Proceedings of the 2002 Winter Simulation Conference
E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.

WHAT'S VIRTUALLY POSSIBLE?

Wayne J. Davis

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

ABSTRACT

This paper continues a sequence of papers discussing futuristic simulation needs and capabilities. These papers focus upon complex systems that evolve by the concurrent execution of processing tasks under the guidance of sophisticated control structures. This paper first provides a detailed state description for such systems from both the perspective of the entities that are being processed in the system and the controllers that manage the task execution. The interrelationship between these two perspectives is next explored. The paper demonstrates the entity-based perspective mainly focuses upon the events that have or should occur in the physical world. The controllers manage when these events will occur by characterizing the feasible alternatives that exist for executing their assigned tasks within a virtual world of future responses.

1 INTRODUCTION

This paper is the third in a recent series of papers addressing futuristic simulation requirements and capabilities. The first paper, Davis (2001) listed several possibilities that were well beyond the capabilities of the present simulations technologies. The second paper, Davis (2002), presented a detailed analytical representation for the state transition of a large-scale discrete-event system that is coordinated by a distributed command and control structure. This paper initially simplifies the state descriptions presented in Davis (2002) and then focuses upon two distinct perspectives describing the state of the system: the entity-based perspective and the controller-based perspective. The original intent for this paper was to employ these system representations to demonstrate how the futuristic simulation capabilities listed in Davis (2001) might be achieved. A principal goal for this paper was to mathematically prove that the massively parallel simulations for complex systems, operating under sophisticated command and control structures, could be achieved without any special synchronizing mechanisms. That is, such simulations would be self-synchronizing.

Achieving this capability represents a major goal for recent development in distributed simulation, including the definition of the High-Level Architecture (HLA). However, after mathematically describing a computational scheme that provided a massively distributed simulation capability, I realized that having this capability would be counterproductive and retracted the existence discussions from the paper. I then elected to contrast the two state representations. Although the two state perspectives share several events, they are not interchangeable. This paper now demonstrates that the entity-based perspective records behavior that has occurred in the real world and constrains the future events that must occur in order for the system to achieve its assigned tasks or goals. On the other hand, the controller-based perspective describes the feasible alternatives that exist for achieving the assigned goals in a virtual world of future responses. Distributed planning among the controllers continuously updates characterization of the future responses while basic process controllers execute the selected alternative in real-time. In this manner, the current time becomes the portal through a differential element of the system's future state trajectory is realized as an actual response. The shared variables between the two state perspectives pertain to the essential constituency relations that enable this temporal portal to exist.

The paper then further argues that the two state representations are actually dual representations for the physical behavior of the system. The entity-based representation is the primal response describing what occurs in the real world. The controller-based representation describes the dual response within a virtual world of future trajectories or possibilities. When one employs a single monolithic model (which itself may be a composition of several submodels) for simulating entire system, one can only address a single response, usually the primal response. The other or dual response is essentially ignored. Such an omission prevents one from assessing the consequences that planning upon the performance of the systems. This, in turn, implies that command and control issues cannot be correlated to the system response.

Because my interest includes the distributed intelligent control of complex systems, I cannot accept such draconian limitations upon my future research. To that end, I have eliminated the simulation (distributed or otherwise) of a monolithic model for a complex system from my list of desired futuristic simulation capabilities. If I remove this capability from the set of futuristic capabilities described in Davis (2001), then I anticipate that the desirability of the other listed capabilities must be reassessed. This will be the focus of a future paper.

2 BASIC SYSTEM ELEMENTS

Define the set of items $i \in \mathbf{I}$ that a system can process and the set of jobs $j \in \mathbf{J}$, each requesting an item type $i(j) \in \mathbf{I}$. Each job of type $i = i(j) \in \mathbf{I}$ is processed with a Task Sequence TS^i specified as

$$TS^i = \left\{ \left(T_k^i, X\left(\bullet \langle T_k^i \rangle \right) \right) \mid k = 1, \ldots, K^i \right\}$$

where

T_k^i is the k^{th} task in processing an item of type $i \in \mathbf{I}$,
$X\left(\bullet \langle T_k^i \rangle \right)$ is the required state of the item before T_k^i can be initiated, and
$X\left(\langle T_k^i \rangle \bullet \right)$ is the desired state after T_k^i is completed.

Define the set of processes $p \in \mathbf{P}$ that can execute tasks upon items and, in particular, let $\mathbf{P}_k^i \subset \mathbf{P}$ represent the subset of processes that can execute T_k^i. Given these process definitions, we can refine our definition for the Task Sequence TS^i where

$$TS^i = \left\{ \left(T_k^i, X\left(\bullet \langle T_k^i \rangle \mid p \right) \right) \mid p \in \mathbf{P}_k^i \ \& \ k = 1, \ldots, K^i \right\}$$

where

T_k^i is the k^{th} task in processing of an item of type $i \in \mathbf{I}$.
$X\left(\bullet \langle T_k^i \rangle \mid p \right)$ is the required state of an item of type $i \in \mathbf{I}$ before T_k^i can be initiated at process $p \in \mathbf{P}_k^i$
$X\left(\langle T_k^i \rangle \bullet \mid p \right)$ is the desired state after T_k^i is completed at $p \in \mathbf{P}_k^i$.

This refined definition allows one to specialize the required initial state, given that a particular process p has been selected to execute T_k^i.

Davis (2002) further decomposes the set of defined tasks into a serial set of instructions that are to be executed at the selected process in order to implement a given task. Usually these instructions would be defined in the dedicated machine language for the employed process. Although this paper will not address this extra level of detail, the discussed conclusions still apply when this additional detail is considered.

Control is obviously critical to the system's state evolution. Assume that the dsitributed control structure for the system consists of the controllers $c \in \mathbf{C}$. For each controller $c \in \mathbf{C}$, define $\mathbf{C}^+(c) \subset \mathbf{C}$ to be its set of Assignors (those controllers that can assign tasks to the given controller) and $\mathbf{C}^-(c) \subset \mathbf{C}$ to be its set of Acceptors (those controllers that can accept tasks from the given controller). Next, define $\mathbf{C}^0 \subset \mathbf{C}$ to be the master controllers such that $c \in \mathbf{C}^0$ implies $\mathbf{C}^+(c) = \varnothing$ (i.e. a master controller has no assignors within the considered system). At the other extreme, define the process controllers $\mathbf{C}^{\mathbf{P}} \subset \mathbf{C}$ such that $c \in \mathbf{C}^{\mathbf{P}}$ implies $\mathbf{C}^-(c) = p \in \mathbf{P}$. Conversely, we denote the controller for a process $p \in \mathbf{P}$ as $c(p) \in \mathbf{C}^{\mathbf{P}}$.

We now introduce the processor domain for a controller $c \in \mathbf{C}$, denoted as $\mathbf{D}(c)$, as the set of processes that can be reached from controller c through one of its acceptors contained within $\mathbf{C}^-(c)$. We can characterize $\mathbf{D}(c)$ with the following assertions:

$$\mathbf{D}(c(p)) = p \text{ for } p \in \mathbf{P}$$
$$\mathbf{D}(\mathbf{C}^0) = \bigcup_{c \in \mathbf{C}^0} \mathbf{D}(c) = \mathbf{P}$$
$$\mathbf{D}(c) \subseteq \bigcup_{c' \in \mathbf{C}^-(c)} \mathbf{D}(c')$$

An individual controller can employ the control domains in order to determine where to send the job for the execution of its next task.

3 BASIC STATE TRANSITION MECHANISMS

The system evolves using two mechanisms. The primary mechanism is the execution of processing tasks upon items, issued as jobs. A supporting transition mechanism arises as the coordinated subsystems configure themselves into a required initial state for executing the next processing task. Clearly, both mechanisms relate to a particular job. Therefore, when no jobs reside within the system, the system typically returns to a rest state where no state transitions occur.

The physical execution of any task necessarily involves a physical process that operates in real time. Often the employed process's real-time behavior during the execution of task can be described with differential equations. By inte-

grating such differential equations, the composite state of the process and job can be modeled as it transfers from a specified initial state toward a desired final state. The interval over which the task execution occurs is further punctuated by task initiation and completion events. Such discrete events represent interface points between the continuous state response associated with the real-time execution of a physical task and discrete-event nature associated with the resource allocation for and scheduling of each task.

We will now precisely define the events associated with a job's state evolution. We previously defined that the task sequence

$$TS^{i(j)} = \left\{ \left(T_k^{i(j)}, X\left(\bullet \langle T_k^{i(j)} \rangle \mid p \right) \right) \mid p \in \mathrm{P}_k^{i(j)} \ \& \ k = 1,...,K^{i(j)} \right\}$$

for the given item type $i(j) \in \mathrm{I}$ associated with job *j*. As the controller of the selected process p_k^j that manages the execution of $T_k^{i(j)}$, $c\left(p_k^j\right)$ encounters the following events:

- $A_k^j\left(c\left(p_k^j\right)\right)$ is the Accept event where controller $c\left(p_k^j\right)$ assumes the responsibility for executing $T_k^{i(j)}$,
- $S_k^j\left(c\left(p_k^j\right)\right)$ is the Start event for executing the k^{th} task upon the j^{th} job at the process p_k^j, and
- $F_k^j\left(c\left(p_k^j\right)\right)$ is the subsequent Finish event for completing the k^{th} task upon the j^{th} job at the process p_k^j.
- $R_k^j\left(c\left(p_k^j\right)\right)$ is the Return event where physical ownership of the j^{th} job is transferred from controller $c\left(p_k^j\right)$ to its Assignor $c^+\left(c\left(p_k^j\right)\right)$.

The Accept event $A_k^j\left(c\left(p_k^j\right)\right)$ and the Return event $R_k^j\left(c\left(p_k^j\right)\right)$ require a coordinated interaction between the controller $c\left(p_k^j\right)$ and its Assignor $c^+\left(c\left(p_k^j\right)\right)$. The Start event $S_k^j\left(c\left(p_k^j\right)\right)$ and the Finish event $F_k^j\left(c_k^j\right)$ require an interaction between the controller $c\left(p_k^j\right)$ and its Acceptor, the process p_k^j. However, because the process p_k^j is entirely subordinate to its controller $c\left(p_k^j\right)$, controller $c\left(p_k^j\right)$ retains direct control over the processing actions that occur at p_k^j.

After the controller $c\left(p_k^j\right)$ assumes control of the j^{th} job at the Accept event $A_k^j\left(c\left(p_k^j\right)\right)$, it directs its supporting processes to reconfigure the combined state of itself with the associated j^{th} job into the state $X\left(\bullet\langle T_k^{i(j)}\rangle \mid c\left(p_k^j\right)\right)$, which represents its total ownership of the j^{th} job prior to executing task $T_k^{i(j)}$. This event also initiates a wait state that occurs during the interval between the Accept event $A_k^j\left(c\left(p_k^j\right)\right)$ and the start event $S_k^j\left(c\left(p_k^j\right)\right)$. At the start event $S_k^j\left(c\left(p_k^j\right)\right)$, controller $c\left(p_k^j\right)$ further directs the transformation of the composite state into required initial state $X\left(\bullet\langle T_k^{i(j)}\rangle \mid p_k^j\right)$. After that required initial state is realized, the physical execution of the task $T_k^{i(j)}$ may commence. When task $T_k^{i(j)}$ is completed, the finish state $X\left(\langle T_k^{i(j)}\rangle \bullet \mid p_k^j\right)$ is achieved. Controller $c\left(p_k^j\right)$ then directs its supporting tasks to regain immediate control of the j^{th} job, designated by the state $X\left(\langle T_k^{i(j)}\rangle \bullet \mid c\left(p_k^j\right)\right)$. When this latter state is achieved, the Finish event $F_k^j\left(c\left(p_k^j\right)\right)$ occurs. Controller $c\left(p_k^j\right)$ then notifies its Assignor $c^+\left(c\left(p_k^j\right)\right)$ that the assigned task $T_k^{i(j)}$ has been completed. The controller $c\left(p_k^j\right)$ and its Assignor $c^+\left(c\left(p_k^j\right)\right)$ must coordinate their supporting processes in order to reconfigure the combined state of controllers with the associated j^{th} job into required state $X\left(\langle T_k^{i(j)}\rangle \bullet \mid c\left(p_k^j\right) \rightarrow c^+\left(c\left(p_k^j\right)\right)\right)$ so that the job can be returned to the Assignor $c^+\left(c\left(p_k^j\right)\right)$. The physical transfer of the job occurs at the Return event $R_k^j\left(c\left(p_k^j\right)\right)$.

4 THE STATE EVOLUTION OF A JOB

My intent is to describe the system's behavior using recursive relationships whenever it is possible to do so. The controller $c\left(p_k^j\right)$ assumes the physical control of the j^{th} job at the Accept event $A_k^j\left(c\left(p_k^j\right)\right)$ and returns that both job and control at the Return event $R_k^j\left(c\left(p_k^j\right)\right)$. After that Return event occurs, the Assignor $c^+\left(c\left(p_k^j\right)\right)$ may take total ownership of the job causing the composite state of Assignor and the j^{th} job to become $X\left(\langle T_k^{i(j)}\rangle \bullet \mid c^+\left(c\left(p_k^j\right)\right)\right)$, and the Finish event $F_k^j\left(c^+\left(c\left(p_k^j\right)\right)\right)$ to occur. There is clearly a similarity between the response at the controller

$c\left(p_k^j\right)$ prior to the Finish event $F_k^j\left(c\left(p_k^j\right)\right)$ and the response at its Assignor $c^+\left(c\left(p_k^j\right)\right)$ prior to the Finish Event $F_k^j\left(c^+\left(c\left(p_k^j\right)\right)\right)$.

A similar recursive behavior also occurs with respect to the Start events. At the Start event $S_k^j\left(c^+\left(c\left(p_k^j\right)\right)\right)$, controller $c\left(p_k^j\right)$ and its Assignor $c^+\left(c\left(p_k^j\right)\right)$ must coordinate their efforts to change the state of the j^{th} job from $X\left(\langle T_k^{i(j)}\rangle \bullet \mid c^+\left(c\left(p_k^j\right)\right)\right)$ to $X\left(\bullet\langle T_k^{i(j)}\rangle \mid c^+\left(c\left(p_k^j\right)\right)\rightarrow c\left(p_k^j\right)\right)$ so the job can be transferred. The Assignor $c^+\left(c\left(p_k^j\right)\right)$ then instructs the controller $c\left(p_k^j\right)$ to accept the job. This causes controller $c\left(p_k^j\right)$ to change the state of the job from $X\left(\bullet\langle T_k^{i(j)}\rangle \mid c^+\left(c\left(p_k^j\right)\right)\rightarrow c\left(p_k^j\right)\right)$ to $X\left(\bullet\langle T_k^{i(j)}\rangle \mid c\left(p_k^j\right)\right)$, and Accept event $A_k^j\left(c\left(p_k^j\right)\right)$ to occur. As discussed above, this behavior is repeated at the Start event $S_k^j\left(c\left(p_k^j\right)\right)$ when the controller $c\left(p_k^j\right)$ installs the job upon the process p_k^j.

Let us now consider how this recursive behavior might be generalized. Assume that the Assignor $c^+\left(c\left(p_k^j\right)\right)$ is assigned two subsequent tasks $T_k^{i(j)}$ and $T_{k+1}^{i(j)}$ that will be executed at the distinct processes p_k^j and p_{k+1}^j. Further assume that the associated process controllers $c\left(p_k^j\right)$ and $c\left(p_{k+1}^j\right)$ have a common Assignor, which we denote as

$$c^+\left(c\left(p_k^j\right)\right)= c^+\left(c\left(p_{k+1}^j\right)\right)=c_{k\rightarrow k+1}^+.$$

In this case, the Assignor $c_{k\rightarrow k+1}^+$ encounters the Accept Event $A_{k\rightarrow k+1}^j\left(c_{k\rightarrow k+1}^+\right)$. Because task $T_k^{i(j)}$ will be executed at process p_k^j, the subsequent Start event $S_{k\rightarrow k+1}^j\left(c_{k\rightarrow k+1}^+\right)$ will be followed by the Accept event $A_k^j\left(c\left(p_k^j\right)\right)$. The subsequent processing of the task $T_k^{i(j)}$ at p_k^j will eventually generate the complete Event Sequence

$$ES_k^j\left(c\left(p_k^j\right)\right)= \left\{A_k^j\left(c\left(p_k^j\right)\right), S_k^j(\bullet), F_k^j(\bullet), R_k^j(\bullet)\right\}$$

at the controller $c\left(p_k^j\right)$. Normally, the Assignor $c_{k\rightarrow k+1}^+$ would seek to regain complete control of the job after the Return event $R_k^j\left(c\left(p_k^j\right)\right)$. However, the Assignor $c_{k\rightarrow k+1}^+$ is also responsible for executing $T_{k+1}^{i(j)}$. Therefore, the Assignor interacts with controller $c\left(p_{k+1}^j\right)$ to schedule the Accept event $A_{k+1}^j\left(c\left(p_{k+1}^j\right)\right)$ to occur, which in turn, produces the Event Sequence

$$ES_{k+1}^j\left(c\left(p_{k+1}^j\right)\right)= \left\{A_{k+1}^j\left(c\left(p_{k+1}^j\right)\right) S_{k+1}^j(\bullet), F_{k+1}^j(\bullet), R_{k+1}^j(\bullet)\right\}.$$

After the task $T_{k+1}^{i(j)}$ is completed, the Finish and the Return events, $F_{k\rightarrow k+1}^j\left(c_{k\rightarrow k+1}^+\right)$ and $R_{k\rightarrow k+1}^j\left(c_{k\rightarrow k+1}^+\right)$, occur at the Assignor. The composite sequence of events would be

$$ES_{k\rightarrow k+1}^j\left(c_{k\rightarrow k+1}^+\right)= \left\{A_{k\rightarrow k+1}^j\left(c_{k\rightarrow k+1}^+\right) S_{k\rightarrow k+1}^j(\bullet) \left\{ES_k^j\left(c\left(p_k^j\right)\right)\right\} \left\{ES_{k+1}^j\left(c\left(p_{k+1}^j\right)\right)\right\} F_{k\rightarrow k+1}^j(\bullet), R_{k\rightarrow k+1}^j(\bullet)\right\}$$

By induction assume that tasks $T_k^{i(j)}$ through $T_{k'}^{i(j)}$ will be executed at processes $p_k^{i(j)}$ through $p_{k'}^{i(j)}$, whose controllers share a common Assignor $c_{k\rightarrow k'}^+$. In this case, the composite event sequence

$$ES_{k\rightarrow k'}^j\left(c_{k\rightarrow k'}^+\right)= \left\{A_{k\rightarrow k'}^j\left(c_{k\rightarrow k+1}^+\right) S_{k\rightarrow k'}^j(\bullet) \left\{ES_k^j\left(c\left(p_k^j\right)\right)\right\} \cdots \left\{ES_{k'}^j\left(c\left(p_{k'}^j\right)\right)\right\} F_{k\rightarrow k'}^j(\bullet), R_{k\rightarrow k'}^j(\bullet)\right\}$$

would be generated.

Now let us generalize the evolution of these event sequences further. Assume that controller $c_{k\rightarrow k"}$ has been assigned tasks $T_k^{i(j)}$ through $T_{k"}^{i(j)}$. Assume that the domain of one of its Acceptor $c_{k\rightarrow k'}^-$ includes processes $p_k^{i(j)}$ through $p_{k'}^{i(j)}$ while the domain of another Acceptor $c_{k'+1\rightarrow k"}^-$ contains processes $p_{k'+1}^{i(j)}$ through $p_{k"}^{i(j)}$. Then the following event sequence will be generated

$$ES_{k\rightarrow k"}^j\left(c_{k\rightarrow k"}\right)= \left\{A_{k\rightarrow k"}^j\left(c_{k\rightarrow k"}\right), S_{k\rightarrow k"}^j(\bullet) \left\{ES_{k\rightarrow k'}^j\left(c_{k\rightarrow k'}^-\right)\right\} \cdots \left\{ES_{k'+1\rightarrow k"}^j\left(c_{k'+1\rightarrow k"}^-\right)\right\} F_{k\rightarrow k"}^j(\bullet), R_{k\rightarrow k"}^j(\bullet)\right\}$$

When the master controller c^0 is assigned the j^{th} job requesting type $i(j)$, the master event sequence

$$ES^j\left(c^0\right)= \left\{A_{1\rightarrow K^{i(j)}}^j\left(c^0\right) S_{1\rightarrow K^{i(j)}}^j(\bullet), F_{1\rightarrow K^{i(j)}}^j(\bullet), R_{1\rightarrow K^{i(j)}}^j(\bullet)\right\}$$

is immediately initiated. By recursively, applying the generalized event sequence generator, the master sequence can be fully defined to contain every event that will occur during the completion of the j^{th} job, which we term the job history for j^{th} job or JH^j. If we assume that task $T_k^{i(j)}$ is executed on a process $p_k^{i(j)}$ that is distinct from the process used to execute the preceding or following tasks, we can make the following assertions:

- **Assertion 1**: The beginning and ending events are given by the master event sequence $ES^j(c^0)$, implying

$$JH^j = \left\{A^j_{1\to K^{i(j)}}(c^0)\; S^j_{1\to K^{i(j)}}(\bullet)\{\cdots\}\; F^j_{1\to K^{i(j)}}(\bullet), R^j_{1\to K^{i(j)}}(\bullet)\right\}$$

- **Assertion 2**: The initial sequence of events can be described as

$$JH^j = \left\{A^j_{1\to K^{i(j)}}(c^0)\; S^j_{1\to K^{i(j)}}(\bullet) \quad \cdots \quad \left\{ES^j_1\left(c\left(p^j_1\right)\right)\right\}\cdots \quad \right\}$$

- **Assertion 3**: The ending sequence of events can be described as

$$JH^j = \left\{ \cdots \left\{ES^j_{K^{i(j)}}\left(c\left(p^j_{K^{i(j)}}\right)\right)\right\}\cdots F^j_{1\to K^{i(j)}}(c^0), R^j_{1\to K^{i(j)}}(c^0)\right\}$$

- **Assertion 4**: The job history should contain the detailed event sequence at each process or

$$\left[ES^j_k\left(c\left(p^j_{k1}\right)\right) \mid k = 1,\cdots, K^{i(j)}\right]$$

- **Assertion 5**: The events appearing in the job history will be in chronological order.
- **Assertion 6**: Because each event reflects a change of state and state changes can only be accomplished by performing primary or supporting tasks with real-world processes, no two events can occur at the same time. Therefore, the events in a job history must have strictly monotonically increasing event times.

5 STATE EVOLUTION OF A CONTROLLER

This section describes the state from the controller's perspective. The controller's perspective is obviously related to a job's perspective because the state of the controller necessarily depends upon the state of the jobs that it currently manages. The question then arises as to how the management of a particular job by a given controller depends upon the state of that job.

In the subsequent development, we will often refer to a set of jobs rather than a particular job *j*. From a historical perspective, we first define following job sets:

- $\mathrm{J}^i_k(c \mid (t_1, t_2])$ is the set of all jobs of type *i* that visited controller *c* for the execution of task T^i_k between t_1 and t_2 where t_1 might approach $-\infty$.
- $\mathrm{J}^i_k(\mathrm{C} \mid (t_1, t_2])$ is the set of all jobs of type *i* that visited controller $c \in \mathrm{C}$ for the execution of task T^i_k during a specified time interval as described above.
- $\mathrm{J}(\mathrm{C}^0 \mid (t_1, t_2])$ is the set of all jobs of type *i* that have visited any master controller, and hence the system, during a specified time interval as described above. Observe in this case, it does not make sense to limit consideration to a single task T^i_k because we can assume that every job that enters the system will have all tasks $\{T^i_1,\cdots,T^i_{K(i)}\}$ executed upon it.

We now define the following job sets for a particular controller *c* at a particular time, which is commonly the present time:

- $\mathrm{J}^A(c \mid t)$ is the set of all jobs that that are actively managed by controller *c* at time *t* where

$$j \in \mathrm{J}^A(c \mid t) \Rightarrow A^j_k(c) \le t \;\&\; F^j_k(c) > t$$

- This definition assumes that the controller *c* relinquishes active control when its Finish event occurs.
- $\mathrm{J}^S(c \mid t)$ is the subset of in-process jobs that upon which controller *c* has started the next task T^j_k at time *t* or

$$j \in \mathrm{J}^S(c \mid t) \Rightarrow S^j_k(c) \le t \;\&\; F^j_k(c) > t$$

and

$$\mathrm{J}^S(c \mid t) \subset \mathrm{J}^A(c \mid t)$$

- $\mathrm{J}^{A_k(c^-_k)}(c \mid t)$ is the set of in-process jobs that have been reassigned to an acceptor $c^-_k \in \mathrm{C}^-(c)$ for the execution of task T^j_k

$$j \in \mathrm{J}^{A_k(c^-_k)}(c^-_k \mid t) \Rightarrow A^j_k(c^-_k) \le t \;\&\; R^j_k(c) > t$$

- $\mathbf{J}^{F_k(c_k^-)}(c \mid t)$ is the set of in-process jobs upon which the execution of task T_k^j has been completed at the acceptor $c_k^- \in \mathbf{C}^-(c)$

$$j \in \mathbf{J}^{F_k(c_k^-)}(c_k^- \mid t) \Rightarrow F_k^j(c_k^-) \le t \;\&\; F_k^j(c_k^-) > t$$

- $\mathbf{J}^F(c \mid t)$ is a subset of $\mathbf{J}^A(c \mid t)$ where the currently assigned task(s) has been completed. In this case, controller c has returned control of a job to its Assignor c^+, but the job is not yet physically returned to the Assignor's immediate control domain.

Based upon these definitions, controller *c* has direct control of a given job during one of these mutually exclusive subsets:

- The set of waiting jobs at the controller *c* with tasks $T_k^j \to T_{k'}^j$ assigned for execution (i.e. $A_{k\to k'}^j(c) \le t$) where $1 \le k \le k' \le K_{i(j)}$ (i.e. event $S_{k\to k'}^j(c)$ has not occurred). This set is represented as

$$\mathbf{J}^W(c \mid t) = \mathbf{J}^A(c \mid t) - \mathbf{J}^S(c \mid t)$$

- The set of jobs being transferred from controller *c* to the acceptor $c_k^- \in \mathbf{C}^-(c)$ for executing the first assigned Task T_k^j (i.e. $S_{k\to k'}^j(c) \le t$ and $A_k^j(c_k^-) > t$). This set is represented as

$$\mathbf{J}^{c\to c_k^-}(c \mid t) = \mathbf{J}^{S_{k\to k'}(c)}(c \mid t) - \mathbf{J}^{A_k(c_k^-)}(c_k^- \mid t)$$

- The set of jobs being transferred from acceptor $c_k^- \in \mathbf{C}^-(c)$ to acceptor $c_{k+1}^- \in \mathbf{C}^-(c)$ implying that Task T_k^j has been completed (i.e. $F_k^j(c_k^-) \le t$), but j^{th} job has not been accepted by the Acceptor $c_{k+1}^- \in \mathbf{C}^-(c)$ for executing task T_{k+1}^j. This set is represented as

$$\mathbf{J}^{c_k^-\to c_{k+1}^-}(c \mid t) = \mathbf{J}^{F_k(c_k^-)}(c_k^- \mid t) - \mathbf{J}^{A_{k+1}(c_{k+1}^-)}(c_{k+1}^- \mid t)$$

- The set of jobs where the last assigned task $T_{k'}^j$ has been completed (i.e. $F_{k'}^j(c_{k'}^-) \le t$), but the job has not yet returned to controller *c* (i.e. $F_{k\to k'}^j(c) \ge t$). This set is represented as

$$\mathbf{J}^{c_k^-\to c}(c \mid t) = \mathbf{J}^{S_{k\to k'}(c)}(c \mid t) - \mathbf{J}^{F_{k'}(c_{k'}^-)}(c_{k'}^- \mid t).$$

From these definitions, we can conclude that the controller *c* only relinquishes direct control of the j^{th} job to its Acceptor $c_k^- \in \mathbf{C}^-(c)$ during the interval between $A_k^j(c_k^-)$ and $F_k^j(c_k^-)$. With respect to its Assignor, controller *c* has direct control during the interval between $A_{k'\to k''}^j(c)$ and $F_{k'\to k''}^j(c)$. Clearly, a recursive relationship again exists between the controller *c* and its Acceptor and between its Assignor and controller *c*. Also observe that the transfer of ownership that occurs between a controller and it acceptor is effectively hidden from its assignor. This feature allows different controllers to view the world at different levels of resolution.

Earlier, we defined the set of jobs that visited the system on the interval $(t_1, t_2]$ as $\mathbf{J}(\mathbf{C}^0 \mid (t_1, t_2])$. Each of these visiting jobs must belong to a particular class $i \in \mathbf{I}$. Let $\mathbf{J}^i(\mathbf{C}^0 \mid (t_1, t_2])$ be the set of visiting jobs of type *i*. For each of these jobs, a complete processing history JH^j would have been generated that records the time that each event occurs. In order to simplify our development, let us assume that the system is time invariant such that

$$ecdf(t_E \mid t_{\bullet(E)}) = ecdf(t_E - t_{\bullet(E)}).$$

This equation states that $ecdf(t_E \mid t_{\bullet(E)})$, (the empirical cumulative distribution function for the time of event *E* or t_E given the time of a prior event $\bullet(E)$ or $t_{\bullet(E)}$), is equal to the empirical cumulative distribution function for the difference between the two event times. This assumption is probably not true for most complex systems. However, the time-variant nature introduces additional complexity that we prefer avoid whenever possible in this paper.

If we employ the time invariant assumption, then we do not need to be particular in specifying the interval $(t_1, t_2]$ to be considered. Rather, we can consider any job of a particular type *i* that has ever visited the system or $\mathbf{J}^i(\mathbf{C}^0 \mid (-\infty, t])$.

Assume that the j^{th} job arrives at controller *c* for the execution of task T_k^j. Assume further that the j^{th} job belongs to type $i(j) \in \mathbf{J}$. Given these assumptions and using completed processing histories contained in

$\mathbf{J}^{i(j)}\left(\mathbf{C}^0 \mid (-\infty, t]\right)$ for which events $A_k^j(c)$, $S_k^j(c)$ and $F_k^j(c)$ exist, one computes

$$ecdf\left(S_k^j(c) \mid A_k^j(c)\right) = ecdf\left(S_k^j(c) - A_k^j(c)\right)$$
$$ecdf\left(F_k^j(c) \mid A_k^j(c)\right) = ecdf\left(F_k^j(c) - A_k^j(c)\right)$$

and

$$ecdf\left(F_k^j(c) \mid S_k^j(c)\right) = ecdf\left(F_k^j(c) - S_k^j(c)\right).$$

The first *ecdf* represents the waiting time at controller *c* before processing will be initiated at the start event $S_k^j(c)$. The second *ecdf* represents the interval between when controller *c* accepts the task T_k^j and when it notifies its Assignor that the task has been executed. The third computation estimates the completion time after the processing of task T_k^j has been initiated.

Controller *c* employs the latter two *ecdf*s as the primary feedback to its Assignor, which we will denote as c^+. Given the estimated *ecdf* for the Finish event $F_k^j(c)$, the Assignor c^+ can develop an *ecdf* for its Finish event $F_k^j\left(c^+\right)$. As discussed above, the Assignor c^+ coordinates the return of the j^{th} job from controller *c* using the supporting processes that either controller manages. Moreover, the interval between these two finish events typically does not depend upon the type of job $i(j)$. Rather, the time interval depends primarily upon the available supporting processes that allow the assignor c^+ to interact with controller *c*. Thus, the assignor c^+ can typically compute

$$ecdf\left(F\left(c^+\right) \mid F(c)\right) = ecdf\left(F\left(c^+\right) - F(c)\right)$$

which is independent of job type. The *ecdf* for the $F_k^j\left(c^+\right)$ can now be computed as the sum of two random variables whose *ecdf*'s have been described.

Figures 1 through 3 depict several situations of increasing complexity where statistical projections are to be made. In Figure 1(a), controller *c* initially has accepted the execution of task T_k^j, (implying $j \in \mathbf{J}^W(c \mid t)$), and provides the statistical estimates, (indicated by the shaded arrows), for when the assigned task will be completed by the selected acceptor c_k^-, $ecdf\left(F_k^j\left(c_k^-\right) \mid A_k^j(c)\right)$, and by itself, $ecdf\left(F_k^j(c) \mid A_k^j(c)\right)$. After controller *c* initiates task T_k^j at event $S_k^j(c)$, (implying $j \in \mathbf{J}^{c \to c_k^-}(c \mid t)$), it estimates $ecdf\left(F_k^j\left(c_k^-\right) \mid S_k^j(c)\right)$ and $ecdf\left(F_k^j(c) \mid S_k^j(c)\right)$. In Figure 1(b), the Acceptor c_k^- has accepted the task T_k^j at $A_k^j\left(c_k^-\right)$ (implying $j \in \mathbf{J}^{A_k\left(c_k^-\right)}(c \mid t)$). The Acceptor c_k^- then estimates when it will complete T_k^j using $ecdf\left(F_k^j\left(c_k^-\right) \mid A_k^j\left(c_k^-\right)\right)$. Upon receiving this estimate from its Acceptor c_k^-, controller *c* generates statistical estimates when it will subsequently notify its Assignor that it has completed the task $ecdf\left(F_k^j(c) \mid F_k^j\left(c_k^-\right)\right)$.

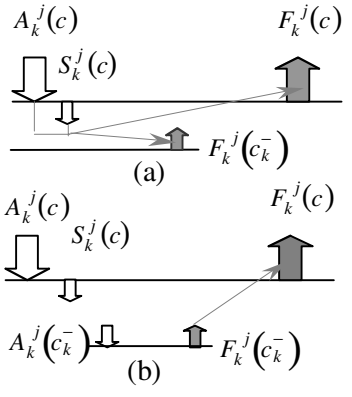

Figure 1: Event Estimation for Single Task

Figure 2 illustrates a slightly more complex situation. In Figure 2(a), controller accepts the job with two sequential tasks, T_k^j and T_{k+1}^j at Accept event $A_{k\to k+1}^j(c)$ and estimates when it will have completed both tasks using the $ecdf\left(F_{k\to k+1}^j(c) \mid A_{k\to k+1}^j(c)\right)$ and when each acceptor will complete it assigned task using $ecdf\left(F_k^j\left(c_k^-\right) \mid A_{k\to k+1}^j(c)\right)$ and $ecdf\left(F_{k+1}^j\left(c_{k+1}^-\right) \mid A_{k\to k+1}^j(c)\right)$, respectively. After the Start event $S_{k\to k+1}^j(c)$ occurs, controller *c* updates the corresponding *ecdf*'s conditioned upon the event $S_{k\to k+1}^j(c)$. In Figure 2(b), the Accept event $A_k^j\left(c_k^-\right)$ has occurred and controller c_k^- computes the $ecdf\left(F_k^j\left(c_k^-\right) \mid A_k^j\left(c_k^-\right)\right)$. Controller *c* then constructs ecdf's for the future events $A_{k+1}^j\left(c_{k+1}^-\right)$, $F_{k+1}^j\left(c_{k+1}^-\right)$ and $F_{k\to k+1}^j(c)$. The *ecdf* for $F_{k\to k+1}^j(c)$ is provided to its Assignor as feedback information on when it expects tasks T_k^j and T_{k+1}^j will be completed. The *ecdf* for $A_{k+1}^j\left(c_{k+1}^-\right)$ provides feed-forward information to the Acceptor c_{k+1}^- pertaining to when it should

expect the j^{th} job to arrive. The ecdf for $F_{k+1}^{j}(c_{k+1}^{-})$ would likely be used for its internal planning. In Figure 2(c), event $A_{k+1}^{j}(c_{k+1}^{-})$ has occurred and Acceptor c_{k+1}^{-} projects the ecdf for $F_{k+1}^{j}(c_{k+1}^{-})$. Given this *ecdf*, controller *c* then projects the *ecdf* for $F_{k\to k+1}^{j}(c)$.

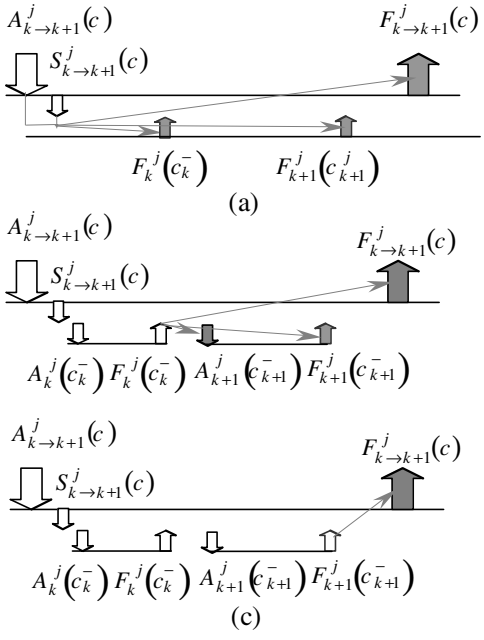

Figure 2: Event Estimation for Two Consecutive Tasks

Figure 3 generalizes the above recursive relationship. In Figure 3(a), Accept event $A_{k\to k'}^{j}(c)$ has occurred and controller *c* predicts *ecdf*'s for the Acceptors' Finish events $F_k^j(c_k^-)$ through $F_{k'}^j(c_{k'}^-)$ and its Finish event $F_{k\to k'}^{j}(c)$. In Figure 3(b), Accept event $A_k^j(c_k^-)$ has occurred, and Acceptor c_k^- predicts when its Finish event $F_k^j(c_k^-)$ will occur. Using this prediction, controller *c* predicts *ecdf*'s for the Acceptors' Finish events $F_{k+1}^j(c_{k+1}^-)$ through $F_{k'}^j(c_{k'}^-)$ and its Finish event $F_{k\to k'}^{j}(c)$. In Figure 3(c), Accept event $A_{k'}^j(c_{k'}^-)$ has occurred, and Acceptor $c_{k'}^-$ predicts when its Finish event $F_{k'}^j(c_{k'}^-)$ will occur. Using this prediction, controller *c* computes the *ecdf* for its Finish event $F_{k\to k'}^{j}(c)$.

The ability to compute these *ecdf*'s at each controller clearly supports planning for the future response of the system. From the moment the job enters the system via a master controller, predictions can be made on when each task will be completed and when the entire job will be finished.

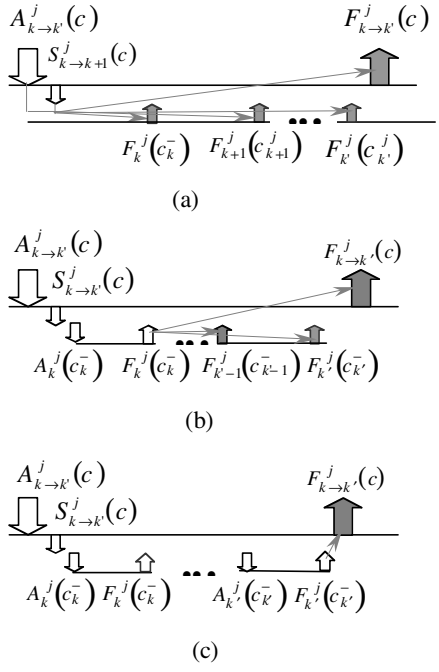

Figure 3: Event Estimation for Series of Tasks

Moreover, as tasks are initiated, these estimates can be updated using feedback information from the chain of controllers that are responsible for executing their particular task assignments. This capability can be further enhanced using on-line simulation procedures (see Davis 1998) that would allow one to project future consequence for alternative task assignment strategies. However, before demonstrating how such capabilities are possible, one should demonstrate how one would coordinate the planning among the controllers. This is again beyond the scope of this paper.

6 CONTRASTING TWO PERSPECTIVES

Section 4 defined the state evolution from the perspective of the job upon which tasks are being executed. In Section 5, the perspective focused upon the controllers that managed the task executions. Section 5 also discussed the interconnection between planning for and the control of task executions. It was demonstrated that a specific controller manages when each event occurs during the task execution process. It also estimates when future events associated with the execution of its assigned tasks and then employs these estimates in planning its future responses. In more complex situations, a given controller might also employ on-line simulation to project its future response as it interacts with its own Assignor(s) and Acceptors. Because the overall planning and execution functions are distributed across an ensemble of interacting controllers, several con-

trollers could be concurrently performing their specialized on-line simulations, each using its own dedicated model.

Obviously, these dedicated models should integrate into a model for the overall system, but it is uncertain what use for the overall model might exist, particularly during the operation of the system. During the last decade, new standards and approaches for the distributed simulation of complex systems have been sought (see Fujimoto 1999). With respect to military simulations, the High-Level Architecture (HLA) now defines the interfacing requirements for insuring that individual models for specialized military activities can be integrated into a comprehensive model for the combined military response. HLA further provides the support for simulating this comprehensive model upon a set of distributed computing processes. Underlying the HLA perspective, there is an assumed need or goal for simulating behavior of the entire system as a monolithic object using distributed computational procedures. The utility for achieving this goal appeared intuitive. However, intuition can be deceiving.

HLA provides minimal consideration of command and control constraints (see Davis and Moeller 1999). Moreover, HLA's procedures for synchronizing the distributed simulation impose significant overhead upon the distributed computation and intrinsically limit the number of computational threads that can be employed. Given such inherent limitations that exist for the HLA, it is improbable that massively distributed simulations with the detailed consideration of both the employed tactics and command and control structure can ever be achieved.

The latter observation is derived from knowing how one can achieve the overall goal. That is, using the state descriptions included in this paper, I have mathematically described how one can address the command and control constraints within massively distributed simulations, which are inherently self-synchronizing. Given that billions of dollars have been spent seeking such a capability, some might consider this accomplishment to be a major achievement. Unfortunately, I believe the simulation of the entire system in a monolithic fashion is useless, if not deleterious. In this regard, I am withholding these mathematical existence proofs for this capability, which incidentally have been checked by other researchers in simulation. My conscience will not allow me to show others how to do something that I believe that they should not do.

Let us return to the two distinct perspectives for viewing state as discussed in Sections 4 and 5. Beginning from the job-based perspective, each job's state delineates what tasks have been performed (by whom and when) and what tasks remain. If you look across the collection of finished and current jobs, one can reconstruct the physical behavior of the system as it reached the current state. That is, the past (real-world) response has been recorded and can be replayed. Now for the active jobs, the remaining tasks delineate what tasks should be executed in the future. Let us view the future as a virtual world of anticipated or planned responses. Ironically, the jobs upon which the tasks will be performed cannot plan that future response.

The system's future/virtual response is planned by the controllers as they interact with each other to establish the ideal assignments and schedule for executing the remaining tasks. Because that planning must always consider the future, the planning horizon for any controller never includes the current time. In order to act in real time, a process must know its action at each moment. The process's real-time response then becomes a part of the job's history of what happened in the real world. In this regard, the current time represents the continuously advancing portal between the real world of what has happened and the virtual world of what might occur.

In order to view the two state perspectives as being interchangeable representations for the same system, the system must be time invariant and the strategies that each controller employs to determine every future action must be predefined. The latter requirement eliminates the potential for any planning or deliberation pertaining to the system's future response. (Incidentally, this is the traditional approach to conducting simulation analyses where the focus is upon the processing of entities while employing predefined strategies or prioritization schemes.) It also reflects HLA's intent to simulate the system as a single monolithic entity. Since addressing the system as a single monolithic entity inherently excludes planning, command and control concerns can never be addressed.

Even though the two state perspectives share events, the two state representations are not interchangeable. Because they are distinct, we can conclude that the current definition for each representation is probably incomplete. With regard to the job, the detailed physical instructions/constraints for executing each remaining tasks are essential. In most time-variant systems, current processing tasks are updated, and entirely new tasks might be defined. Moreover, only the process controller needs to know the detailed instructions to be executed. The Assignors above the process controllers need only know which processes can execute a given task and what initial state should be achieved before the task can be executed. An Assignor's planning usually involves selecting which process will execute a given task upon a given job, scheduling when the task is to be performed and defining the essential supporting task for achieving the specified initial state.

The required time to execute a task also changes in most time-variant systems. In this manner, a present job is intrinsically linked to the historical record for performing similar tasks upon the previous jobs. In general, the statistics for the most recent completions are more meaningful for estimating the time required to complete a remaining task. Thus, after each execution of a given task, the task's

statistics should be updated based upon the most recent observations. The *current* statistical estimates for the time needed to complete a given task and the probability that the task can be successfully executed represent an essential component for specifying each remaining task and, consequently, the state of the job associated with the given remaining task.

Now consider the controller's perspective. Certainly, its current state and future behavior depends upon the jobs under its control and the tasks it has been assigned to execute. In fact, these components of the state represent the collective outcome of the controller's prior interactions with its Assignor(s). Given this information, each controller must then be able to project its response in coordination with its managed subsystem's under any feasible scheme for executing the tasks. The projected future behavior of its Acceptors is dependent upon statistical information imbedded within each assigned job's state, their assigned tasks, and the feedback they receive from their Acceptors.

Given these observations, the state of each controller depends upon the state of the controllers with which it interacts *and* its prior assessment of planned alternatives for executing its remaining tasks. This implies that the controller's state with its associated planning process is entirely dynamic as the state of controllers with which itinteracts changes and the set of considered alternatives continues to grow. Each controller's state is dynamic. Therefore its planning is dynamic and incessant.

A controller's projection of its future performance under a given alternative will likely require the on-line simulation with a specialized model of that controller as it interacts with its Assignor(s) and Acceptors. Moreover, this same situation occurs at each controller other than the most basic process controllers. Therefore, it is essential that each controller concurrently conducts its specialized simulation with its specialized models. Clearly, the specialized model that each controller employs should include the submodel for the controller's behavior that would be included within a monolithic model for the entire system. However, the models of an interaction between any two controllers cannot be independent of each other. Moreover, there are alternative means through which such interactions might occur. During the real-time operation of the system, the consequences of a potential interaction between two controllers might be determined by permitting one controller to ask another controller to access the potential consequences resulting from a particular assignment. It might even be possible for an Acceptor to provide a model of how it would behave to its Assignor(s) along with the feedback information pertaining to its current state. Given these possibilities, it is uncertain how one would model interactions among the controllers within a monolithic model. What is clear, however, is the simulation of the monolithic system operating with on-line planning will require on-line simulations to be embedded within the primary simulation for the entire system. Today, we do not know how to recursively embed simulations within other simulations.

Returning to the two different state representations, both representations address the basic physical task execution process that is central to the systems response. However, both state representations must be further enhanced, and these enhancements distinguish the representations from each other. Consequently, the representations are not alternatives for each other. Actually one can show that they are dual representations, the job's state describing the world of realities and the controller's state describing the world of virtual possibilities. These dual worlds are linked at the current time where a next state along the virtual trajectory of a potential response becomes next state in the observed (real) response. Therefore, it is essential that these dual representations share common variables.

The dynamism of the duality between real and virtual worlds can never be described with a single simulation. Because prior simulation development has focused primarily upon the processing of entities and has devoted little or no attention to planning, this duality has ignored. However, if planning is to be addressed within an adopted command and control structure for managing the system this dualism must be recognized.

7 CONCLUSION

After completing this paper in a sequence of papers pertaining to the future needs in simulations, I am still learning. I expect new needs to evolve and the specifications of previously defined needs to be modified. During recent years, I devoted several man months to proving that massively distributed simulation of command and control systems was possible. Indeed, the original draft of this paper included that proof as the sixth section. Given the effort required to develop this demonstrate, it was difficult for me to accept that it was detrimental to future simulation evolution and should be omitted.

The issue then arises as to whether others should continue to seek such capabilities. That is a question that each researcher must answer for herself. However, the experience that I gained from demonstrating how massively parallel simulations of a monolithic model can be achieved, I have drawn the following conclusions:

- Mathematically, it is impossible to achieve this capability under the HLA standards. HLA cannot address the interactions required to consider command and control constraints. Moreover, the overhead that it incurs in synchronizing concurrent event processing limits the number of computational threads that can be employed.

- The goal of performing distributed simulations upon monolithic models of large systems should be abandoned. Such simulations can assess only the primal behavior of the real world real-world response while ignoring the dual behavior in the virtual planning world.
  - When planning is ignored, one implicitly assumes that command and control structures do not influence the behavior of the system.
  - Moreover, if the consequences of planning cannot be considered, then effective tools for supporting planning cannot be created.

Clearly, much effort and resources have been devoted to the development of HLA and establishing it as the standard for distributed simulation. The original intent for developing HLA was genuine and plausible. Progress produces change and an opportunity to revisit prior decisions. The simulation community has faces two mutually exclusive options. If they continue to support the desire to perform monolithic simulations of large systems, then it will be virtually impossible to support system management functions. On the other hand, if one recognized the intrinsic primal-dual nature of the state evolution for these complex systems, then we can explore what new capabilities become virtually possible in the discovered world of future responses.

REFERENCES

Davis, W. J. 1998. On-line Simulation: The Need and the Evolving Research Requirements. In the *Simulation Handbook*, ed. J. Banks, 465-516. New York: John Wiley and Sons, Inc.

Davis, W. J. 2001. Distributed Simulation and Control: The Foundations. *Proceedings of the 2001 Winter Simulation Conference*, 187-198, San Diego: Society for Modeling and Simulation International.

Davis, W. J. 2002. The State Evolution and Simulation of Distributed Systems. *Proceedings of the 2002 Advanced Simulation Technologies Conference*, 34(3), 114-119, San Diego: Society for Modeling and Simulation International.

Davis, W. J. and G. L. Moeller. 1999. The High-Level Architecture: Is There a Better Way, *Proceedings of the 1999 Winter Simulation Conference*, 1595-1601, San Diego: Society for Modeling and Simulation International.

Fujimoto, Richard. 1999. Parallel and Distributed Simulation Systems. New York: Wiley Interscience.

AUTHOR BIOGRAPHY

Wayne J. Davis is a professor of general engineering at the University of Illinois at Urbana-Champaign. His research addresses the distributed intelligent control architectures for complex systems. In this effort, he has developed several new modeling paradigms and on-line simulation approaches.