Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

HYBRID-SYSTEM SIMULATION FOR NATIONAL AIRSPACE SYSTEM SAFETY ANALYSIS

Amy R. Pritchett
Seungman Lee
David Huang
David Goldsman

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

ABSTRACT

Analysis of large, complex systems requires simulations of hybrid-system dynamics, i.e., dynamics which are best described by a combination of continuous-time and discrete-event models, and their interactions. To serve as valuable research tools, such simulations need also be computationally efficient, readily modifiable, and open to a wide range of component modules. This paper describes the development of a simulation architecture meeting these criteria. The issues with its development are described conceptually, and its application to the task of safety analysis of the national airspace system is discussed. In particular, an object-oriented approach to hybrid-system simulation is detailed, and computationally efficient methods of updating the simulation are described and compared.

1 INTRODUCTION

The behavior of many large, complex systems is hybrid in nature; i.e., it includes both continuous-time and discrete-event models, and the behaviors these models represent are not separable, but instead can interact in significant ways. A simulation capable of recreating these hybrid-system dynamics provides an analysis tool that can dramatically change the design process. In electronics, for example, integrated circuits were purposefully designed to have components spaced far apart until simulations capable of predicting electromagnetic interference could be used to analyze and re-design smaller chips (Saleh, Jou, and Newton 1994). Many aerospace systems are best captured by hybrid-system simulations, ranging from aircraft with flight control systems that change modes, to on-board systems with discontinuous behaviors such as open-closed mechanisms.

Take, for instance, the task of performing safety analysis on the National Airspace System (NAS). Merely simulating the trajectories of the aircraft would not capture the discrete actions of controllers; likewise, continuous-time simulation architectures would not be well suited for the tasks of inserting aircraft into the airspace being simulated at random rates, or for the stochastic injection of disturbances. Elements of NAS dynamics have been simulated using purely discrete-event models (Odoni, et al. 1997); however, such models do not have the resolution to capture safety issues. Similarly, hybrid-system simulations have been made of the NAS, but have been limited to specific applications or parts of the NAS (Odoni, et al. 1997, Jim and Chang 1998).

In order to serve as an effective design tool, simulation of large-scale systems must also meet a number of practical considerations. Most obviously, the simulation should be sufficiently computationally efficient that it can provide a time-effective analysis tool, even when large numbers of runs are required. In addition, the simulation should be rapidly reconfigurable, so that the development time of the simulation is not prohibitive, and so that the simulation can be applied to a range of applications and can accommodate models of varying fidelity.

This paper discusses issues involving hybrid-simulation, with the thesis that many of these issues can be solved by an object-oriented software architecture. This architecture handles the communication between objects without needing to treat objects differently based on the type of their underlying model. Likewise, this architecture can be constructed to control the timing and updating of the elements in a computationally efficient manner.

First, a comparison of fundamental differences and similarities between continuous-time and discrete-event models is made. Then we describe the test case used in this paper – the safety analysis of the NAS. Conceptual issues in, and requirements of, hybrid-system simulations are

discussed, and then we present an example simulation software architecture. Methods of controlling simulator timing are also described, illustrated by comparisons from the test case.

2 BACKGROUND: DISCRETE-EVENT AND CONTINUOUS-TIME MODELS

Important, fundamental differences exist between discrete-event and continuous-time models. These differences are shown in Table 1.

Discrete-event models typically attempt to define the state of a system by categorizing whether conditions exist, or by quantifying the number of entities within a category. As such, they can describe the system without attempting to explain internal dynamics. Their defining parameters stipulate how and how often states will transition from one to another. These parameters are set by experimental observation of existing systems, and can capture the stochastic nature of the system. Discrete events are well suited for modeling systems made of multiple entities with no internal dynamics of relevance – and discrete-event models usually require study of an established system to ascertain their parameters.

Continuous-time models, on the other hand, typically attempt to model the internal dynamics within the system. The state is usually defined as a measure of energy within the system, such as measures of position and velocity. These models are often physics-based; i.e., they can be developed before the system can be measured as a form of analysis. However, these models are described by differential equations, which can be numerically complex to propagate forward through time. (A related model type, that of discrete-time systems, is modeled in similar ways, e.g., via the use of difference equations, and can treated as a special class of continuous-time models; see Kheir 1996.)

These two types of models, applied to the same problem, tend to have very different rates at which the states need to be updated. Continuous-time models are solved through numerical integration (or transition) algorithms that approximate the continuous variations by updates at discrete intervals. These intervals (or time steps), at the very least, must be at twice the rate that dynamics of interest occur in order to capture their basic properties (Beltrami 1987) – but in most applications, the time step is set much smaller to reduce error in the numerical solutions (Press, et al. 1992). Discrete events, on the other hand, typically capture fairly large changes in dynamics, and need to occur less often. In some cases, the update rates for the two types of models may be separated by several orders of magnitude.

Comparisons of these two types of models are conceptual distinctions only. It has been proven possible to incorporate models of either type into simulation software intended for the other. For example, continuous-time models have been merged into discrete-event simulations by fitting updates in their state values into mechanisms for discrete transitions with fixed, small transition times. However, it has also been noted that such cross-implementation often requires restrictive assumptions on the models, limits their accuracy, increases the complexity of the software, and does not result in a computationally efficient simulation (Fahrland 1970). Thus, these differences are of dramatic import to the simulation designer, as the simulation architecture is typically tailored to the type of model implicit in the simulation.

Table 1: Comparison of Continuous-Time and Discrete-Event Models

	Continuous-Time Models	**Discrete-Event Models**
System Being Simulated	Specific mechanical unit with complex functioning	Multiple, often simple, entities
Definition of System State	Distribution of energy within system	Categorization of current system properties
Typical Measures of System State	Position, attitude, velocity (Deterministic)	Queue size, incidence (Statistical properties thereof)
Typical Factor Driving Updates	Time	Transitions from state to state
Capability of Simulation	Analyze deterministic dynamic behavior of a mechanical unit	Analyze stochastic nature of interactions between entities
Common Uses	Design and analysis of unit, (real-time) training	Analysis and planning of operations with multiple entities

3 TEST CASE: SAFETY ANALYSIS OF THE NATIONAL AIRSPACE SYSTEM

At its full extent, the NAS is a system of overwhelming complexity. Thousands of aircraft may be aloft at one time; hundreds of controllers are monitoring and directing them with the assistance of many communication and surveillance technologies.

Complexity of the NAS is also due to the large number of different entities involved in its operation. Only at a high-level can the NAS be modeled as consisting of controllers, pilots, and aircraft. When more resolution is required, distinctions must be made between the different types of controllers (ground, tower, terminal area, en-route, etc.) and their hierarchies; likewise, aircraft can be of many different types with different performance, and their pilots may have widely different goals and levels of experience.

The behavior of the NAS, to a great extent, is defined by the interaction of these different elements. The NAS can not be modeled as a collection of independent aircraft flying simultaneously; instead, controllers (and pilots) are constantly changing direction of flight in response to the actions of others. These interactions may meet a number of goals, ranging from time-critical collision avoidance maneuvers, to strategic plans for air traffic flow management.

In order to meet increasing capacity demands and stricter safety demands, the NAS is being updated. These updates range in scale from near-term equipment upgrades, to longer-term calls to change the manner in which controllers and pilots interact.

These upgrades create a design problem of vast scale. For both cost and safety, as much analysis should occur before implementation as possible. The worth of this analysis will be measured by its ability to predict problems, allowing for their correction, before implementation. Simulation and modeling have been recognized as a critical part of this analysis (National Research Council 1997).

The NAS has been simulated before (Odoni, et al. 1997, Jim and Chang 1998). However, most of these simulations have not been suitable for large-scale safety analysis. Many NAS simulations have been motivated by studies of efficiency. These simulations have been concerned with values such as airport inter-arrival times or flight delays. These concerns are best abstracted by discrete events, and hence have largely been covered by discrete-event simulations.

Safety, on the other hand, is largely determined by continuously evolving interactions which discrete-event simulations can not capture. For example, a discrete-event simulation may model the times at which two aircraft arrive at an airport; but without monitoring their continuous trajectories, it is nearly impossible to measure with certainty whether these two aircraft come unacceptably close during their flights. Therefore, simulation suitable for safety analysis needs better resolution of some parts of the system than can be readily provided by discrete events alone.

The most notable element requiring adequate resolution is the trajectory of the aircraft itself. Typically represented by differential equations, aircraft trajectories are usually best modeled as continuous-time objects. Fortunately, many excellent models are currently available at all levels of fidelity, ranging from outer-loop models of the aircraft's guidance, to detailed inner-loop models of the aircraft's flight dynamics (Hanke 1971, Johnson and Hansman 1994, Stevens and Lewis 1992). These models are physics-based, which allows their parameters to be estimated (if not exactly known), and brings predictive power to the simulation where novel NAS changes are to be simulated before measurements of actual dynamics can be observed.

Other elements remain best modeled as discrete events, for a variety of reasons. Some elements of NAS behavior can be predicted to occur discretely; for example, the generation of a cue of aircraft waiting for taxi clearance to the take-off runway is discrete in nature. Other elements of the NAS are not needed at a fine resolution, so computational effort can be saved by reducing them to a notable state – for example, a detailed, computationally expensive continuous-time model of an aircraft waiting for take-off can be temporarily replaced by a notation in the take-off cue. Finally, discrete events can be used to interject stochastic elements into the simulation, including the dramatic conditions caused by errors and failures, as well as disturbances into aircraft model parameters representing variations in aircraft behavior.

Of critical importance in NAS safety simulation is inclusion of human performance models, for their behavior drives system dynamics. Their behavior should not be typecast into either continuous or discrete forms. For example, many human performance models can be based on procedures or expert systems that call for isolated actions to be triggered by conditions in the environment (discretely) while also maintaining a continuously evolving valuation of workload or working memory content.

Measurements of NAS safety can also be treated as discrete events that occur when conditions in the environment are met, such as loss of separation between two aircraft. Unlike 'normal' discrete events, these measurements do not need to trigger subsequent events in the simulation, beyond recording the event to an output file. However, these measurements can share in the computational structures that are used for other discrete events.

4 HYBRID-SYSTEM SIMULATION CONCEPTS

In cases where an elegant, analytic solution can not be found to analyze a system represented by discrete-event

and continuous-time models, a hybrid-system simulation is required to calculate system dynamics through time. Several approaches have been suggested to the design of hybrid-system simulations. Some create larger 'meta-model' frameworks in which each 'type' of simulation functions separately (Saleh, Jou, and Newton 1994, Friedman 1996). Other solutions have adapted existing simulations of one type to include the other (Wieting 1996, Cellier 1986).

A third approach, sometimes called the 'fully-integrated approach' (Saleh, Joe, and Newton 1994), seeks to create new software that inherently accepts the two model types. Such approaches have been undertaken with special modeling languages (Kettenis 1997). The current paper will instead focus on software architectures (not necessarily written in special languages) that can accept models of any type, control their timing in computationally efficient mechanisms, and handle communication between the objects.

Previous sections have highlighted the differences between the models used in hybrid-system simulation. A simulation architecture can capitalize upon the various abilities these models share: to update themselves when required; to report when their next update is required; and to report interactions with other objects that warrant a joint update.

At a high-level, a simulation architecture can require components to meet these three interface requirements. All other dynamics of the components can remain internal to their models, without requiring intervention by the larger simulation architecture. This internalism can be considered a feature. It prevents fundamental restrictions on the type of model allowed in the simulation – and it allows for the simulation to include components of various resolutions as required by the task at hand (Bezdek, Halley, and Hummel 1997, Davis and Bigelow 1998).

Without placing restrictions on components' models, the simulation architecture also needs to support their interactions. These interactions may take on several forms. Traditional to continuous-time simulation is coupling between different continuous-time objects, such as two aircraft flying in formation (or executing avoidance maneuvers) reacting to the movements of each other. A discrete event may also impact a continuous-time object in several ways: it may enact a discrete change in the variables maintained by the continuous-time object (such as a sudden change in acceleration due to the application of brakes or engine failure); it may change a parameter's value within the continuous-time object (such as a change in stability derivatives in response to a discrete change in aircraft configuration); or it may 'swap in' a whole new continuous-time model better suited to the situation (such as inclusion of a higher fidelity aircraft model at the start of an avoidance maneuver, or a switch to a 'taxiing' aircraft model after landing). Continuous-time objects can interact with discrete events when their values are the events' triggers.

Within a simulation framework that supports such interactions, the simulation designer is given the ability to make components work efficiently on their own. For example, in modeling an aircraft with on-board systems that have discrete dynamics, the simulation designer has choices beyond the usual requirement that these two behaviors be kept common in one model, despite their two different time scales. Instead, the designer would have the option of making the on-board systems into a separate discrete-event model that communicates appropriately with the continuous-time model of the aircraft dynamics. Such an approach allows the simulation designer to separate behaviors according to the times at which they will need to be updated without substantial re-workings within individual components (Kettenis 1997).

While a simulation's operation does not depend on measurements, the desire for accurate measurements is typically the motivation for the simulation. In many cases, the measurements are temporal in nature and hence it is important that a measurement be taken exactly when it occurs. One approach is to make measurements an active element by treating them as discrete events that must report when they must next be updated. This projected update time can be a conservative estimate of when the conditions wanting recording may next occur. While this process requires measurements to have a prediction power, it also reduces the need for unnecessary measurements. This also mirrors a duality between measurements and conditionally based discrete events – while the former have no lasting impact on simulation dynamics, the latter are measurements with a consequence.

Based on this discussion, we can summarize several requirements for a hybrid-system simulation architecture. First, the simulation architecture should not place unnecessary limits on the types of objects, but instead accommodate any components that can list their update times and update themselves upon command. Second, the simulation architecture should facilitate communication between objects, so that it is easy for the simulation designer to break apart models according to their functionality and update rates, without requiring lengthy communication standards to be developed. Finally, the simulation architecture should be capable of timing the updates of the individual components in a computationally efficient manner.

5 SIMULATION ARCHITECTURE

The previous sections discussed conceptual issues with hybrid-system simulation. This section discusses a particular simulator architecture design. This simulation is based on extensions to and adaptations of the Reconfigurable Flight Simulator (RFS), which was

originally designed for more traditional applications of flight simulation (Pritchett and Ippolito 2000).

This simulation allows for the inclusion of several broad classes of objects, as shown schematically in Figure 1. An arbitrary number of vehicles can be added to the vehicle list. These components must fit a base interface for vehicles, which was designed to support continuous-time models of vehicle dynamics. For specific applications, base interfaces have also been defined for aircraft and for spacecraft. Further, several basic components, including 6DOF and waypoint following aircraft, have been developed and can be used or extended by other developers.

Beyond these capabilities as a 'normal' flight simulator, an arbitrary number of controller and measurement objects can be added to a dedicated list. The base interface standards for these objects are less specified, allowing flexibility in the objects' behaviors. Components that have been added to date include: 'Random Aircraft Generators' that place aircraft into airspace according to a given distribution of inter-arrival times; basic air traffic controllers 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 look for and record events specified in a text script. The flexibility of this type of component allows for many other types of discrete-event and measurement components to be added as desired.

Other types of components are also available in the simulation architecture as support for hybrid-system simulation needs. Most of the components needed for this application are already established, but these components can be modified or added to as desired. Input-output (IO) objects provide mechanisms for graphical output of the simulation (such as an 'air traffic control screen' and text output of commands given by controllers), and for data recording of the measurements. The environment controller and database (ECAD) provides a common simulation environment for all the components by giving common axis definitions and conversions, and by allowing for the inclusion of atmospheric and terrain models as desired. Finally, a networking object handles communications between simulations run on multiple machines over a network; this networking is transparent to the rest of the simulation and therefore does not require re-compilation of components to use or access networking.

To function as a hybrid-system simulation, each continuous-time, discrete-event, and measurement object is required to meet the minimal standards of a standard interface. In particular, each of these components must update its state on command, report the time of its next update (or, in the case of some discrete-event and measurement objects, the next time at which an update might occur conditionally upon other events), and identify whether any other objects must also be updated.

For the continuous-time objects, a standard input that needs to be provided is an upper bound on the numerical error allowable in each time step. With this input, the next update time is calculated based upon knowledge of the interior dynamics; algorithms for such flexible time step calculations are well established (Beltrami 1987).

Communications between objects are handled by the high-level simulation object. It follows that the designer of a hybrid-system simulation using this architecture does not need to develop communication standards beyond giving individual components the ability to send and receive messages. Many standard messages can be passed through the base interface standards of vehicle and controller / measurement objects.

Those messages that do not fit within the base interface standards can be sent through the simulator object via the Object Data / Methods Extensions (OD/ME) protocol, which requires sending a message to the simulator object along with a request for its destination (Pritchett and Ippolito 2000). This mechanism also allows for objects to request the addition or destruction of other objects; for example, a random aircraft generator can request the addition of a new aircraft to the vehicle list, with subsequent messages to that new aircraft that provide it with initialization data.

To facilitate efficient timing of the objects' updates, a 'state updater' object within the controller list maintains a list of the vehicle and controller objects, sorted by the times of their next desired updates (see Figure 2). This state updater object can identify the object next to be updated, regardless of type, can query that object as to whether it requires other objects to also be updated, and can command the appropriate objects to update. Once objects have been updated, they each are asked for their new update time, and are sorted accordingly.

6 EFFICIENCY AND TIMING METHODS

In large-scale simulation, concerns with computational efficiency extend past standard efforts to make each component individually efficient. Overall efficiency is achieved when each object updates only when needed to meet several criteria: accurate modeling of its interior dynamics; correct interaction with other objects; and timely measurements.

Any unnecessary updates of objects may be considered wasted use of the processor. However, methods of deciding when an update may be required for correct interactions or measurements is usually non-exact once the simulation at hand is non-trivial in size. Likewise, the overhead computation to enable the most sophisticated timing methods may be non-trivial and can slow down the simulation by itself.

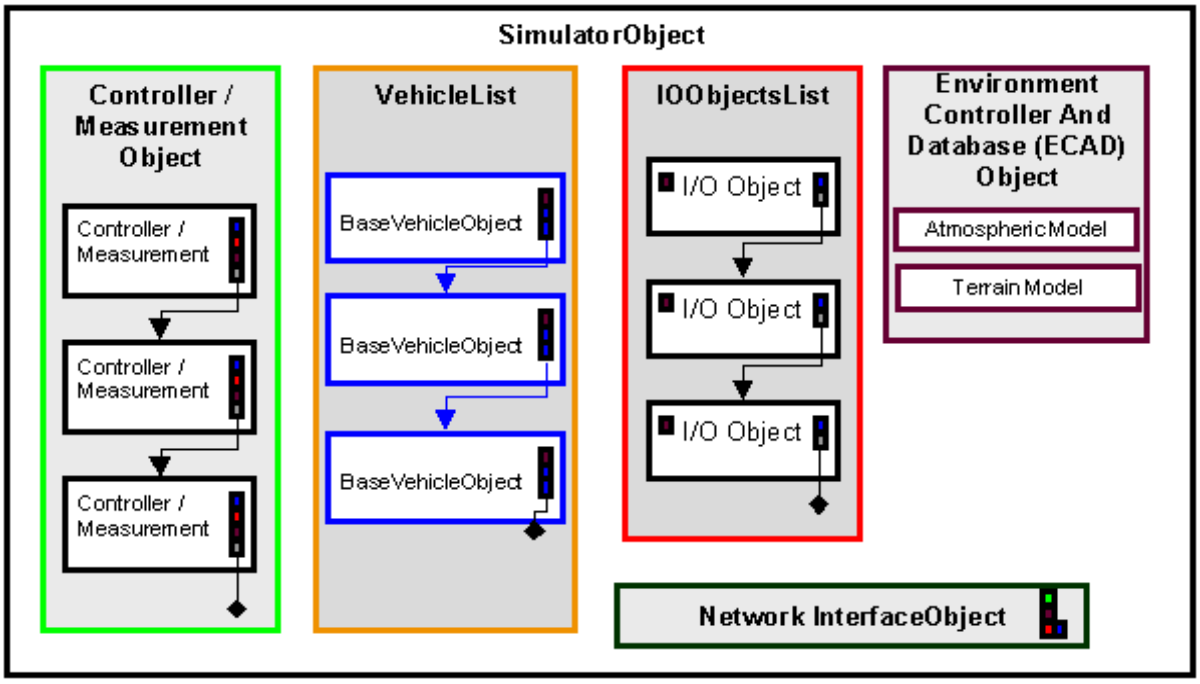

Figure 1: Schematic of Component Classes within the Simulation Architecture

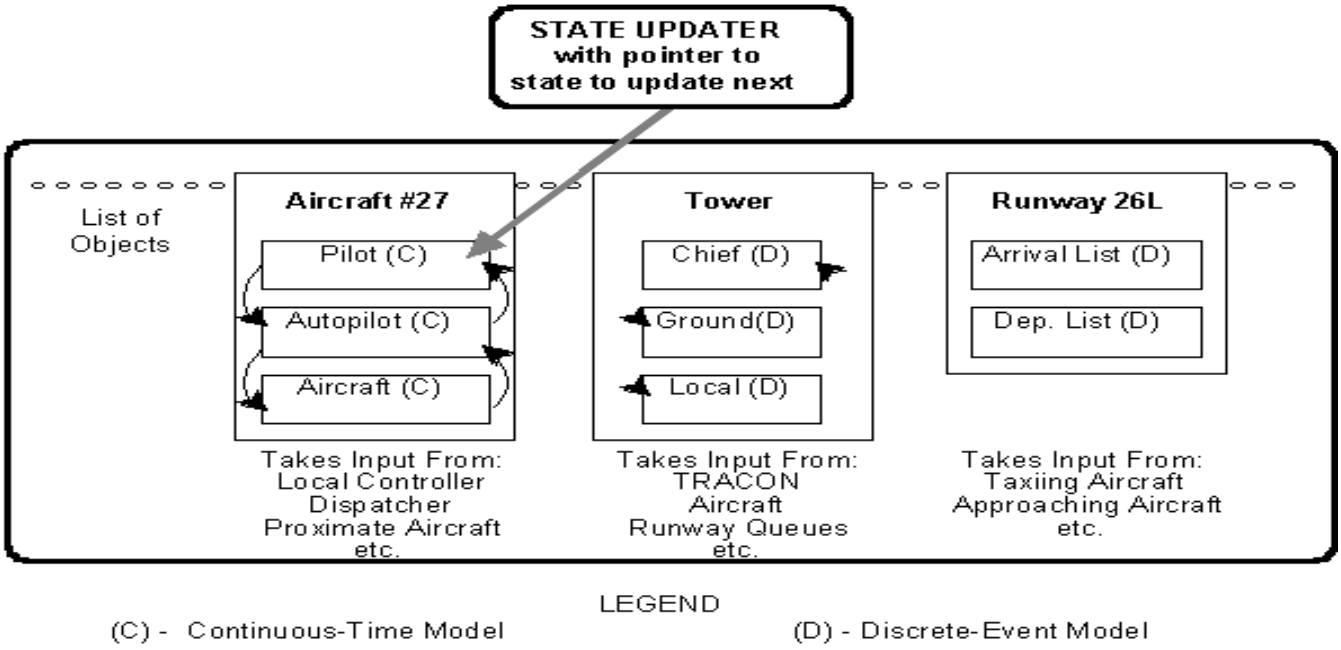

Figure 2: Schematic of Sorting of Simulation Components by Update Time, with Access by a State Updater Component

Even measuring the computational efficiency of a simulation can be non-trivial. Once the simulation includes stochastic elements, it can be difficult to compare with certainty the relative speed of different update timing methods, which may take the simulation though different system dynamics due to the inclusion of disturbances or anomalies.

This section will compare different timing methods, and illustrate their effect on a representative NAS simulation using the architecture described in the previous section. Then tradeoffs in the fundamental characteristics

of these methods are discussed, and alternative methods are commented on.

6.1 Timing Methods

Two major variables define the variety of methods for determining the timing of simulation components. First is the selection of how the time step is set (next-event time advance or fixed-increment time advance) (see Law and Kelton 2000). Second is the choice whether the simulation will be entirely synchronous (i.e., all components update at the same time), partially synchronous and partially asynchronous, or entirely asynchronous (i.e., all components update individually). Several timing methods can be defined by these two variables, as follows.

6.1.1 Fixed Time-Step Synchronous

This timing method has all objects update at the same time, and this update time is based upon a fixed time step. This method is commonly used in current flight simulation techniques, where the time-step may be fixed by conservative analysis of the fastest dynamics in the system, or by the system clock in real-time simulation. This method is very basic and is often the first step in the development of a hybrid-system simulation. It also provides conservative results that can be guaranteed to not miss any measurements or interactions by the setting of an arbitrarily small time step, without requiring predictions from discrete-event or measurement objects. However, it also forces all objects to update at a rate governed by the worst-case dynamics of the component with the fastest response, which is computationally inefficient.

6.1.2 Variable Time-Step, Synchronous

This timing method has all the objects update at the same time, but varies the update time from one time step to the next to meet the needs of the simulation. For instance, the update time may be chosen by polling all objects for their desired time step, and then selecting the worst-case (smallest). This method still forces some objects to update more times than normally required, but can relax the time step when conditions allow.

6.1.3 Asynchronous With Resynchronization

This timing method allows for components to be updated independently following their own update times. This is shown schematically in Figure 3 for a simulation with four aircraft, a random aircraft generator (RAG) and a measurement object; the aircraft and RAG update at their own rates until a measurement requires a complete synchronization. This will allow for objects with fast dynamics to update frequently without requiring other objects to be bound by such small time steps. However, this method also allows for objects which interact, or which measure interactions, to require all objects (or some objects) to resynchronize when it is time for their update, with the result that interactions and measurements can be based on values from temporally co-located objects.

6.2 Case Study: Simulation of a Standard Terminal Arrival Route

To demonstrate the computational efficiency of these methods, a numerical simulation was conducted using the simulation architecture described in the previous section. The simulation modeled the stream of arriving aircraft flying the Macey Two Standard Terminal Arrival Route (STAR) into Atlanta-Hartsfield airport. Aircraft were injected into the simulation stochastically with a specified inter-arrival rate. A controller scheduled the aircraft from the multiple entry streams into one arrival flow by selecting the appropriate order of the aircraft, and commanding speeds to the aircraft that created this desired traffic pattern. The aircraft were removed from the simulation when they reached the runway.

Figure 4 illustrates the results of the simulation. Efficiency is measured by the average number of times the aircraft objects are called to update during the course of the run. We use this efficiency measure since aircraft are the most computationally intensive objects – due both to their underlying update rate as well as to forced resynchronizations.

We present data for two timing methods. The ‘Fast Time’ method used the variable time step synchronous method. The ‘Asynchronous’ method used the asynchronous with resynchronization method; the controller and measurement objects commanded a complete resynchronization at times when they predicted a conflict might occur or the next command might be warranted.

The inter-arrival rate of aircraft into the arrival route was also varied. The highest inter-arrival mean (500 seconds) created a fairly low traffic intensity, with commensurately few interactions. At this inter-arrival mean, the benefits of asynchronous simulation are noticeable, but not dramatic.

The lowest inter-arrival mean (100 seconds) created a high traffic intensity, in which controller commands and potential conflicts were possible. The aircraft often needed to maneuver, and conflicts indeed manifested themselves. In the ‘Fast-Time’ method, this had the effect of requiring many more updates for all aircraft on average. In the ‘Asynchronous’ method, fewer updates were required overall as the aircraft needing updates at small intervals were able to work independently.

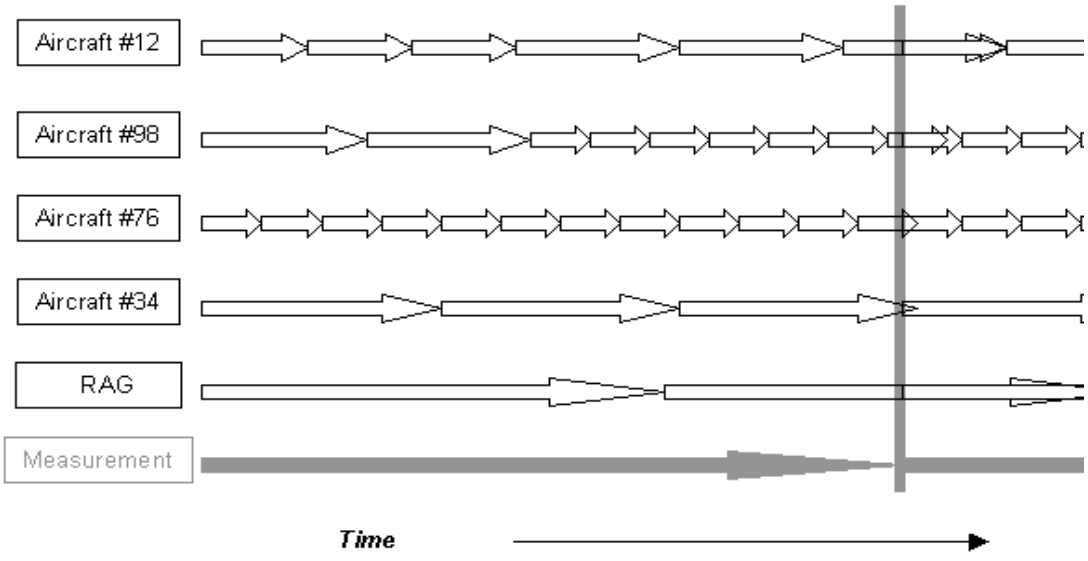

Figure 3 Schematic of Asynchronous Simulation With Resynchronization

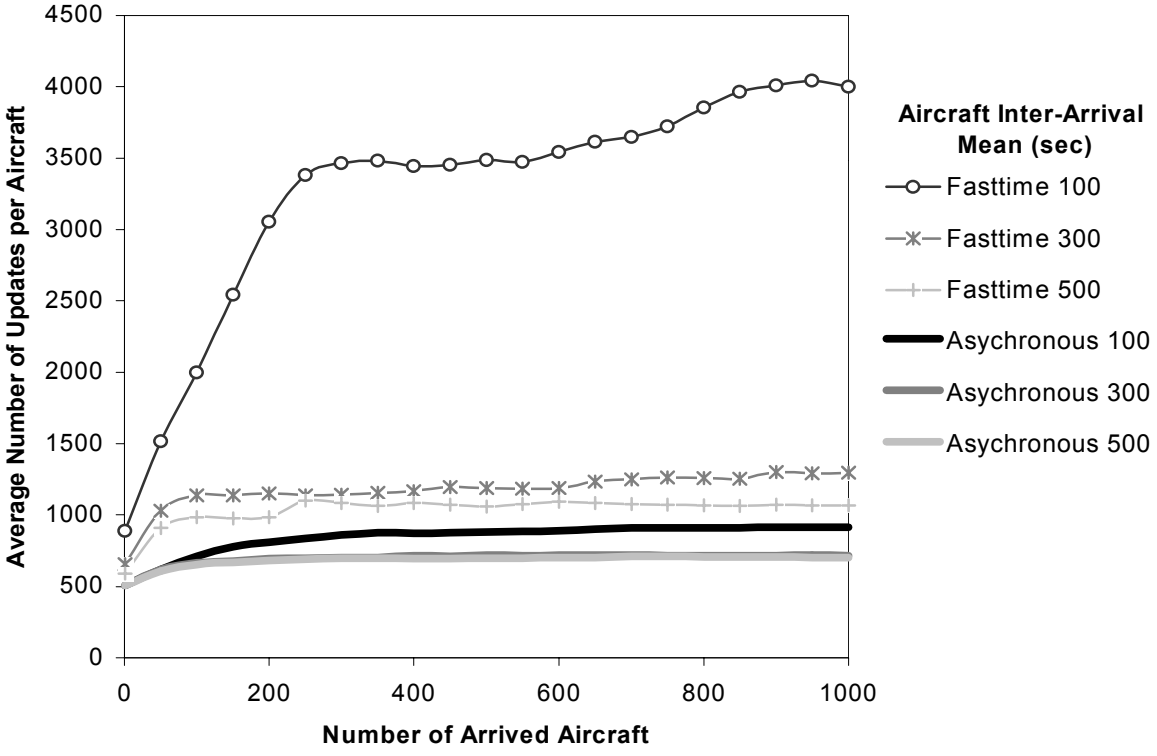

Figure 4: Experimental Results from a STAR Simulation Showing Computational Efficiency (avg. number of calls to aircraft) for Synchronous, Variable Time Step and for Asynchronous with Resynchronization Timing Mechanisms

6.3 Tradeoffs Between Resynchronization Intervals and Efficiency

In applications such as just shown in the case study, there appear to be benefits with allowing simulation objects to run, at least for intervals, asynchronously. At first glance, this seems to imply that the best efficiency will arise with the largest resynchronization intervals. However, two main issues limit the size of resynchronization intervals.

First, larger resynchronization intervals require better (and more computationally expensive) predictions by the individual components about when a resynchronization may occur. Better predictions require more extensive calculations – at an extreme, the predictor would need to internally simulate other objects in order to accurately predict when a problem might occur! As such, the value of better predictions can reach a point of diminishing returns, where the additional computations in the predictions offset any savings in computations by other objects.

Second, larger resynchronization intervals require better (and model specific) predictions by the individual components about when a resynchronization may occur. Simple predictions about a potential aircraft collision, for example, can be made based on commonly available aircraft position and velocity; more accurate predictions require knowledge of the aircraft's internal dynamics and likely future actions. This imposes an obvious development cost on the simulation. It also makes such 'smart' predictors difficult to use in simulations where a large variety of objects may be involved in the prediction, and limits the use of the predictors to specific cases.

6.4 Alternatives to Resynchronization

So far, this discussion on simulation timing has assumed that accurate measurements and interactions can only occur when the objects involved are temporally co-located, with the implication that occasional resynchronization is always required. It is also possible for measurements and interactions to be calculated from temporally disjoint objects. Of course, such calculations tend to be more complex, but with such a capability fewer resynchronizations are needed solely to make measurements or predictions about the future. However, at least partial resynchronizations will still be needed when predicted interactions require other objects to jointly manifest a new behavior at a certain time (e.g., a predicted collision avoidance alert requiring two aircraft to synchronize and communicate at the start of the alert). Likewise, in a simulation with stochastic elements, such predictions can not be made with certainty and hence remain susceptible to inaccuracies.

Similarly, we have assumed that the simulation always runs 'forward' in time. This assumption generates conservative timing intervals to avoid 'missing' any important interactions. For some applications, simulations capable of running 'backwards' to a potential missed interaction are possible, with the benefit of relaxing timing intervals (Mirtich 2000, Jefferson 1985). However, these 'rollback' or 'timewarp' simulations can fit better in some domains than others – some types of models are simply easier to either 'run' backwards or store their recent state space so that the simulator can be backed up to before the missed problem (such methods have most commonly been applied to systems with purely discrete dynamics or very simple continuous-time models). Likewise, these methods incur a computational hit, and so should be used wisely.

7 CONCLUSION

This paper has discussed issues related to simulating large, complex systems as an analysis method during their design. Hybrid-system simulation is an emerging field of interest with the potential to provide such an analysis tool.

Simulation of the NAS for safety analysis was used as an example and a test case throughout the paper. This application shares many of the qualities (and requirements) of other aerospace systems. For example, large-scale simulations of many operational systems are now being proposed, including military mission planning and spacecraft Launch and Range Operations. Further, details of a single vehicle's avionics systems now present a complex analysis task of both aircraft dynamics and discrete transitions in mechanical, electronic, and software on-board systems.

Several open issues remain with hybrid-system simulation. Some can be addressed by a software architecture specifically designed to allow this purpose. This paper suggested that such an architecture should place few restrictions on the types of models allowed, so that it can be used for a variety of purposes and with components of varying fidelity and resolution.

The behavior and performance metrics of hybrid systems both rely on interactions between individual components. As such, a simulation architecture also needs to accurately capture and/or create these interactions.

Methods of making the simulation as computationally efficient as possible are important. Rather than reducing the need for computational efficiency, recent improvements in computational power have, for the first time, allowed the research community to hope that very large, very complex systems can be simulated. As these simulations become more widely used, there may be increasing demand for more fidelity, more accuracy, and for more simulation runs in an analysis for wider or more statistically verifiable results.

This paper discussed the use of timing the updates of individual objects within a large-scale simulation. We considered two specific mechanisms: variable time steps,

and asynchronous simulation with occasional synchronization to capture measurements and interactions.

As a test case, we discussed a simulation architecture which met the requirements and mechanisms described in the paper. This simulation architecture uses an object-oriented framework to accept objects of a wide variety of types, easily incorporating both continuous-time and discrete-event models. This simulation architecture was used to simulate the dynamics of the NAS. Even the simple methods of improved simulation timing were found to have significant benefits in this application.

ACKNOWLEDGMENTS

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. We also thank the people who have contributed to the development of the simulation, including Serhan Ziya, Corey Ippolito, and Ted Chen.

REFERENCES

Beltrami, E. 1987. *Mathematics for Dynamic Modeling*. Boston: Academic Press.

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, Louisiana, 376–386.

Cellier, F. E. 1986. Combined continuous/discrete simulation applications, techniques, and tools. In *Proceedings of the 1986 Winter Simulation Conference*, ed. J. R. Wilson, J. O. Henriksen, and S. D. Roberts. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

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

Fahrland, D. A. 1970. Combined discrete event continuous systems simulation. *Simulation*, 61–72.

Friedman, L. W. 1996. *The Simulation Meta-Model*. Norwell, Massachusetts: Kluwer.

Hanke, C. R. 1971. The simulation of a large jet transport, NASA Contractor Report CR-1756, Volumes 1 and 2.

Jefferson, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7(3):404–425.

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

Johnson, E. N. and R. J. Hansman. 1994. Multi-agent flight simulation with robust situation generation, MIT Aeronautical Systems Laboratory Report ASL-95-2, MIT, Cambridge, Massachusetts.

Kettenis, D. L. 1997. An algorithm for parallel combined continuous and discrete-event simulation. *Simulation Practice and Theory* 5:167–184.

Kheir, N. A. 1996. Continuous-time and discrete-time systems, In *Systems Modeling and Computer Simulation*, ed. N. A. Kheir. New York: Marcel Dekker.

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

Mirtich, B. 2000. Timewarp rigid body simulation, *SIGGRAPH 00.*

National Research Council. 1997. *Flight to the Future: Human Factors in Air Traffic Control,* ed. C. D. Wickens, A. S. Mavor, and J. P. McGee, Washington, DC: National Academic 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, Cambridge, Massachusetts.

Press, W. H., B. P. Flannery, S. Teukolsky, and W. T. Vetterling. 1992. *Numerical Recipes in C – The Art of Scientific Computing*, 2d Ed. New York: Cambridge University Press.

Pritchett, A. R. and C. Ippolito. 2000. Software architecture for a reconfigurable flight simulator, *AIAA Modeling and Simulation Technologies Conference* Denver, Colorado.

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

Stevens, B. and F. Lewis. 1992. *Aircraft Control and Simulation*. New York: John Wiley.

Wieting, R. 1996. Hybrid high-level nets. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 848–855. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

AUTHOR BIOGRAPHIES

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>.

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_lee@hotmail.com> and <www.isye.gatech.edu/~seungman>.

DAVID HUANG is a graduate student in the School of Industrial and Systems Engineering at the Georgia Institute of Technology. He received a B.S. (with a triple major) from Duke University and an M.S. from Georgia Tech. His e-mail address is <david.huang@alumni.duke.edu>.

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 was the Program Chair for the 1995 Winter Simulation Conference. His e-mail and web addresses are <sman@isye.gatech.edu> and <www.isye.gatech.edu/~sman>.