Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

SIMULATING AIRCRAFT DELAY ABSORPTION

Justin Boesel

The MITRE Corporation
Center for Advanced Aviation Systems Development
7515 Colshire Drive
Mail Stop N370
McLean, VA, 22102, U.S.A.

ABSTRACT

An airplane's ability to absorb delay while airborne is limited and costly. Because of this, the air traffic control system anticipates and manages excessive demand for scarce shared resources, such as arrival runways or busy airspace, so that the delay necessary for buffering can be spread out over a larger distance, or taken on the ground before departure. It is difficult to model these important dynamics in a standard queue-resource simulation framework, which does not account for limited delay absorption capacity. The modeling methodology presented here captures these dynamics by employing a large number of independent threads of execution to monitor and enforce a large number of relatively simple mathematical relationships. These relationships calculate feasible time windows for each portion of each flight. The model was implemented in the SLX simulation language. The speed and scalability of SLX are essential to the approach, which would otherwise be impractical.

1 INTRODUCTION

From a capacity modeling perspective, airplane traffic is fundamentally different from automobile, rail or ship traffic; while a car, train, or ship can stop and wait for an essentially unlimited amount of time in the middle of its journey, an airplane cannot. In other words, once airborne, an airplane's ability to absorb delay is limited and costly, complicating the problem of modeling delays and capacities.

Because of this limited delay absorption capacity, the air traffic control system anticipates and manages excessive demand for scarce shared resources (e.g., arrival runways, busy terminal airspace) so that the delay necessary for buffering can be spread out over a larger distance, or taken on the ground before departure. These actions, however, can ripple back and block resources upstream, such as departure runways and controller attention in busy sectors.

It is difficult to model these dynamics in a standard queue-resource simulation framework. In a standard queue-resource model, there is no concept of limited delay absorption capacity. For instance, in a factory setting, a part moving from one work station to another may wait for one minute or one week before receiving service. The *number* of parts waiting for service (queue size) may be explicitly limited, but the wait time per part is not.

To model airplane traffic, one needs to be able to anticipate excessive demand for a resource well *before* it occurs, so that the flight to be delayed has adequate distance over which to absorb the required delay.

This article describes a modeling methodology for simulating the limited delay absorption capacity of airborne flights. The remainder of this paper is organized as follows: section 2 describes how aircraft are delayed for buffering; section 3 explains why it is important to capture the dynamics caused by limited buffering capacity; section 4 describes the network and object elements of the model; section 5 describes the central "monitor and enforce" mechanism; section 6 provides an example modeling two merging aircraft; section 7 looks at the problem from an information flow perspective; section 8 describes the elements of the SLX simulation language that are central to this approach; and section 9 summarizes and draws some conclusions.

2 DELAY ABSORPTION BUFFERING

In almost any capacity-constrained system, the ability to buffer demand during busy periods is key to increasing utilization of scarce server resources. For instance, during lunchtime at a fast food restaurant, customers wait in line while the cashier/server takes orders from other customers. When one customer is done, the next in line receives service, and the server remains busy. As wait time increases, a customer can choose to remain in line or can opt out, and leave the restaurant. In the air traffic control system, some similar situations exist: departing airplanes wait on taxiways for their turn on a runway. Once aircraft are in the

air, however, the situation changes. An aircraft cannot opt out of landing and, because of fuel constraints, it cannot wait for an arrival runway indefinitely. Within these constraints, buffering airborne flights–making them wait in the air–even for relatively short periods of time is costly in a number of ways.

- Buffering requires controller work. Overloading a controller is undesirable because it can compromise safety.
- Buffering usually increases mileage, which burns fuel and increases aircraft wear and tear.
- Buffering requires airspace.

Air traffic controllers have four basic methods in which they can delay aircraft to keep them from overwhelming a resource such as a runway or a controller downstream. These methods, and their relative costs and benefits, are described below:

- *Ground Delay.* Delaying a flight on the ground before it departs is relatively cheap in terms of fuel and controller workload, even though it is not free to the airlines, their passengers or cargo. Ground delay can absorb practically unlimited amounts of time. Ground delay is often used to prevent congestion in the air, which, if left unchecked, could overburden controllers and compromise safety. Because of departure runway congestion and flight-time variability, however, it is not practical to use ground delay to fine-tune a flight's arrival time at a distant airport.
- *Airborne Holding.* Placing a flight into airborne holding is expensive in terms of fuel and controller workload. Furthermore, holding requires reserved airspace; the locations at which airborne holding can take place are limited. Holding is most commonly used to delay arriving flights close to (within 50 miles of) their destination airport. Despite the costs, holding allows controllers to delay airborne flights for relatively large amounts of time (tens of minutes). At large airports with relatively unconstrained airspace, such as Atlanta Hartsfield, controllers use the buffering capacity provided by airborne holding to make more efficient use of arrival runways (Voss and Hoffman, 2001).
- *Vectoring.* Vectoring means extending a flight's path (thereby delaying it) by turning it. Vectoring allows controllers to delay aircraft more precisely and with less expense (in terms of workload and fuel cost) than they could with airborne holding. The amount of delay that can be achieved with vectoring is closely related to the amount of airspace a controller can use. Vectoring is a very common technique, especially for sequencing flights onto an arrival runway.
- *Speed Control.* Slowing a flight down to delay it requires little airspace, but the amount of delay that can be absorbed with speed control is not great.

3 WHY MODEL LIMITED DELAY ABSORPTION?

Because airborne flights can absorb only a limited amount of delay, buffering caused by contention for a downstream resource, such as an arrival runway, can quickly ripple back upstream and cause congestion in an upstream resource, such as an en route sector. The subsequent congestion upstream can delay departures from and arrivals to other airports.

If a model fails to capture the limits on delay absorption, it will miss these blocking effects upstream. This makes it important to model aircraft taking delay not only in the correct amount, but also at the correct place and time.

Figure 1 illustrates this problem. Suppose airport D sends flights to airport A, and airport C send flights to airport B, all via en-route sector Y. If runway congestion at airport A delays arrivals, they may spend more time in sector Y, especially if the delay absorption capacity between Y and A is small. If this causes sector Y to become too busy, departures from C may be held on the ground. A model that accounts for limited delay absorption would capture this dynamic.

On the other hand, in a model that overstates the delay absorption capacity of flights between sector Y and airport A, the arrivals to A will quickly pass through sector Y, which will not become too busy, thus allowing departures from C to D to proceed undelayed. This model will understate delay.

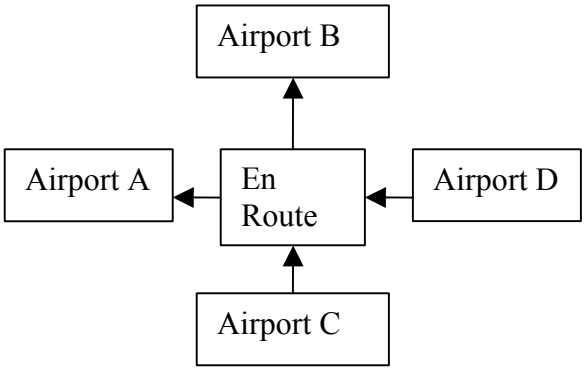

Figure 1: En Route Sector as a Constraint

4 ELEMENTS OF THE MODEL

In the model presented here, aircraft move along a link-node structure. Each flight requires a minimum time to traverse each link, and each flight can absorb only a limited amount of additional delay on each link. To represent an airspace that has plenty of room for vectoring, this

maximum delay parameter can be set high, while a narrower, more constrained airspace would have a lower maximum delay parameter.

As flights merge on to common links or cross each other's paths, minimum separation between aircraft is maintained. Each link and node in the model is assigned a minimum required separation (in minutes) that defines its capacity. The model presented here deals only with inter-aircraft separation, and does not explicitly model controller workload as a constraint.

The model is essentially a network model with nodes, links, and flights moving and absorbing delay on links. Unlike most network models, however, this model anticipates contention for resources long before it occurs and spreads the required delay absorption out across the links, rather than in a buffer immediately in front of the constrained resource. Four object types define the model:

- *Flight Object.* Each Flight object represents a single flight. It has the flight's aircraft ID, aircraft type, desired departure time, and a flight plan, defined as a list of Links.
- *Link Object* Flights use Links to get from one place to another. A Link object can be used to represent an airway at a particular attitude. Links are defined to be one-way only, and while a Link can be shared by several Flights, passing is not permitted on a Link. Each Link has pointers to its starting and ending Nodes, pointers to all of the Flights that will pass over it, and minimum required separations (in minutes) that define its capacity.
- *Node Object.* A Node Object is used to connect Links. A Node object, which can be thought of as a point in 3-D space, can be used to represent a waypoint or a fix at a particular altitude. Nodes represent crossings, merges, and split relationships between Link Objects. Each Node has a list of the Links coming into and out of it. Like a Link, a Node has several minimum required separations that define its capacity.
- *Flight-By-Link Object.* As each Flight crosses each Link on its flight path, the model generates a great deal of timing information. Flight-by-Link objects keep track of all of this information. A Flight-by-Link object represents a particular Flight on a particular Link. For example, if a Flight has *n* Links on its path, then *n* Flight-By-Link objects will be created for that Flight.

The Flight-By-Link object is the workhorse of the simulation. It has pointers to three other Flight-by-Link objects, which define the object's relationship with the rest of the model. One pointer refers to the Flight immediately ahead of it on the same Link, and the other two pointers refer to the same Flight on the next and previous Links. Figures 2 and 3 illustrate these relationships.

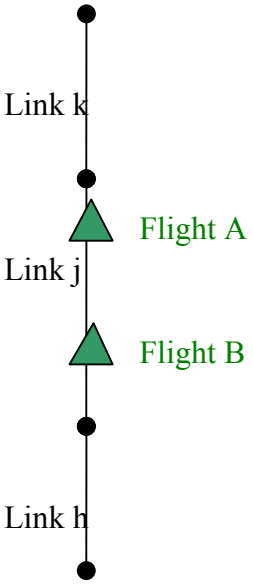

Figure 2: Flights A and B on Links h, j, and k

Suppose Flight B follows Flight A across Links h, j, and k, as shown in Figure 2, above. To represent this in the model, one would need Flight objects for A and B, and Link objects for h, j, and k. To represent the flights' movement over these Links, one would need to create six Flight-By-Link objects, A_h, A_j, A_k, B_h, B_j, and B_k. Figure 3 illustrates the pointer relationships of Flight-By-Link B_j to its "adjacent" Flight-By-Link objects B_k (same Flight, next Link), A_j (leading Flight, same Link), and B_h (same Flight, previous Link).

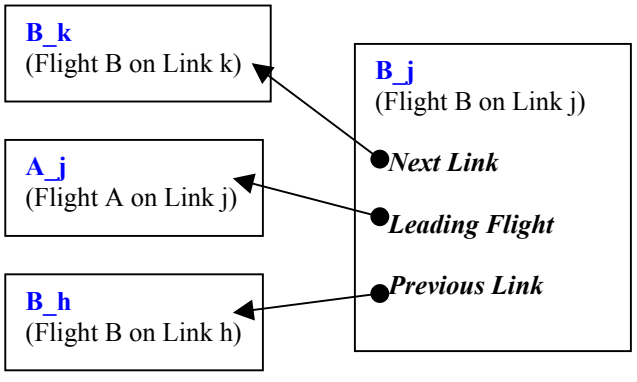

Figure 3: Flight-by-Link Object B_j's Pointers to Adjacent Flight-by-Link Objects

Each Flight-by-Link object has two quantities -- minimum traversal time and maximum delay -- that determine the minimum and maximum amounts of time the Flight can spend on the Link. The minimum traversal time represents the amount of time a Flight needs to cross a Link, and the maximum delay represents the amount of additional time a Flight could absorb on a particular Link.

5 MONITOR AND ENFORCE

In the model presented here, each object monitors and enforces a number of mathematical relationships. The following example illustrates such a relationship, and describes its enforcement.

Suppose the two Flights depicted in Figure 2, above, need to increase separation before entering Link k. Let T_A and T_B be the clock times at which Flights A and B (respectively) enter Link k, and let S_k be the minimum required separation (in minutes) between Flights entering Link k. To respect this separation, one must ensure that:

$$T_B \geq T_A + S_k. \quad (1)$$

Flight-By-Link object B_k can read all three quantities in (1): T_B is local to B_k, and the others can be read via pointers. B_k gets its own independent thread of execution to enforce the separation requirement expressed in (1), illustrated by the following pseudo-code :

```
fork {// fork command creates new thread

    forever //Enters Loop
    {
    wait until (TB <  TA + Sk);
    //relationship is violated

    TB = TA + Sk;
    //Enforce relationship

    }//end forever loop
}//end fork
```

The new thread immediately enters a "forever" loop, the only purpose of which is to wait until inequality (1) is violated, and then to correct it. This basic mechanism monitors and enforces a mathematical relationship.

At its heart, the model is essentially a system of hundreds of thousands of such relationships, each one monitored and enforced by its own independent thread and a "wait until" statement.

6 MODELING MOVEMENT AND LIMITED DELAY ABSORPTION: AN EXAMPLE

The following example illustrates how the objects and the monitor-and-enforce mechanism work together to model delay pass back due to limited delay absorption.

Consider Figure 4. Suppose Flight C traverses the airspace represented by Links v, w, and z, and Flight D, which starts just a little bit later, traverses the airspace represented by Links x, y, and z. Suppose that the Flights will need to be separated by one minute when crossing their merge point, S.

Suppose further that each Flight can traverse each Link in a minimum of 2 minutes, but can only absorb 45 seconds (0.75 minutes) of delay on each Link. To respect the separation requirement at point S, one of the Flights will need to be delayed.

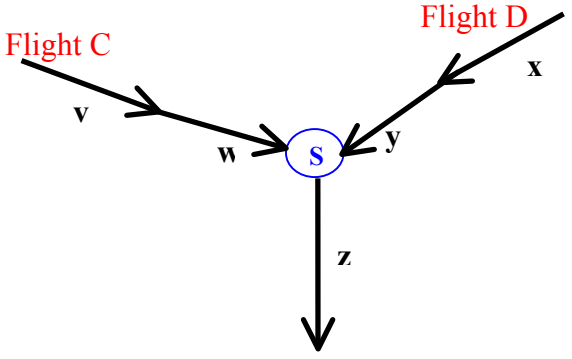

Figure 4: Merging Flights at Point S

To model this, one creates Flight objects representing C and D, Link objects representing v, w, x, y, and z, a Node object representing point S, Flight-By-Link objects C_v, C_w, and C_z, representing Flight C on Links v, w, and z, and Flight-By-Link objects D_x, D_y, and D_z representing Flight D on Links x, y, and z.

Each Flight-By-Link object has three pointers to other Flight-By-Link objects that are "adjacent" (in space or sequence):

- The ***Next Link*** pointer points to the same flight on the next Link in the flight plan. For instance, C_v has a pointer to ***Next Link*** C_w,
- The ***Previous Link*** pointer points to the same flight on the previous Link in the flight plan. For instance, C_w has a pointer to ***Previous Link*** C_v.
- The ***Leading Flight*** pointer points to the previous flight on the same Link. In this example, there is only one such meaningful instance of this; D_z has a pointer to ***Leading Flight*** C_z.

Each Flight-By-Link object has several attributes, specifically:

- *Minimum Traversal Time* -- 2 minutes for all Flight-By-Link objects in this example
- *Maximum Delay* -- 45 seconds for all Flight-By-Link objects in this example
- *Best Guess Time of Entry* -- the current best estimate of when (in simulation clock time) the Flight will enter the Link;
- *Best Guess Time of Exit* -- the current best estimate of when (in simulation clock time) the Flight will exit the Link.

To model simple movement across a single Link, there is a thread that monitors and enforces the relationship (M1):

My *Best Guess Time of Exit* $\geq$
My *Best Guess Time of Entry* +
My *Minimum Traversal Time*.

To model movement from Link to Link, there is a thread that monitors and enforces the relationship (M2):

My *Best Guess Time of Entry* ≥
***Previous Link*'s** *Best Guess Time of Exit*.

Each Link object has a constant *Minimum Separation Required at Entry*, which is used to separate merging and in-trail Flights. This is set to one minute on Link z. Each Flight-By-Link object compares its *Best Guess Time of Entry* to that of the Flight in front of it. The mechanism for doing this is a thread that monitors and enforces the relationship (M3):

My *Best Guess Time of Entry* ≥
***Leading Flight*'s** *Best Guess Time of Entry* +
Shared_Link's
Minimum Separation Required at Entry.

To pass back constraint information from Link to Link, there is a thread that monitors and enforces the relationship (M4):

My *Best Guess Time of Exit* ≥
Next Link's Best Guess Time of Entry.

To model the limited delay absorption capacity on each Link, there is a thread that monitors and enforces the relationship (M5):

My *Best Guess Time of Entry* ≥
My *Best Guess Time of Exit* –
(*Minimum Traversal Time* +
Maximum Delay).

Suppose Flight C starts at time 100.0 and Flight D starts at time 100.1. The threads enforcing relationships M1 and M2 will cascade forward through the system, setting *Best Guess Time of Entry* at 104 for Flight-By-Link object C_z (see Fig.5)

A similar chain of events takes place on objects D_x, D_y, and D_z, so that D_z's *Best Guess Time of Entry* will equal 104.1. But because the entry times are less than one minute apart, this causes a separation violation on entry to Link z. This violation is caught by M3, which will force D_z's *Best Guess Time of Entry* to 105, causing M4 to force D_y's *Best Guess Time of Exit* to 105. (See Fig.6)

Flight D needs to absorb a total delay of 0.9 minutes overall, but only 0.75 minutes can be absorbed on any single Link. M5 forces D_y's *Best Guess Time of Entry* down to 102.25. This means that 0.75 minutes of delay, the maximum, is absorbed on Link y. The remaining 0.15 minutes is pushed back to Link x by M4 which forces D_x's *Best Guess Time of Exit* to 102.25.

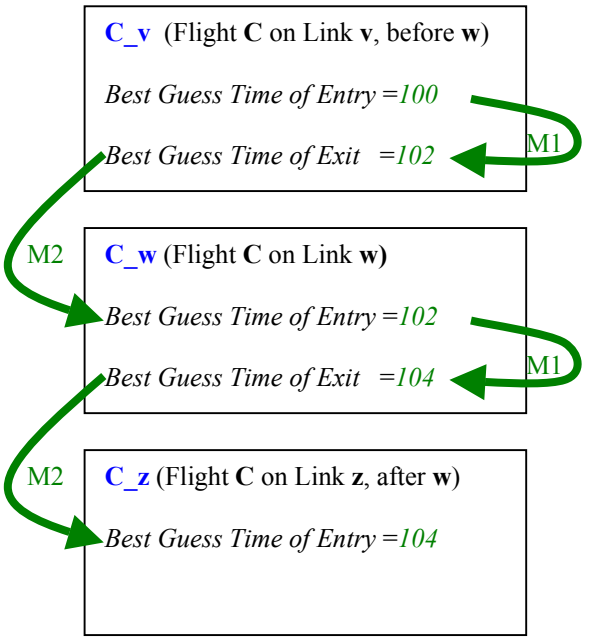

Figure 5: Sending Flight C's Timing Information Forward via Flight-By-Link Objects C_v, C_w, and C_z

At this point, the monitoring and enforcing stops, because D_x's *Best Guess Time of Exit* of 102.25 is well within bounds: M5 is unviolated because D_x's *Best Guess Time of Entry* is 100.1, its *Minimum Traversal Time* is 2, and its *Maximum Delay* is 45 seconds (0.75 minutes).

Note that these changes, which essentially put the system of equations back into balance, take place in zero simulated time. In this case, the changes should occur *before* the simulation clock reaches 100, the time at which the Flights start on their first Links. The simulation clock advances to the next scheduled "Link movement", that is, the next scheduled *Best Guess Time of Entry* among all of the Flight-By-Link objects defined in the model. Subsequent balancing actions occur between every advance of the simulation clock.

Because each Flight-By-Link object only has to directly account for the three "adjacent" Flight-By-Link objects, the model scales up well. Once one has set up the rules governing these relationships, changes that take place far away are allowed to ripple through the system automatically.

6.1 Crossing Flights

The example above illustrates the fundamental monitor-and-enforce mechanisms and the system of relationships required to simulate a merge. Flights whose paths cross, rather than merge, however, share only a common Node object, not a common Link object. As a result, an additional set of relationships among Flight-By-Link objects is required. Specifically, whereas a merging Flight requires a

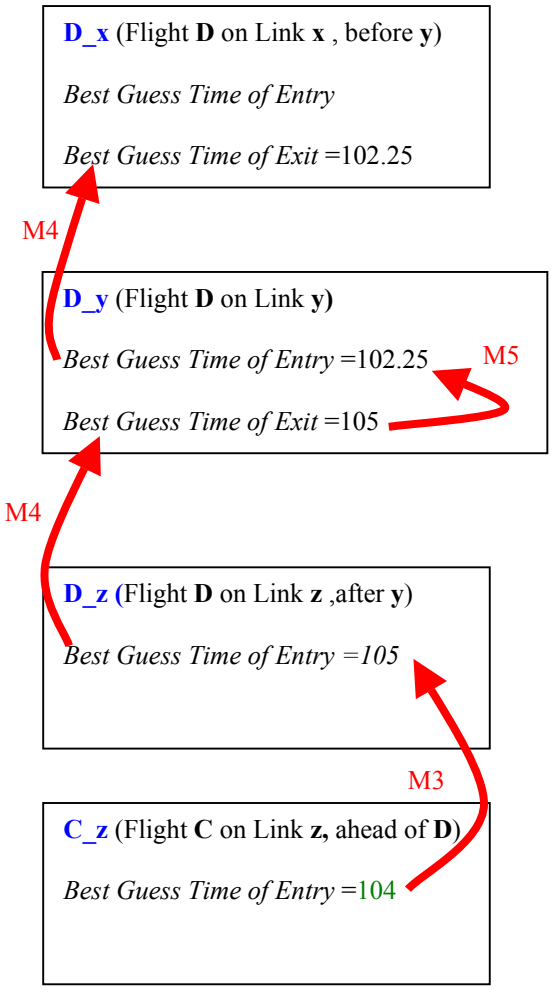

Figure 6: Flight C Delays Flight D, and Delay is Passed Back via Flight-By-Link Objects D_z, D_y, and D_x

pointer to the leading Flight on the shared Link, a crossing Flight requires a pointer to the leading Flight through the shared Node. Apart from that, the mechanisms for maintaining separation and absorbing delay due to crossing are practically identical to those for merging.

7 INFORMATION FLOW

Looking at the problem from a slightly different perspective, the model is doing two things: pushing demand information *forward* through the network (spatially and through simulated time) and pushing capacity and delay information *backward*. Two questions emerge:

- How are the Flight-By-Link pointer relationships established in the first place?
- How far ahead can, and should, this information be transmitted?

A Flight-By-Link object's ***Leading Flight*** pointer remains unassigned until the expected arrival time at the Link is within a time window. The pointers are ordered according to each Flight's initial claim time on the Link. A Flight-By-Link object's inter-*link* pointers (***Next Link*** and ***Previous Link***) are determined by the flight plan, but the threads that monitor and enforce the mathematical relationships are not "activated" (turned on) until the expected arrival time at the Link is inside of a (different) time window. This window should be larger than the window that assigns the inter-*flight* pointers because position information must be pushed forward before constraint and delay information can be pushed backward. In the near future, we plan to implement a model with a more dynamic flight plan, in which the inter-link pointers remain unassigned until the model has chosen a particular route among several options.

Obviously, there are tradeoffs in the selection of time window sizes. Make the time windows too small, and Flights will be unable to absorb the required buffering delay, defeating the purpose of the model. Make the time window too large, and computational performance suffers. Moreover, from a modeling perspective, making time windows very large equates to simulating an air traffic control system with perfect inter-facility coordination and perfect long-range information flow. This is a poor representation of reality: controllers in California do not know the precise arrival time of a trans-continental flight just leaving New York because there is too much uncertainty in travel times.

8 FUNDAMENTALS OF THE SLX SIMULATION LANGUAGE

A model employing the concepts described above was implemented in the SLX simulation language. SLX (Wolverine Software) is a PC-based simulation language and development environment that has some unique capabilities that enable the user to build and run very large, complex models (Henriksen, 1998). While a detailed description of the language is beyond the scope of this paper, three fundamental aspects of the language, necessary for implementing the delay absorption model, are described below.

- *Pucks*. An SLX puck is like an independent thread, or stream of execution, within the simulation model. Pucks allow the modeler to simulate many things happening in parallel. The SLX pucks run extremely quickly, and each one consumes relatively little memory. Each of the mathematical relationships in the model presented above was monitored and enforced by its own puck. A model employing the concepts presented in this article used over 500,000 pucks, up to 250,000 running simultaneously, and ran in about 9 seconds on a PC with a 2-gigahertz processor.
- *Wait Until*. SLX has a "wait until" statement that allows the user to model time- and state-based delays. While a state-based delay capability is not unique to SLX, the speed with which it

executes is noteworthy. Almost all of the 500,000 pucks in the model mentioned above were controlled with "wait until" statements.

- *Active Objects*. SLX is an "object-based" language that allows each object to have an unlimited number of independent pucks (threads of execution), thereby making the object "active". This means that the user can model several things happening in parallel within a single object. Each Flight-By-Link object in the model described above had multiple pucks, one puck for each of the mathematical relationships monitored and enforced.

These modeling constructs, combined with the speed and scalability of SLX, open up many modeling possibilities -- like the approach described above-- that would otherwise be impractical.

9 CONCLUSIONS

This article described a methodology for modeling limited delay absorption capacity, which is a fundamental problem in air traffic control. At its heart, the methodology is based on a large number of independent streams of execution monitoring and enforcing a large number of relatively simple mathematical relationships. This approach relies heavily on the speed and scalability of the SLX simulation language. The examples presented in this article dealt only with delays imposed to maintain separation between aircraft, but the approach could be extended to handle delays imposed for other reasons, such as maintaining a manageable flow through busy sectors.

ACKNOWLEDGMENTS

The author would like to acknowledge Carla Gladstone and Mike White of the MITRE corporation for their help.

This work was produced for the U.S. Government under Contract DTFA01-01-C-00001 and is subject to Federal Aviation Administration Acquisition Management System Clause 3.5-13, Rights in Data-General, Alt. III and Alt. IV (Oct., 1996).

The contents of this document reflect the views of the author and The MITRE Corporation and do not necessarily reflect the views of the FAA or the DOT. Neither the Federal Aviation Administration nor the Department of Transportation makes any warranty or guarantee, expressed or implied, concerning the content or accuracy of the views.

REFERENCES

Henriksen, James O. 1998. Stretching the boundaries of simulation software. In *Proceedings of the 1998 Winter Simulation Conference*, ed. Medeiros, D.J., E. Watson, M.S. Manivannan, and J. Carson, 227-234. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Voss, William R, and J. Hoffman. 2001. Analytical Identification of Airport and Airspace Capacity Constraints. In *Air Transportation Systems Engineering*, ed. G.L. Donohue, A.G. Zellwegger, H. Rediess,and C. Pusch, 409-419. Reston, Virginia: American Institute of Aeronautics and Astronautics.

AUTHOR BIOGRAPHY

JUSTIN BOESEL is a Senior Simulation Modeling Engineer at The MITRE Corporation's Center for Advanced Aviation System Design (CAASD). He received his Ph.D. from Northwestern University and won the INFORMS George B. Dantzig Dissertation Award in 1999. He has been working on air traffic control problems since 2000. His e-mail address is <boesel@mitre.org>.