Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

HYBRID AGENT-BASED SIMULATION FOR ANALYZING THE NATIONAL AIRSPACE SYSTEM

Seungman Lee
Amy Pritchett
David Goldsman

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205, U.S.A.

ABSTRACT

Hybrid agent-based simulation is required to provide a mechanism for analyzing large-scale complex systems, such as the National Airspace System (NAS). The dynamic behavior of many complex systems is, in general, hybrid in nature and can be best described by a combination of discrete-event and continuous-time models, and their interactions. Correspondingly, hybrid agent-based simulation capable of incorporating different types of models provides an accurate means of evaluating the reliability and performance of complex systems. However, in order to serve as a design and analysis tool, a number of issues must be addressed. This paper outlines issues in the development of hybrid agent-based simulation architectures capable of providing a scaleable mechanism for simulating the NAS. In particular, an object-oriented approach is described. In addition, methods of improving computational efficiency of updating the simulation are described and compared.

1 INTRODUCTION

Analysis of large-scale complex systems, such as the NAS, requires the use of hybrid agent-based simulation. Simulation is an important tool for evaluating the performance of systems, and provides a safe and cost-effective way of examining the impact of potential changes (National Research Council 1998). Typically, large-scale complex systems include different types of entities interacting with each other in significant ways. For example, the NAS can be defined as a collection of mutually dependent entities, such as aircraft, controllers, pilots, ground systems, and communication, navigation and surveillance technologies. Controllers (and pilots) are constantly changing aircraft flight paths in response to the actions of other entities and to changes in the environment. Each of these entities requires a different type of simulation model with varying fidelity based on the behavior of the entity. For instance, aircraft in the NAS may require a high-fidelity continuous-time simulation model to predict the dynamic behavior of the aircraft, while certain stochastic events such as aircraft arrivals or machine failures require discrete-event simulation models.

The individual behavior of these different entities and overall behavior of the NAS, therefore, can be modeled by a combination of both continuous-time and discrete-event models, and their interactions. Although systems of real-world applications are often characterized by a combination of discrete-event and continuous-time models, the systems have been mainly simulated by either entirely discrete-event simulation models capable of capturing and predicting stochastic effects within a system of interest, or entirely continuous-time simulation models capable of predicting the dynamic behavior of physical systems such as aircraft trajectories and mechanical system performance. The NAS, for example, has been simulated before using entirely discrete-event models (Odoni, et al. 1997). However, such simulations have been limited to specific applications or parts of the NAS (Jim and Chang 1998). Correspondingly, hybrid agent-based simulation capable of incorporating different types of models provides a means of more-accurately evaluating the reliability and performance of large, complex systems. This combined simulation methodology was first proposed by Fahrland (1970), and the need for such combined simulations has been addressed previously (Cellier 1979, Dessouky and Roberts 1997).

In order for hybrid agent-based simulation to serve as a valuable design and analysis tool, a number of important issues must be addressed. First, a hybrid agent-based simulation of a large-scale system must be sufficiently flexible and extensible so that different types of agents can be easily added and modifiable to a wide range of scenarios of varying scope and fidelity. The most important function of agent-based simulation is to efficiently integrate different types of models interacting with each other in complex ways at unpredictable times. An agent-based simulation architecture for large-scale complex systems, therefore, should be able to efficiently incorporate different types of simulation models, such as continuous-time and discrete-event models, into a combined simulation model for representing and analyzing the system of interest more accurately and more completely.

Second, the agent-based simulation of large-scale systems must also be computationally efficient in order to provide statistically meaningful data within a reasonable amount of computation time. The different models in hybrid agent-based simulation use different timing methods to advance the simulation and also tend to require substantially different update rates. For example, a continuous-time model of aircraft flight dynamics might require very small time steps and frequent update rates, whereas a discrete-event model for generating aircraft objects into the airspace might require relatively large update intervals. This disparity between the various timing mechanisms of different models must be resolved efficiently in the agent-based simulation.

Despite these differences, hybrid agent-based simulation must also ensure that agents in the simulation are synchronized at any time when they must interact or exchange data between different types of models. In particular, the different types of models have to be properly incorporated and synchronized, even when dramatically different time steps are used for numerical or logical reasons. However, computationally efficient mechanisms have not been definitively established for timing the updates of each individual agent contained in agent-based simulation.

This paper outlines issues in the development of (1) fully integrated agent-based simulation architecture using an object-oriented approach and (2) computationally efficient timing mechanisms for the agent-based simulation. The performance of different timing mechanisms is demonstrated in an experiment comparing three timing mechanisms using the NAS simulation as a test case.

2 AGENT-BASED SIMULATION ARCHITECTURE

A hybrid agent-based simulation model represents a simulation architecture for modeling and simulation of complex systems consisting of a mixture of discrete and continuous components. In the simulation of large-scale complex systems, multiples of these different types of models interact with each other in significant ways. The behavior of the systems can be defined as hybrid dynamics incorporating discrete-event and continuous-time models. The most important feature of the hybrid agent-based simulation model is the efficient integration of different types of models interacting with each other. The different modeling and simulation approaches used for continuous-time and discrete-event models must be integrated in a consistent manner so that entire systems can be simulated with an appropriate speed/accuracy tradeoff.

Several approaches have been suggested to integrate the different types of models within combined simulation. Augmented approaches, such as the discrete-augmented approach and the continuous-augmented approach, have adapted existing simulations of one type to include the other (Saleh, Jou, and Newton 1994). For example, Klingener (1996) described an approach for the incorporation of interacting continuous processes into a discrete-event simulation model. Several modeling languages have also been developed for combined system simulation (Dessouky and Roberts 1997). These approaches usually force the systems into either a strictly discrete or continuous model, and may lead to an inadequate representation of the system under study.

The simulation architecture to appropriately represent the behavior of hybrid systems should be able to (1) accept different types of models with varying fidelity without placing unnecessary restrictions on the type of models, (2) control their timing in a computationally efficient manner, and (3) handle interactions between entities within the system.

Object-oriented (O-O) simulation is becoming the appropriate approach for modeling large-scale, complex, and/or distributed systems (Roberts and Dessouky 1998). The agent-based approach to simulation can benefit from O-O modeling because it provides constructs and concepts that support the definition of agents. An entity in the real world is modeled as an agent, which itself is represented as a class. An agent is an abstract representation of an entity that interacts with and contributes to its environment. Instances of the agent are represented as objects of the class. For example, in the simulation of the NAS, aircraft in the NAS are modeled as agents and represented by a class. The NAS simulation includes hundreds of aircraft as agents of the same type; these are represented as instances, or objects, of the aircraft class. The interactions between agents in agent-based simulation will become clear by using the O-O approach.

Agent-based simulation architecture using an O-O approach, therefore, can efficiently represent the collective dynamic behavior of a large number of agents interacting with each other. Further, this agent-based architecture using an O-O approach allows for fast prototyping and simulation of large, complex systems due to the ease of modeling. With reusable and modular component design as provided by concepts such as encapsulation, inheritance, and polymorphism, an agent-based simulation can be efficient to develop and maintain (Sichman, Conte, and Gilbert 1998). In addition, this approach can be used to control the timing and updating of agents in a computationally efficient manner.

Instead of forcing agents to fit into one of various different types of models, each agent in the agent-based simulation architecture is required to satisfy the minimal requirements of a standard interface. Specifically, each agent must update its state upon command, report the time of its next update, and identify whether its own update requires any other objects to also update. Each agent, therefore, has to know when significant changes have occurred in the actions of its own process. All other dynamics of the components can remain internal to their models; this internalism prevents restricting the types of models allowed in the simulation (Bezdek, Halley, and Hummel 1997, Davis and Bigelow 1998). Based on this argument, the simulation architecture should not place unnecessary restrictions on the types of objects, but instead each object must satisfy the base standard interface requirements.

The Reconfigurable Flight Simulator (RFS) software, designed using principles of O-O analysis, is used as the simulation architecture for an agent-based simulation application (Pritchett and Ippolito 2000, Pritchett, Lee, and Goldsman 2001). This architecture allows for the inclusion of several broad classes of objects. First, an arbitrary number of controller, event, and measurement (CEM) objects can be added; these classes of objects can include human performance models, as well as measurement and discrete-event models. Likewise, an arbitrary number of vehicle objects and input-output objects can be included to provide continuous-time representations of aircraft dynamics and data output / storage / entry, respectively. The environment controller and database (ECAD) provides a shared simulation environment by establishing axis definitions and by allowing for the inclusion of atmospheric and terrain models as needed.

A practical concern in agent-based simulation is that the simulation should be sufficiently computationally efficient that it is a time-effective analysis tool. In agent-based simulation of large-scale systems, overall computational efficiency can be achieved only when each agent is updated as needed for its own internal dynamics, for correct interactions with other agents, and for timely measurements. The simulation architecture, therefore, should be capable of timing the updates of the individual agents in a computationally efficient manner. Specifically, if a timing mechanism commands agents to update only when required, it can dramatically speed up the simulation while maintaining its fidelity. Of course, methods of deciding when an update may be required for correct interactions or timely measurements are generally non-trivial once the simulation at hand contains stochastic elements.

3 TIMING MECHANISMS

Research on timing mechanisms has been focused on making each simulation model computationally efficient. However, timing mechanisms for an agent-based simulation incorporating different types of models have not been developed thus far. As discussed in previous sections, agent-based simulation includes different types of agent models such as discrete-event models and continuous-time models. These different models require different timing methods and considerably different update rates. By developing a timing mechanism to efficiently control the different update rates of different simulation models, it is possible to reduce any unnecessary updates of agents and consequently to improve the computational efficiency of the simulation. More generally, this section focuses on various aspects of timing mechanisms for agent-based simulation of large, complex systems and describes their implementation using an O-O approach.

3.1 Characteristics of Different Timing Methods

There are fundamental distinctive differences between timing methods for discrete-event models and continuous-time models. Historically, there are two principal mechanisms for advancing discrete-event simulations: *next-event time advance* and *fixed-increment time advance* (Law and Kelton 2000). With the fixed-increment timing mechanism, the simulation moves forward in time at fixed intervals regardless of whether anything happens within the time intervals. With the next-event timing method approach, the simulation is advanced from one event time to the next significant event time and these intervals may be variable. Discrete-event models update their state at discrete points in time, when an event occurs to cause a change in the state of the system.

On the other hand, continuous-time models typically attempt to represent internal dynamics, such as aircraft trajectories, that are governed by physical laws, which are expressed as differential equations (Zeigler, Praehofer, and Kim 2000). For most continuous-time models, numerical-analysis techniques that integrate the differential equations numerically are used to compute the evolution of the system states. In continuous-time models, the integration proceeds using a large number of small time steps. The time step can be fixed at a small value to reduce error in the numerical solutions and to capture the simulation's basic properties, or the time step can be variable in size within an error tolerance.

The timing mechanisms can also be defined as *synchronous* or *asynchronous* as typically used in parallel and distributed system simulation (Fujimoto 2000). While synchronous timing methods require all agents in the simulation to update at the same time, asynchronous timing methods allow each agent to update individually and independently. For large-scale or repeated runs of the simulation, synchronous timing methods will be computationally inefficient since the timing method requires all agents to update at every time step, whether they need to or not. The synchronization between agents interacting with one another is also an important factor in determining the accuracy and speed of agent-based simulation.

Based on the characteristics of those different timing methods, several timing mechanisms to synchronize the time of simulation updates can generally be categorized into three schemes: synchronous fixed time interval, synchronous variable time interval, and optimistic time-warp methods.

3.1.1 Synchronous Fixed Time Interval

This timing method requires all agents to update at a predetermined fixed time interval. The advantage here is that synchronization will always be maintained since all different types of agent models are simulated at the same time. By the setting of a sufficiently small time step, this method provides accurate results that can be guaranteed not to miss any measurements or interactions. The obvious drawback is that a small time step results in a large number of update points. Although fixed time interval mechanisms are conceptually very simple, they lead to unnecessary updates of agents since the system needs to examine the simulation at the fixed intervals even in cases where interactions or measurements have not occurred. Thus, this method degrades the overall simulation speed. Another limitation is that it is not easy to predetermine a single time step for the simulator as the smallest time interval among those associated with every agent under worst-case conditions.

3.1.2 Synchronous Variable Time Interval

This method requires all agents in the simulation to update at the same time, usually selecting the most restrictive time step demanded by any of the agents in the simulation. The time interval varies from one time step to the next to meet the needs of the simulation. This method still forces some agents to update unnecessarily even though simulators with variable time intervals offer more efficiency and better flexibility than simulators with fixed time intervals.

3.1.3 Optimistic Time-Warp Method

Asynchronous, optimistic 'time-warp' or 'roll-back' algorithms have been applied to parallel simulations using discrete-event and simple continuous-time models (Fujimoto 2000). These methods allow entities to advance forward independently and asynchronously until it is recognized that synchronization should have occurred earlier due to an interaction between agents. At that point, the relevant agents are rolled back to the time of the interaction and re-evaluated. This method may degrade the overall simulation speed, depending on the degree of roll back and frequency with which it is used. These types of timing methods are ill-suited to complex aerospace simulations where individual models (such as high-fidelity aircraft dynamic models) can not be easily converted to run backwards or to save (and re-initialize) the full description of their past state space.

3.2 Asynchronous with Resynchronization Method

In theory, asynchronous timing mechanisms should be much more computationally efficient than synchronous timing methods since asynchronous timing methods do not require all agents to update at every time step. However, a completely asynchronous method is not appropriate for agent-based simulation because it may not fully capture interactions between agents. In an agent-based simulation, it is common for one agent to collect state information from other agents. For example, in the simulation of the NAS, an air traffic controller agent might need the current values of the location, speed, and heading state variables of aircraft to determine a desired speed for the aircraft to avoid loss of separation. Therefore, it is necessary to resynchronize all aircraft at the time when the air traffic controller calculates the desired speed for the aircraft.

Pritchett, Lee, and Goldsman (2001) demonstrated the potential of a fourth timing method, *asynchronous with resynchronization*. This timing method allows agents to update asynchronously following their own update times, but also makes conservative estimates of when interactions may occur in the future, and requires the relevant agents to jointly update at these resynchronization intervals. The remainder of this paper further examines this timing method and outlines the considerations in setting the resynchronization interval.

In an asynchronous with resynchronization simulation, a 'state updater' object (a simulation executive) within the agent-based simulation architecture must include a mechanism for efficiently carrying out the updates of each agent contained in the simulation and must guarantee that all agents are updated in correct chronological order. To do this, the state updater object maintains a list of the agents active in the simulation. The mechanism for advancing simulation time is based on the sorted object list, which is ordered by the time of the agents' next desired update. As described in previous sections, all agents are required to report the time of their next desired update as well as which agents are also needed to update together. As such, the state updater object can identify the agent next to be updated and check whether it requires other agents to jointly update, and then command the appropriate agents to update. Once agents have been updated, they are sorted accordingly on the sorted object list, and the simulation is advanced to the time of the next update.

3.2.1 Complete Resynchronization

The asynchronous with complete resynchronization mechanism allows all agents in the simulation to update independently until any agent requires resynchronization. With this timing method, the state updater object synchronizes all agents at each resynchronization interval. This method is shown schematically in Figure 1 for a simulation with three agents. For example, agent$_1$ and agent$_2$ update at their own rates until agent$_3$ requires resynchronization.

In large-scale agent-based simulation, the asynchronous with complete resynchronization method can be computationally inefficient because it updates all agents when any agent requires resynchronization.

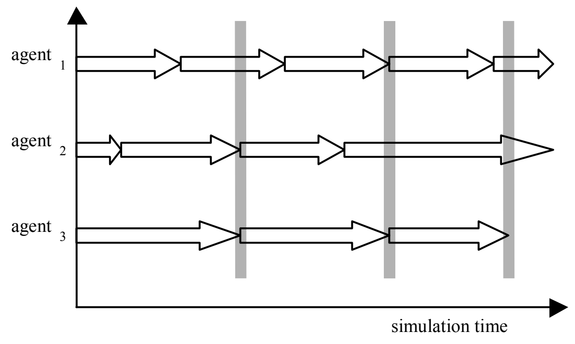

Figure 1: Asynchronous with Complete Resynchronization

3.2.2 Partial Resynchronization

At the times of resynchronization, only some of the agents are required to update due to the interactions or measurements involving them. Therefore, a better approach is to update only those agents interacting with each other at resynchronization times. The asynchronous with partial resynchronization timing method allows agents in the simulation to update at their own update times independently until an agent specifically requires some of the other agents to be resynchronized. With this timing method, the state updater agent synchronizes only those agents that another agent requires to also update at the resynchronization time. This method is shown schematically in Figure 2: $agent_2$ requires only $agent_1$ to update at time 5; $agent_3$ requires only $agent_2$ to be updated at time 20; and $agent_3$ requires both $agent_1$ and $agent_2$ to update at time 40. This timing method can improve computational efficiency by updating just some of the agents interacting with each other at resynchronization intervals.

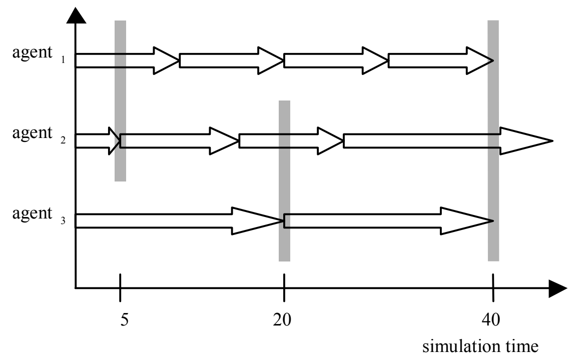

Figure 2: Asynchronous with Partial Resynchronization

Several issues need to be addressed in the development of the asynchronous with partial resynchronization timing mechanism, some of which are beyond the scope of this paper. One issue concerns cyclic dependencies between agents. For example, if $agent_1$ requires $agent_2$ to be updated, $agent_2$ requires $agent_3$ to also be updated, and $agent_3$ requires $agent_1$ to be updated, then the simulator might be in an infinite loop. Another issue lies in identifying the proper sequence in which agents need to be updated for proper interactions.

3.3 Resynchronization Intervals

The computational efficiency of the asynchronous with resynchronization strategy depends upon the setting of the resynchronization intervals. For example, if an interval is conservatively set to be very short, the simulation will resynchronize too frequently, causing unnecessary updates of individual entities in the simulation; conversely, if an interval is set to be too long, the simulation may miss or skip important interactions or measurements such as the conflict or loss of separation between aircraft. Typically, larger resynchronization intervals require better predictions by individual agents about when an interaction may occur. However, the most accurate predictions require significant computation in their own right, negating their benefit, and are fundamentally limited by the unpredictability of future stochastic events. Likewise, the most accurate predictors may require significant knowledge of the underlying dynamics of many agents in the simulation. The development of very accurate predictors can thus be laborious and expensive, as well as application specific.

Setting the resynchronization interval may be modeled as a signal-detection problem, in which it is desired to minimize both false alarms (early resynchronization) and missed detections (resynchronizations that skip over interactions and measurements) in the face of uncertainty about the future dynamics (Kuchar 1996). Therefore, methods of setting resynchronization intervals can be measured in two ways: by their ability to predict when an interaction or measurement may occur; and by their implicit trade-off between false alarms and missed detections.

4 MEASUREMENT AGENTS

To make asynchronous with resynchronization timing methods more computationally efficient, it is important to capture interactions or measurements exactly when they occur. It is also necessary to develop an object that can predict when interactions between agents might occur, for purposes of setting the resynchronization intervals and to identify which agents will interact with each other at the resynchronization intervals. Our approach makes active 'measurement agents' that report when they must next be updated. This projected update time can be a conservative estimate of when a measurement may be needed or when an interaction may next occur.

Since the interactions generally involve two agents, it is most efficient to develop a modular measurement object for predicting the interaction times between only two agents of any type. As a benefit, by making a measurement object that can take any types of two agents in the simulation, the resynchronization intervals in the asynchronous with resynchronization timing methods can be predicted more accurately and efficiently.

A 'measurement management agent' (MMA) is required for creating and managing modular measurement objects for all pairs of relevant agents in the simulation. The MMA maintains a list of modular measurement objects to efficiently manage the update of the measurement objects. The MMA is also required to meet the interface standards as described earlier. The MMA identifies the measurement object that has the minimum value of the next update time in the list of measurement objects and reports the next desired update time to the state updater object. The base standard interfaces stipulate that each of these modular measurement objects must report the time of their next update and return the two relevant agents to be jointly updated. When the MMA is called to update in the simulator, it updates only the measurement object with the minimum value for its next update time. Therefore, only the two agents that are associated with the measurement object need to be resynchronized at the time.

The MMA uses an O-O modular structure, which makes it easy to add and develop new measurement objects that collect different statistical data and predict different interactions between two agents. The MMA can be given the capability to monitor and modify resynchronization intervals, to develop its own predictive power based on experience, and to adjust its measurement objects' resynchronization intervals accordingly through several mechanisms. For example, neural networks can be used to predict when measurements or interactions will occur as the simulation progresses (Wu 1994). Once a neural network is trained, it can provide more accurate resynchronization intervals quickly and without substantial computations, and can be stored for future simulation runs.

5 EXPERIMENT RESULTS

5.1 Test Case: Simulation of a Standard Terminal Arrival Route

To demonstrate the computational efficiency of these different timing mechanisms, an experiment was conducted using the simulation architecture and measurement agents described in the previous sections. The simulation modeled the stream of arriving aircraft flying the Macey Two Standard Terminal Arrival Route (STAR) into Atlanta Hartsfield airport.

For the simulation, several different types of agent models were developed using the simulation architecture described in the previous section. These models include waypoint following aircraft that traverse the desired trajectory defined by a list of waypoints, random aircraft generator agents that generate aircraft agents stochastically with a specified inter-arrival time distribution, air traffic controller agents that determine aircraft sequences in merging arrival streams and then command speeds to aircraft to maintain proper spacing within the traffic streams, and measurement objects that predict the time to the violation of minimum separation between aircraft and collect statistical data of interest.

5.2 Performance Measurement

It is not a trivial problem to measure computational efficiency between different timing mechanisms. In particular, it can be difficult to compare the relative computational efficiency of different timing mechanisms if the simulation includes stochastic elements. Each agent requires some time to execute its internal dynamics. The cost of computation is proportional to the number of agent executions performed. Therefore, the number of agent executions performed during the simulation can be used as a performance measure of the agent-based simulation. In the simulation of the NAS, for example, many aircraft agents are generated, and this type of agent is quite computationally intensive due to both its own continuous-time internal dynamics and enforced resynchronization by other agents. Thus, the average number of updates for each aircraft agent is a good performance measure of the different timing mechanisms in this test case.

5.3 Simulation Results

The simulation results for three different timing mechanisms are shown in Figure 3. The 'Fast-Time' label represents the synchronous with variable time interval method. The 'Complete Resynch.' moniker denotes the asynchronous with complete resynchronization timing mechanism, and 'Partial Resynch.' represents the asynchronous with partial resynchronization timing mechanism. In the asynchronous with complete resynchronization simulation, the air traffic controller agents and measurement objects require all aircraft agents to also be updated at the resynchronization intervals. In the asynchronous with partial resynchronization simulation, the controller agents require only the aircraft in their sector to be updated, and the measurement objects require only two aircraft to be updated at times when there may be a potential loss of separation.

The computational efficiency of the simulation is significantly improved overall with both asynchronous with resynchronization timing methods by allowing aircraft agents to update independently until the resynchronization is required. Specifically, as illustrated in Figure 3, the asynchronous with partial resynchronization timing method is more computationally efficient than the asynchronous with complete resynchronization. The inter-arrival time distribution of aircraft into the arrival route also affects the performance of the timing mechanisms. The lower average inter-arrival time (150 seconds) creates a high traffic volume, in which there are often conflict possibilities and measurement objects predict shorter resynchronization intervals. In this case, the asynchronous with resynchronization methods require relatively frequent resynchronization.

In the asynchronous with partial resynchronization method, fewer updates were required for all aircraft on average, as the method required only two aircraft objects to update at the frequent resynchronization intervals, while the asynchronous with complete resynchronization method required all aircraft objects to update at every resynchronization interval.

In summary, when agents have widely varying update times and need to interact frequently, the asynchronous with partial resynchronization method can greatly improve computational efficiency. Even with the most conservative and simple methods of estimating these resynchronization intervals, significant performance gains were found in the time required to run the simulation of the NAS test case.

6 CONCLUSIONS

Hybrid agent-based simulation using an objected-oriented approach provides a mechanism for more accurately and efficiently analyzing large-scale, complex systems such as the NAS. Such a simulation model represents well the dynamic behavior of large, complex systems involving substantial numbers of entities that can be best described by either (or both) continuous-time models or discrete-event models. The application of hybrid agent-based simulation can be used for *a priori* safety analysis of large-scale complex systems. For example, the impact of human error or machine failures on the safety of such systems can be evaluated by using more realistic hybrid agent-based simulation.

As noted herein, hybrid agent-based simulation architecture should be sufficiently flexible and extensible to easily incorporate different types of models with varying fidelity and resolution. With O-O software architecture, an agent-based simulation could accept models of any type, control their different timing in a computationally efficient manner, and handle interactions between the agents within the simulation.

In order to serve as a valuable design and analysis tool, the agent-based simulation of large, complex systems must also be computationally efficient to provide statistically meaningful results within a reasonable amount of computation time. This paper has discussed timing mechanisms for hybrid agent-based simulation to improve computational efficiency. Such efficiency can be achieved when agents are updated only when needed for their internal dynamics, for accurate interactions with other agents, and for timely measurements.

This paper focused on timing the updates of individual agents in the agent-based simulation. Different types of timing mechanisms were reviewed, and a comparatively new timing method, asynchronous with resynchronization, was detailed. Its accuracy and computational efficiency

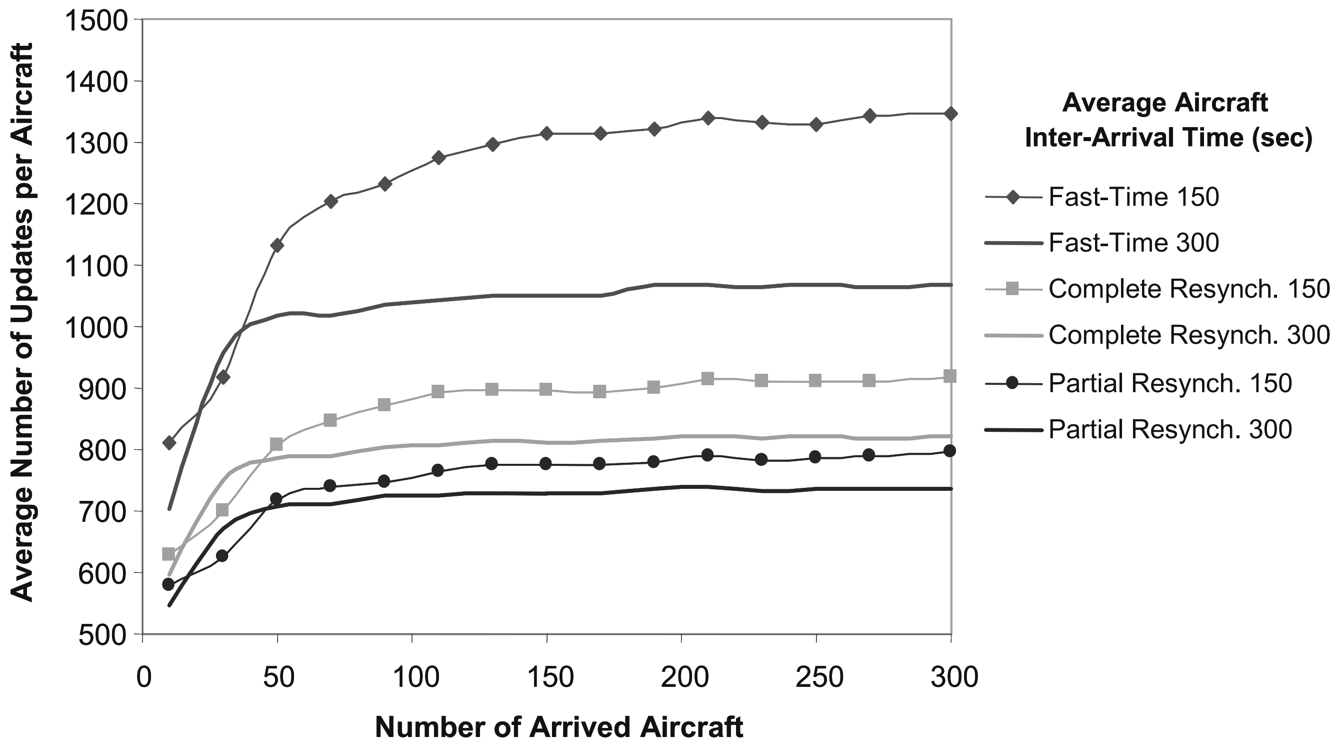

Figure 3: Experimental Results of the Simulation with Different Timing Mechanisms and Avg Aircraft Inter-Arrival Times

depends on the setting of resynchronization intervals. In particular, considerations in setting the resynchronization intervals were outlined and O-O modular measurement agents capable of predicting resynchronization intervals were developed and described to facilitate the implementation of new timing mechanisms. This measurement object has the direct benefit of allowing for easy implementation of new measurements in the simulation, as this high-level object monitors all pairs of relevant agents and interacts with the larger simulation architecture, including commanding the relevant agents to resynchronize. These developments are expected to enable more accurate and computationally efficient simulations.

As described earlier, the resynchronization intervals depend strongly on the update time of modular measurement objects. Measurement objects, therefore, need to be improved to be capable of estimating efficiently and intelligently when interactions may occur, in order to improve the computation efficiency of the asynchronous with resynchronization simulation method. Currently under development is an intelligent and efficient module to predict via neural nets the next update time for each measurement object.

Even with the conservative and simple measurement object that estimates resynchronization intervals, significant computational gains were achieved using the new timing mechanisms in a test case simulation of the NAS.

ACKNOWLEDGEMENT

This work was funded by the NASA Ames Research Center under Grant NAG-2-1291, with Irv Statler, Mary Conners and Kevin Corker administering and serving as technical points of contact. The authors would also like to thank those who have contributed to the development of the simulation, including Corey Ippolito, Ted Chen, and Satchidanand Kalaver. Parts of this paper appeared in the *Proceedings of the 2001 Digital Avionics Systems Conference.*

REFERENCES

Bezdek, W. J., T. A. Halley, and P. C. Hummel. 1997. Model reuse for software development and testing: The application of common interfaces to support variable fidelity models. In *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, New Orleans, LA, 376–386.

Cellier, F. E. 1979. Combined continuous / discrete system simulation languages – usefulness, experiences and future development. *Methodology in Systems Modelling and Simulation*: 201–220.

Davis, P. K. and J. H. Bigelow. 1998. Experiments in multiresolution modeling (MRM). Report MR-1004-DARPA, Rand, Santa Monica, CA.

Dessouky, Y. and C. A. Roberts. 1997. A review and classification of combined simulation. *Computers and Industrial Engineering* 32 (2): 251–264.

Fahrland, D. A. 1970. Combined discrete event continuous systems simulation, *Simulation* 14 (2): 61–72.

Fujimoto, R. M. 2000. *Parallel and Distributed Simulation Systems.* New York, NY: Wiley.

Jim, H. K. and Z. Y. Chang. 1998. An airport passenger terminal simulator: A planning and design tool. *Simulation Practice and Theory* 6: 387–396.

Klingener, J. F. 1996. Programming combined discrete-continuous simulation models for performance. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 833–839, Piscataway, NJ: Institute of Electrical and Electronics Engineers.

Kuchar, J. K. 1996. Methodology for alerting-system performance evaluation. *AIAA Journal of Guidance, Control, and Dynamics* 19 (2): 438–444.

Law, A. M. and W. D. Kelton. 2000. *Simulation Modeling and Analysis*, 3d Ed. New York, NY: McGraw-Hill.

National Research Council. 1998. *The Future of Air Traffic Control: Human Operators and Automation*, ed. C. D. Wickens, A. S. Mavor, R. Parasuraman, and J. P. McGee, Washington, DC: National Academy Press.

Odoni, A. R., et al. 1997. Existing and required modeling capabilities for evaluating ATM systems and concepts. Technical Report, MIT International Center for Air Transportation.

Pritchett, A. R. and C. Ippolito, 2000. Software architecture for a Reconfigurable Flight Simulator. In *Proceedings of the AIAA Modeling and Simulation Technologies Conference*, Denver, CO.

Pritchett, A. R., S. M. Lee, and D. Goldsman. 2001. Hybrid-system simulation for safety analysis of the National Airspace System. Accepted for publication in *AIAA Journal of Aircraft.*

Roberts, C. A. and Y. M. Dessouky. 1998. An overview of object-oriented simulation. *Simulation* 70 (6): 359–368.

Saleh, R. A., S.-J. Jou, and A. R. Newton. 1994. *Mixed-Mode Simulation and Analog Multilevel Simulation*, Boston, MA: Kluwer.

Sichman, J. S., R. Conte, and G. L. Gilbert. 1998. Multi-agent systems and agent-based simulation. In *First International Workshop*, MABS '98, New York, NY: Springer.

Wu, J.-K. 1994. *Neural Networks and Simulation Methods.* New York, NY: Marcel Dekker.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems.* San Diego, CA: Academic Press.

AUTHOR BIOGRAPHIES

SEUNGMAN LEE is a Ph.D. student in the School of Industrial and Systems Engineering at the Georgia Institute of Technology. He received a B.S. from Hanyang University and M.S. degrees from Pohang University and Carnegie Mellon University. His research interests include air traffic control, flight simulation, and large-scale agent-based simulation of hybrid systems. His e-mail and web addresses are <seungman@isye.gatech.edu> and <www.isye.gatech.edu/~seungman>.

AMY R. PRITCHETT is an Assistant Professor in the Schools of Industrial and Systems Engineering and Aerospace Engineering at the Georgia Institute of Technology. She received S.B., S.M., and Sci.D. degrees from the Department of Aeronautics and Astronautics at the Massachusetts Institute of Technology. Her research specialties include cockpit design, air traffic control, flight simulation, and large-scale agent-based simulation of hybrid systems. Her e-mail and web addresses are <amyp@isye.gatech.edu> and <www.isye.gatech.edu/~amyp>.

DAVID GOLDSMAN is a Professor in the School of Industrial and Systems Engineering at the Georgia Institute of Technology. His research interests include simulation output analysis and ranking and selection. He also studies applications arising in the healthcare field. Dave has been an active participant in the Winter Simulation Conference — he is currently on the Board of Directors, and was the 1995 Program Chair and 1992 Associate *Proceedings* Editor. His e-mail and web addresses are <sman@isye.gatech.edu> and <www.isye.gatech.edu/~sman>.