Performance Experiments with the High Level Architecture and the Total Airport and Airspace Model (TAAM)

David J. Bodoh
Dr. Frederick Wieland

Center for Advanced Aviation Systems Development
The MITRE Corporation
7515 Colshire Drive, McLean, VA 22102
{fwieland, dbodoh}@mitre.org

Introduction

The purpose of this paper is to present a series of performance experiments in which the United States' Department of Defense (DoD) High Level Architecture (HLA) has been used to improve the performance of a sequential aviation simulation. As it was originally envisioned and commonly used, the HLA is a mechanism for interconnecting disparate simulations over a network. Its main application has been distributed wargaming, where simulations prepared by different organizations are combined in a virtual environment for a specific training exercise or study objective. The individual simulations are called *federates* in the HLA world, while the collection of federates that interoperate in the virtual world is called a *federation.* The HLA specifies both procedures by which organizations define a federations, as well as software protocols to which the federates adhere for interoperation.

In this paper we investigate an alternate use of the HLA. It is possible to imagine that the federates are not disparate simulations, but a single simulation that has been partitioned along some logical axis and federated with itself. Such a self-federation has many attractive features. First, the data structures are identical among the various federates. Secondly, there are no problems with common definitions of data elements, messages, interactions, or algorithms, as the federates all derive from the same source code. Finally, the mechanism can be viewed as a lightweight parallel simulation engine, as all of the details of synchronization, time management, message passing, and monitoring have already been well thought out and implemented in the HLA system.

The Total Airport and Airspace Model (TAAM) is a large air traffic simulation that is a worldwide standard for aviation analysis. Traditionally it has been used for regional studies, consisting of a small subset of airports and airspace. Recently, there has been interest in using TAAM for much larger scenarios, such as simulations of traffic throughout the entire United States. It is possible, but not practical, to run such simulations with TAAM. A simulation of a large fraction of the traffic in the United States requires at least 35 hours of run time; simulations of the entire country would require at least double that time.

The study question we seek to address here is, can the HLA be used as a parallel simulation mechanism to allow large TAAM simulations to execute in a reasonable amount of time? In other words, to what extent can the HLA be used as a self-federation mechanism for TAAM? As noted above, the use of the HLA has many attractive properties, not the least of which is that many of the issues involved in parallel simulation engines have a solution inside the HLA.

Details of the Problem

The United States' National Airspace System (NAS) is a collection of airports, navigation aids, airspace, equipment, controllers, pilots, airlines, procedures, and other associated equipment, organizations, and policies that facilitate the safe operation of air traffic throughout the United States [Nolan1990]. Many organizations have built simulations of the NAS, ranging from abstract to more detailed [Wieland 2002]. The Total Airport and Airspace Model (TAAM), a product of Preston Aviation Services in Melbourne, Australia, is regarded as a very detailed and standard model for analysis of aviation issues.

TAAM represents airports in one of three modes: as points (sources and sinks of flights), in "runway only" mode, and in full layout mode which includes taxipaths and arrival/departure gates. TAAM also models the airspace in some detail, providing the user with various options for airspace configuration. Relevant to this study, the most important option allows the user to turn on (or off) the conflict detection function, which determines when two aircraft will eventually be within conflict. Also, the conflict resolution function, which allows TAAM to determine a vectoring path for one of the two aircraft that

are in conflict, can also be switched on (or off). Conflict resolution is determined by scanning a rulebase that specifies how controllers would react in similar situations.

A typical simulation of the whole NAS would require detailed airport layouts for up to thirty-two airports of interest, runway only mode for up to twenty other airports, and point sources for the remaining airports.[1] Airspace would typically be modeled with conflict detection turned on, and conflict resolution would be used if a rulebase is available that adequately describes conflict avoidance maneuvers.

The experiments outlined herein use a geographic partitioning strategy for parallelization. That is, the scenario space is divided into two regions, one "east" and one "west," and the HLA is used to coordinate the flow of aircraft as they cross the boundary. The details of this geographic parallelization strategy, and how it was implemented, are provided below.

Related Work

The only other self-federation study of which we are aware involves federating a network simulation with itself [Riley 99]. That paper describes an effort where an HLA-like system was used for the self-federation. The modeling issues involved are discussed at length in that paper. The network simulator needs to determine how to route a packet from a source to a sink. The model has to handle routing problems caused by a federate being unaware of the connectivity and link status in remote federates. In our system, routing is not an issue: the routing is done in three-dimensional space, and an aircraft can always point itself at a destination, even if the details about the destination are in a remote federate. Additionally, an aviation simulation generally requires a number of rules and procedures that involve controlling the flow of traffic, runway assignment, arrival sequencing, taxipath computation, gate allocation, and so forth.

Before we began federating TAAM with itself, we conducted our own study of the feasibility of the approach [Bodoh 2001]. We developed a meta-simulation of a self-federation in Java, and found that the speedup was a sensitive function of computation, communication, and their associated variances. While this result was no surprise to us, we were able to verify that, given sequential performance data on TAAM's expected computation and communication, the HLA self-federation is feasible for TAAM..

[1] It should be noted that there is considerable flexibility in how a TAAM scenario is constructed, so the configuration stated here is for illustrative purposes only.

Geographic Partitioning and the HLA

Although the HLA software protocol provides many services for connecting simulations [Kuhl 1999], in the TAAM self-federation, only some of them are needed.. We need federation management, to create the federation and have the individual TAAM federates join it, declaration management to handle the routing of messages, object management for sending and receiving interactions (events), and finally time management for synchronizing the advancement of the simulation clock across federates. Each of the TAAM federates is identical, except for the federate ID (an RTI identifier unique to each TAAM instance), the geographical area they control, and the flights they own, which is based upon their geographic area of control.

Thus the basis for self-federation is splitting the scenario space into distinct non-overlapping geographical regions, such that each TAAM federate handles all the air traffic within a region. As the traffic moves across region boundaries, the federates hand off the data using the HLA protocols.

A software realization of the HLA protocol is called the Runtime Infrastructure, or RTI. The RTI we are using was built at the Georgia Institute of Technology in Atlanta. It is called the DRTI, for Distributed Runtime Infrastructure. It is implemented in C++, and is easily compilable using the "gcc" compiler. Only one modification of the DRTI software was needed to support TAAM (we increased the size of the message buffers). The DRTI system requires two UNIX environment variables to be defined for it to work; beyond that, all the software modifications and changes were to TAAM itself.

Although the HLA specification allows for interoperability of federates on a heterogeneous network, there is no support in DRTI for heterogeneous networks. Practically, this means that the self-federated TAAM that we built will only run on networks of identical processors (i.e., all Sun workstations, or all Linux boxes—it will not run on a mixed Sun/Linux network). There are two software constraints that prohibit heterogenerous operation: (1) the DRTI does not include support for binary translation between "little-endian" and "big-endian" platforms; (2) we encoded the TAAM interactions as binary data structures, as opposed to ASCII text.

The individual TAAM federates must share the same scenario data, which means they all read the same project file. We have added two new scenario files to the system: the federation file (FED file) required by the HLA RTI, and a geography mapping table file. The FED file

describes the data structures that are shared between instances of TAAM—the flight plan, segments, waypoints, and so forth. The mapping file defines the geographical extent of each federate's responsible area in the federation.

Each federate must execute from a separate directory, even if the directories are cross-mounted. This is required because each federate writes a different report file. Finally, even if the federation is hosted on a single mulit-processor mahine, the federates cannot share memory, which means that the data structures sent between TAAM instances must contain the actual data—not pointers to memory locations. As a result, the TAAM federation must "deep copy" each data structure it sends.

The FED file defines the type of data to be shared. Generally, the FED file can store either objects or interactions or both, but in our self-federated TAAM all communication is done via interactions. As a sidenote, "interactions" in HLA are transient messages that are sent to the RTI, while object interaction represents an update to a persistent data structure. The self-federated TAAM has two types of interactions that transfer data: an *aircraft handoff* interaction and one that transfers pending scheduled events for an aircraft previously handed off. In addition, there are four types of interactions that are instructional in nature. The first is a *kill aircraft* interaction that informs all federates when one federate has deleted an aircraft. The second is a *set aircraft approach* interaction, while the third is a *sequencing* interaction. The fourth is a *shift time marks* interaction.

The second added file—the geography mapping file—defines rectangular geographic boundaries for each federate. Each federate must have an entry in the file, and the entire scenario space must be covered by the union of each federate's individual space. It is not necessary, or even desirable, for each federate to have the same quantity of scenario space in its geographic region. Rather, it is necessary to evenly distribute the work across federates, as a function of simulated traffic load bound by geographic region.

Data Marshalling Logic

With a distributed simulation, HLA provides the vehicle for transferring data among federates, but it is the responsibility of the programmer to ensure the sent data properly represents the state of the model, and the data received properly reflects the state of the model. Also, the data messages need to be encoded in a form recognized by the HLA constructs for transmittal. This set of tasks is called Data Marshalling.

Resident data structures in TAAM cannot be transmitted over HLA in their original form. Most of the existing structures contain a mixture of static and pointer data that, if exhaustively transmitted, would require transmittal of virtually all of the data in memory. We discovered that sending all such aircraft data is unnecessary.

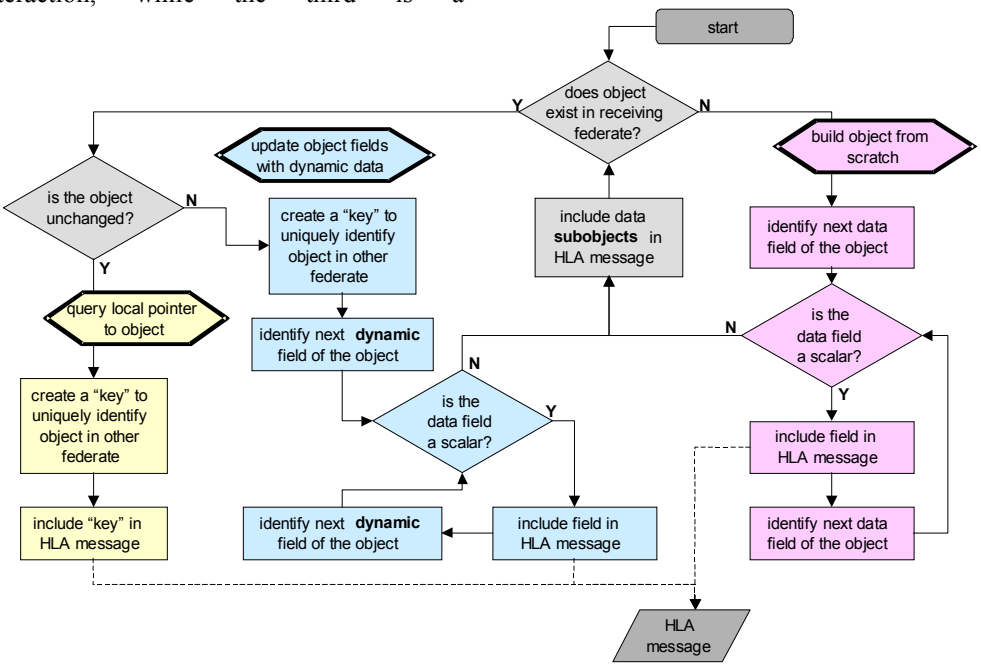


Figure 1. Data Marshalling Logic in the TAAM self-federation

At a minimum, the data message had to include all dynamic data (any data items that could be different between federates), while also harnessing any dynamic data of the sub-elements of the structure that are in turn dynamic. Any data that was not dynamic, yet still needed to be identified in the message, was sent using a "key." Keyswere created to uniquely identify data objects based on some field property or as an index in a list.

One primary use of the "key" method is to locate pointers to aircraft structures. A federate cannot send the address of an aircraft to another federate because the other federate is using a different memory block, perhaps even a different computer. The receiving federate would make no sense of a foreign address. So the sending federate could just send the aircraft ID, which can then be used by the receiving federate to query a hash table to find that aircraft pointer. To this end, all federates read the same time table information, and assign identical unique Ids to the same aircraft.

Software Ambassadors

In the HLA specification, there are two ambassador classes that are used for communicating with the other federates. The *RTI ambassador* provides functions that the federates employ for communicating with the RTI. Likewise, the *federate ambassador* provides the callback functions used by the RTI for communicating with the federates.

In the DRTI, the RTI ambassador class is already defined. The TAAM federate ambassador only overloads functions required for operation within the federation. In this case, it only implements **`timeAdvanceGrant()`** and **`receiveInteraction()`**.

Initializing the Federation

HLA defines a set of features to support federation management. In order to get a set of federates to cooperate in a common simulation, they need to agree on a number of issues including what types of data can they share and when they can start executing simulation events. HLA also defines a set of features for time management and declaration management that prepares the federate for how it will participate in the group.

In TAAM, the **`start_sim()`** function is used once to initialize the simulation. In this function we added a step to initialize this copy of TAAM as a federate in a federation, which prepares it by using the three management services mentioned above.

First, the federate joins the federation by specifying three values: a common name that all federates will use to access the same federation, a FED file that contains the federation object model for all available data types, and a reference to the federate ambassador instance for this federate. TAAM then initializes some internal states related to its participation within the federation. This includes assigning its federate space, assigning handles to the interaction classes, setting time management parameters, and publishing and subscribing to interactions. Finally, the federates enters a wait state until all federates have reached this stage of initialization. Once that is attained, the federation can proceed with the remainder of **`start_sim()`**.

Initializing Aircraft

Federating TAAM required the introduction of three new fields as part of the aircraft structure. The **`federate`** field represents the ID of the federate that has control of the aircraft. This field will be a known federate ID, or –1 if the owning federate is undetermined. The **`on_loan`** flag indicates that the aircraft is temporarily under the control of the local federate, and will be returned to its true owning federate after some calculations (this is explained more in the section on approach settings). The **`relay_kill`** flag indicates the origin of where the aircraft was killed in the federation (this is explained more in the section on terminating an aircraft).

These fields are initialized when the aircraft object is instantiated in the simulation At the same point, the aircraft is registered in a new hash table built for quick retrieval of aircraft pointers using their ID as a hash key. Because all federates read the same data from identical time tables, the resulting hash table entries will contain the same Ids for those aircraft, yet maintain distinct values for the local aircraft addresses. This initialization stage is also the point where TAAM determines which federate will control the aircraft. Using the initial position of the aircraft and the geographic space assigned to the federate, the aircraft will either be included or excluded from the federate's control. All aircraft in a federate's control are explicitly stored in a global linked list. Only aircraft in the federate's space are added to the this list; aircraft not in the federate's space are excluded from that list. In fact, nearly half of an aircraft initialization stage is skipped if it is excluded from the federate's space. This filtering process guarantees that every aircraft will be assigned to no more than one federate at initialization. Control of any aircraft can be later transferred between federates as described below.

Advancing Time

HLA provides management services for time advancement, though HLA itself does not represent any notion of a "simulation-wide" clock, nor does it regulate

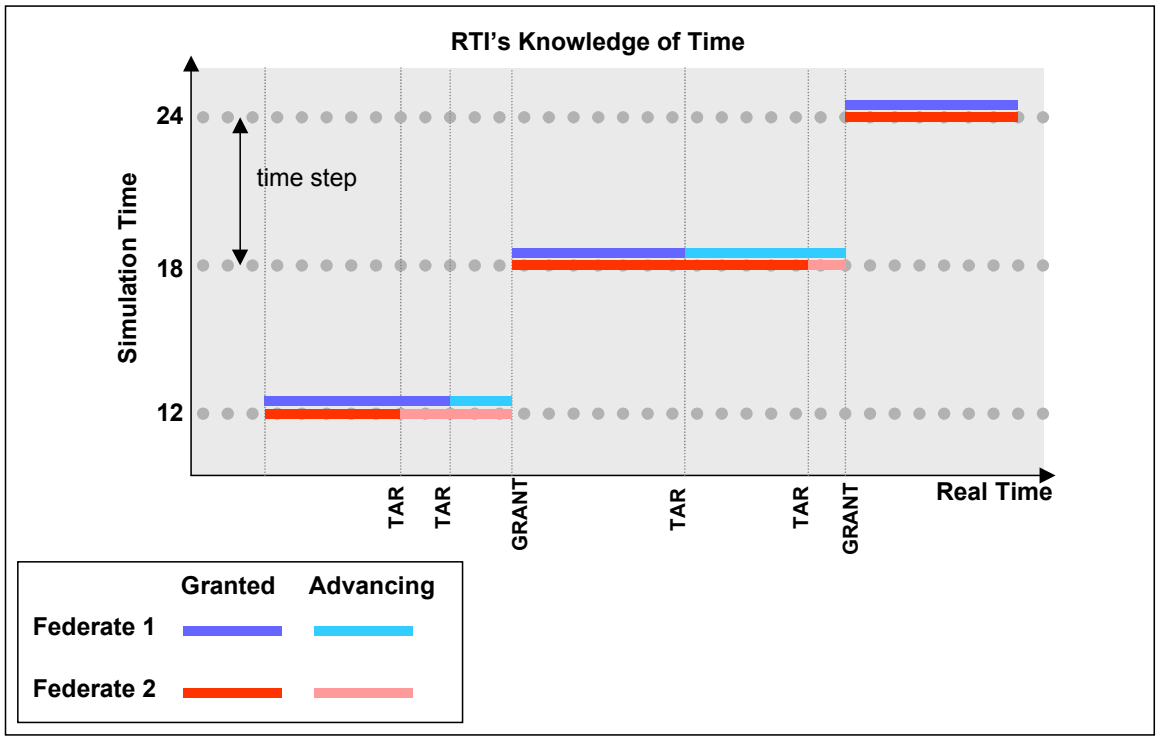


Figure 2. Time management in the HLA

the speed of the federation. The federates themselves advance their internal clocks, and depending on how they are registered with the RTI, can influence or be influenced by other federates' perceived advancement of time. The internal clock value that is presented to the RTI by the federate is known as that federate's "logical time." The RTI keeps track of every federate's logical time so that it can safely issue grants to time advance requests. The federation as a whole does not have a single logical time.

The TAAM federates are both time-regulating and time-constrained. Time regulating federates request time advances from the RTI. When a federate is ready to advance to a new logical time, it must first get permission from the RTI, which will only grant permission based on the current logical times of all the federates in the simulation. Time constrained federates are constrained by time-regulating federates. When all federates are established in this configuration, the simulation is known as "conservatively synchronized."

Time-regulating federates alternate state between the granted state and the advancing state. The granted state is when the simulation is performing calculations at some logical time. The advancing state is when the federate issues a time advance request from the RTI and is awaiting a grant. Once the grant is received, the federate resumes the granted state. While waiting for the grant, the federate may receive and process any time-stamped interactions delivered by the RTI from other federates.

The core of TAAM's aircraft processing function updates each aircraft's position every six simulation seconds. At each iteration, TAAM sends its request to the RTI to advance to the current_time. Note that TAAM can and does perform events at other times between these six-second processing cycles, but they are transparent to the RTI because the federate's logical time is only presented with each time advance request. Any messages that are received and processed by the federate are executed during the advancing state. Federates can send messages while in either state. The primary difference is that messages sent during the granted state will be received by the other federate(s) during the very next advancing state. Messages sent during the advancing state will be received by the other federate(s) during the subsequent advancing state.

Figure 2 illustrates how two federates affect each other's advancement of time. At the start of each time cycle, both federates perform the calculations in the processing function, and any other events that follow the processing event. When the next processing event is ready to commence, each federate requests a time advance from the RTI, handles any and all events pending with that timestamp, and receives the grant. The grant will normally be sent immediately after the last federate has

requested the advance. Any delay in receiving the grant is due to message process time spent during the advancing state. Also, because all federates are using the same time step, all time advance requests sent to the RTI will be for identical time values. In other words, the federates will not be "stair-stepping" each other to advance time. Note that in figure 2, the dark-shaded states represent the time TAAM would be spending in a sequential run. The light-shaded states represent the overhead associated with processing HLA messages. The chart is for illustration only, do not draw any conclusions about the scale of event times.

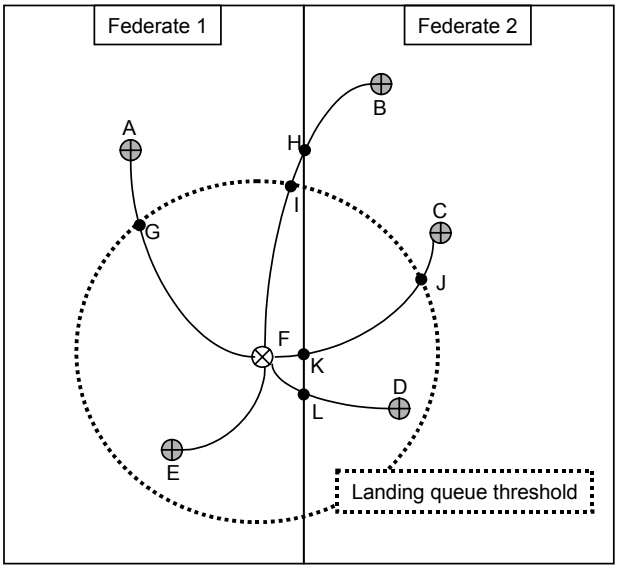
Aircraft Handoff

Because aircraft are moving about the simulated airspace, it is inevitable that several will leave the designated airspace of the federate in which they were initialized and enter another federate's airspace. Aircraft may even cross the boundaries of several federates' spaces during its flight. The use of HLA interactions facilitates this transition of ownership called handoff.

The basic operation of a handoff is that each time the aircraft changes position, a query is made to the owning federate's space to determine if the new position is outside that space. If so, then collect all the dynamic information about that aircraft's state into an HLA message and send it. At that instant, the aircraft is removed from the aircraft linked list, and any associated events still pending in the event list are also purged, and sent in a similar fashion. The receiving federate locates its local instance of that aircraft, and populates that aircraft data structures with the data that was in the message. The events associated with that aircraft are also re-scheduled in the local schedule list, and the aircraft is added to the local linked list.

The original intent for aircraft handoffs was to perform them only at federate border crossings. We later discovered exceptions to that rule. First, to prevent aircraft from entering holding patterns that breach federate borders, handoffs will wait until the aircraft leaves a holding state before a handoff can be completed. Without that rule, TAAM could generate several handoffs between two federates in short succession. This would add unnecessary overhead in communication. Second, we noticed that once the aircraft performs its approach setting, it will subsequently be compared and contrasted to other aircraft in its landing queue. If any of the aircraft in that queue are controlled by different federates, each federate may not have the complete state of all the aircraft at any given moment. Therefore we decided that only one federate will maintain the landing queue for each arrival airport, and any aircraft in that airport's landing queue must be controlled by that same federate. To accomplish this, a handoff must be performed when an aircraft is about to calculate its approach setting, and its arrival airport is located in a different federate's space. This may result in handoffs that happen well before the aircraft crosses federate boundaries, but it maintains integrity of all the flights approaching the same airport.

In Figure 3, five different flight situations are observed. The perimeter around point "F" represents the time-based approach threshold for aircraft approaching airport F.



AGF: Approach set enroute at G, no handoff.

BHIF: Handoff at H, then approach set enroute at I.

CJKF: Handoff at J. Federate 1 sets approach upon receipt. No event at K.

DLF: Temporary handoff to federate 1 for approach set on ground, return handoff to federate 2 immediately. Handoff again at wheels-off to federate 1. No event at L.

EF: Approach set on ground, no handoff.

Figure 3. Aircraft Handoff in the federated TAAM

TAAM calculates approach settings when that threshold is crossed. The line between federate 1 and federate 2 represents the border between their federate spaces. Airports *A, B, C, D* and *E* represent points of origin for aircraft landing at airport *F*.

With regard to airport *A*, TAAM calculates the approach at point *G*, however there is no handoff because the aircraft is landing in the same federate where the approach is being set. From airport *B*, the aircraft first crosses from federate 2 to federate 1 by means of handoff, and then reaches the approach threshold. Again, the aircraft continues to perform all calculations in federate 1. In case *C*, the aircraft reaches the approach threshold before it enters federate 1's space. Federate 2 initiates a handoff at point J to federate 1, which will then calculate the approach setting, and continue controlling the aircraft to the destination airport. Note that there is no handoff event at point *K*. In case *D*, the origin airport for the aircraft is already within the approach threshold for airport F, but not in the same federate as *F*. In this case, the aircraft may remain on the ground for some time after the approach is set. Because federate 2 controls the aircraft separation on the ground at airport *D*, federate 1 must receive the handoff, perform the approach setting, and handoff the aircraft back to federate 1 immediately. Federate 2 continues to control the aircraft on the ground until wheels-off at which point it performs another handoff back to federate 1. There is no handoff event then at point *L*. In case *E*, both departure and arrival airports are in the same federate, so there are no handoff events there.

A similar set of circumstances exists for the case of aircraft crossing the sequencing threshold, which is a smaller radius about the arrival airport. However in this situation, the aircraft should already have had its approach set, and would then already be controlled by its final destination federate. The exception here is that the aircraft could still be on the ground, as in case *D*, and require (another) episode of bi-directional handoffs while on the ground.

Aircraft Termination

Each TAAM federate maintains two lists of aircraft, One represents the list of aircraft currently controlled by TAAM. The other is list of all known aircraft in the simulation, active or not. Because this seconds list is unaffected during aircraft initialization, it will remain consistent among federates until an aircraft is deleted or killed. To maintain consistency, TAAM must communicate "terminate aircraft" events so that all federates can perform the same operation. A simple HLA interaction was designed to transmit the aircraft ID from any federate that initiated the aircraft termination. All other federates who received this interaction would immediately terminate their local instance of that same aircraft. Note that in order to prevent recursive interactions from second-level aircraft delete events, a flag is used within the aircraft data structure which get set by only the original federate. Receiving federates will note that the flag is set, and bypass the step used for sending that HLA interaction.

Federation Termination

Near the end of a TAAM simulation, it will reach the point where only one aircraft is still active. In this case, only one of the federates will be actively controlling aircraft, while the others have stopped their processing cycles. Normally, TAAM would end the simulation at the termination of the last active aircraft. If that were the case in a federation, the terminating federates would cause an early resign error which the RTI would propagate to any remaining active federates. To prevent early resign errors, the terminate simulation function will be interceded by a call to request a time advance to an ultimate simulation time. Therefore any non-active federates will be in the advancing state while waiting for the active federates to either send messages, or also request advances to the terminal time. The active federate(s) will not need to wait as long for time advance grants, because the RTI has already confirmed the inactive federates are beyond any logical time still in use. Once the last active federate has reached the advance request to the terminal time, all federates will receive the grant, and terminate simultaneously.

Performance of the Federation

TAAM, as modified above for self-federation, was executed as two federates on a single multiprocessor Solaris server. Two difference scenarios were used for the experiment. Both scenarios were considered large enough to warrant efforts in reducing the overall run time. The first scenario is a 6,600 flight scenario, representing traffic for about 8 hours in the northeastern United States. The second scenario is a 34,000 flight scenario, representing traffic for an entire day in the same region. In both scenarios, the geographic extent included all of New England, south to Washington DC, and as far west as Chicago.

The experiments were conducted on a 700 Mhz 6-processor Intel-chip server running Solaris 2.9 with 2GB RAM and 4GB of swap space. This server was ideal because each federate could be assigned a unique processor, and there is no overhead associated with network latency. We used the final release version 1.2 of TAAM, as well as version 3.0 of the Federated

Simulations Development Kit (FDK) from Georgia Tech. This kit contains a functioning RTI called the DRTI which was used to support the HLA features of the experiment. As a proof of concept, some test federations were deployed across a network with two Intel-based Solaris servers, to conclude that the distributed simulation can be executed over a network.

The unmodified sequential version of TAAM, running as a single federate, required approximately 25 minutes for the 6,600 flight scenario, and 3.5 hours for the 34,000 flight scenario.

Figures 4 and 5 present the results of the experiment with two federates (vs. the baseline) in a graphical format. The speedup for the 6,600 flight scenario was a tiny 1.2%, while that for the 34,000 flight scenario was 14.4%. We measured the performance gain by inserting additional timing code into the TAAM federates. The additional code allowed us to assign the time spent during federation execution into one of five different bins: both federates in granted state (they are running the simulation simultaneously), one federate busy (either advancing or granted) while the other waits; both federates in advancing state; and advance/grant cross-parallel time. The results for the two scenarios are shown in figures 6 and 7.

Discussion and Conclusions

The results of this experiment are simultaneously pleasing and disappointing. We are pleased that TAAM, a sequential simulation that has evolved over ten years, can be self-federated using the HLA technique. We are somewhat disappointed with the low speedup achieved by the federation. In inspecting Figures 6 and 7, it is apparent that only 26% of the time the 6,600 flight scenario was computing in parallel. This corresponds to a maximum

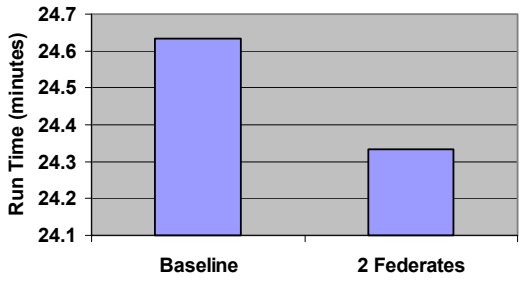


Figure 4. 6,600-flight scenario performance

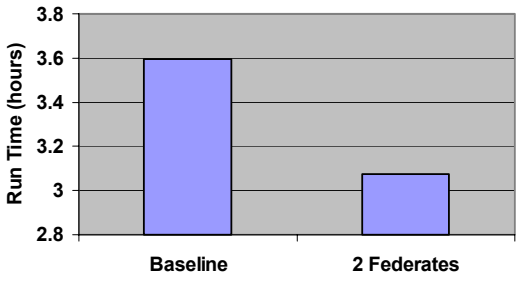


Figure 5. 34,000-flight scenario performance

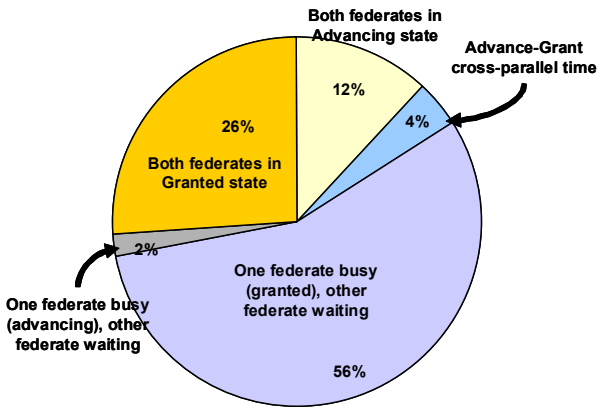


Figure 6. Time Assignment for the 6,600-Flight Scenario

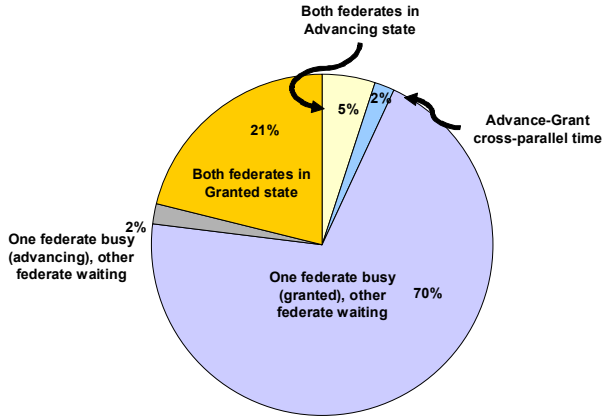


Figure 7. Time Assignment for the 34,000-Flight Scenario

speedup of 1.26. For the 34,000 flight scenario, the maximum speedup would be 1.21. The discrepancy between the maximum available and the observed speedup (1.01 and 1.18, respectively) is most likely due to low computational granularity, but we have not made the measurements to confirm this belief.

The rest of the system represents overhead due to either message passing by the RTI, or due to poor load balancing among the federates. The latter is shown by the time one federate is busy while the other waiting; this consumes 58% of the time in the 6,600 flight scenario, and 72% of the time in the 34,000 flight scenario.

		SPACE		
		Federate A	Federate B	Federate C
time	t1	workload At1	workload Bt1	workload Ct1
	t2	workload At2	workload Bt2	workload Ct2
	t3	workload At3	workload Bt3	workload Ct3

Table 1: Workload Distribution

When the goal of federating a simulation with itself is to let the federates operate in parallel to achieve a faster simulation, it is ideal to keep all the federates as busy as possible. The workload divided among the federates is based on geographic space. Further, each of those workloads are partitioned to operate within the granted state of the federate, as seen in table 1. The workload within each federate's granted state is a function of time and space. For each time step, each federate has some simulation calculation to perform, ranging from none whatsoever, to everything, for that particular time step. Recall that all federates will receive the time advance grant at the same time, therefore the measured amount of time to process the federation's workload at a given time step will be no less than the maximum federate workload for that time step. Any federates that complete their workload early will simply wait and do nothing until all federates are done. If the maximum workload is significantly greater than the average workload for the majority of time steps, then the simulation is considered to be load-imbalanced. A load-imbalanced simulation is inefficient based on the amount of time federates are idle.

The user has limited control of balancing the workload by adjusting and fine-tuning the boundaries that separate the geographic space of the federates. However this can only solve the problem in one dimension (space). Over time, the workload could shift from East to West, making the federate space balanced early in the simulation, but imbalanced later in the simulation, or vice-versa. The user is left to attempt to balance the federation's workload by approximating the space boundaries around the busiest and most time-intensive points of the simulation. This is tenuous at best. Most of the time step intervals could still be left as heavily imbalanced which produces less chance for simulation speed up. As it stands, there is no way to dynamically alter the federate boundaries during the simulation. That would require an extra set of communication for bulk handoffs of aircraft, which would add more overhead than it is worth.

Thus we conclude that the major source of overhead in these scenarios is load balancing. If this conclusion stands, then TAAM scenarios which possess better load balancing between the federates should show much larger speedup than reported here. Such experiments will remain the focus of future investigations.

Acknowledgements

We would like to thank Preston Aviation Services of Australia for their support, interest, and collaboration in this work. In particular, we would like to thank Dr. Sasha Klein, Dr. Leigh Samphier, and Mr. Paul Gargett from Preston. In addition, we gratefully acknowledge the DRTI software from the Georgia Tech College of Computing, Dr. Richard Fujimoto's research group, and in particular assistance from Dr. Thomas McLean. Finally we would like to thank The MITRE Corporation's Center for Advanced Aviation System Development for their continued participation and support for this research.

The contents of this paper reflect the views of the authors. Neither the Federal Aviation Administration nor the Department of Transportation makes any warranty or guarantee, or promise, express of implied, concerning the content or accuracy of the views expressed herein.

References

[Bodoh 2001] David J. Bodoh, Frederick Wieland, "Self-Federating an Aviation System Using HLA: Is it Feasible?" *Distributed Interactive Simulation-Real Time (DiS-RT) Conference*, August, 2001 IEEE:Cincinnati, Ohio.

[Kuhl 1999] Frederick Kuhl, Richard Weatherly, Judith Dahmann, Anita Jones, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture*, Prentice Hall: 1999.

[Nolan 1990] Michael S. Nolan, *Fundamentals of Air Traffic Control,* (Wadsworth:1990).

[Riley 1999] George F. Riley, Richard M. Fujimoto, Mostafa H. Ammar, "A Generic Framework for Parallelization of Network Simulations," *Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems,* October, 1999.

[Wieland 2002] Frederick Wieland, "Modeling the NAS: A grand challenge for the simulation community," *Grand Challenges in Modeling and Simulation*, San Antonio, TX, 2002.