

## INITIALIZING ON-LINE SIMULATIONS FROM THE STATE OF A DISTRIBUTED SYSTEM

Fernando G. Gonzalez

Department of Electrical and Computer Engineering  
University of Central Florida  
Orlando, Florida 32816 USA

Wayne J. Davis

Department of General Engineering  
University of Illinois at Urbana-Champaign  
Urbana, Illinois 61801 USA

### ABSTRACT

In this paper, we address the complex task of initializing an on-line simulation to a current system state collected from an operating physical system. The paper begins by discussing the complications that arise when the system model employed by the controller and the planner are not the same. The benefits of using the same model for control and planning are then outlined. The paper then discusses a new simulation paradigm that models controller interactions and provides a single model that is capable of supporting planning and control functions. Next, issues arising from performing a distributed simulation of the distributed control architecture that is being employed to manage the system are addressed. The definition of the state for the distributed system is then discussed and the collection of the real-time state information from the elements of this distributed system is outlined. Finally, the procedure for initializing the distributed on-line simulation from the collected real-time state information is given.

### 1 INTRODUCTION

In most intelligent control applications associated with the operation of complex discrete-event systems, the control system must be able to employ the state information that is collected from the managed hardware in order to initialize on-line simulations, see Davis (1992). These on-line simulations can be employed within the intelligent controller to project the future performance of the system for both planning and control purposes. For example, on-line simulations may be used to project the future performance of the system while operating under one or more alternative plans. They also may be used in order to permit the controller to employ predictive control practices where the future consequences of a given control action is assessed before it is implemented. This real-time collection of the state information from the physical system under operation and the subsequent use of this real-time

state information to initialize on-line simulation runs are often complex tasks. In this paper, we demonstrate the benefits of using a simulation tool that is capable of providing a model that is applicable to both the analysis and control of the system has on fulfilling the initialization requirements for on-line simulation.

In general, the state of any computer program is dependent upon its data, including the values currently assigned to its variables and data structures. Initializing a program to a specific state necessarily requires one to assign values to every variable. This requirement holds for simulations as well. In order to initialize a simulation to the state of the physical system, one must initialize each variable and data structure such that the state of the program represents the current state of the physical system. In most simulation applications, this is a complex task because the variables employed to define the state of the system and the state of the simulation model are different. In addition, if one employs commercial simulation languages, many of the variables that must be initialized are inaccessible to the user.

The task of gathering the state information from the controller can also be difficult. In many cases, the internal data structures employed by the controller are also hidden from the user. Even if these data are accessible, the user still must determine the relationship of the data to the current state of the system as it is being modeled. That is, complete data dictionaries for the variables being employed by the controller and the simulation model must be known. For most commercial controllers, this information is not available. In summary, when one employs commercial simulation and control tools, one seldom is capable of defining a congruent representation for the system state which both the simulation model and controller can employ. Hence, initializing the simulation model to the current system state becomes extremely difficult if not impossible to achieve.

One obvious solution to this problem is to employ the same state representation in both the simulation model and

the controller. An even more obvious solution would be to employ the same model for the system state transition functions in both the simulation and controller functions. Thus, the model that the controller employs to manage the system would be the same as the simulation model employed in the planning process. We believe that this capability can be achieved by considering the interactions among the controllers as the primary basis for system evolution rather than the flow of entities through a stochastic queuing network, the approach which has been adopted by most commercial simulation tools. Although most simulators do not have the capability to model controller interactions or control the system, the integration of simulation and control has been gaining popularity in recent years, see Peters, Smith, Curry, and LaJimodiére (1996). Recently, our research team has developed a new simulation tool that explicitly models the interactions among the controllers in order to describe the state transition mechanisms that govern the evolution of the system, see Gonzalez and Davis (1997). The models developed by this simulation tool can either be used to project the performance of the system for planning purposes or be employed by the controllers in managing the response of the system. Since the state representation is the same for both the planner and the controller, the task of collecting real-time state information and then employing the collected information to initialize any on-line simulations is significantly simplified. However, there are still significant challenges to be addressed, which we will discuss later in the paper.

Our tool has two modes of operation: a control mode and a simulation mode. These two modes of operation differ only in the way they handle the advancement of time and in the order in which they execute events. In order to provide the simulation tool with this dual capability, a new data structure called the *pending event list* was added. This list contains the events that are waiting for a response from the hardware before they can be executed.

A simulation or control program consists of two basic components: the simulation tool which manages the execution of events and the model which describes which events can occur. Our simulation tool, like most simulation tools, consists of three basic parts:

- The executive function, which manages the execution of events under the constraints of the employed system model.
- The data structures, which include the scheduled event list, the pending event, and the resource list.
- The set of modeling elements that is used to construct the system model.

Our tool encapsulates the system model represented by a collection of C++ objects into a comprehensive simulation executive object. This encapsulation has been

developed in a manner that will support the simulation of distributed systems, a capability which will be discussed later. The included objects contain the executive function, the scheduled and pending event lists, a list of available resources, and two pointers, one to the system model and one to the communication program (See Figure 1). The collection of objects also contains the modeling elements which are used to construct the system model.

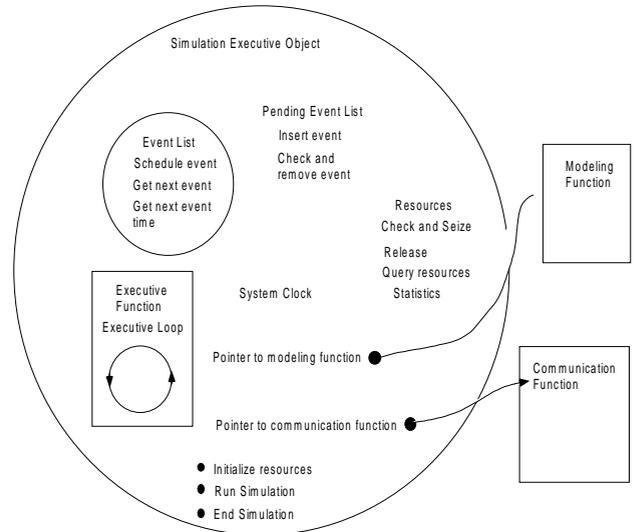


Figure 1: The Executive Object. Note: the Circles Represent the Data Structure Objects, the Square Represents the Executive Function, and the Flat Rectangles Represent Pointer Variables.

A problem arises when one simply copies the data from the controller to the simulator in order to initialize the simulation to the state of the controller. Since the controller uses both the pending and the scheduled event lists while the simulator only uses the scheduled event list (the simulation does not interact directly with the equipment), one must make some accommodations for the events that are contained within the controller’s pending event list. Since the simulation does not receive messages from the hardware, there is no direct means for determining when these events will occur. Rather, the initialization process must estimate when these pending events will occur and then transfer the events from the pending event list to the scheduled event list. The scheduled event time for each pending event is calculated by sampling a random deviate from a previously specified distribution of the duration time for each pending event type contained within the simulation model. Once a random duration is sampled for the particular pending event, this duration minus the amount of time the event has already been waiting provides the time in the future when the event will occur. If the event time is computed to have occurred before the current

time, then the event is scheduled to occur now. The controller may also help the simulator in this matter by computing an estimated time for every new event that is inserted into the pending event list and storing it into the event as an attribute. If this approach is adopted, the pending event can be immediately transferred to the scheduled event list and scheduled to occur at the time predicted by the controller.

## 2 DISTRIBUTED SYSTEMS

One of the focuses of our research has been the development of distributed simulation capabilities that can more faithfully emulate the distributed control environment that occurs in the management of real-world systems. To this end, we have developed an even more comprehensive modeling paradigm called the Hierarchical Object-Oriented Programmable Logic Simulator (HOOPLS), which simulates the distributed control architecture by modeling the interactions among the controllers, see Davis, Macro, and Setterdahl (1997) and Davis, Setterdahl, Macro, Izokaitis, and Bauman (1993).

The HOOPLS paradigm is based upon the belief that interactions among the controllers must be considered by the simulation model in order to accurately model a system with a distributed control architecture. The single most important characteristic of the HOOPLS-based methodology, and what separates it from other object-oriented simulation approaches, is its attention to modeling the flow of messages among the controllers included within the distributed control architecture.

This paradigm is implemented by including all of the controllers into a single simulation program. The independence of the controllers is maintained by encapsulating each controller into an object. The included controllers execute independently of each other by having a supervisory function simulate the communication network that passes messages among the controllers and manage the order in which the messages get transmitted, see Gonzalez and Davis (1997). In its emulation of the communication network, the supervisory function receives all of the control messages that are transmitted by the control object and places them into a message queue where they are ordered based upon their simulated delivery times. The supervisor then sends each message to its recipient control object at its scheduled delivery time (see Figure 2).

The primary reason for developing the HOOPLS methodology was to more accurately model systems that have a distributed control architecture. In order to test the ability of a single model to support both planning and control, we constructed a physical emulator of a flexible manufacturing system, see Gonzalez and Davis (1997). The control architecture for the physical emulator is distributed across 25 independent controllers, each

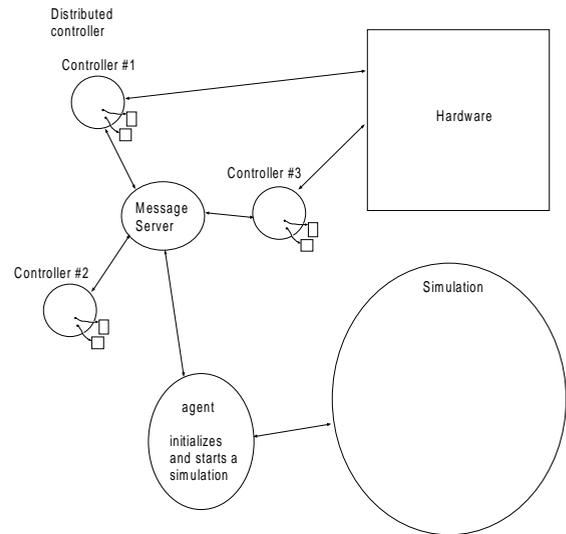


Figure 2: The overall system. On the left is the controller with the message server. On the right is the on-line HOOPLS simulation. The agent connecting the two tells the server to get the system state, then gives this state to the simulator in order to initialize a simulation run with the system state.

consisting of an instantiation of our software tool (see **Figure 3**). The distributed system was then simulated under the HOOPLS methodology. The state of the distributed controllers was then successfully gathered and used to initialize several on-line simulation runs which were used for planning purposes.

In the control of distributed systems, the control objects communicate with each other. The pending event list is used in the same manner that it was used for single systems. At each controller, the pending event list holds the events that are waiting for a response from an external agent (which can be either another controller or a controlled piece of equipment). When simulating the distributed system using the HOOPLS methodology, the control objects continue to communicate with each other. In fact, when we simulate the system using the developed distributed simulation capabilities, the software tool is actually running in control mode. The only difference between simulation and control mode is in the way the clock is incremented and in the way events are scheduled to be executed. For simulation of distributed systems, the time is advanced by the supervisory function and not independently by each control object as is the case when the simulation tool operates in the control mode. In addition, the models communicate with each other as they do when they are physically controlling the system.

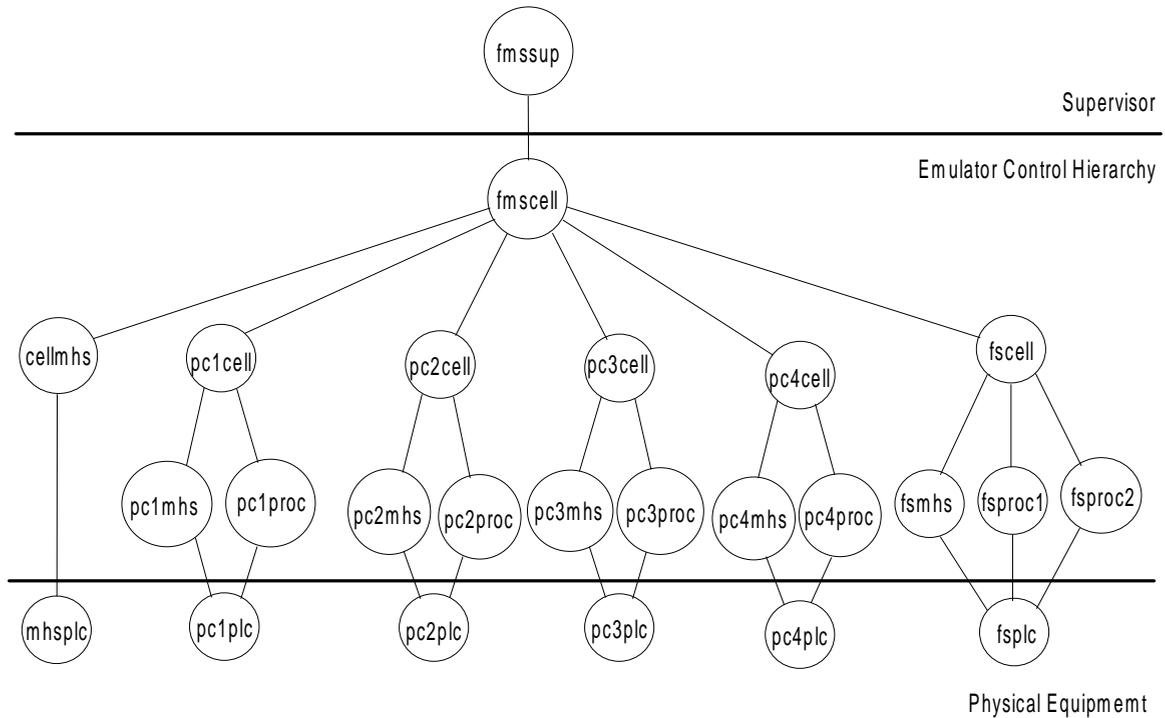


Figure 3: The Implemented Control Hierarchy for the Physical FMS Emulator. The Submodels That are Presented Between the 2 Horizontal Lines are the Actual Controllers Used Both to Control and Simulate the System. The Rest of the Models Were Created to Provide a Complete Model for Simulation Only; They Are Not Used in Control Mode.

distributed systems, the time is advanced by the supervisory function and not independently by each control object as is the case when the simulation tool operates in the control mode. In addition, the models communicate with each other as they do when they are physically controlling the system.

In initializing the distributed model for an on-line simulation, the events in the pending event list can remain there as opposed to moving them to the scheduled event list as is done for the simulation of single thread systems (i.e. a non-distributed simulation). This avoids the task of having to move the pending events into the scheduled event list and makes the initialization task one of simply copying the variables and data structures of the controller's program into the simulation's program. However, since some of the controller objects must communicate with the physical hardware, for the purpose of simulation, objects representing the physical machines must be added to the collection of models. These equipment objects are the submodels that are shown below the bottom horizontal line in Figure 3. These latter models exist for simulation purposes only. During control mode implementations of the model, these models would be replaced by the actual machines. The inclusion of these equipment models complicates the task of defining the state of the overall system. For these equipment objects, one can not copy the

data structures from their controller models since these models do not exist. Fortunately, these models represent the most basic task-implementing elements of the system at the lowest level of the control architecture. They generally consist of one simple queue and a process. Initializing their models is usually a matter of inserting an entity into their queue and possibly an event into their event list. In our implementation, generating a state representation for these equipment models proved to be a straightforward task.

### 3 THE DEFINITION OF THE STATE

The state of the system provides a complete description for all the variables that change as the system evolves in time. Some variables are initialized at the beginning of the simulation run and then remain constant. For example, the total number of Automatic Guided Vehicles (AGVs) within a given submodel is typically constant. Thus, these variables are not considered to be a part of the state because they do not change with time. The following grammar has been developed in order to define the state for any model or system of models developed using our simulation tool where the control model and the simulation model are the same.

## Initializing On-Line Simulations from the State of a Distributed System

system state :	( (msg) ... (msg) ) ( (name objstate) ... (name objstate) ) ( time )	; the messages ; the control object's states ; the system time
msg:	fromaddr toaddr msg operand1 sn ( str1 str2 ... )	; the source address ; the destination address ; the message ; the operand ; the serial number of the destination entity ; the operand that is a queue
objstate:	( ( (ent) ... (ent) ) ... ( (ent) ... (ent) ) ) ( ( (ent) ... (ent) ) ... ( (ent) ... (ent) ) ) ( ( (ent) ... (ent) ) ... ( (ent) ... (ent) ) ) ( (evt) ... (evt) ) ( (evt) ... (evt) ) ( results ) ( resmax ) ( resalter ) ( needalter ) ) ( time ) ( model )	; the states of the QUESEIZE elements ; the states of the PENDING elements ; the states of the DELAY elements ; the state of the scheduled event list ; the state of the pending event list ; the state of the list of available resources ; the state of the list of total resources ; the state of the list of alterations to be made ; the state of the alterations request flag ; the time of the control object ; state information used by the model
ent :	sn attr1 attr2 ... attr10 strattr1 strattr2... strattr10 timein param1 param2 ... param10 (pp1 pp2...) (retaddr1 retaddr2 ...) pri	; the serial number of the entity ; the 10 attributes ; the 10 string attributes ; the time in attribute ; the 10 parameters ; the process plan queue ; the return address queue ; the entity's priority
evt:	type whichone eventtime key1 key2 key3 sn	; the event type ; the modeling element number ; the time the event is to occur ; the keys for matching pending events ; the event serial number

Figure 4: The State of the System

The state of the overall system, which includes all of the sub-models in a distributed system, consists of the following three items. The messages that were in the network at the time the control objects generated their state data, the state of each of the control objects in the control architecture, and the current time. The rationale for including the messages as part of the state will be discussed later. Each control object state is preceded by the object's name. This allows the process for building the system state to gather the state for each controller in any order.

The state of the data structures (msg, ent, and evt) is simply the data contained in the structures. The label msg is for the structure that contains the messages, ent for entity, and evt for event. Note that if the data structure contains a queue, then the data for the items in the queue are enclosed by parentheses. The parentheses are used to

tell the parser which data are associated with the queue, because the queue can contain any number of items. In interpreting the system state, the parser employs a finite-state machine. The parentheses in the system state description are used to guide the parser through the state information.

In our tool there are three modeling elements that have the capacity to store entities. The QUESEIZE element models the entities that are waiting for resources to become available. The DELAY element models entities that are being processed and are waiting for a scheduled completion event to occur. The PENDING element holds entities that are waiting for a message from an external agent to arrive. Thus, the QUESEIZE, PENDING and DELAY elements must have a queue to hold the entities. These are also the only data structures associated with the modeling elements that need to be included in the state

definition. The rest of the data are either statistical variables, which are reinitialized before each on-line simulation run, or variables whose value remain constant throughout the simulation. Because the queue is the only data that is relevant in the state, the state of these modeling elements can be defined by the list of the state variables for each entity that is in its queue. In general, there may be any number of a given type of modeling element in the model. For example, the model may include three QUESEIZE, four DELAY and one PENDING element. The first line of the control object's state definition specifies its QUESEIZE elements. Following this line, the first set of parentheses is for the QUESEIZE1 element, the second is for the QUESEIZE2 element, and so on. Within each set of parentheses is a list of the entities that are in the associated element's queue, as specified in the first line for each control object. The state data for the PENDING and DELAY elements are organized in a similar fashion.

The scheduled event-list and pending event list contain only the list of events as their object's data. Therefore, the state for these lists includes each event that is currently contained in the list. The resource states are implemented as tables of integers. Each resource has two integers associated with it. The first is the number of available resources of that type and the second is the total number of resources of that type in the system.

Some of the queues that are stored in the state representation have a particular order based upon an employed priority scheme. Even though the order of each queue is always maintained in the state, these associated priorities must be saved separately. In the case of the event and the pending event list, the priority can be obtained from the events themselves. The queue associated with a DELAY modeling element uses the time that the delay event will occur for the order of the entities in the queue, and this event time must be stored along with the entity.

#### 4 THE GATHERING OF THE STATE

The gathering of the state is a delicate operation that requires some thought. Any message that is in the network at the time the state is generated must be accounted for by including it as part of the state. If these messages are not correctly placed into the state description, then the following situation can occur. One controller's state description reflects the object as waiting for a message from a sending object. However, the sending object's state description reflects the message as already being sent and, thus, does not send the message. This situation can put the simulation into a deadlock state. Hence, the messages that are part of the state must be put into the message relay before starting the on-line simulation. This initialization task correctly identifies the messages that were in the network at the time the state was gathered. Because gathering the system state requires knowing where

messages are at the time the state is measured, the best agent to gather the state is the message server. In the control architecture, all of the messages going from one control object to another are routed through this message server (see Figure 2). The message server is the agent that routes all of the messages and keeps a log of all messages that are currently being transmitted. It is also responsible for collecting the state data from each control object, along with the anticipated messages, and then generating the complete system state.

Since the controller must continue controlling the system while it gathers the state information, it is necessary to minimize the disruption that gathering the state has upon the controller. It is also essential that the controllers be able to continue communicating with each other while the state is being gathered. For this reason, one cannot simply tell all of the controllers to stop sending messages and wait until all of the messages in the network arrive at their destination before starting the state gathering process. This method of emptying the network before gathering the state is intuitively desirable, but no controller can be suspended for the amount of time it takes to clear the network. The network must continue to transmit messages and the controllers must continue to operate while the state is being gathered. Thus, the agent that gathers the state must do some bookkeeping in order to determine which messages should be included as part of the state and which should not.

Our controllers were distributed across several UNIX workstations. Since the UNIX sockets preserve the order in which messages travel from one point to the next, the message server can use this fact to determine which messages must be included in the current state definition. In order to gather the system state, the message server sends a message to each of the control objects requesting its state. For each communication link, all of the messages that arrive after the state request message is sent and before the state itself is received are messages that will not be accounted for in the individual object sub-states and therefore must be included as part of the overall system state. Because all of the control objects process messages in the order that they are received, the controller knows which messages were sent before the request to submit state occurred. As mentioned above, the messages that are received between the time the state request message is sent and the state itself is received must be made part of the system state. However these messages can be sent to their destination as soon as a copy is written to the system state even though the state of the destination object has not yet been received. Since UNIX preserves the order in which messages are received, the destination object will not process this message until after it has gathered its state. There is never a need to hold a message. This method of collecting the state information causes only a minor disruption for each controller. That is, the controller is

only interrupted for the length of time it takes it to gather its state and does not need to pause while all messages reach their destinations. Furthermore, a given controller does not wait for other controllers to finish gathering their state before continuing with its operation.

Once the message server has received the state for every object, the overall state description is generated and sent to the HOOPLS simulation executive function, where it is decomposed into the messages and the state description for each simulated object. To initialize the simulation, the HOOPLS function first creates a message object for each message in the state description and inserts it into the message relay. Then, each object within the simulation contains a function which allows it to initialize itself to the given state. The HOOPLS function calls each simulation object and passes it the portion of the system state that corresponds to that object.

## 5 CONCLUSION

In this paper, we have shown the benefit that employing the same model for control and simulation has upon defining and transferring the system state. The state of a program consists of the data stored in the variables and data structures. If the model that is used to control a system has the same variables and data structures as the simulation model, the definition of the state is simply the state of the program which implements the model. This approach completely avoids having to produce a state description that is compatible with both the simulation and control program. Producing such a description requires one to translate the state description in the control model to that of the simulation model. If the models are not the same, this translation may be impossible.

In order to employ the same controller and simulation model, we implemented a new software tool that has the capability to control the system as well as simulate it using the same model. We briefly discussed our software tool and described the state definition employed by our tool. We also discussed how our HOOPLS-based modeling methodology allows us to address both distributed control and simulation of the same system. Finally, we outlined the procedure we adopted for collecting the current state description.

## REFERENCES

- W. J. Davis, "A concurrent computing algorithm for real-time decision making," *ORSA Computer Science and Operation Research: New Developments in their Interfaces Conference*, Jan. 8-10, 1992, pp. 247-266.
- W. J. Davis, J. Macro and D. Setterdahl, "An integrated methodology for the modeling, scheduling and control of flexible automation," 1997, To appear in the *Journal on Robotics and Intelligent Control*.

- W. J. Davis, D. Setterdahl, J. Macro, V. Izokaitis, and B. Bauman, "Recent advances in the modeling, scheduling and control of flexible automation," *Proceedings of the 1993 Winter Simulation Conference*, pp. 143-155.
- F. G. Gonzalez, and W.J. Davis, "A simulation-based controller for distributed discrete-event systems with application to flexible manufacturing," *Proceedings of the 1997 Winter Simulation Conference*, pp. 845-853.
- F. G. Gonzalez, and W. J. Davis, "A simulation-based controller for a flexible manufacturing cell," *Proceedings of the 1997 International Conference on Systems, Man and Cybernetics*.
- B. A. Peters, J. S. Smith, J. Curry and C. LaJimodiere, "Advanced tutorial - simulation based scheduling and control," *Proceedings of the 1996 Winter Simulation Conference*, Eds. J.M. Charnes, D.J. Morrice, D.T. Brunner and J.J. Swain, pp. 194-198.

## AUTHOR BIOGRAPHIES

**FERNANDO G. GONZALEZ** is Assistant Professor of Electrical and Computer Engineering at the University of Central Florida in Orlando Florida. He received his Ph.D. in Electrical Engineering at the University of Illinois at Urbana-Champaign. Earlier, he received his B.S. in Computer Science and M.S. in Electrical Engineering at Florida International University. His current research addresses the real-time management of discrete-event systems.

**WAYNE J. DAVIS** is Professor of General Engineering at the University of Illinois at Urbana-Champaign. He received his degrees in Engineering Sciences from Purdue University. His current research addresses the intelligent control architectures for large-scale discrete-event systems. In this effort, he is collaborating with the Intelligent Systems Division at the National Institute of Standards and Technology. He is also developing several new simulation languages in order to support this development.