*Proceedings of the 2004 Winter Simulation Conference*
*R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.*

# DISTRIBUTING A LARGE-SCALE, COMPLEX FAB SIMULATION USING HLA AND JAVA: ISSUES AND LESSONS

Leon F. McGinnis

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

## ABSTRACT

Distributed simulation promises a range of benefits and opportunities, especially for modeling large-scale complex systems, such as wafer fabs. However, as with many promising technologies, the devil is in the details. This paper describes some experiences in distributing a high-fidelity, full fab simulation model with federates implemented in Java representing manufacturing processes, automated material handling systems, and control systems. Some informal comparisons with AutoMod/ASAP are included.

## 1 INTRODUCTION

Distributed simulation is not a new idea. In fact, the relevant archival literature goes back at least twenty five years. One of the most often cited papers (Chandy and Misra, 1981) cites earlier work in 1979. The Defense Modeling and Simulation Office, by distributing the High Level Architecture (see, e.g., (DMSO, 2004a)) has precipitated a large number of prospective applications, motivated by objectives as diverse as faster runtimes, integration of disparate legacy models, easier model maintenance, information sharing/hiding, and integration of simulation with real control systems.

The notion of distributing a simulation model among a collection of independent programs, or federates, is especially appealing when the system being modeled is large in scale, complex, and subject to frequent change. If only we could focus model revision and maintenance activities on a set of smaller, better defined models, we would be able to achieve much greater productivity in model development and maintenance. And if the computational overhead of distribution is not too high, it will be well worthwhile.

Distributed simulation would seem to be an ideal technology for manufacturing and supply chains, at least judging by the number of related papers. A search of the Inspec database using the search string "((((("distributed simulation") WN All fields) AND ((manufacturing) WN All fields)) OR (("supply chain") WN All fields))" and limiting the search to conference articles returns 1136 "hits".

When the interested reader digs into this abundant literature, what he or she is likely to find are two predominant themes. In the first, authors describe the use of specific software technologies to implement a federated simulation, e.g., the language(s) used to code the federates, the protocols used for messaging, and the architecture of the various ambassadors. In the second, authors describe a class of applications at a high conceptual level, and often provides some screen shots of a small-scale, simple application.

Much more difficult to find are papers that discuss in useful detail the tricky aspects of integrating the concepts and technologies of HLA with the structuring, modeling, and implementation of the federates. This is not to say such papers do not exist, but they are difficult to find.

The goal of this paper is to address one specific example of the interaction between the use of HLA and the modeling of a large scale complex system. The story has four parts: a brief tutorial on synchronization in HLA; a brief description of the domain being modeled; a somewhat more involved discussion of the key elements of the resulting federated simulation; and a summary of the lessons learned.

## 2 SYNCHRONIZATION

An HLA-based simulation has two kinds of components, the federates themselves, and the runtime infrastructure, or RTI. (For a good introduction to HLA, see, e.g., (Kuhl, et al., 1999) or (DMSO, 2004b)) A federate is a stand-alone simulation model, with its own time advance mechanisms, and a message-passing protocol that serves two functions it instantiates the simulated interactions between federates, and it implements the method used for insuring that the federates are *synchronized*.

Synchronization, as a fundamental process in distributed simulation, has attracted a vast array of research (see, e.g., (Fujimoto, 2000) or (Fujimoto, 2004) for a fully detailed discussion of synchronization.) There are various approaches to synchronization, but for our purposes, the discussion will be limited to those designated *conservative*. We also will restrict our attention to event-oriented time advance (versus time-increment oriented time advance).

Synchronization is the primary function of the RTI, at least in terms of the application to manufacturing and supply chains. As explained in (Fujimoto, 2004), each federate is a *logical process*, and has its own *current time*. If federate A at time $t_A$ sends a message to federate B representing an interaction, it is important that the interaction message appear to federate B before its logical time has advanced beyond $t_A$. That way, the interaction can be handled as an event in the pending event list and can be processed when federate B's current time advances to $t_A$. Failure to enforce this requirement would mean that federate B has to process an interaction occurring in its past. Since most of the systems we want to model don't admit time travel, this would violate a fundamental requirement for model fidelity.

The challenge of synchronization is to devise a method for implementing this rather straightforward requirement in such a way that computational efficiency is not overly compromised.

In HLA, interactions between federates are mediated by the RTI, using messages. When a federate, say federate A, interacts with another federate, say federate B, a *time stamp ordered* (or TSO) event, or message, is sent to the RTI. The RTI holds this message until it is eligible to be delivered to federate B. The conditions for eligibility for sending a message (interaction) to federate B with a timestamp of, say, $\tau$, are that:

1. no other messages are waiting for delivery to federate B with a timestamp smaller than $\tau$ , and
2. the RTI can guarantee that no new messages will be sent to federate B by any other federate with a timestamp smaller than $\tau$

The fundamental question is, "How does the RTI know when these two conditions are satisfied?"

In effect, the RTI tells a federate when it can advance its logical time. For the purposes of this discussion, suppose federate A requests permission to advance its logical time to its next scheduled event (a *next event request*, or NER) at time $\tau$. The RTI determines the minimum time at which it is still possible for the federate to receive an interaction from some other federate, say $\tau$' and grants time advance to the minimum of these two times.

The RTI has all the TSO messages sent by other federates, but it needs to determine the earliest time at which it might still receive TSO messages. It does this using a *lookahead* value for each federate. According to (DMSO, 2004b) "Lookahead is a non-negative value that establishes a lower value on the time stamps that can be sent in TSO messages by a time-regulating joined federate." Suppose federate B has been granted time advance to time $\tau^*$, and has a lookahead value of $\lambda$. The earliest time at which federate B can generate a TSO message is $\tau^*+\lambda$. By considering this earliest new message time for each other federate, the RTI can determine a "safe" time advance for federate A.

In summary (and somewhat oversimplified) synchronization is achieved in HLA by using a lookahead value for each federate to determine the earliest time at which it may have a new interaction with other federates, and limiting the time advance granted to each federate by insuring that it does not advance beyond the time at which it may be guaranteed that no new interactions will be originated by other federates.

How is the value of the lookahead determined? Logically, it is the minimum simulated time between processing an event from the federate's pending event list and sending a TSO message. From a naïve perspective, the lookahead for a federate is the minimum processing time for any federate task which results in a TSO message.

## 3 WAFER FABS AND MODELS

A wafer fab is a large-scale, complex system, with dynamic behavior that is not amenable to analytic description at the single tool-job level. We are focusing on latest-technology fabs, based on 300mm wafers, and fully automated material handling. The most common configuration of such a wafer fab has a number of processing departments (bays, farms, cells) with dedicated internal material handling, linked via a stocker to a fab-level material handling system for moving product between bays. In state of the art 300 mm wafer fabs, product moves in environmentally-controlled pods or FOUPs (for Front Opening Unified Pod).

The typical pattern of processing for a FOUP in a traditional *through-stocker* system is as follows:

- Delivery to a bay stocker
- Retrieval from bay stocker and delivery to a process tool
- Processing on the process tool
- Retrieval from the process tool and delivery to the bay stocker
- Retrieval from the bay stocker and delivery to the next bay stocker in the process route

There may be a tool-to-tool move within the bay, if more than one process occurs in the same bay.

A major concern is to provide the right amounts of capacity, for both processing and material handling, and to coordinate the assignment of capacity to FOUPs to insure a proper balance between throughput and cycle time.

Simulation has been the preferred tool for evaluating fab performance, and today the most popular simulation tool for studying wafer fabs appears to be AutoMod/ASAP. In this combination, the material handling portions of the model are implemented in AutoMod, while the routing, scheduling, and processing portions of the

model are implemented in ASAP. A proprietary synchronization method is used to couple the two models.

Kim, et al. (2001) describe a high fidelity, Java-based approach for simulating a wafer fab. Their approach is inherently object-oriented, with objects representing process tools and stockers, intra-bay material handling, bay controllers, inter-bay material handling, and a fab controller. Because the implementation was based on Java, it was relatively easy to incorporate graphical tools for representing the fab layout and for animating the simulation results.

A key feature of the HiFiVE implementation was the level of fidelity. Machines were represented using "virtual machines" having a well defined state graph, and a set of controller interactions that would mimic the "real" interactions between a controller and process tool.

The computational burden of the original implementation led us to develop a distributed version of the HiFiVE simulation, using HLA as the distribution platform. In the course of converting to HLA, the HiFiVE simulation was reconfigured into six federates:

1. fab model: includes all process tools and stockers
2. fab controller: manages lot release
3. bay controller: manages all lot flow
4. intrabay material handling: overhead hoist transportation system in all the bays
5. interbay material handling: overhead hoist transportation system between bays
6. simulation manager: initialization, termination

Note that there is one bay controller federate, but it models all the bays in the fab. Likewise, there is one intrabay material handling federate, but it models all the bay-specific material handling.

In the transition to HLA, it was necessary to formally define the TSO messages representing interactions between the federates. Table 1 summarizes the messages that were defined in a straightforward way from the earlier HiFiVE model. Note the "command/acknowledge" structure of the messages, which follows directly from the earlier virtual machine architecture. "Delay lot move" is a command that allows a lot to wait in its originating bay to see if a tool becomes available in that bay, potentially avoiding an intrabay move for the next operation.

## 4 PROGRESS AND LESSONS LEARNED

As of this writing, we are in the process of benchmarking the distributed version of HiFiVE against Automod/ASAP, using the Sematech 300mm fab model (Campbell and Ammenhauser, 2000). This model has 24 bays, with 53 tool groups and 310 process tools. There are 14 product types, but only two different processes. Each process involves 316 process steps.

For the computing platforms we are using, AutoMod/ASAP full fab (processing and material handling, but no animation) simulation speed for this model is about 100x "real time", i.e., 100 hours of simulated time requires approximately 1 hour of compute time. The corresponding speed factor for the distributed version of HiFiVE, in its initial incarnation was between 3x and 4x.

Table1: TSO Messages, Version 1

| PUB | SUB | Note |
|---|---|---|
| FC | FM | Release lot |
| FM | BC | Lot released |
| BC | IT | Request interbay lot move |
| IT | FM | IT vehicle available to load/unload |
| FM | IT | IT Lot loaded/unloaded |
| IT | BC | Interbay lot move complete |
| BC | ET | Request intrabay lot move |
| ET | FM | ET vehicle available to load/unload |
| FM | ET | ET Lot loaded/unloaded |
| ET | BC | Intrabay lot move complete |
| BC | FM | Move lot into stocker from IT port |
| FM | BC | Lot moved into stocker from IT port |
| BC | FM | Move lot from stocker to ET port |
| FM | BC | Lot moved from stocker to ET port |
| FM | BC | ET vehicle loaded at stocker |
| FM | BC | Lot moved from ET port to stocker |
| BC | FM | Move lot from stocker to IT port |
| FM | BC | Lot moved from stocker to IT port |
| FM | BC | IT vehicle loaded at stocker |
| FM | BC | Start process operation |
| FM | BC | Finish process operation |
| BC | FM | Delay lot move |
| FM | BC | Delay ended |
| FC: Fab controller; BC: Bay controller<br>FM: Factory model; IT: interbay transport<br>ET: Intrabay transport | | |

Needless to say, the team was perplexed by the poor performance of the distributed version of HiFiVE, and began to look for opportunities to improve it. The search revealed a number of factors, not the least of which was our

naiveté regarding the intelligent development of a federated simulation.

The time regulating function of the RTI interacts with the structure of the federation (the events and the TSO messages) to determine how large a time advance is granted to each federate . In version 1 of the distributed HiFiVE implementation, the acknowledgement TSOs, represent not physical processes but simply information updates, and have a devastating effect on simulation efficiency, because they become events representing very short duration processes—in our case, approximately 6 seconds of simulated time.

**Lesson #1:** whenever possible, model the interactions between federates so that no acknowledgement TSO is required.

For example, when a vehicle arrives to be unloaded, and announces that it is ready to be unloaded, have both the transporter federate and the factory federate schedule the unloading event in their respective pending event lists, but do not generate a TSO announcing the completion of unloading. This avoids the overhead of a TSO message, which would also have the effect of scheduling an event.

This models federate interactions as "open loop' control processes. As a result, there is a potentially significant implication of lesson #1. If one hopes to use HLA to support a federation including both a simulation and actual control software, then the role of acknowledgment messages will need to be considered carefully. It may be possible to "spoof" the acknowledgements coming from the simulated factory without making them part of the time regulating function of the RTI.

Another opportunity to apply Lesson #1 is in the design of the stocker simulation. In the first incarnation of HiFiVE, stockers were essentially "dumb" automation, and had to be managed by the bay controller. However, the behavior of stockers is pretty simple, and it would be relatively straightforward to give their models enough intelligence to manage the movement of FOUPs between load ports and internal storage locations. Doing so would eliminate a significant number of TSO events between the factory federate and the bay control federate.

**Lesson #2:** Structure the simulation so that TSO events are minimized.

The modeler always has a great deal of freedom in designing the structure of a simulation, whether it is federated or not. In the case of the distributed fab model, for example, two original design decisions turn out to have negative impacts on computational efficiency.

First, the loading and unloading of transport vehicles is "done" by the factory federate. This means that when vehicle arrives to a load port, the transport federate must send an arrival TSO to the factory federate, and then the factory federate must send a load/unload completion TSO to the transport federate. In retrospect, a better design would have been to have the arriving vehicle perform the load/unload operation, and then have the transport federate send a load/unload completion TSO to the factory federate. This design is feasible, provided the destination port for a FOUP is reserved at the time the transport request is issued.

Another example of poorly structuring the federation was the decision to include the stockers in the factory model, along with other "machines" such as process and metrology tools. Superficially, this seems to be a reasonable design, because both stockers and process tools have load ports and interface with the transport system.

A case could be made, however, that a better design would be to include the stockers in the bay controller federate. The bay controller must maintain a "data image" of the stocker, in order to track lot queues for machines, so from a modeling point of view, there is duplication of data if the stockers are in the factory federate. From a pragmatic perspective, stockers exist primarily as a mechanism to improve the control of lot flow, by allowing lots to accumulate away from the process tools, and providing a buffer between transport systems. Functionally, they serve a control purpose rather than a processing purpose. Incorporating stockers into the bay control federate would, again, eliminate some TSO events. On the other hand, treating the stockers in this fashion conflicts with the goal of integrating simulated physical resources with real fab control or management software, because it eliminates the explicit representation of stockers and control/acknowledgement messages.

## 5 CONCLUSIONS

Successful implementation of HLA for distributed factory simulation is much more than simply federating a set of models. The interplay between modeling decisions and the methodology instantiated in HLA must be understood and exploited to achieve acceptable computational results.

We are currently using lessons 1 and 2 to revise the implementation of the distributed version of HiFiVE, and expect to see significant improvements in simulation speed.

Looking to the next steps, a fundamental issue is the structure of the next event request in HLA, and the use of lookahead values. We are investigating whether the federated simulation can be defined in such a way as to maximize the lookahead value seen by the federate which most frequently submits a next event request. A related issue is the difference between "internal" and "external" events in the pending event list for a given federate. Those events that do not result in an interaction with other federates are, in effect, transparent to those other federates, so perhaps they can be excluded from the RTI-based time advance considerations.

## ACKNOWLEDGMENTS

A number of graduate students at Georgia Tech have played key roles in the ongoing HiFiVE project. For the federation work, Ke Wang, Ying Wang, and Sheng Xu

have taken the lead. For AutoMod/ASAP, Dima Nazzal has been the lead.

## REFERENCES


Campbell, Elizabeth and Jim Ammenheuser. 2002. 300 mm Factory Layout and Material Handling Modeling: Phase II Report. Technology Transfer # 99113848B-ENG. International SEMATECH.

Chandy, K. M., and J. Misra. 1981. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*. 24(4), April: 198 – 206.

DMSO. 2004a. High Level Architecture. <https://www.dmso.mil/public/transition/hla/> [accessed July 23, 2004].

DMSO. 2004b. Introduction to the HLA: Glossary. <http://www.aegistg.com/adl/adl_courses/intro/intro_glossary1.htm> [accessed July 23, 2004].

Fujimoto, Richard M. 2000. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, Inc.

Fujimoto, Richard M. 2004. Time management in the high level architecture. <http://www.cc.gatech.edu/computing/pads/PAPERS/Time_mgmt_High_Level_Arch.pdf> [accessed July 23, 2004].

Kim, H., J. Park, S. Sohn, Y. Wang, S. Reveliotis, C. Zhou, D. A. Bodner and L. F. McGinnis. 2001. "A High-Fidelity, Web-Based Simulator for 300mm Fabs," in *Proceedings of the 2001 IEEE International Conference on Systems, Man, and Cybernetics*, Piscataway, NJ: IEEE, pp. 1288-1293,.

Kuhl, Frederick, Richard Weatherly, and Judith Dahmann. 1999. *Creating Computer Simulation Systems*. Prentice Hall.

Lutz, R. 1998. High Level Architecture Object Model Development and Supporting Tools. *Simulation.* 71 (6): 401-409.


## AUTHOR BIOGRAPHY


**LEON McGINNIS** is Eugene C. Gwaltney Professor of Manufacturing Systems at Georgia Tech, where he also serves as Associate Director of the Manufacturing Research Center and founding Director of the Keck Virtual Factory Lab. His research focuses on the application of operations research and computer science to solve decision problems arising in the design and operation of industrial logistics systems. His email address is <leon.mcginnis@isye.gatech.edu>.