Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

DEVELOPING A REAL-TIME EMULATION/SIMULATION CAPABILITY FOR THE CONTROL ARCHITECTURE TO THE RAMP FMS

Wayne J. Davis
Joseph G. Macro
Andrew L. Brook
Michael S. Lee
Guoyan S. Zhou

Department of General Engineering
University of Illinois at Urbana-Champaign
Urbana, IL 61801 U.S.A

ABSTRACT

This paper first discusses an object-oriented, control architecture and then applies the architecture to produce a real-time software emulator for the Rapid Acquisition of Manufactured Parts (RAMP) flexible manufacturing system (FMS). In specifying the control architecture, the coordinated object is first defined as the primary modeling element. These coordinated objects are then integrated into a Recursive, Object-Oriented Coordination Hierarchy. A new simulation methodology, the Hierarchical Object-Oriented Programmable Logic Simulator, is then employed to model the interactions among the coordinated objects. The final step in implementing the emulator is to distribute the models of the coordinated objects over a network of computers and to synchronize their operation to a real-time clock. Specialized displays have also been developed to allow one to monitor the detailed, real-time operation of each coordinated object. The research will also convert the real-time emulation model into a real-time simulation model for the RAMP FMS. Both the real-time emulation and simulation capabilities will be demonstrated at the presentation.

1. INTRODUCTION

Although simulation remains the tool of choice for modeling the behavior of FMSs, the accuracy of current simulation tools in modeling modern manufacturing systems such as Flexible Manufacturing Systems (FMSs) has been questioned. Mize et al. [1991] have concluded that current simulation tools cannot accurately estimate the true performance of an FMS. Rather, these merely permit the modeler to compare the relative performance differences among alternative designs for the FMS.

In our research, we have consistently observed that available simulation tools overestimate the performance for an operational FMS (see Flanders and Davis [1995] for a real-world example). We believe that the inherent inaccuracy of current simulation tools arises from the inability to model *all the constraints* associated with the operation of an FMS. Current simulation tools focus on modeling the primary job entity as it flows through a stochastic queueing network representation of the FMS. Few, if any, simulation tools readily permit the modeler to consider the flow of the supporting resources (e.g. tooling, part kits, fixtures, and processing plans). The coordination of all entity flows in an FMS is crucial, and it is the interactions among the controllers within the FMS that coordinate these flows (see Davis et al. [1993]).

A new simulation methodology has been developed to explicitly model the controller interactions. This methodology also permits the immediate consideration of the detailed processing plans for the parts to be produced in the FMS. In this manner, all entity flows are considered. This paper discusses an application of this simulation methodology to construct a software emulator for the control architecture of a RAMP FMS that is operated by the Naval Air Warfare Center/Aircraft Division/Indianapolis (NAWC AD Indpls). This FMS will produce new and spare electronic circuit boards for defense systems using through-the-hole assembly technologies. The complexity of this system is evidenced by the fact that the cell controller requires nearly a million lines of computer code to implement, excluding the numerous off-the-shelf software packages contained within the controller. Given this complexity, no prior attempt to simulate the entire system has been successful, and production capabilities are still unconfirmed. Furthermore, given the current failure to model the dynamics for this FMS, the ability to schedule this FMS has been compromised. The development of the emulator is the first step toward the development of a new scheduler for this system. This scheduler will be developed within the framework of a comprehensive methodology for the modeling, scheduling and control of FMSs as discussed in Davis et al. [1993].

In this State-of-the-Art tutorial, a detailed discussion of the development of the real-time emulator for the RAMP FMS will be provided. The conversion of the real-time emulator into a real-time simulator for the RAMP FMS will also be addressed. Finally, an on-line demonstration of the real-time emulation and simulation capability will be provided.

2. MODELING APPROACH

2.1 The Coordinated Object

In order to define the set of subsystems contained within a FMS, we sought to find a single modeling template which could be recursively applied to decompose the system into its component subsystems. Working in collaboration with the Government Systems Group at Motorola, we first published the fractal architecture as a mechanism for decomposing the factory into its constituent processing elements (see Tirpak et al. [1992]). During the past few years, the basic fractal unit defined in that architecture has been generalized to become the coordinated object, and we now refer to the fractal architecture as the Recursive Object-Oriented Coordination Hierarchy.

The generalized coordinated object (CO) is depicted in Figure 1. It represents the most fundamental hierarchical element where integrated planning and control are implemented. Each CO contains one or more subordinate task-capable resources or subsystems, P_n (n=1,...,N), which can be allocated to execute the tasks that have been assigned to the CO. In order to execute these tasks, entities, (including both jobs or supporting resources), enter the CO through its input port and eventually exit through its output port. These entities are assumed to be under the control of the CO from the time of their arrival at the input port until they exit through the output port. Consequently, any entity residing in the CO's output queue is controlled by the CO's supervisor since it can be assumed that the CO no longer has any assigned tasks to be performed upon an entity in its output queue.

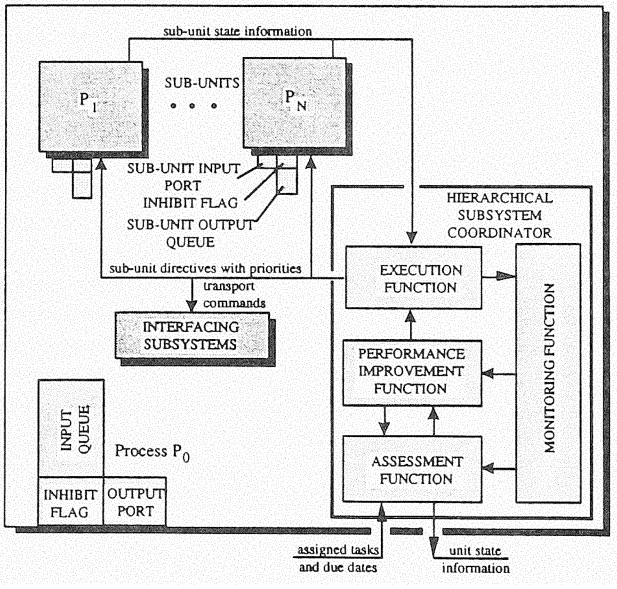

Figure 1. Schematic of the Coordinated Object: A Basic Module for Planning and Control

In general, the CO does not perform the tasks itself. Rather, it decomposes its assigned tasks into subtasks using detailed process plans. These subtasks are then implemented at the subordinate processes or subsystems. When the CO allocates a given entity with an assigned subtask(s) to one of its subordinate subsystems P_n, the physical control of that entity is relegated to that subsystem. Note that in Figure 1 the Input Port and the Output Queues of each subordinate subsystem belong to the CO, while the Input Queue and the Output Port belong to that subordinate subsystem. Therefore, a consistent chain of command for the control of a given entity is defined as the entity flows among the various subsystems contained within the CO.

To move the entities from one subordinate subsystem to another, we assume that each CO has the essential interfacing (material handling) subsystems which contain the transport processes within the CO. These interfacing subsystems are also task-capable resources and must be under the control of the CO. In general, a subordinate subsystem cannot execute its assigned tasks until essential entities and resources are delivered to its control domain. The CO must be able to manage this flow of entities and resources through the control of its subordinate Interfacing Subsystems. Again, the control of the Interfacing Subsystems is manifested via the CO's assignment of tasks to be executed by the interfacing subsystem.

2.2 The Recursive Object-Oriented Coordination Hierarchy (ROOCH)

The recursive nature of the ROOCH arises from the fact that any subordinate subsystem or object within a coordinated object, (including the Interfacing Subsystems), can also be a coordinated object. This recursive approach may be applied to construct the ROOCH with the essential number of hierarchical levels needed to model any FMS.

For an application of the ROOCH, consider the Rapid Access to Manufactured Parts (RAMP) FMS. The ROOCH for the RAMP FMS is pictured in Figure 2.

The recursive nature of the ROOCH is immediately apparent. The RAMP FMS forms the primary CO. Included in the RAMP CO are ten individual subordinate processing centers which perform all the tasks that are required to assemble a circuit board. For the RAMP CO, the primary interfacing subsystem is the central Automated Storage and Retrieval System (AS/RS). The AS/RS is also a CO which includes the primary AS/RS controller and 30 additional controllers to manage the subordinate transport processes within the AS/RS. Another interfacing subsystem is the communication network which downloads the essential processing plans to a given processing center.

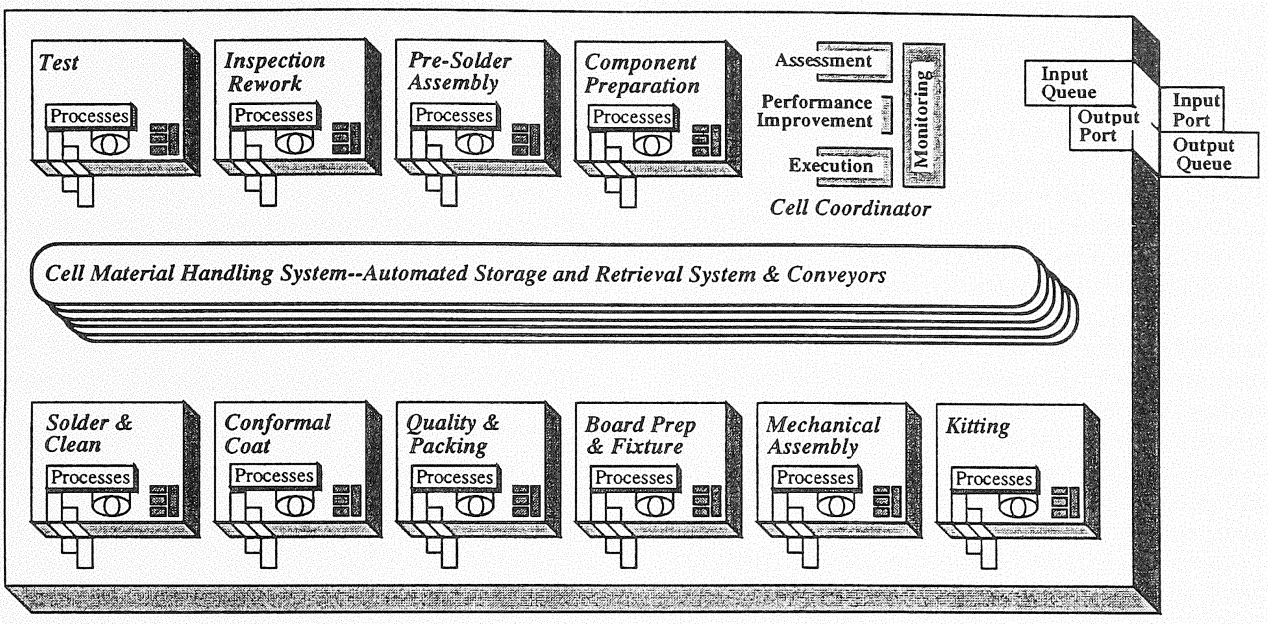

Figure 2. The ROOCH for the RAMP FMS

Each processing center is also a CO containing one or more primary unit processes and a material handler or interfacing subsystem. At most processing centers in the RAMP FMS, the human operator provides the primary material handling. In many cases, the human operator also functions as one of the unit processors. It should be noted, however, that the human operator typically functions only in one capacity at a time. The notion of a unit or transport process is therefore very generic as it can describe a human, a machine or any other type of subsystem that can execute a task.

Despite the apparent complexity of Figure 2, there is a simplicity in the proposed ROOCH. Each included function is a CO except the lowest level process controllers. Each CO is responsible for executing tasks assigned by its supervisor. In most cases, however, the CO cannot execute these tasks itself but rather, defines subtasks to be executed by a subordinate subsystem.

In addition to the assignment of tasks to its subordinate COs, the CO must also direct the flow of jobs and supporting resources to the subordinate COs and processes. This flow of entities is implemented by the CO assigning tasks to its subordinate interfacing subsystems. As is the case for the AS/RS in the RAMP FMS, a given interfacing subsystem may also be a coordinated object. It, too, is responsible for scheduling the execution of its assigned transport tasks. Hence, interfacing subsystems may also possess a dedicated coordination hierarchy which is responsible for both planning and execution of transport tasks.

We usually prefer to distinguish the Processing Subsystem, P_n (n=1,...,N), from the Interfacing Subsystems. The Processing Subsystems are responsible for planning the execution of assigned tasks which physically modify the state of an entity using instructions contained in a process plan. The interfacing subsystems, on other hand, address tasks that support the processing subsystems in generating the desired change in state for a given entity. For example, a material handling system such as the AS/RS may change the location of an entity or store an entity until the next processing task is scheduled. These Interfacing COs do not execute the processing steps contained within the processing plan for the product being manufactured. Rather, the manner in which these execute their assigned tasks is known by the controller and is inherent within the subsystem itself. Therefore, the ensemble of tasks that the Interfacing COs can execute is independent of the product being manufactured. For example, there is a physical layout and control system for most material handling systems which specifies which tasks the material handling system can address and how it will execute each task.

The lowest-level control object in the ROOCH is either a Transport or Unit Process Object. Unit processes are always subordinate to a CO for processing subsystems while transport processes reside within an interfacing subsystem. Neither transport nor unit process objects can have subordinates.

2.3 The Hierarchical Object-Oriented Programmable Logic Simulator (HOOPLS)

As is the case for most large-scale, discrete-event systems, numerous types of entity flows must be considered for the RAMP FMS, including the several types of totes that contain both the production jobs and the supporting resources (tools, fixtures, part kits and so forth).

The flow (location) of all entity types must be modeled by the simulation, and the control of every entity must also be explained. The entities in a modern FMS do not

flow of their own volition. Rather, their flows are managed by controllers. Furthermore, whenever there is an interaction among the controllers, there is likely an exchange of control for one or more entities among the interacting controllers. A new simulation approach, the Hierarchical Object-Oriented Programmable Logic Simulator (HOOPLS), has been formulated to explicitly address these controller interaction concerns.

A HOOPLS-based model for a FMS consists of four primary frames. The first frame is the **model frame** which contains the specifications for the ROOCH associated with the modeled FMS. The second frame is the **control frame** which provides for the exhaustive definition of the control messages that will be issued or received by each controller contained within the ROOCH. This frame also defines the state transition mechanisms which occur upon the receipt of a control message and the subsequent control message(s) that are issued.

Given that HOOPLS explicitly models the interaction among the controllers within the ROOCH and considers the flow of all entities to be a consequence of these interactions, *HOOPLS has abandoned the use of a traditional event calendar.* Instead, HOOPLS employs a message relay which stores the control messages that will be passed among the controllers, chronologically-ordered based upon their delivery time. Each control message designates the controller that issued the message, the recipient controller for the message, the message content, and the scheduled delivery time. The message relay is responsible for delivering the control message to the recipient controller at the appropriate simulated time. It is obvious that the prescribed operation of the message relay physically mimics the communication network which links the controllers in an actual FMS (for the RAMP FMS, this is a dedicated Local Area Network).

The third simulation frame in the HOOPLS-based model is the **processing plan frame**. The processing plan details not only which manufacturing processes will be required, and in which order, but also details which supporting resources will be required to complete a processing task.

The fourth simulation frame is the **experimental frame** which specifies the experimental parameters governing the simulation study. The experiment frame also includes extensive capabilities for initializing the simulation to a known system state.

To demonstrate the modeling objectives for HOOPLS, a simulation/emulation model developed for the RAMP FMS will now be summarized. HOOPLS intrinsically employs an object-oriented architecture, and the C++ programming language was selected for implementing of the simulation model. In addition, the object-oriented design and analysis methods detailed by Booch [1991] were used to specify the class hierarchy for the objects.

3. IMPLEMENTING THE EMULATOR

3.1 Prior Modeling Effort

Space limitations do not permit a detailed discussion of all the essential steps that are required to construct the emulator. Rather, we can only provide an overview of our basic approach. In an earlier paper, we discuss an initial effort to construct the basic object classes for the controllers, entities, and control messages that are needed to describe the RAMP FMS (see Davis et al. [1994]). The ROOCH used for this paper is depicted in Figure 2. This same ROOCH is also employed in the definition of the model frame for the HOOPLS-based emulator.

The distinction between an emulation and simulation is simple. Both employ the same model. The primary difference arises with the manner in which the model is executed. In a HOOPLS-based simulation approach, the messages are stored chronologically on a message calendar. After the state transitions arising from the receipt of a given message at a given controller are implemented and the new controlled messages derived from the processing of the current message are chronologically stored in the message calendar, the next message is popped from the message calendar for processing. At this moment, the simulation time is advanced to the delivery time for the next control message to be popped. A HOOPLS-based emulation uses the same approach save one fundamental difference. The emulation is also tied to a real-time clock. When the next message is removed from the event calendar, it will not be executed until the actual time at which that message is to be delivered to the recipient controller. Thus, the difference between an emulation and simulation may simply be summarized by saying that under a simulation, message processing procedures control the advancement of time while under an emulation, the advancement of time is controlled by a real-time clock.

In the development of the control frame, over seventy controllers were described. As shown in Table 1, the AS/RS alone includes 31 controllers. These controllers are differentiated into four classes of objects. The Coordinate Nodes model the cell and station controllers. The Transport Nodes, (a special case of the Coordinate Node), model the AS/RS and station-level material handler controllers. The Unit Process Nodes model the process controllers at each station. Finally, Transport Process Nodes model the dedicated transport process controllers within the AS/RS. Using this class definition schema, Coordinate Nodes can have other Coordinate Nodes, Transport Nodes and Unit Process Nodes as subordinates. A Transport Node can have only Transport Process Nodes as subordinates. Unit and Transport Process Nodes cannot have subordinates.

In the control frame, we establish the communication scheme for the controller interactions. Here, we define two classes of message. An Action Message requests a subordinate to take execute an action. A Status Mes-

RAMP Cell Controller
- Kitting Station
  - Processor : Human Operator
  - Material Handler : Human Operator
- Board Preparation Station
  - Processor : Human Operator
  - Material Handler : Human Operator
- Component Preparation Station
  - Processor : Human Operator
  - Material Handler: Human Operator
  - Part Carousel
  - Tinning Robot
- Pre-Solder Assembly Station
  - Processor : Human Operator
  - Material Handler: Human Operator
  - Part Carousel
  - Part Location Indicator
- Test Facility Station
  - Processor : Human Operator
  - Material Handler: Human Operator
  - Burn In Tester
  - Conductivity Tester
- Inspection/Rework Station
  - Similar to Board Preparation Station
- Mechanical Assembly Station
  - Similar to Board Preparation Station
- Conformal Coat Station
  - Similar to Board Preparation Station
- Quality & Packaging Station
  - Similar to Board Preparation Station
- Wave Solder and Clean Station
  - Similar to Board Preparation Station
- Automated Storage and Retrieval System
  - Storage layer Conveyors (6)
  - Inserter/Extractor (5)
  - Machine Station Input/Output Conveyors (20)

Table 1. Controllers in the RAMP FMS

sage provides feedback to the supervisor at the completion of the requested action.

With the interaction of either a Coordinate Node or Transport Node with a subordinate of any class other than a Transport Node, we have three basic action messages. AcceptItem instructs the subordinate that an entity is in its input queue and that the subordinate now has control of that entity. ReturnItem instructs the subordinate to place the entity in its output queue and return the control to the requesting supervisor. ExecuteTask requests the subordinate to execute a specific task upon a given item. If the subordinate is a Coordinate Node or Unit Process Node, this task will be defined within the process plan database. If the subordinate is a Transport Node, then the instruction is predefined within the process's capabilities.

Kitting: Generate the parts kits for the order
- Input: KIT tote containing all order components
- Output: Bare Board Tote containing up to 10 boards
  Pre- and Post-Solder Assembly Tote(s)
  Mechanical Tote holding large parts

Board Preparation: Place board into fixture
- Input: Bare Board Tote with bare boards
- Output: One Pallet Tote for each fixtured board

Component Preparation: Bend/tin component leads
- Input: Pre- and Post-Solder Assembly Tote(s)
- Output: Same

Pre-Solder Assembly: Place components on board
- Input: Pre-Solder Assembly Tote(s) and all Pallet Totes for given order
- Output: Same

Pre-Solder Inspect/Rework:
- Input: Pallet Totes and Pre-Solder Assembly Tote(s) if rework is needed

Output: Same

Wave Solder and Clean:
- Input: All Pallet Totes
- Output: Same

Post-Solder Assembly: Mount small parts
- Input: All Pallet and Post-Solder Assembly Tote
- Output: Same

Post-Solder Inspection
- Input: All Pallet Totes with Pre- and Post-Solder Assembly Totes if reworked
- Output: Same

Mechanical Assembly: Remove board from fixture and mount larger mechanical parts.
- Input: All Pallet and Mechanical Part Tote
- Output: Printed Wiring Assembly Tote for each board and Mechanical Part Tote

Test: Perform bed-of-nails and burn-in test.
- Input: Printed Wiring Assembly Totes for job and various part totes if reworked
- Output: Same

Conformal Coat: Apply protective coating to board.
- Input: Printed Wiring Assembly Totes for order
- Output: Same

Final Quality Control and Inspection/Packaging:
- Input: Post-Wiring Assembly Totes
- Output: Packaged boards

Table 2. Processing Steps for Production in the RAMP FMS

When the subordinate to the Coordinate Node is a Transport Node, there are two action messages. DelieverItem instructs the Transport Node to deliver an entity to a requested station while PickupItem instructs the Transport Node to pickup an entity at a specified station. Each of the above action messages has a corresponding status message which notifies the supervisor when the requested action is completed.

The process planning frame specifies the details for all the processing instructions needed to manufacture a given part. A generic processing plan for any part manufactured in the RAMP FMS is given in Table 2. Note, however, that the actual process plan contains considerably more detail pertaining to the execution of each task. For example, detailed instructions on how to bend the leads for each component and where the component is to be placed upon the board are provided.

Finally, the experiment frame is very similar to that of most simulations detailing the length of the simulation run and other parameters that are needed to specify the simulation experimental trial. The experiment frame under the HOOPLS paradigm also provides extensive provision for initializing the simulation to a known state.

3.2 On-Going Development of RAMP Emulator

The development of the emulator is initiated as a "clean-sheet" design. All that is being retained from the prior development of the HOOPLS-based simulation model is the basic structure of the ROOCH and the control messages that will be employed. However, the list of control messages will probably be expanded. A distributed object architecture is being employed in the development of the emulator such that each controller can be positioned on a different computer if desired. The controller objects are being coded in C++.

In developing the new emulation model, the greatest effort has been devoted to the definition of the state variables for each coordinated object, especially the state variables for the AS/RS. There are several reasons for the focus upon the AS/RS. The AS/RS includes 30 transport processes. The complete AS/RS will have a total of 31 controllers, nearly half the total number of controllers in the entire RAMP FMS.

In defining the state variables for each control object, there are fundamental issues to be addressed. For example, should the AS/RS controller know the contents of the totes which contain jobs and supporting resources while they are being stored and transported by the AS/RS? This single issue has been the subject of hours of debate. Our current design assumes that the AS/RS will know the contents of the totes. This decision extends the general capability of the formulation as the cell controller can request to the AS/RS that a given item be moved without specifying the tote that it is in. By making this design decision, the AS/RS can easily be replaced by another material handling system, one which may not employ totes. In fact, there are presently other versions of the RAMP FMS which do not include an AS/RS.

Distributing the controllers across several computers creates several other problems. First and foremost, a method of communication between the objects on different computers must be developed. The computers, (in this case, Sun workstations), are networked. If a control object wants to send a message to another control object across this network, it needs to know on which computer the object resides. To fulfill the communication requirements, we have written our own mail system which knows the address of every control object. When an object desires to send a message, it must specify the recipient of the message, the time that the message is to be delivered, and the contents of the message. The message is then placed into the local computer's mail box. If the recipient object resides on the same computer, the message-passing is handled by the local mailer. If the recipient controller is located at another computer, then the message is routed to the mailer on the computer where the recipient object resides.

Messages are retained in the recipient object's mailbox until the scheduled time of their delivery. When the messages are delivered, the recipient controller executes the appropriate state transitions and generates its response messages which it then places in its mail box for transmission to other controllers at the desired time.

Using the same message service, we have also included programmed features which will permit the emulation to proceed at a rate that is faster than real time. Specifically, we have included a clock object which monitors a real-time clock and advances the emulated time at a time-scale factor greater than real time. The clock object sends messages to each computer's mail service specifying the current emulated time, which the mail service then uses to determine when messages are to be forwarded to their recipient controllers.

Each controller type also has its own specialized display, which was developed under X-Windows/Motif®. Within each controller display, the state of the controlled subsystem will be constantly updated. Much time has also been devoted to the development of the displays. Our desire is for an individual to be able to view a given controller's window and visualize the operation of the subsystem that is being controlled. These displays also are especially useful when validating the model. These provide a far more detailed depiction of the operation of the system than that which can be achieved using current simulation and animation techniques. Current animation capabilities provide iconic displays which permit only the basic entity flow among the various objects to be depicted. Using our emulation capabilities, the modeler will be able to access the detailed state information for each modeled object within the simulation as it evolves in real-time. The interaction among the primary modeled controller objects can also be visualized by monitoring the control messages that are being transmitted among the controllers.

3.3 The Demonstration

At the State-of-the-Art tutorial, we plan to demonstrate the real-time emulation of the RAMP FMS. The ability to execute this demonstration is derived from another unique capability which we have included within the emulator. As stated above, we chose to employ a distributed-

object programming architecture within the emulation which permits each controller to be executed on a different platform. Each platform is connected to the Internet. To provide a future capability to model all types of RAMP FMSs, it was also necessary to implement the display window to each controller as a distinct programmed object from the programmed object representing the controller itself. Thus, a given controller and its associated control window are separate programmed objects which can also run on different computers. This design decision also enhanced the virtual manufacturing capability by permitting the operations of the RAMP FMS to be viewed and eventually controlled from any computer on the Internet.

The computer code for the control displays is situated at the computer where the displays are viewed. When this code is executed, the display object initially logs into the message service for the emulation and provides the message service with its IP address on the Internet. Eventually, the message service will request the viewer's name and password which it will then employ to control access to certain state information and to limit the viewer's capability to interact with the operating RAMP FMS. Once the viewer has gained access, the primary control window for the RAMP FMS cell controller is opened at the remote location. The message service then notifies the cell controller for the RAMP FMS to begin sending state information to the requesting window object at intervals of approximately 1 second duration.

The displaying software is contained at the distal location. Therefore, when the display object receives the state information, it then employs the state information based upon the control window being viewed. It is important to observe that only textual state information is being transmitted across the Internet, not the entire pixel information that is needed to construct the control window. Therefore, the information transmission requirements are minimal.

The remote viewer can interact with the system as various elements of a given controller's window are user-selectable icons. For example, when the viewer clicks the icon of the AS/RS within the cell controller window, the viewing programming immediately opens a window for the AS/RS controller. The program then identifies the message service that the viewer wishes to monitor the operations of the AS/RS. The message service then tells the cell controller to temporarily suspend state updates to the remote viewer and simultaneously tells the AS/RS controller to start sending state information updates. Within the AS/RS controller window, the viewer may choose to have detailed information presented for a given insertor/extractor or level of the storage conveyors.

As stated above, a simulation and an emulation differ only in the manner in which they advance time. When the emulation of the RAMP FMS is complete, the emulation model will immediately be converted to a simulation model for the RAMP FMS. Multiple instances of the resulting RAMP FMS simulation model will then be assigned to distinct processors. Each instance of a simulation model (referred to as a simulation engine) will operate under a different control strategy and each simulation engine will receive current state information updates from the control objects executing the real-time emulation. Operating under the selected control strategy, each simulation will perform a real-time simulation of the future response of the system over a future planning horizon given the current state of the emulated system. Each simulation engine will attempt to generate real-time simulation trials as quickly as possible.

Using the included message service, the output of these real-time simulation trials will be forwarded to the various programmed objects which perform the essential real-time statistical and compromise analyses which are needed to select the best control strategy among the considered strategies for implementation. This capability will be demonstrated at the tutorial.

Given space limitations, we cannot discuss all the components of the real-time simulation analysis in this paper. Our basic approach to real-time simulation has been previously discussed in Davis, Wang and Hsieh [1991] and Tirpak, Deligiannis and Davis [1992]. In a chapter of the forthcoming Handbook on Simulation (see Davis [1997]), we discuss evolving directions in simulation tools and real-time simulation. This chapter will provide an overview of the real-time simulation and emulation technology which will be demonstrated at the State-of-the-Art tutorial. Harmonosky [1995] also provides an abridged summary of ongoing research in real-time simulation and scheduling.

5. FUTURE WORK

The emulator will be completed in time for this paper to be presented. After the emulator is completed, the emulation code will be modified such that it can operate as a second generation, HOOPLS-based simulation model for the RAMP FMS. This simulation model then will be employed to perform real-time simulation (see Davis, Wang and Hsieh [1991]). This is a necessary first step to develop a real-time production scheduler for the RAMP FMS (see Davis [1992] and Davis et al. [1993].

REFERENCES

Booch, G. 1991. *Object Oriented Design with Applications*, Benjamin Cummings, Redwood City, California.

Davis, W. J., H. Wang and C. Hsieh. 1991. Experimental Studies in Real-Time, Monte Carlo Simulation. *IEEE Systems, Man and Cybernetics*, 21(4), 802-81.

Davis, W. J. 1992. "A Concurrent Computing Algorithm for Real-time Decision Making." *Proc. of the ORSA Computer Science and Operations Research: New Developments in their Interfaces Conference*, eds. O. Balci,

R. Sharda and S. Zenios, 247-266, Pergamon Press, London.

Davis, W. J., Setterdahl, D., Macro, J., Izokaitis, V. and Bauman, B. 1993. "Recent Advances in the Modeling, Scheduling and Control of Flexible Automation." *Proc. of the 1993 Winter Simulation Conference*, eds. G. W. Evans, M. Mollaghasemi, E. C. Russell, W. E. Biles, 143-155, The Society for Computer Simulation, San Diego, CA.

Davis, W. J., J. G. Macro and D. L. Setterdahl. 1994. "An Object-Oriented, Coordination-Based Simulation Model for the RAMP Flexible Manufacturing System." *Proc. of the Flexible Automation and Integrated Manufacturing Conf.*, eds. M. M. Ahmad and W. G. Sullivan, 138-147, Begell House, Inc., New York.

Davis, W. J. 1997. "Reat-Time Simulation: The Need and the Evolving Research Requirements." To appear in *The Simulation Handbook*,Wiley, new York.

Flanders, S. W. and W. J. Davis. 1995. "Scheduling a Flexible Manufacturing System with Tooling Constraints: An Actual Case Study." *Interfaces,* 25, 42-55.

Harmonosky, C. M. 1995. Simulation-Based Real-Time Scheduling: Review of Recent Developments. *Proc. of the 1995 Winter Simulation Conf.*,eds. C. Alexopoulos, K. Kang, W. R. Lilegdon and D. Goldsman, 220-225, The Society for Computer Simulation, San Diego, CA.

Mize J. H., H. C. Bhuskute, and M. Kamath. 1992. " Modeling of Integrated Manufacturing Systems." *IIE Transactions* , 24(3):14-26.

Tirpak, T. M., S. M. Daniel, J. D. LaLonde and W. J. Davis. 1992. "A Fractal Architecture for Modeling and Controlling Flexible Manufacturing Systems." *IEEE Trans. on Systems, Man and Cybernetics*, 22(5), 564-567.

Tirpak, T. M., S. J. Deligiannis and W. J. Davis. 1992. Real-Time Scheduling of Flexible Manufacturing. *Manufacturing Review* (ASME), 5(3), 193-212.

ACKNOWLEDGMENT

This research is supported by the US Army Research Laboratory/Battelle Grant TCN 95-300 and the corporate members of the Hyper-Linked Manufacturing Consortium.

BIOGRAPHICAL SKETCHES

WAYNE J. DAVIS is a professor of General Engineering at the University of Illinois at Urbana-Champaign. His research areas include distributed planning and control architectures for large-scale, discrete-event systems; computer-integrated manufacturing; and simulation.

JOSEPH G. MACRO is a Ph.D. candidate in Mechanical and Industrial Engineering at the University of Illinois at Urbana-Champaign. His research interests include advanced simulation methodologies and computer-integrated manufacturing.

ANDREW L. BROOK is an undergraduate student in Computer Science at the University of Illinois at Urbana-Champaign.

MICHAEL S., LEE is an undergraduate student in Computer Science and General Engineering at the University of Illinois at Urbana-Champaign.

GUOYAN ZHOU is a master's degree candidate in Computer Science at the University of Illinois at Urbana-Champaign.