried out have been investigated.

Most Graphical User Interface toolkits allow designers to develop graphical displays with little or no programming, however in order to provide dynamic simulation of an interface more effort is required. Most tools provide an Application Programmer's Interface (API) which allows the designer to write callback routines to interface with databases, library calls, processes, and equipment. These callbacks can also be used to interface with a simulator for purposes of evaluation. However, utilizing these features assumes programming language knowledge and some knowledge of networking. Interface designers may not have this level of expertise and therefore need to be provided with a friendlier method of producing simulations to drive the interface.

This research is supported in part by the Mission Operations Laboratory, NASA, Marshall Space Flight Center, MSFC, AL 35812 under Contract NAS8-39131, Delivery Order No. 25. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of NASA.

A rapid prototyping environment has been developed which allows for rapid prototyping and evaluation of graphical displays [2]. The components of this environment include: a graphical user interface development toolkit, a simulator tool, a dynamic interface between the interface and the simulator, and an embedded evaluation tool. The purpose of this environment is to support the process of rapid prototyping, so it is important that the tools included within the environment provide the needed functionality, but also be easy to use.

This paper describes two options for simulation within the dynamic prototyping environment: petri nets and rule-based simulation. The petri net system, PERCNET [3], is designed to be used as a knowledge-based graphical simulation environment for modeling and analyzing human-machine tasks. With PERCNET, task models (i.e., simulations) are developed using modified petri nets. The rule based system is a CLIPS [1] based system with an X windows interface for running the simulations. CLIPS executes in a non-procedural fashion making it ideal for representing random and concurrent events required by the simulation. Its C language-based design allows external communication to be programmed directly into the model. In order to compare the two approaches for simulation, a prototype of a user interface has been developed within the dynamic prototyping environment with both simulation architectures. This paper compares the two systems based upon usability, functionality, and performance.

ARCHITECTURE OF THE DYNAMIC PROTOTYPING ENVIRONMENT

There are four components of the Human Computer Interface (HCI) Prototyping Environment: (1) a Graphical User Interface (GUI) development tool, (2) a simulator development tool, (3) a dynamic, interactive interface between the GUI and the simulator, (4) an embedded evaluation tool. The GUI tool allows the designer to dynamically develop graphical displays through direct manipulation. The simulator development tool allows the functionality of the system to be implemented and will act as the driver for the displays. The dynamic, interactive interface will handle communication between the GUI runtime environment and the simulation environment. The embedded evaluation tool will collect data while the user is interacting with the system and will evaluate the adequacy of an interface based on a user's performance. The architecture of the environment is shown in figure 1.

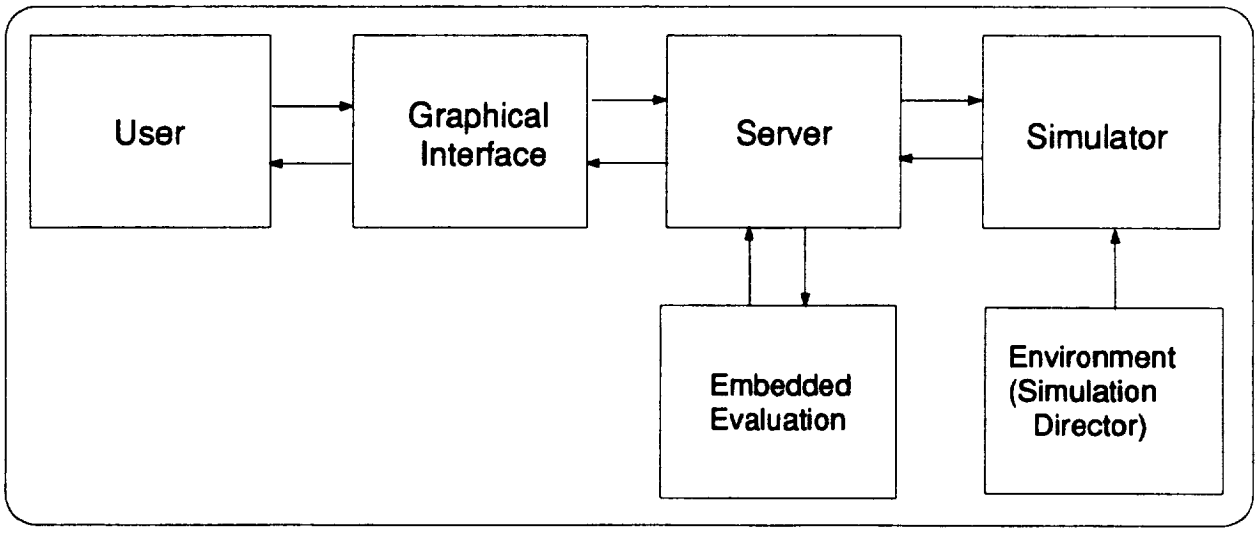

Figure 1 - HCI Prototyping Environment Architecture

Interface Development Tool

The Graphical User Interface (GUI) tool for the prototyping environment will allow the designer to create the display through direct manipulation. This includes the creation of static and dynamic objects, windows, menus, and boxes. The tool also allows objects created to be linked to a data source. During execution, the interface objects send and receive data and commands to the simulator by way of the data server. The user interface objects and their associated data access description are defined independent of the actual source of data. This first allows the development of the interface and the simulator to occur concurrently. Second, an interface developed with the GUI tool can later be connected to a high fidelity simulator and then to the actual flight software.

Simulator Development Tool

The simulator development tool provides the capability to develop a low fidelity simulation of a system or process. The development of a simulation has two important functions. First, the simulation helps the designer identify and define basic system requirements. Second, potential users can evaluate both the look (in terms of the screen layout, color, objects, etc.) and feel (in terms of operations and actions which need to be performed) of a system. The simulator provides realistic feedback to the interface based on user inputs.

Dynamic, Interactive Interface

This interface will handle communication between the GUI prototyping tool and the simulation tool during execution. The interface is a server which has been developed using the GUI's Application Programmer's Interface. Messages and commands can be sent and received both ways between the GUI and the simulator. The server also services requests from the embedded evaluation process, providing information as to which actions the user has taken and which events and activities have fired.

Embedded Evaluation Tool

An important aspect of the prototyping process is the ability to evaluate the adequacy of the developed graphical user interfaces. The embedded evaluation tool communicates with the server to receive information on the interaction between the user and the system. The types of data collected include user actions, simulator events and activities, and the times associated with these items. The collected data is analyzed to determine task correctness, task completion times, error counts, and user response times. The data is then analyzed to provide feedback as to which features of the interface the user had problems with and therefore need to be redesigned.

An Example: The Automobile Prototype

In order to assess the architecture described above a system was chosen to be prototyped in the environment. The system chosen for empirical evaluation of the HCI prototyping environment was an automobile. An automobile has sufficient complexity and subsystems' interdependencies to provide a moderate level of operational workload. Further, potential subjects in the empirical studies would have a working understanding of an automobile's functionality, thus minimizing pre-experiment training requirements.

An automobile can be considered a system with many interacting components that perform a task. The driver (or user) monitors and controls the automobile's performance using pedals, levers,

gauges, and a steering wheel. The dashboard and controls are the user interface and the engine is the main part of the system. Mapping the automobile system to the simulation architecture calls for a model of the dashboard and driver controls and a separate model of the engine. Figure 2 demonstrates how an automobile system could be mapped into the architecture described. The main component of the automobile is the engine which responds to inputs from the driver (e.g. the driver shifts gears or presses the accelerator pedal) and factors in the effects of the environment (e.g. climbing a hill causes a decrease in the speed of the car). The driver changes inputs to obtain desired performance results. If the car slows down climbing a hill, pressing the accelerator closer to the floorboard will counteract the effects of the hill.

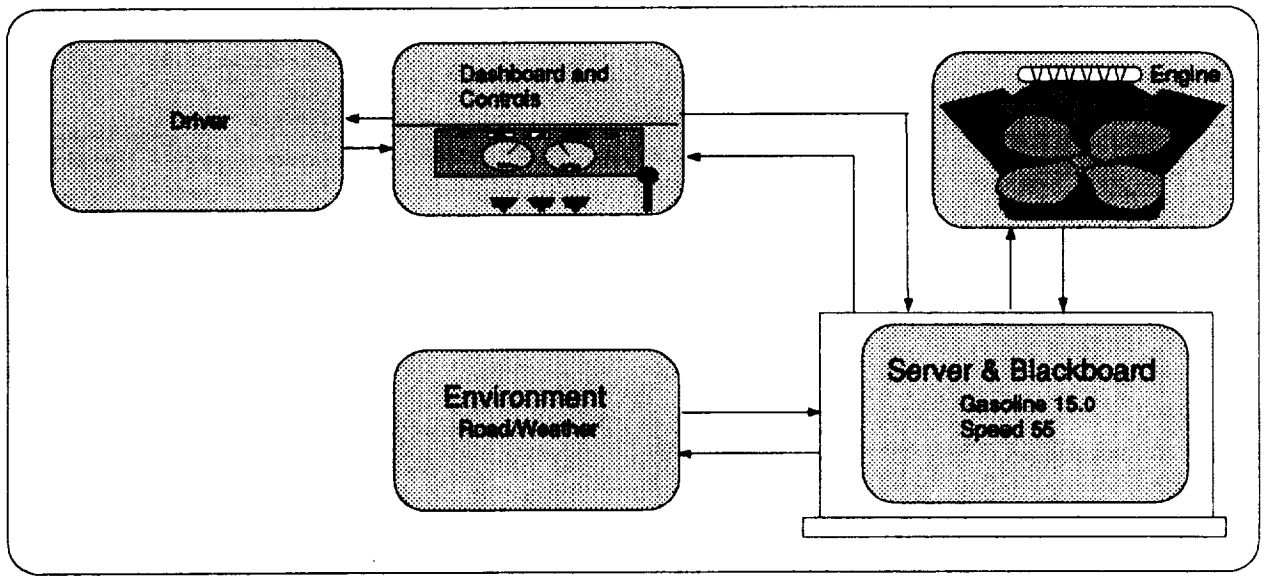

Figure 2 - Automobile Prototype

The dashboard and controls have been modeled using Sammi [5], a graphical user interface development tool developed by Kinesix. Two options have been investigated for simulation: petri nets and rules. Petri nets provide a graphical model of concurrent systems. The petri net system which has been used is PERCNET [3], developed by Perceptronics. PERCNET is designed to be used as a knowledge-based graphical simulation environment for modeling and analyzing human-machine tasks. With PERCNET, task models are developed using modified petri nets, a combination of petri nets, frames, and rules. The rule based system which has been used is CLIPS [1], a rule based language primarily used for the design of expert systems, developed by NASA. CLIPS executes in a non-procedural fashion making it ideal for representing random and concurrent events. The automobile system has been prototyped using both the petri net and rule-based systems as simulators and comparisons have been made based upon functionality, usability, and performance.

SIMULATION IN THE DYNAMIC PROTOTYPING ENVIRONMENT

Because of the need to provide dynamic evaluation of an interface rather than just static evaluation, there must be support provided for producing active simulation. Most GUIs, including Sammi, provide some sort of Application Programmer's Interface (API) which allow the developer to write call back routines which interface with databases, library calls, other processes and equipment. We would like to provide a means of building a low fidelity simulation of the system to drive the interface which requires little programming.

Basic simulation requirements include the ability to model events and activities, both sequentially and concurrently. The system should provide the ability to create submodels within the main model. The simulator clock must be linked to the system clock, and support should be provided for the creation of temporal events. The process must be able to communicate with UNIX processes using the TCP/IP protocol. Real-time communication must also be provided to allow the tool to communicate with the GUI tool on a separate platform via Ethernet. The ability for two-way asynchronous communication between the runtime versions of the interface and the simulator must be provided. The simulator must be capable of receiving data from the GUI tool to dynamically control temporal events, to modify the values of variables, and trigger events and activities. The ability to specify and send commands, data, and alarms to the GUI tool must also be provided. A simulator director should be able to send commands (e.g., start simulation, trigger scenario event, etc.) to the simulator from a monitoring station. An interface should be provided in order to bind interface objects to simulation objects in order to set the values of variables, trigger events or activities, and set temporal variables.

Simulation Using Petri Nets

PERCNET is a very powerful system analysis software package designed by Perceptronics, Inc. It provides an easy-to-use, graphical interface which allows users to quickly lay out a petri net model of the system. PERCNET uses "modified" petri nets, which allow each state to describe preconditions for state transitions, modify global variables, perform function calls and maintain a global simulation time.

Pictorially, Petri nets show systems of activities and events. Ovals represent activities which describe actions performed by the system. Activities are joined by events, represented by vertical bars, that occur during execution. Events are associated with user actions and environmental conditions. Execution is shown by tokens propagating through the system. Flow of control passes from activities to events. Before an event can fire all incoming arcs must have tokens. When this occurs, the event places tokens on all outgoing arcs passing control to activities. The behavior that an event exhibits during execution is dependant on the data contained in its frame. Frames record data related to each activity and event. Event frames may contain rules and functions. Activity frames allow the designer to specify a time to be associated with each activity. Figure 3 shows the top-level petri net of the automobile simulator.

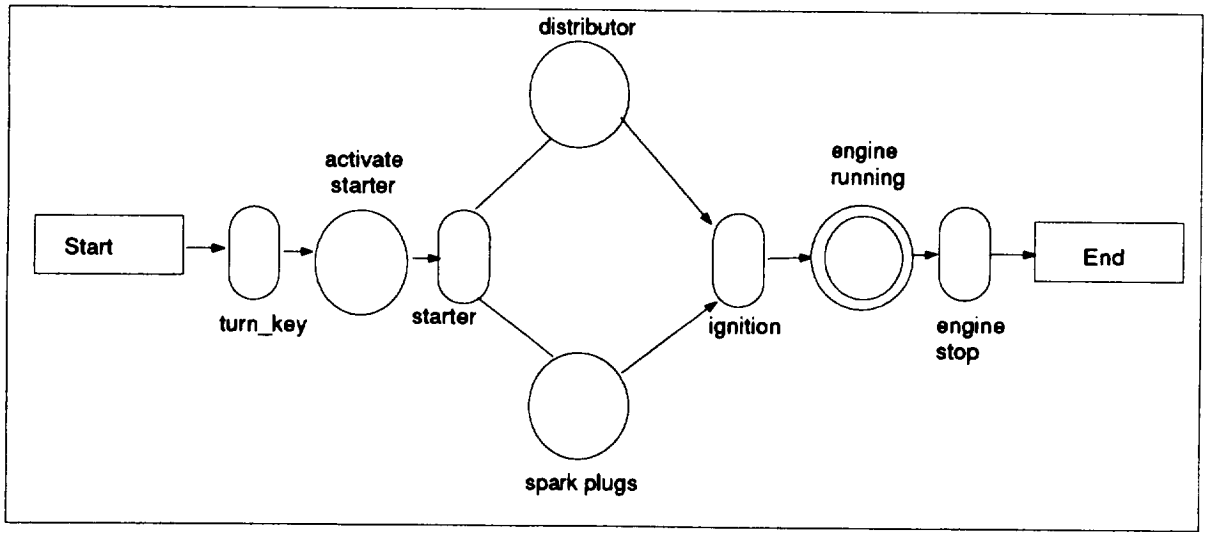

Figure 3 - Top-Level Petri Net of the Automobile Simulator

The starter is the component that is activated by the turning of the key. Before the starter can begin working, however, the key should be turned on, the driver must be wearing his/her seat belt, the car must be in neutral and the battery must have a sufficient charge to start the starter. When all three pre-conditions are true, the starter is activated and control advances to the right in the Petri net. Once the starter has been activated, it must do its part to start the automobile. The starter allows electricity to flow into the distributor where it is channeled into the spark plugs. As long as the starter is functioning, the distributor and spark plugs are activated. Finally, as long as the spark plugs and distributor are working properly and there is gasoline, the spark from the spark plugs ignites the gasoline mixture in the engine and ignition is achieved. Now that ignition has been accomplished, the engine is running. The concentric circles representing the engine_running activity in Figure 3 indicate that the state is shown in a sub-net.

The petri net representing the automobile passes from the ignition portion to the engine running state and remains in the running state until some condition causes the engine to stop running. The engine will stop running if the engine runs out of gas, runs out of oil, the temperature rises above a certain threshold, the key is turned off, the engine stalls (when the automobile is in some gear and the rpms fall below a threshold amount), the battery loses its charge or the fuel pump, oil pump, spark plugs or alternator fail.

The major components of the engine modeled are: fuel pump, oil pump, water pump, distributor, spark plugs, starter, battery, alternator, and fan. The condition of these components is modeled using a boolean variable indicating either that they are functioning or they are not. The boolean variables are then used as conditions within events occurring during the simulation. Details of the Petri Net implementation can be found in [2].

Simulation Using Rules

Since CLIPS is rule-based, it is completely non-procedural. Furthermore, it allows programmers to pick the strategy by which successive rule-firings are chosen. Certain rules may be designated fired by different priority levels (rules with the highest priority fire before rules with lower priority). Other rule-selection strategies govern how rules with equal priority are selected. Events and activities are represented by the pre- and post-conditions of rules. For example, the rule for activating the starter is:

```
(defrule TURN_KEY
                ?tick <- (clock_tick)
                (test (= 1 ?*key*))
                (test (= 1 ?*seatbelt*))
                (test (= ?*gear* 0))
                (test (> ?*battery* 10.0))
                (test (= ?*state* ?*READY*))
=>
                (bind ?*state* ?*STARTER*)
                (retract ?tick)
                (assert (clock_tick))
                (printout t "ACTIVATE STARTER (" ?*time* ")" crlf)
                (tick_tocks 2)
                (assert (updated TRUE))
)
```

In this project, CLIPS has been extended to include communication capabilities [4]. Two sockets have been provided for reading from and writing to the server. C functions have been developed to eliminate redundant information from the messages passed to the server. Another improvement compiled into the CLIPS executable has been a control process that allows a user to start, stop and quit CLIPS execution through a graphical interface.

The project also demonstrates some programming techniques used in CLIPS to support the simulation. A global simulation time should be maintained and a mechanism for keeping simulation execution time has been demonstrated. Another important feature that makes use of the timer is the periodic update feature. This ensures that CLIPS execution pauses (i.e., no rules may fire) every few seconds to send and receive information from the server. When this happens, control returns to the main routine which initializes communication with the server.

Writing CLIPS programs to take advantage of this strategy requires the incorporation of several techniques. These techniques include rules, variables, and functions which may be used in subsequent simulation designs. The first choice involves determining which values will be passed to or received from the server. All global variables (defined using the "defglobal" command) are passed to the server. No other values are passed. Facts and local variables may be used to store values which do not need to be passed to the server. It will be shown later how communication has been further streamlined for efficiency. The most important rule is the clock rule.

The clock rule stays ready at all times, but because the salience (i.e. priority) of the rule is kept low, it will not block the firing of other rules. When execution begins, the current system time is retrieved and stored. The current simulation time is always known by retrieving the system time and comparing it to the starting time. The new simulation time is temporarily stored in a variable called "new_time"and is compared to the last calculated time. If the two values are the same, then the clock rule has fired more than once within one second. In that case, the time is not printed and facts are reset to allow the clock rule to fire again.

A "clock_tick" fact is used in the preconditions of rules to allow them to become ready for firing. Without the clock_tick fact, a rule may never fire. Another time feature provided is the tick_tocks function. Often a programmer would like to force a rule to consume clock time. A call to the tick_tocks function forces execution to enter a side loop where the required time elapses before execution continues.

COMPARISON

Usability

Most features of PERCNET are easy-to-learn and use While some study of petri-net theory would benefit designers, much could be done with very minimal knowledge of petri-nets. One difficulty in working with PERCNET was the lack of available documentation on the Tool Command Language (TCL). All function calls, calculations, communication and ad-hoc programming are done using this language. Perceptronics provides only minimal documentation on the use of the language within PERCNET making it very difficult to perform anything more than the most basic operations. However, PERCNET's graphical interface is very appealing to users.

CLIPS is a rule-based language, which means that there may be a larger learning curve than there is with PERCNET's point-and-click interface. After the initial learning stages, however, CLIPS leaves a developer with an immensely powerful simulation tool. The main advantage is flexibility.

CLIPS was written in the C programming language and is completely compatible and extendible with C functions. Knowing C in advance can significantly lessen the learning curve. Many of the "non-C" features of CLIPS resemble LISP. CLIPS has been a tremendous surprise to work with. A basic proficiency with CLIPS may be gained quickly and one can learn to do very useful things with the language. Writing the rules for the simulation was actually the easiest part of the project. As proficiency with the language developed, more advanced features provided tremendous possibilities. The manuals present the language in a very easy to read format, contained extensive reference sections and sample code. Furthermore, the manuals outline how CLIPS may be easily extended to include C (and other) functions written by programmers.

Functionality

As this project began, PERCNET was a closed package, that is, there was no provision for communicating with other applications. NASA contracted Perceptronics to modify PERCNET to allow for such a feature. The final result was a revision of PERCNET which would allow communication with other applications through the use of sockets. Applications are allowed to request that global variables be retrieved and/or modified. PERCNET essentially opened it's blackboard (i.e., global data store) to other applications. The other application in this case being the server.

After several functions were added to CLIPS (see descriptions in previous sections), the CLIPS system performed the same functions as the Petri Net simulator. If a new system is prototyped, the only changes which would be needed are to the knowledge base. The communication link developed for the Sammi-CLIPS architecture uses the blackboard paradigm to improve modularity, flexibility, and efficiency. This form of data management stores all information in a central location (the blackboard), and processes communicate by posting and retrieving information from the blackboard. The server manages the blackboard, allowing applications to retrieve current values from the board and to request that a value be changed. The server accepts write requests from valid sources and changes values. The comparison of the two architectures goes much further than comparing the two simulation designs. The design of the communication link significantly affects the flexibility and performance of the architecture.

Performance

The performance within the Petri Net architecture was not acceptable for real-time interface simulation. Interfaces running within this architecture exhibit a very slow response rate to user actions when PERCNET is executing within its subnets. The PERCNET execution is also using excessive amounts of swap space and memory which also affect the refreshing of displays.

Early analysis attempted to find the exact cause of the poor performance; however, only limited work could be done without access to PERCNET's source code. Since PERCNET's code was unavailable, we could only speculate about what was actually happening to cause the slow responses. It was determined that the cause of much of the problem was that PERCNET was trying to do too much. In the PERCNET simulation architecture, PERCNET is actually the data server for the environment. The global blackboard is maintained within PERCNET. The server only provides a mechanism for passing information between PERCNET and other applications. The server is connected to PERCNET by a socket and the server is actually on the "client" end of the connection-oriented socket. The server establishes connections with PERCNET and Sammi and then alternately receives information from each. Any data or commands received from Sammi are passed immediately to PERCNET. Commands from PERCNET for Sammi are passed immediately through, as well. Finally, the server sends Sammi copies of all variables. Since PERCNET is the blackboard server, as well as the simulator, PERCNET's performance would naturally be affected by the added burden.

Lastly, the method provided for sending variables to the server was terribly inefficient. When a calculation was performed in the simulation model for a variable that was needed by the interface, that variable was passed to the server whether or not it's value had changed from the previous iteration. No mechanism was provided for restricting the number of redundant values passed across the communication link. As a result, PERCNET passed every value back to the server when only a few had actually changed.

Each of these limitations was addressed in the design of the server and blackboard in the rule-based architecture. The server program may be divided into three portions: blackboard management, Sammi routines, CLIPS routines. The Sammi and CLIPS routines are provided to communicate with the respective applications. These routines map data into a special "blackboard entry" form and pass the data to the blackboard management routines. The blackboard routines also return information to the Sammi and CLIPS routines for routing back to the applications. The blackboard management routines require that each application (many more applications may be supported) register itself initially. Applications are assigned application identification numbers which are used for all subsequent transactions. This application number allows the blackboard to closely monitor which variable values each application needs to see. It also provides a mechanism for installing a priority scheme for updates.

The overwhelming advantage of the CLIPS and blackboard combination is the flexibility and potential they provide. Features are provided that allow modifications which can affect performance. The ability to tune the performance has allowed the simulation architecture to be tailored to specific running conditions (e.g., machine limitations, network traffic and complexity of the interface being simulated). Several parameters may be modified to alter performance. Tuning tests have improved performance. More detailed performance testing is planned to verify the results.

CONCLUSION

The goal of the architecture has been to provide simulation of user interfaces so that they may be designed and evaluated quickly. An important portion of the dynamic prototyping architecture is therefore the simulator. Ease-of-use is very important, but performance is critical. The Petri Net architecture's ease-of-use is currently its only advantage over the Rule-Based architecture. The Rule-Based design overcomes this with power and flexibility. Work currently in progress include a detailed analysis of the performance of the communication link and a design of a graphical interface to CLIPS.

REFERENCES

1. CLIPS Reference Manual, NASA Johnson Space Flight Center, Houston, Texas, 1993.

2. Moore, Loretta, "Assessment of a Human Computer Interface Prototyping Environment," Final Report, Delivery Order No. 16, Basic NASA Contract No. NAS8-39131, NASA Marshall Space Flight Center, Huntsville, Alabama, 1993.

3. PERCNET User's Manual, Perceptronics Inc., Woodland Hills, California, 1992.

4. Price, Shannon, "An Improved Interface Simulation Architecture", Final Report for Master of Computer Science and Engineering Degree, Auburn University, Auburn, Alabama, 1994.

5. Sammi API Manual, Kinesix Corporation, Houston Texas, 1992.