Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)

EMULATION AS A DESIGN TOOL IN THE DEVELOPMENT OF REAL-TIME CONTROL SYSTEMS

Howard M. Bloom, Cita M. Furlani, and Anthony J. Barbera
Industrial Systems Division, Center for Manufacturing Engineering
National Bureau of Standards, Gaithersburg, MD 20899

ABSTRACT

A major facility for manufacturing research is being established at the National Bureau of Standards. The Automated Manufacturing Research Facility (AMRF) will provide a testbed where measurement research of computer integrated manufacturing systems can be performed. The control architecture of the facility is based on a sensory-interactive, modular, hierarchical, feedback system. Each module is represented as a finite state machine that interacts through a shared time-sliced common-memory where command, feedback and database information is stored.

A hierarchical control system emulator (HCSE) has been developed that allows the system to be designed and tested before implementation on the actual hardware. The HCSE has been successfully used in the AMRF project as a design management tool, providing a complete specification of the control software. It is also used as a testing aid that allows a given module (i.e., a robot control system) to interact with emulated control modules substituting for unavailable AMRF hardware.

INTRODUCTION

A research project, the Automated Manufacturing Research Facility (AMRF), is underway at the National Bureau of Standards (NBS). One goal of the effort is to identify potential standard interfaces between components of automated small batch manufacturing systems. A second goal is to develop measurement techniques and standard reference materials that will provide users of the new manufacturing technologies with a means for ensuring, where applicable, the traceability of their processes and products to national standards [1,2,3].

A portion of the NBS machine shop has been set aside for the construction of an testbed automated manufacturing system to support experimentation. The testbed will be made available for selected research projects by academia, industry, research institutions, and other government agencies. The AMRF is being designed to handle the bulk of the part mix manufactured in the NBS instrument shop. The research supported by the AMRF will address only chip forming metal removal manufacturing. Within this constraint, the intent is to completely automate the production process from the transfer of near net-shape-blanks from inventory through the delivery of finished, cleaned, and inspected parts. The machine tools in the AMRF were chosen to be representative of the general purpose machine tools in common use throughout the United States. Each of the machines, along with a single industrial robot, will be configured into a work station. The AMRF will have six self-contained work stations, each with a well defined function. The functions are:

1. Horizontal machining
2. Vertical machining
3. Turning
4. Cleaning and deburring
5. Inspection
6. Materials handling

A major portion of the research effort addresses the development of methods for effectively controlling the operation of all the work stations. A control system is required that takes orders for a part to be made, describes the part using a CAD system, uses a process planning system to define how to make the part, and then schedules and monitors the actual production process.

DESCRIPTION OF THE HIERARCHICAL CONTROL TECHNIQUE

The control architecture design for the AMRF is based on the the NBS robot control system which consists of a collection of control modules arranged in an hierarchical multi-level system and interacting through a communication network [4]. A control module can be represented by a generic control structure. At each instant, the particular output will be determined by the input command and the feedback data processed (Figure 1). Each controller takes commands from the next higher level system, but may direct several others at the next lower level. Long range goals or tasks enter the system at the highest level and are decomposed into sequences of subtasks to be executed as procedures at that level or output as commands to the next lower level.

The hierarchical control architecture concept has been extended to apply to the entire research facility. The control structure of the AMRF is composed of five major hierarchical levels: Facility, Shop, Cell, Workstation, and Equipment [5]. Each level has one or more instances of

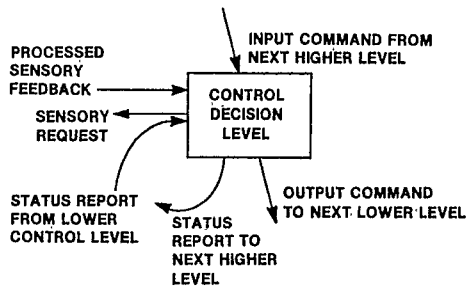


Figure 1: Generic Control Level

controllers which are further decomposed into sublevels or modules (Figure 2). The functions of each level are as follows:

(a) Facility - This highest level of control comprises three major subsystems: Manufacturing Engineering, Information Management, and Production Management. Manufacturing Engineering provides user interfaces for the computer-aided-design of parts, tools, and fixtures; and for the planning of production processes. Information Management provides interfaces to and support for the administrative functions. Production Management generates long range schedules based on production resource requirements and available production capacity.

(b) Shop - This level is responsible for the real-time management of resources and jobs on the shop floor through two major modules: Task Management and Resource Management. The Task Manager schedules job orders, equipment maintenance, and shop support services, such as housekeeping. Job orders are grouped into part batches according to a group technology (GT) classification scheme. Virtual manufacturing cells are created (through software) to manage the part production and are removed when assigned tasks are completed [6]. The Resource Manager allocates workstations, storage buffers, tools, and materials to cell level control systems and to particular production jobs. The scheduling of resources is updated as status information about task completion is reported from the cells.

(c) Cell - This level controls the sequencing of batch jobs of similar parts through workstations and the supervision of various other support services, such as material handling. The virtual cell permits the time sharing of workstation level processing systems [7].

(d) Workstation - This level coordinates the activities of the shop floor equipment typically consisting of a robot, a machine tool, a material storage buffer and a control computer. Its task is to process a tray of parts that has been delivered by the material transport system. The workstation controller sequences equipment through job setup, part fixturing, cutting, chip removal, in-process inspection, job takedown, and cleanup operations.

(e) Equipment - This level is identified with particular pieces of equipment on the shop floor, such as robots, NC machine tools, and delivery systems. These systems perform the functions of material storage, transportation, handling, material processing, cleaning, and inspection.

In addition to hierarchical control architecture, the AMRF control systems utilize several other ideas derived from the development of the NBS Real-Time Control System [8]. These are: state-machine implementation, control cycles, and common memory.

Within the hierarchical control structure each control module operates as an independent finite state machine and may reside on one or more processors. All inputs, outputs, states, and state transitions of each subsystem are identified in state tables that offer a convenient form for user interaction and modification (Figure 3).

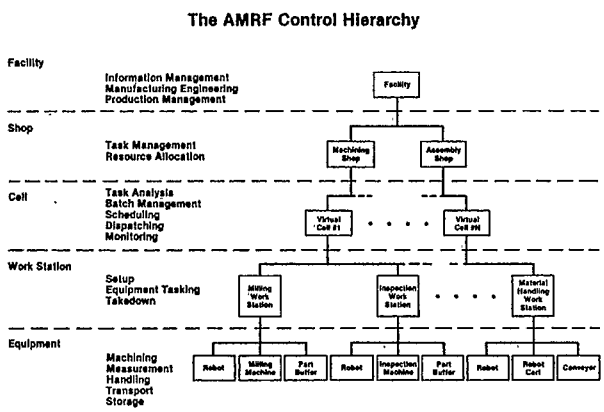


Figure 2

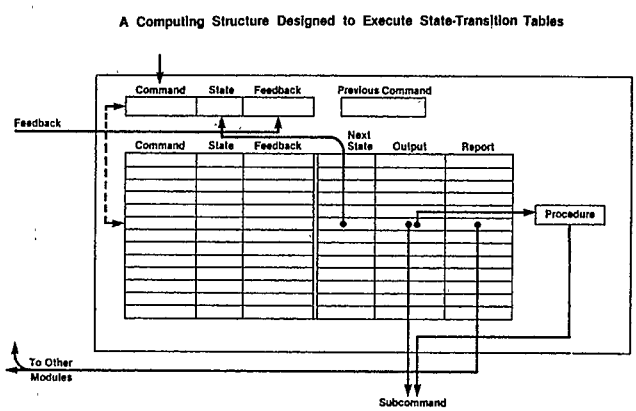


Figure 3

A time interval (called a control cycle) that determines how often its table is processed is defined for each control subsystem. Processing a state table involves examining all inputs and state variables, locating the current state in the table, and then executing the procedures and generating the outputs associated with the current state. The control cycle at each level must be short enough to maintain system stability. That is, each processor must be able to identify the current state and generate appropriate outputs before the behavior of the system deviates from acceptable ranges.

Communications between the various controllers is accomplished by writing messages in a database designated as "common-memory" which is common to all modules which either compute or make use of those messages. Each message area (or mailbox) within the database is restricted so that only one system may write to it, although many can read its contents. If the cycles of the state-clock at all levels are synchronized, information transfer into and out of the common memory occurs at predictable time increments and each message carries a time tag to allow for proper identification.

EMULATION AS A SOFTWARE DEVELOPMENT TOOL

The AMRF is being designed cooperatively by several divisions of NBS working in various areas of automation technology. Approximately 50 major control systems are being simultaneously implemented. The NBS researchers come from a variety of disciplines--computer science, electrical engineering, mechanical engineering, industrial engineering, etc. -- and are applying technology to a wide range of manufacturing areas.

These efforts must be integrated together as a well-structured facility implementation. Interfaces between different modules must be worked out through intensive negotiation. It is important to have an up-to-date design specification of all the interfaces and control modules in order to perform the integration function. Emulation, through the use of state-table construction, can be an effective management tool to accomplish this purpose. As the system is developed, more detailed state-tables can be created that define modules by "step-wise refinement". These state-tables are then available to other project members for obtaining an understanding of the system functions. Structured walk-throughs that are used as a management review process can be acccomplished by simply testing the control logic of the state-tables even before the output procedures are developed.

The basic principle that makes the state-table structure an effective software engineering tool is the rigid adherence to the definition of each variable (including command and state information) that might affect which line in the table is to be executed. A column is defined for specifying the condition of the variable value (e.g. COUNT < 10). Each column relationship is then processed in an AND operation with the other input columns. This corresponds to using only IF-THEN logic. The designer is thus forced to consider every variable even if the result is a "don't care" condition. If IF-THEN-ELSE structure was used, it would be difficult to follow the control logic when the nesting reached beyond the second level. Using the state-table approach leads to the following benefits:

(a) Problem partitioning - The system is decomposed into simple, well-defined modules with clearly specified inputs, outputs, internal states, and rules for state-transitions.

(b) Extensibility - State-tables are first developed with few or no error conditions. As new problems or cases are identified, new states can be added to the module.

(c) Structured code - Each line in the state-transition table for any module is an IF/THEN production rule.

(d) Conditional testing - As new feedback data is added, it can be added as new columns to the state-table to test the new variable values.

(e) Debugging - Diagnostic routines can be used to read state conditions from common memory which makes it easy to perform traces, and to reason back from error states.

The development effort for the real-time control system has identified the principal software tools necessary to create such an environment for design and programming. This programming environment includes a state-table editor that allows convenient user interaction; a state-table interpreter or compiler; a data dictionary to manage the state-tables, variables, routines, including the relationships between them; and diagnostics which allow for monitoring control cycles and examination of data.

EMULATION DEVELOPMENT

The robot control system serves as the basis for the implementation of an emulator environment that can effectively handle the system development requirements for the entire AMRF. Although the major thrust of the development effort is in system integration through interface specifications, the research nature of the laboratory causes the facility to be in a constant state of change as equipment is added or modified. For this reason it was decided to develop a system that, by accurately reflecting the structure of the

AMRF, could be used as a real-time interface to individual modules to allow them to be tested as desired. The Hierarchical Control System Emulator (HCSE) was designed to satisfy these goals.

The HCSE is a collection of computer programs written for NBS in the high-level PRAXIS [9] language developed at Bolt, Beranek, and Newman, Inc., to run under the DEC VAX-11* VMS Operating System [10]. The software provided allows the emulation to follow the structure of the AMRF modular hierarchical feedback control system, with the modules conceptualized as finite state machines (FSM) that interact through a shared time-sliced synchronized common memory (Figure 4). The features of the emulator can be used to establish the feasibility of the overall AMRF control structure, as well as to assess the computational and communications requirements.

Emulation Structure

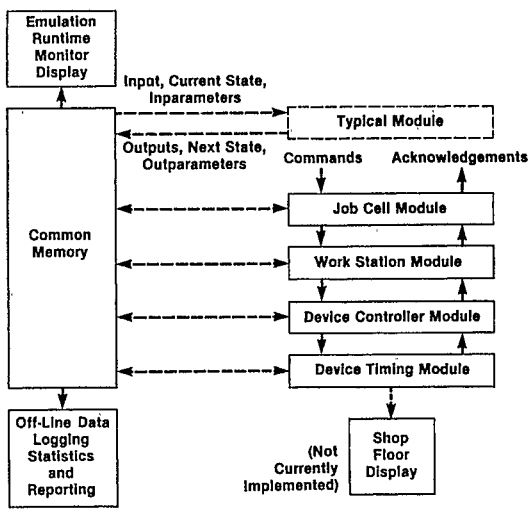


Figure 4

The emulator can serve as a prototype for the real system to allow design development before the actual system hardware is available. When the emulation speed is equal to actual clock speed, the emulator can function as a mock-up of the actual shop-floor control software, and any subset of the simulated machine functions could be substituted for physical devices. In bringing the job shop on-line, new components (including man-machine interfaces) may be tested with existing components and simulations of components which are not yet installed.

The features of the HCSE were developed to allow for effective emulation of the entire AMRF control structure. Each module (e.g. workstation) at each control level is distinctly identifiable which facilitates functional interchangeability with the actual hardware. The database structure reflects the common-memory approach to data transfer. The ability to perform local computational processing is available. Communication and computing delays are represented by using the mail-box technique [11]. Modules are allocated to different physical processes to simulate putting them in different computers. Finally, the control structure of each emulation module reflects the state-table design.

The basic unit of the emulator is the finite state machine (FSM) transition table which is used to describe each module. A standard format is used based on a generalized state-machine description with named variables (Figure 5). The first part consists of lines with declared input, output; and internal variable names and types. This is followed by a sequence of condition-action pairs that implement the rows of a state table. Procedures that are used in the action statements appear in the last part of the FSM. The FSM module is translated into a PRAXIS program module which is then compiled.

Communication between modules, which is transparent to the user, is achieved through

Finite State Machine (FSM) Format

```
/ / Name          Modulename
/ /
/ / Input         Inputvariable  Type
/ / Internal      Internalvariable  Type
/ / State         Statevariable  Type
/ / Output        Outputvariable  Type
. . . (Other Variable Declarations)
/ / Conditions    Condition; Condition; . . .
/ / Actions       Statement; Statement; . . .
. . . (Other Condition-Action Pairs)
/ / Multimatch    Statement; Statement; . . .
/ / Nomatch       Statement; Statement; . . .
/ / Procedures
. . . (Procedure Declarations)
/ / Doc
/ / Documentation of this FSM
/ / Enddoc
  [End of File]
```

Figure 5

*Certain commercial equipment and software is identified in this paper in order to adequately specify the experimental facility. Such identification does not imply recommendation or endorsement by the National Bureau of Standards.

Sequential List of Logging File

0: 0: 0.00	COUNT1_CURS		NOMATCH
0: 0: 0.00	COUNT2_CURS		RUNNING
0: 0: 0.10	COMMAND	up	
0: 0: 0.20	COUNT		1
0: 0: 0.30	COUNT		2
0: 0: 0.40	COUNT		3
0: 0: 0.50	COUNT		4
0: 0: 0.60	COUNT		5
0: 0: 0.70	COUNT		6
0: 0: 0.80	COUNT		7
0: 0: 0.90	COUNT		8
0: 0: 1.00	COUNT		9
0: 0: 1.10	COUNT		10
0: 0: 1.20	COUNT		11
0: 0: 1.20	COMMAND	RESET	
0: 0: 1.30	COUNT		-10
0: 0: 1.40	COMMAND	up	
0: 0: 1.50	COUNT		-9
0: 0: 1.60	COUNT		-8
0: 0: 1.70	COUNT		-7
0: 0: 1.80	COUNT		-6
0: 0: 1.90	COUNT		-5
0: 0: 2.00	COUNT		-4
0: 0: 2.10	COUNT		-3
0: 0: 2.20	COUNT		-2
0: 0: 2.30	COUNT		-1
0: 0: 2.40	COUNT		0
0: 0: 2.50	COUNT		1

Figure 8

Data Attribute Listing of Variable Names

Command String
Written by: Count2
Read by: Count1
Comments:
Count1 — Command from Count2
Count2 — Command to Count1

Count Integer
Written by: Count1
Read by: Count2
Comments:
Count1 — Counting Variable
Count2 — Counting Variable

Figure 9

the variable value (only one module is allowed to write into the variable value). Comments in the FSM module relating to the variable are also presented.

Another feature of the emulator is a special documentation format that allows for commenting in the FSM module. Statements inclosed in "//DOC" and "//ENDDOC" can be stripped out of the FSM and used in a formal design specification document.

USE OF EMULATION IN THE AMRF

The first emulation version of the AMRF was designed to study the concept of the Virtual Cell. A virtual job cell is created each time that the production of a new part is scheduled to begin. The job cell acquires the necessary resources for each stage in the production of the part, assures that the correct sequence of subtasks is performed to machine the part, and then disappears once the part has been returned to inventory. A simple graphics interface to the common-memory was developed to allow the user to view the operation of the facility (Figure 10). Three workstations are available in this example: an inventory workstation, a transportation workstation, and a milling workstation. The inventory workstation controls a carousel, the transport workstation controls a cart, and the milling workstation controls the transfer of trays, a robot, and a vertical milling station.

Each job cell in this example first retrieves a specified blank from inventory and transports it to the milling workstation. It then takes control of the milling workstation which activates the tray transfer. The robot picks up the part and places it on the milling machine. The piece is then machined into a part, and these steps are repeated in reverse order to return the finished part to inventory. Each control module is enclosed in a box with the command being executed and the current status displayed. The lines reflect the chain of control with the connections between cells and workstations dynamically changing with the reallocation of resources. In the example, four instances of emulated time are shown and two virtual cells appear and control different workstations. All levels of the facility are emulated with state tables except the equipment level which is simulated with simple timing functions. The actual hardware could be substituted for the simulated systems with the emulator running in real time.

The emulator is also being used as a design tool for computer-aided-design (CAD) directed inspection [12]. This project involves integrating the emulator with a constructive solid geometry modeling system, and with a coordinate measuring machine and with a program development environment. This emulation includes a front end through which commmands can be entered, a machine simulation module, and a dynamic graphic display of the emulated inspection machine.

The robot vision system developed at NBS has also been implemented on the HCSE. The basic control structure was developed within the emulator methodology. A link to VAX 'C' compiled code was developed to allow the vision application routines to be used without any modification.

In the fall of 1983, and again in June, 1984, there was a major test of the integration of robots, machine tools, robot carts and workstations. This test demonstrated the implementation of the control hierarchy from the equipment level through the cell control level. The emulator played the roles of the cell control system and the Material Handling Workstation (MHW) because the actual hardware was not yet available. The emulated cell control system interfaced with the actual workstations, while the emulated

storage (by name) of common input and output variables in a shared (common) memory. Access to the memory is time-sliced synchronized. The pattern and sequencing of input/output transactions between modules may be specified by the user to define a hierarchical relationship of the control system modules.

In order to run the emulation, the compiled modules are combined using the BUILD feature of the HCSE that permits one or more modules to be processed together (as if they were on the same computer). Each BUILD module runs as an independent process. These processes are synchronized through common memory. A run-time display runs as another process, allowing the user to monitor the real-time progress of the various processes of the emulation. The user can control the actual rate of progress of the emulation through the run-time display to achieve single-cycle operation, wall-clock synchronization with variable time-scaling, or free-running (maximum-speed) simulation. The user can select variables from common memory to be displayed, and may stop the emulation or record "snapshots" of common memory at any time.

The use of the emulator can be illustrated through a simple example (Figure 6a, 6b). Module COUNT1.FSM resets the names common variable COUNT to -10 and thereafter increments it on each tick until it is reset again. Module COUNT2.FSM observes the output of COUNT1 through common memory and issues a "RESET" command to COUNT1 when its count reaches +10. These two modules are combined into a single emulation process which will only work correctly if the two modules successfully communicate via common memory.

Each time the value of a variable changes in common memory, it is recorded in the logging file. Upon completion of a run, these logging files are processed to produce summary statistics that show the values taken by each logged variable and the amount

Two Module Example Using the HCSE: Count 1

```
//Name Count1 | Rests to -10 and Then Increments on Each
//           | Tick Until Reset
//
//Input Command String | Command From Count2
//
//Output Count Integer  | Counting Variable
//
//Conditions Command = "UP"
//Actions Count: = Count +1          | Do the Counting
//
//Conditions Command = "RESET"
//Actions Count: =.-10
//
//Multimatch Nexts: = "MULTI"
//Nomatch Nexts: = "NOMATCH"
```

Figure 6a

Two Module Example Using the HCSE: Count2

```
//Name Count2  | Observes Count1 . fsm Output Through
//             | Common Memory and Issues a Reset
//             | Command to Count1 When the Variable
//             | Count  > = 10
//Input Count Integer      | Counting Variable
//
//Output Command String    | Command to Count1
//
//Conditions First_Entry
//Actions Nexts: = "RUNNING"
//
//Conditions Curs = "RUNNING"; Count < 10
//Actions Command: = "UP"
//
//Conditions Curs = "RUNNING"; Count  > = 10
//Actions Command: = "RESET"
//
//Multimatch Nexts: = "MULTI"
//Nomatch Nexts: = "NOMATCH"
```

Figure 6b

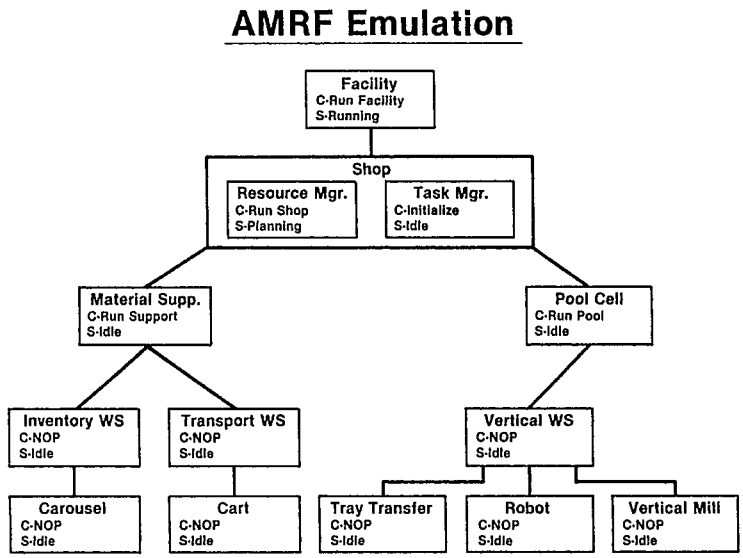
of time spent at each value (Figure 7), along with a sequential logging file (Figure 8).

A particularly useful feature of the emulator is the generation of a rudimentary data dictionary containing a list of all the data elements in the control modules. An illustration of this is given for the simple two-module example described above (Figure 9). The current state of each module (CURS) is always available from common memory, even if it has not been explicitly mentioned. The data dictionary allows the user to rapidly check that modules are consistent in their naming conventions and that minor typographical errors have not occurred in variable names. The system pulls together the occurrence of each variable into one representation that gives both a list of the modules that read and the module that writes

Summary Statistics

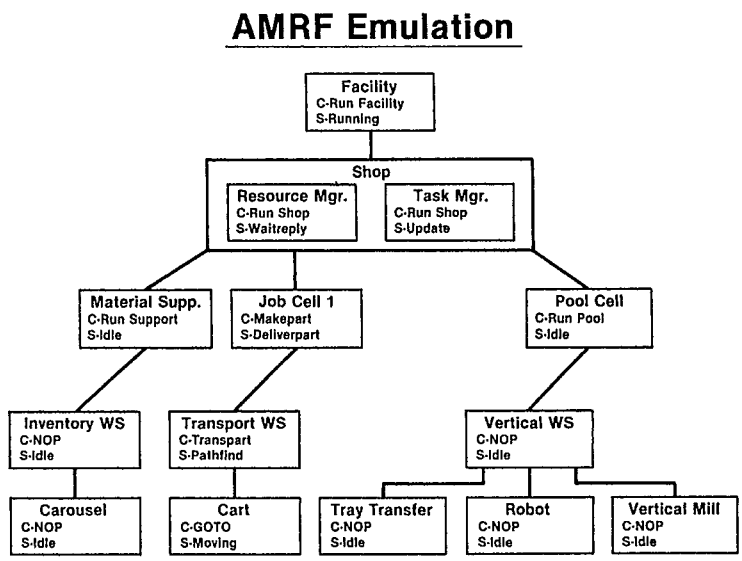
```
There Were a Total of 636 Transitions.
Page Faults Last Value = 95
Direct I/O Last Value = 6
Buffered I/O Last Value = 6
CPU Time (10 msec units) Last Value = 1058
Elapsed Time Last Value = 00:00:22.74
Total Common Memory Writes Last Value = 2436
Total Common Memory Reads Last Value = 2440
COUNT Last Value = 9
      Number of Transitions = 581
      Minimum -10 Maximum 11
COMMAND Last Value = Up
      Number of Transitions = 53
      Other Values
        Duration     %____     Value
        0:00:05.2    8         RESET
        0:00:55.6    91        Up
COUNT2_CURS Last Value = RUNNING
COUNT1_CURS Last Value = NOMATCH
```

Figure 7



Snapshot at Emulated Time of 38 Clock Ticks

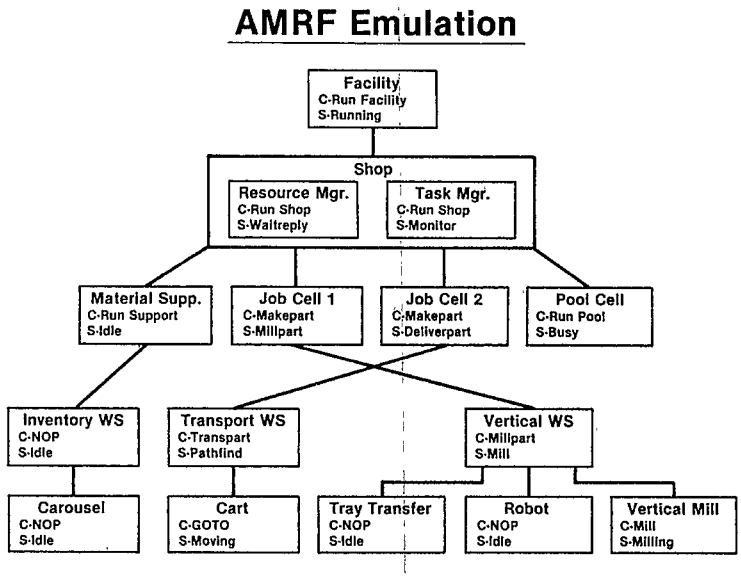
Initialization of the Facility



Snapshot at Emulated Time of 152 Clock Ticks

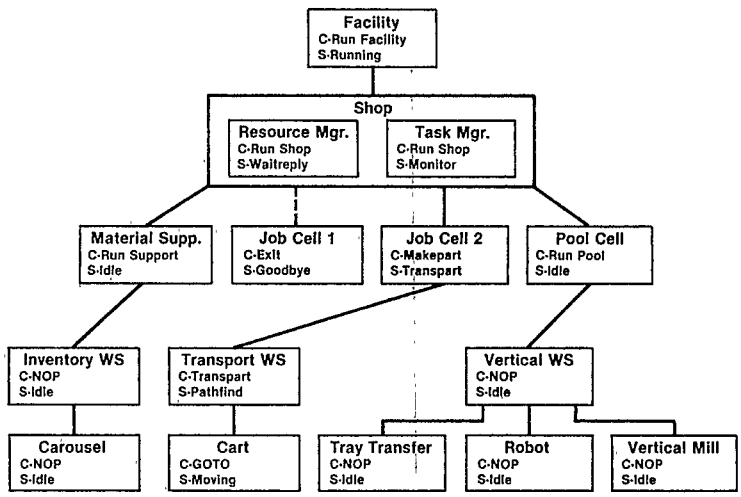
First Job Cell Controlling Transportation Work Station

Figure 10: Emulation of AMRF Control Structure



Snapshot at Emulated Time of 358 Clock Ticks

First Job Cell Controlling Vertical Mill Work Station
Second Job Cell Controlling Transportation Work Station



Snapshot at Emulated Time of 769 Clock Ticks

First Job Cell has Exited
Second Job Cell Controlling Transportation Work Station

Figure 1Ø (Continued): Emulation of AMRF Control Structure

material handling system interfaced with the robot cart. The cell control system was greatly expanded beyond the capabilities described in the above example. With the use of queue managers within the cell it handled the scheduling of many part types (designated as orders by the visitor) and the distribution of orders to the various workstations. The Material Handling Workstation directly controlled the robot cart through the use of an RF communication link. It also interfaced to an inventory system operated by a tray tender who manually loaded blanks (into a tray based on tray configuration instructions from the MHW) and unloaded finished parts.

During the June test run, the graphic display shown in Figure 11 was used. It was run as a separate process to read common memory and display information as that information changed. Each module is represented with its input command (C:) and its current status (S:). In addition, as a mailgram was sent across the network, an arrow appeared to indicate that occurrence, with the direction indicated. In Figure 11 an arrow can be seen indicating that a status mailgram has just been sent from the Horizontal Workstation Controller to the Cell. If the Data Base Manager was active, that information was also displayed by having its module appear, with an arrow indicating which module in the hierarchy it is servicing. In Figure 11, it is servicing the Material Handling System.

Several new features have been added to the AMRF emulator model. These include the following:

(a) Database Management System Interface - An interface was developed between the HCSE and RIM [13] to handle the transferring of data between the different control modules [14].

ACTIVITY OF AMRF CONTROL SYSTEM

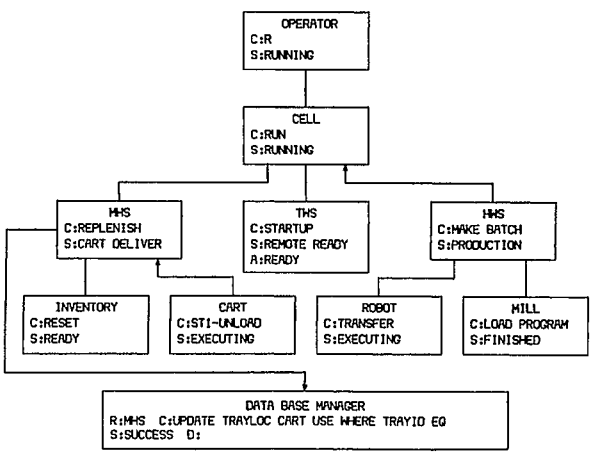


Figure 11

(b) Mailbox communication - An interface was established between the emulator and the AMRF network using the mailbox communication feature [14].

(c) Operator/visitor interface - A full screen display capability using the features of FMS (Forms Management System - a product of DEC) has been interfaced to the emulator to allow for easy viewing and communication to the emulated modules.

FUTURE EMULATOR DEVELOPMENT

Future efforts will be primarily in the area of developing more detailed design descriptions of the AMRF control modules. To accomplish this goal, many new features and capabilities need to be added to the emulator enviroment. An interactive state-table editor will be the user interface to the emulator. This full-screen editor will display state-tables in tabular form and will allow the user to input and modify state-tables in text format. The editor will output the state-transition matrix to an interpretor that implements the common-memory, database interface, and network functions that are necessary for the AMRF control system.

The interpreter will permit the user to dynamically build and interchange state-tables. It will be possible to "collect" state-table programs to implement a model. A library of predefined modules or routines will be implemented. These routines will permit the simulation of processes in any language that uses a standard calling routine. After compilation, these link-edited libraries will be installed in a "shareable library" which will permit the interpreter to dynamically invoke the action routines on demand from a finite state machine. This state-table interpreter is being designed to be transportable, but speed for real-time control is not necessary as the state-tables that are created with this design tool will be translated into the HCSE FSM module format to meet real-time response requirements.

CONCLUSION

The emulator has proven to be an extremely useful tool in the design and implementation of control systems for the AMRF. It has allowed control structures to be tested without concern for the possible destruction of equipment. It has also resulted in the early development of a design specification for the AMRF, a project that involves interfacing control systems being developed by many different groups.

ACKNOWLEDGEMENT

The emulation project is supported primarily by funding from the Air Force/DARPA ITAS program.

REFERENCES

[1] Simpson, J.A., Hocken, R.J., Albus, J.S., The Automated Manufacturing Research Facility of the National Bureau of Standards, Journal of Manufacturing Systems, Vol. 1, No. 1, 1982.

[2] Nanzetta, P., Update: NBS Research Facility Addresses Problems in Set-Ups for Small Batch Manufacturing, Industrial Engineering, June, 1984.

[3] Albus, J.S., et.al., A Control System for an Automated Manufacturing Research Facility, Proc. Robots 8 Conference and Exposition, Detroit, MI, June 1984.

[4] Barbera, A.J., Fitzgerald, M.L., Albus, J.S., Concepts for a Real-Time Sensory Interactive Control System Architecture, Proc. of the Fourteenth Southeastern Symposium on System Theory, April 1982.

[5] Albus, J.S., McLean, C.R., Barbera, A.J., Fitzgerald, M.L., An Architecture for Sensory-Interactive Control of Robots in a Manufacturing Facility, Proc. Fourth IFAC/IFIP Symposium on Information Control Problems in Manufacturing Technology, October 1982.

[6] McLean, C.R., Bloom, H.M., Hopp, T.H., The Virtual Manufacturing Cell, Proc. Fourth IFAC/IFIP Symposium on Information Control Problems in Manufacturing Technology, October 1982.

[7] Jones, A.T., McLean, C.R., A Cell Control System for the AMRF, Proc. ASME International Computers in Engineering Conference and Exhibit, Las Vegas, NV, August 1984.

[8] Barbera, A.J., Albus, J.S., Fitzgerald, M.L., Haynes, L.S., RCS: The NBS Real-Time Control System, Proc. Robots 8 Conference and Exposition, Detroit, MI, June 1984.

[9] (no author), PRAXIS Language Reference Manual, Bolt Beranek and Newman Report 4582, April 1982.

[10] Johnson, T.L., Milligan, S.D., Fortmann, T.E., HCSE User's Guide, National Bureau of Standards, NBS-GCR-82-413, October 1982. Available from: NTIS, Springfield, VA; PB83-141952.

[11] McLean, C.R., Mitchell, M., Barkmeyer, E., A Distributed Computing Architecture for Small Batch Manufacturing Systems, IEEE Spectrum, May 1983.

[12] Hopp, T.H., CAD-Directed Inspection, CIRP Annals, Vol. 33, Madison, WI, August 1984.

[13] (no author), BCS RIM - Relational Information Management System Version 6.0, TR 70101-03-017, The Boeing Company, May 1983.

[14] Mitchell, M.J., Barkmeyer, E.J., Data Distribution in the NBS Automated Manufacturing Research Facility, Proc. IPAD2 Conference, Denver, CO, April 1984.
