JOURNAL OF AIRCRAFT
Vol. 38, No. 5, September–October 2001

Hybrid-System Simulation for National Airspace System Safety Analysis

A. R. Pritchett,* S. M. Lee,† and D. Goldsman‡
Georgia Institute of Technology, Atlanta, Georgia 30332-0205

Analysis of large, complex systems requires simulations of hybrid-system dynamics (i.e., dynamics best described by a combination of continuous-time and discrete-event models) and their interactions. To serve as valuable research tools, such simulations must also be computationally efficient, readily modifiable, capable of simulating systems using models of a wide range of fidelity, and easily reconfigurable to simulate parts or all of the system of interest. The development of a simulation architecture meeting these criteria is described. Issues with its development are described conceptually, and its application to 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. New asynchronous with resynchronization methods of timing individual objects are applied in an example, demonstrating a significant improvement in simulation efficiency.

Introduction

MANY large, complex systems are hybrid in nature (i.e., they need both continuous-time and discrete-event models to describe their behavior), and these models are not separable, but instead must 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 purposely designed to have components spaced far apart until simulations capable of predicting electromagnetic interference could be used to analyze and redesign smaller chips.[1] Many aerospace systems also 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 example, 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 inserting aircraft into the airspace at random times or for the stochastic injection of disturbances. Elements of NAS dynamics have most often been simulated using purely discrete-event simulations such as SIMMOD[2]; however, such models do not have the resolution to capture a range of safety issues. Similiarly, hybrid-system simulations have been made of the NAS such as TAAM and HERMES, but these are typically limited to specific applications, specific model resolutions, or isolated parts of the NAS.[2,3]

To serve as an effective design tool, simulation of large-scale systems must also meet a number of practical requirements. First, the simulation should be rapidly reconfigurable. As a practical matter, this eases the cost of developing a simulation; rapid reconfigurability also allows for the simulation to be applied to a range of applications and to accommodate models of varying form and fidelity as needed for the task at hand. In addition, the simulation should be sufficiently computationally efficient that it can provide a time-effective analysis tool, even when large numbers of runs are required.

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

First, a comparison is made of the fundamental differences and similarities between continuous-time and discrete-event models. Then the test case used in this paper, the safety analysis of the NAS, is described. Conceptual issues in, and requirements of, hybrid-system simulations are discussed, and then a simulation software architecture is presented. Methods of controlling simulator timing are described, illustrated by comparisons from the test case and a specific demonstration from a simulation of a standard terminal arrival route (STAR).

Background: Discrete-Event and Continuous-Time Models

Important, fundamental differences exist between discrete-event and continuous-time models, as 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. Therefore, they can describe the system without attempting to capture internal dynamics. Their defining parameters stipulate how and when states will transition from one to another. These parameters are typically set by experimental observation of existing systems to capture the stochastic nature of the system. As such, discrete events are well suited for modeling systems made of multiple entities with no internal dynamics of relevance to the type of analysis being conducted, and discrete-event models usually require study of an established system to ascertain their parameters.

Continuous-time models, on the other hand, usually attempt to model the internal dynamics of system components. System state is typically defined as a measure of the magnitude and distribution of energy within the system, such as vector-valued measures of position and velocity. These models are often physics-based, that is, they can be developed before the system is built and can be measured. However, these models are usually described by differential equations, which can be computationally expensive to propagate forward through time. (A related model type, that of discrete-time systems, is modeled using difference equations[4] and can be treated as a special class of continuous-time models.)

These two types of models, applied to the same system, 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 the dynamics of interest

Received 21 December 2000; revision received 11 April 2001; accepted for publication 12 April 2001. Copyright © 2001 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*Assistant Professor, School of Industrial and Systems Engineering and Aerospace Engineering. Member AIAA.

†Graduate Research Assistant, School of Industrial and Systems Engineering.

‡Professor, School of Industrial and Systems Engineering.

Table 1 Comparison of continuous-time and discrete-event models

Attribute	Continuous-time models	Discrete-event models
System being simulated	Specific mechanical unit with complex functioning	Multiple, often simple, entities
Definition of system state	Continuous and vector-valued, for example, magnitude and distribution of energy within system	Discrete, for example, categorization of current system properties
Typical measures of system state	Position, attitude, and velocity (deterministic)	Queue size, incidence (statistical properties thereof)
Typical factor driving updates	Time	Transitions from state to state
Common purpose 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

occur to capture their basic properties[5]; in most applications, the time step is set much smaller than this to reduce error in the numerical solutions.[6] 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 differ 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 often results in a computationally inefficient simulation.[7,8] Thus, these differences are of dramatic import to the simulation designer because the simulation architecture is typically tailored to the type of model implicit in the simulation.

Test Case: Safety Analysis of the NAS

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. Furthermore, distinctions must often 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 a wide range of 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 and to changes in the environment. These interactions may meet a number of goals, ranging from time-critical collision avoidance maneuvers to strategic plans for air traffic flow management.

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

Changing such a large and complex system creates a design problem of vast scale. For both cost and safety reasons, changes should be analyzed thoroughly before implementation. The worth of this analysis will be measured by its ability to predict and correct problems before implementation. Simulation and modeling have been recognized as a critical part of this analysis.[9]

The NAS has been simulated before. However, most of these simulations have not been suitable for large-scale safety analysis. Instead, many NAS simulations have been motivated by studies of efficiency. Therefore, these simulations have been concerned with values such as aircraft interarrival times or flight delays. These concerns are best abstracted by macroscopic models and, hence, have usually been covered by discrete-event simulations or simulations with very simple models of aircraft behavior.[2,3]

Safety, on the other hand, is largely determined by continuously evolving interactions that macroscopic discrete-event simulations can not capture. For example, a macroscopic simulation may model when two aircraft arrive at an airport; without detailed modeling of their continuous trajectories, however, it is nearly impossible to ascertain accurately whether these two aircraft came unacceptably close during their flights, or to model the performance of humans such as pilots and controllers with suitable resolution. 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. Because they are most accurately 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.[10–12] 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 of NAS behavior remain best modeled as discrete events, for a variety of reasons. Some elements can be predicted to occur discretely; for example, the generation of a queue of aircraft waiting for taxi clearance to the takeoff runway is discrete in nature. Other elements of the NAS are not needed at a fine resolution, and 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 takeoff can be temporarily replaced by a notation in the takeoff queue. Finally, discrete events can be used to inject stochastic phenomena into the simulation, such as errors and failures, as well as disturbances into aircraft and human performance model parameters.

Measurements of NAS safety can also be treated as discrete events that occur when threshold 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 the computational structures that are used for other discrete events.

Of critical importance in NAS safety simulation is inclusion of human performance models because their behavior drives system dynamics. Their behavior should not be typecast into either continuous or discrete forms; in addition, many established models would be difficult to convert to fit within any one rigidly defined simulation architecture. 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.[13]

Therefore, a simulation architecture capable of accommodating the many different types of models can provide a valuable research tool. For such a tool to also be cost effective, it must also be rapidly reconfigurable. This reconfigurability can allow for the same architecture and models to be used in simulations of different elements of the NAS and in simulations with different purposes, for example, capacity studies vs detailed safety analysis. Thus, a corresponding benefit of reconfigurability can be a dramatic reduction in the cost of developing a simulation through the reuse of models and computational structures.

Hybrid-System Simulation Concepts

In cases where an elegant, analytic solution can not be found to analyze a system represented by purely discrete-event or continuous-time models, a hybrid-system simulation is required to calculate system dynamics through time. Several approaches have been suggested for the design of hybrid-system simulations. Some create larger metamodel frameworks in which each type of simulation functions separately.[1,14] Other solutions have adapted existing simulations of one type to include the other.[15,16]

A third approach, sometimes called the fully integrated approach,[1] seeks to create new software that inherently accepts the two model types. Such approaches have been undertaken with special modeling languages.[8,17–20] This 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.

Earlier sections have highlighted the differences between the models used in hybrid-system simulation. A simulation architecture can capitalize on the 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 as such, it allows for the simulation to include components of various resolutions as required by the task at hand.[21,22]

Without placing restrictions on components' models, the simulation architecture also needs to support their interactions. These interactions may take 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) and, therefore, 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). Conversely, continuous-time objects can interact with discrete events when their values correspond to the events' triggers.

Within a simulation framework that supports such interactions, the simulation designer is able to make components work efficiently on their own. For example, in modeling an aircraft with onboard 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 timescales. Instead, the designer has the option of making the onboard 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 reworkings within individual components.[17]

Although 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. Although this process requires measurements to have a predictive power, it also reduces the need for unnecessary measurements. This also mirrors a duality between measurements and conditionally based discrete events; whereas the former have no lasting impact on simulation dynamics, the latter are measurements with a consequence.

Based on this discussion, several requirements for a hybrid-system simulation architecture may be summarized. 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 on 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.

Simulation Architecture

The preceding sections discussed conceptual issues with hybrid-system simulation. This section discusses a particular simulator architecture design. This architecture extends the reconfigurable flight simulator software, which was originally designed for more traditional applications of flight simulation.[23]

This architecture allows for the inclusion of several broad classes of objects, as shown schematically in Fig. 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. Several vehicle components, including six-degree-of-freedom 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, event, and measurement (CEM) 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 interarrival 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 human performance models, discrete-event generation, 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.

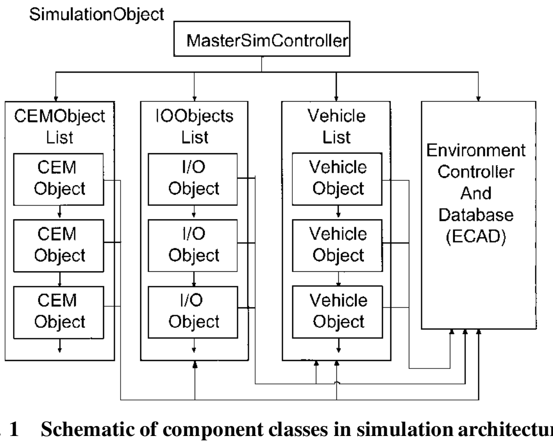

Fig. 1 Schematic of component classes in simulation architecture.

Input–output (I/O) 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 provides a common simulation environment for all of the components by establishing common axis definitions and conversions and by allowing for the inclusion of atmospheric and terrain models as desired. Finally, a networking object can be included to handle 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 recompilation of components to use or access networking functions.

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. Specifically, each of these components must update its state on command, report the time of its next update (or, in the case of some measurement and discrete-event objects, the next time at which an update might occur conditionally on other events), and identify whether its own update requires any other objects to also update.

For continuous-time objects, a required input is an upper bound on the numerical error allowable in each time step. With this input, the next update time is calculated based on knowledge of the interior dynamics; algorithms for such adaptive time step calculations are well established.[6]

Communications between objects are handled by the high-level simulation object. Therefore, 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 protocol, which requires sending a message to the simulator object along with a request for its destination.[23] This mechanism also allows for objects to request the creation 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.

This architecture, therefore, provides an overall framework that can simulate any NAS configuration at any level of fidelity. It is configured by incorporating new components during run time as specified by initialization commands and operator input. Thus, development of a new configuration does not require a new or different simulation architecture; instead, the development of new initialization scripts describes the simulator configuration, and the inclusion of new or modified components can extend the models used in the simulation.

Efficiency and Timing Methods

In large-scale simulation, concerns with computational efficiency extend beyond making 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 are generally nonexact once the simulation is run in a configuration containing stochastic elements and/or is nontrivial in size. Likewise, the amount of computation required by the most sophisticated timing methods may be significant and can slow down the simulation.

Even measuring the computational efficiency of a simulation can be nontrivial. Once the simulation includes stochastic elements, it can be difficult to compare with certainty the relative speed of different update timing methods without a large number of runs because the simulation runs will have different system dynamics due to the inclusion of random disturbances or anomalies. Likewise, measures of simulation efficiency based on run time are highly sensitive not only to the specific scenario under test and basic processor speed, but also to many other aspects of the simulation hardware such as data-access speed, memory size, etc.

This section will compare different timing methods and illustrate their effect on a representative simulation using the architecture described in the preceding section. Then trade-offs in the fundamental characteristics of these methods are discussed, and alternative methods are commented on.

Timing Methods

Two major factors define the variety of methods for determining the timing of simulation components: the first is selection of how the time step is set (next-event time advance, or fixed-increment time advance)[24]; the second is selection as to whether the simulation will be entirely synchronous, that is, all components update at the same time; partially synchronous and partially asynchronous; or entirely asynchronous, that is, all components update individually.

Several timing methods can be defined by these two factors, as follows.

Synchronous Fixed Time Step

This timing method updates all objects at the same time, with the time step externally fixed through the simulation. 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 a conservative, worst-case estimate of the dynamics of the component with the fastest response, which is computationally inefficient.

Synchronous Variable Time Step

This timing method has all of 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) time step. This method still forces some objects to update more often than they would require when running alone, but it can relax the time step when conditions allow.

Asynchronous with Resynchronization

This timing method allows for components to be updated independently following their own update times. This is shown schematically in Fig. 2 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 allows 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 that interact, or that measure interactions, to require all objects (or specific objects) to resynchronize when it is time for their update, with the result that interactions and measurements can be based on values from temporally collocated objects.

This timing method requires a state-updater object to maintain a list of the objects within the simulation, sorted by the time of

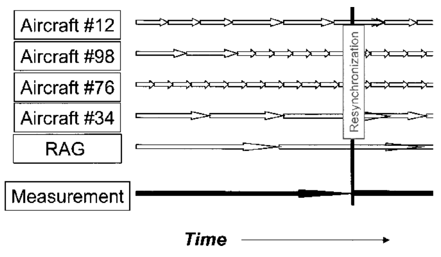

Fig. 2 Schematic of asynchronous simulation with resynchronization.

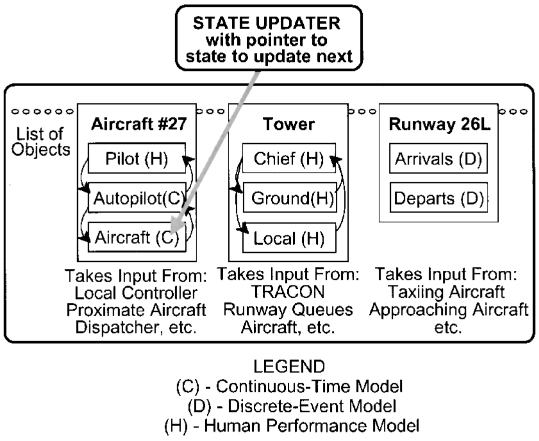
STATE UPDATER
with pointer to
state to update next
List of Objects
Aircraft #27
Pilot (H)
Autopilot(C)
Aircraft (C)
Tower
Chief (H)
Ground(H)
Local (H)
Runway 26L
Arrivals (D)
Departs (D)
Takes Input From: Local Controller Proximate Aircraft Dispatcher, etc.
Takes Input From: TRACON Runway Queues Aircraft, etc.
Takes Input From: Taxiing Aircraft Approaching Aircraft etc.
LEGEND
(C) - Continuous-Time Model
(D) - Discrete-Event Model
(H) - Human Performance Model

Fig. 3 Schematic of sorting of simulation components by update time, with access by state-updater component.

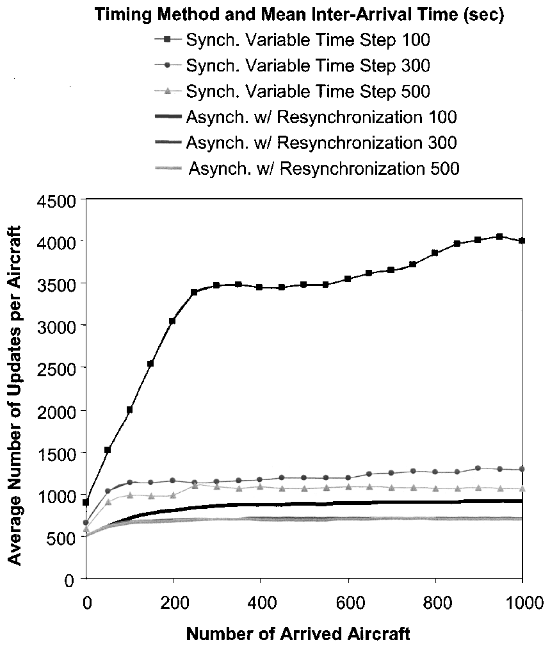

Fig. 4 Average number of calls to aircraft objects during simulation of a STAR comparing two timing methods, with varied mean aircraft interarrival times.

their next desired update. This object is shown schematically in Fig. 3. This state-updater object then controls the simulation timing by identifying the next object to be updated, regardless of type, querying that object as to whether it requires other objects to also be updated, and commanding the appropriate objects to update. Once objects have been updated, they each are asked for their new update time and are sorted accordingly.

Example: Simulation of a STAR

To compare the computational efficiency of these methods, a numerical simulation was conducted using the architecture described earlier. The simulation modeled the stream of arriving aircraft flying the Macey Two STAR into Atlanta–Hartsfield airport. Aircraft were injected into the simulation stochastically with a specified interarrival 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 would create this desired traffic pattern. The aircraft were removed from the simulation when they reached the runway.

The results of this simulation are shown in Fig. 4. Efficiency in this case is measured by the average number of times aircraft objects are called to update during the course of the run due to both their own desired rate as well as forced resynchronizations. Aircraft are the most computationally intensive objects in the simulation; therefore, a lower number of calls, on average, to individual aircraft represents greater computational efficiency.

The data are shown for two timing methods. The synchronous variable time step method required all objects in the simulation to update at the same time, using the worst-case time step identified from all of the objects. In the asynchronous with resynchronization method, the controller and measurement objects were allowed to command a complete resynchronization at times when they predicted a conflict might occur or the next command might be warranted. The aircraft mean interarrival time into the arrival route was also varied. The highest mean interarrival time (500 s) created a fairly low traffic density, with commensurately few interactions. At this mean interarrival time, the benefits of asynchronous simulation are noticeable, but not dramatic.

The lowest mean interarrival time (100 s) created a high traffic density, in which controller commands and aircraft maneuvers in response to potential conflicts were often required. In the synchronous variable time step method, this had the effect of requiring many more updates for all aircraft on average. In the asynchronous with resynchronization method, fewer updates were required overall because those aircraft needing updates at small intervals were able to update independently.

Tradeoffs Between Resynchronization Intervals and Efficiency

In applications such as just shown in the case study, there appear to be benefits to asynchronous timing methods. At first glance, this appears to imply that the best efficiency will arise with the largest resynchronization intervals, which allows the objects to run asynchronously for significant portions of the simulation. 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 be warranted. Better predictions require more extensive calculations; at an extreme, the predictor would need to internally simulate other objects to predict accurately 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 used to set resynchronization intervals offset any savings in computations by the objects that are affected.

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 aircraft 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, limits the use of the predictors to specific cases, and reduces the ease with which the simulation can be reconfigured.

Alternatives to Resynchronization

Thus far, this discussion on simulation timing has assumed that accurate measurements and interactions can only occur when the objects involved are temporally collocated, 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 required when predicted interactions require other objects to jointly manifest a new behavior at a certain time, for example, 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 missed measurements.

Similiarly, it has been 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 backward to a potential missed interaction are possible, with the benefit of relaxing timing intervals.[25–27] However, these rollback or timewarp simulations can fit better in some domains than others; some types of models are easier to either run backward 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 cost and, hence, should be used wisely.

Conclusion

This paper has discussed issues relating 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 a test case throughout the paper, with a specific simulation configuration detailed as an example. 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. Likewise, detailed analysis of a single vehicle's avionics systems requires simulating both aircraft dynamics and discrete transitions in mechanical, electronic, and software onboard systems.

Several open issues remain with hybrid-system simulation. Some can be addressed by software architectures. 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; this also has the practical benefit that existing models can be used without substantial restructuring. The behavior and performance metrics of hybrid systems both rely on interactions between individual components; as such, a simulation architecture also needs to capture accurately 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 in detail. 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 seeking statistically verifiable results.

Methods of timing object updates within a large-scale, hybrid-system simulation have been identified as a research topic requiring investigation.[19] This paper discussed timing the updates of individual objects within a large-scale simulation. Two specific mechanisms were discussed: variable time steps and asynchronous simulation with occasional resynchronization to capture measurements and interactions. A simulation architecture was described that met the requirements and mechanisms discussed 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. As an example, this simulation architecture was used to simulate the dynamics of a STAR; methods of improved simulation timing were found to increase significantly computational efficiency of the simulation as a whole.

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. The authors would also like to thank the people who have contributed to the development of the simulation, including Serhan Ziya, Corey Ippolito, David Huang, and Ted Chen. Portions of this paper were presented at the 2000 AIAA Modeling and Simulation Technologies Conference and the 2000 Winter Simulation Conference.

References

[1]Saleh, R., Jou, S. J., and Newton, A. R., *Mixed-Mode Simulation and Analog Multilevel Simulation*, Kluwer, Boston, 1994, pp. 1–3, 23–25.

[2]Odoni, A. R., Bowman, J., Delahaye, D., Deyst, J. J., Feron, E., Hansman, R. J., Khan, K., Kuchar, J. K., Pujet, N., and Simpson, R. W., "Existing and Required Modeling Capabilities for Evaluating ATM Systems and Concepts," TR ICAT 98-2, Massachusetts Inst. of Technology International Center for Air Transportation, Cambridge, MA, 1997.

[3]Jim, H. K., and Chang, Z. Y., "An Airport Passenger Terminal Simulator: A Planning and Design Tool," *Simulation Practice and Theory*, Vol. 6, No. 4, 1998, pp. 387–396.

[4]Kheir, N. A., "Continuous-Time and Discrete-Time Systems," *Systems Modeling and Computer Simulation*, edited by N. A. Kheir, Marcel Dekker, New York, 1996, pp. 27–88.

[5]Beltrami, E., *Mathematics for Dynamic Modeling*, Academic International Press, Boston, 1987, pp. 66–68.

[6]Press, W. H., Flannery, B. P., Teukolsky, S., and Vetterling, W. T., *Numerical Recipes in C— The Art of Scientific Computing*, 2nd ed., Cambridge Univ. Press, New York, 1992, pp. 707–709, 714–723.

[7]Fahrland, D. A., "Combined Discrete Event Continuous Systems Simulation," *Simulation*, Vol. 14, No. 2, 1970, pp. 61–72.

[8]Fishwick, P. A., "A Taxonomy for Simulation Modeling Based on Programming Language Principles," *IIE Transactions*, Vol. 30, No. 9, 1998, pp. 811–820.

[9]Wickens, C. D., Mavor, A. S., and McGee, J. P. (eds.), *Flight to the Future: Human Factors in Air Traffic Control*, National Academic Press, Washington, DC, 1997, pp. 210–214.

[10]Hanke, C. R., "The Simulation of a Large Jet Transport," NASA CR-1756, 1971.

[11]Johnson, E. N., and Hansman, R. J., "Multi-Agent Flight Simulation with Robust Situation Generation," Massachusetts Inst. of Technology Aeronautical Systems Lab. Rep. ASL-95-2, Cambridge, MA, 1994.

[12]Stevens, B., and Lewis, F., *Aircraft Control and Simulation*, Wiley, New York, 1992, pp. 51–109.

[13]Corker, K. M., and Pisanich, G., "Cognitive Performance for Multiple Operators in Complex Dynamic Airspace Systems: Computational Representation and Empirical Analyses," *Proceedings of the 1998 42nd Annual Meeting Human Factors and Ergonomics Society*, Vol. 1., Human Factors and Ergonomics Society, Santa Monica, CA, 1998, pp. 341–345.

[14]Friedman, L. W., *The Simulation Meta-Model*, Kluwer, Norwell, MA, 1996.

[15]Wieting, R., "Hybrid High-Level Nets," *Proceedings of the 1996 Winter Simulation Conference*, IEEE Publ., Piscataway, NJ, 1996, pp. 848–855.

[16]Cellier, F. E., "Combined Continuous/Discrete Simulation Applications, Techniques, and Tools," *Proceedings of the 1986 Winter Simulation Conference*, IEEE Publ., Piscataway, NJ, 1986, pp. 24–33.

[17]Kettenis, D. L., "An Algorithm for Parallel Combined Continuous and Discrete-Event Simulation," *Simulation Practice and Theory*, Vol. 5, No. 2, 1997, pp. 167–184.

[18]Cubert, R. M., and Fishwick, P. A., "Modeling the Simulation Execution Process with Digital Objects," *Proceedings of SPIE Conference on Enabling Technology for Simulation Science III*, Vol. 3696, Society of Photo-Optical Instrumentation Engineers, Bellingham, WA, 1999, pp. 2–22.

[19]Koo, T. K. J., Ma, Y., Pappas, G. J., and Tomlin, C., "SmartATMS: A Simulator for Air Traffic Management Systems," *Proceedings of the Winter Simulation Conference*, IEEE Publ., Piscataway, NJ, 1997, pp. 1199–1205.

[20]Liu, J., Liu, X., Koo, T. K. J., Sinopoli, B., Sastry, S., and Lee, E. A., "A Hierarchical Hybrid System Model and Its Simulation," *Proceedings of the IEEE Conference on Decision Control*, Vol. 4, IEEE Publ., Piscataway, NJ, 1999, pp. 3508–3513.

[21]Bezdek, W. J., Halley, T. A., and Hummel, P. C., "Model Reuse for Software Development and Testing: The Application of Common Interfaces to Support Variable Fidelity Models," AIAA Paper 97-3799, Aug. 1997.

[22]Davis, P. K., and Bigelow, J. H., "Experiments in Multiresolution Modeling (MRM)," Rept. MR-1004-DARPA, Rand Corp., Santa Monica, CA, 1998.

[23]Pritchett, A. R., and Ippolito, C., "Software Architecture for a Reconfigurable Flight Simulator," AIAA Paper 2000-4501, Aug. 2000.

[24]Law, A. M., and Kelton, W. D., *Simulation Modeling and Analysis*, 2nd ed., McGraw–Hill, New York, 1991, pp. 8–10.

[25]Mirtich, B., "Timewarp Rigid Body Simulation," *SIGGRAPH 00*, Association for Computing Machinery, New York, 2000.

[26]Jefferson, D. R., "Virtual Time," *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, 1985, pp. 404–425.

[27]Carothers, C. D., Perumall, K. S., and Fujimoto, R. M., "Efficient Optimistic Parallel Simulations Using Reverse Computation," *ACM Transactions on Modeling and Computer Simulation*, Vol. 9, No. 3, 2000, pp. 224–253.