Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

INTEROPERATING AUTOSCHED AP USING THE HIGH LEVEL ARCHITECTURE

Boon Ping Gan
Peter Lendermann
Malcolm Yoke Hean Low

D-SIMLAB Programme
Singapore Institute of Manufacturing Technology
71 Nanyang Drive
Singapore 638075, SINGAPORE

Stephen J. Turner
Xiaoguang Wang

Parallel and Distributed Computing Centre
Nanyang Technological University
Block N4, Nanyang Avenue
Singapore 639798, SINGAPORE

Simon J. E. Taylor

Centre for Applied Simulation Modeling
School of Information Systems, Computing and Mathematics
Brunel University
Uxbridge UB8 3PH, UK

ABSTRACT

The High Level Architecture (HLA) is an IEEE standard for interoperating simulation federates. In this paper, we describe a set of requirements that simulation packages need to satisfy in order to be made interoperable using the HLA standard. *AutoSched AP*, a commercial off-the-shelf simulation package (CSP) which is widely used in the semiconductor industry, was used as a case study for this interoperation exercise. We demonstrated that a straightforward customization of the CSP through a middleware that provides standard functions for interoperation may not provide a satisfactory solution. A specially optimized time synchronization mechanism needs to be installed to ensure good execution efficiency. Experimental results using a Borderless Fab model that comprises of two factory models show that an optimized time synchronization mechanism results in an execution time that is ten times better than a straightforward application of the HLA Runtime Infrastructure's time synchronization mechanism.

1 INTRODUCTION

The ever changing business environment in today's global marketplaces requires organizations to adapt to changes promptly to stay competitive. Many organizations have adopted simulation as an enabling technology for this decision support process, evaluating impact of changes to their business/operations before changes are implemented, in particular when experiments on the real system are not feasible because they would disrupt daily operations. These simulation models, representing the current business and operation practice, are mainly built using commercial off-the-shelf simulation packages (CSPs). Integrating and interoperating these individual simulation models to form a larger simulation that represents a supply chain (Gan et al. 2000, Lendermann et al. 2003), for mutual beneficial decision making among organizations, is very difficult. This is because most current CSPs do not support interoperation of simulation components. Some examples of CSPs are: AutoMod, *AutoSched AP*, WITNESS, Arena, Pro-Model, and Simul8.

Interoperation of CSPs can be realized through the adoption of the High Level Architecture (HLA), an IEEE standard developed by the U.S. Department of Defense (DoD) to facilitate interoperability and reusability (IEEE 1516 2000). However, customizing each and every CSP through different means to achieve interoperation using the HLA involves significant effort. The Commercial off-the-shelf Simulation Package Interoperability Product Development Group (CSPI-PDG 2005) endorsed by the Simulation Interoperability Standards Organization (SISO) recently began work to complement the HLA standard by creating standards for interoperation of simulation components/packages. Its objective is to derive a set of standard reference models, data exchange standards and generic interfaces (Taylor, Turner, and Low 2005) for communication between the Runtime Infrastructure (as defined by the interface specification) of the HLA standard and the CSPs. In this paper, we illustrate the exercise of interoperating

AutoSched AP (*ASAP*) simulation models through the adoption of the HLA and the CSPI-PDG draft standard. *ASAP* is widely used in the semiconductor industry. It is a highly flexible CSP where extensions to the simulator can be realized through the customization module (Brooks 2001). The customization module is then integrated into the *ASAP* simulation engine through dynamic linked libraries.

On the modeling aspect, we preserve the way models are built using *ASAP* with the interoperation details hidden from the modeler. The extension also ensures that existing *ASAP* models can be made interoperable with minimum modifications. Straβburger (1999) called this an implicit approach as all the HLA functionalities are hidden from the modeler's point of view.

On the implementation aspect, interoperating simulation components incurs extra overhead as the simulation executes and communicates across a network. Also, the simulation components' time progress needs to be synchronized. This extra overhead should not slow down the simulation execution too much as it will offset any benefits gained in reusing and interoperating the simulation components. Thus, a Borderless Fab model (Lendermann et al. 2004) built using the *ASAP* simulator that involves simulation of two wafer fabrication factories sharing capacity to cope with new product introduction was used to evaluate the performance of the interoperating *ASAP* simulation components.

This paper is organized as follows: Section 2 first gives a brief overview of how a typical discrete event simulator advances time using the HLA, followed by a derived set of requirements that CSPs need to fulfill before they can be made interoperable. In Section 3, we discuss how *ASAP* fulfills the requirements outlined in Section 2, propose a system architecture for the interoperation, and discuss two variations of the time synchronization mechanism. The Borderless Fab model is described in Section 4. This is followed by the performance comparison of a sequential simulation, and a distributed simulation using the two variations of the time synchronization mechanism, in Section 5. Lastly, Section 6 concludes the paper with an outline of future work.

2 INTEROPERATING COMMERCIAL OFF-THE-SHELF SIMULATION PACKAGES (CSP)

2.1 Typical Execution Steps for Distributed Discrete Event Simulation

To interoperate CSPs, we first have to understand how a typical distributed discrete event simulator executes a simulation run. This involves three phases, namely the initialization, execution (state transitions at discrete points in time), and termination phases. During the initialization phase, the CSP joins the simulation and declares its interest in information/events. After this it is suspended until all other simulation components (also known as federates in HLA terminology) join the simulation. This is illustrated by Lines 1 to 3 in Figure 1.

The federate transits into the execution phase once all other federates join the simulation. In the main event loop (illustrated by Lines 4 to 10 of Figure 1), the federate makes a request for time advancement before it simulates any events (Line 5), a necessary action when the conservative synchronization protocol is used. This is to ensure that the causality constraint of the simulation is not violated. The RTI offers two ways of advancing time, namely the *next event request* (*NER*) and the *time advance request* (*TAR*). The most appropriate time synchronization mechanism to be used will be the *NER* as the federate has no information about when the next external event arrives, and when will it trigger an external event. The *NER* will result in a grant for the federate to proceed either to the requested time or the time of the external events that are delivered to the federate. Upon delivering the external events, the RTI returns control to the federate. The federate will then simulate all the events that have timestamp less than or equal to the granted time (Line 8). During this process of simulation, new external or local events will be triggered (Line 9). This process of time advancement and simulation is repeated until the end of simulation time is reached, or when there are no more events to be simulated.

The termination phase is the most straightforward. The federate that exits the simulation last will destroy the simulation.

```
1.  Join simulation
2.  Declare interests of the model
3.  Suspend simulation till all federates join
4.  While not end of simulation
5.   ReqTime = time of next event to be
       processed
6.   TimeGranted = ask for time advance to
       ReqTime from RTI
7.         {receive external messages}
8.   Process all events with timestamp at
       TimeGranted or less
9.         {sending messages or introducing
            new local events}
10. End of loop
11. Destroy the simulation
```

Figure 1: Typical Execution Steps for Distributed Discrete Event Simulation using the HLA Standard

2.2 Interoperation Requirements for CSP

Based on the typical execution steps of distributed discrete event simulation outlined in Section 2.1, CSPs need to provide the following features for interoperation using the HLA standard:

- (R1) Ability to initialize the distributed simulation prior to simulation execution
- (R2) Ability to suspend the simulation execution

- (R3) Access to the time of the next event to be simulated
- (R4) Ability to introduce new events/entities from the external source into the event list
- (R5) Access to information of simulation objects/entities that are shared among federates

Features (R1) and (R2) are needed because the federate needs to first join the simulation and then be suspended from execution until all the other federates join (Lines 1 to 3 in Figure 1) before the simulation begins. The federate is also suspended while it is waiting for a time grant from the RTI (Line 6 of Figure 1).

When the federate requests for time advancement using the *NER* time request, it needs to provide the timestamp of the next event to be simulated (Line 5 of Figure 1). This means that the CSP needs to provide a mechanism by which the timestamp of the next event can be obtained, and hence the (R3) requirement.

New events will be received from other federates during the course of simulation (Line 7 of Figure 1), while the federate is suspended waiting for the time grant. These new events are simulated when the control is returned to the federate as their timestamp is less than or equal to the timestamp of the current local events. Hence the CSP needs to provide a mechanism to simulate these external events and suspend the local events for the time being. One way is to allow these external events to be merged into the internal event list of the federate. With this, external events will be handled just like local events when the control returns to the federate. This gives rise to the required (R4) feature.

When an event is simulated, new external events/entities might be triggered (Line 9 of Figure 1). They have to be packed into a message and delivered to the receiving federate. To pack the content of the event/entity into a message, the CSP needs to provide a mechanism to access the data structure that describes the event/entity, hence the (R5) requirement.

3 INTEROPERATING AUTOSCHED AP (ASAP) MODELS

3.1 ASAP Fulfilling Interoperability Requirements

AutoSched AP (*ASAP*) is a CSP supplied by Brooks Automation that is specially customized to model wafer fabrication plants. It is widely used in the semiconductor industry to answer questions such as the impact of dispatching rule or product mix changes to the factory performance, or identifying bottleneck equipment with varying demand profiles, etc. Simulation models are constructed through input data files. The input data files define the process routes, resources (workstations and operators), preventive maintenance (scheduled and non-scheduled), order arrival patterns, and so on. Simulation progresses as lots move from one workstation to another, following steps defined by the process route (Brooks 2001). In this paper, lots will be the basic entities that are moved from one model to another to keep the illustration of interoperating *ASAP* simple.

The important aspect of *ASAP* that will be covered here is its extensibility for interoperation. *ASAP* uses a publish/subscribe messaging system to allow function calls to be triggered by events that occur during simulation. These functions are known as the notification functions in *ASAP* terms, and they implement extensions to the *ASAP* simulation engine. These extensions are all collected into a library that is loaded into *ASAP* at runtime. The publish/subscribe mechanism of *ASAP* makes it possible to enhance the *ASAP* simulator for interoperation.

Table 1 summarizes the notification messages that the customization module needs to subscribe to, for interoperation of *ASAP* models. A notification function of a simulation component that joins the distributed simulation declares interest in information or events (including information that it shares out) and is associated to the NT_FACTORY_FINISHED_ALL_FILES message. This notification message is published by the *ASAP* simulation engine immediately after all the standard input data files are read. This satisfies (R1) mentioned in Section 2 as we are able to initialize the distributed simulation prior to simulation execution. The handing over of execution control to the notification function also means that (R2) is satisfied as the *ASAP* simulation engine will not proceed until the call to the notification function returns. Thus, the notification function can suspend the federate until all other federates have joined the simulation.

Table 1: Notification Messages by ASAP

Event Type	Description
NT_FACTORY_ FINISHED_ALL_ FILES	Published by the simulation engine immediately after reading all the standard input data files.
NT_FACTORY_ FINISHED_ CURRENT_EVENT	Published by the simulation engine when all events at current simulation time are simulated.
NT_APPLICATION _INTIALIZED	Published by the simulation engine just before the simulation begins, after the model is fully initialized.
NT_APPLICATION _FINISHED	Published by the simulation engine immediately before simulation terminates.

The NT_FACTORY_FINISHED_CURRENT_EVENT is published by the simulation engine after all the events at the current simulation time are simulated. A notification function that advances the simulation time of the federate

through the RTI can be associated with this message. By doing so, we can suspend the simulation from progressing, by not returning from the notification function, until a time grant is given by the RTI. This ensures that the federate does not violate the causality constraint. But in order to issue a time request to the RTI, we need to know the timestamp of the next event that will be simulated. *ASAP* provides access to its underlying simulation engine through a set of classes, collectively known as the AP Framework (Brooks 2001). The timestamp of the next event, which is the first event in the Future Event List, can be obtained through the *SIMEngine* object class. Thus, *ASAP* satisfies the (R3) requirement outlined in Section 2.

As mentioned earlier, lots are the basic entities that are moved from one model to another in *ASAP*. Lots at the source model have to be deleted, and need to be introduced at the destination model when they arrive. The AP Framework offers the *FIFactory* object class that exposes the ability to create and delete a lot dynamically during the simulation run. Having access to this object class, we are able to introduce lots at the destination model and fulfill the (R4) requirement.

The *FILot* object class provided by the AP Framework enables us to retrieve the attribute values of a lot. These attribute values are packed into a message and delivered through the RTI when we model movement of a lot from one factory to another. This means that *ASAP* fulfills the (R5) requirement as well. As a result, the *ASAP* simulator is able to fulfill all the five requirements for interoperability outlined in Section 2.

The NT_APPLICATION_INITIALIZED and the NT_APPLICATION_FINISHED in Table 1 are another two important notification messages that the customization module needs to subscribe to. The former notification message has an associated function that requests for initial time advancement before the simulation begins, after the model is initialized. This kicks start the simulation and subsequent time advance requests are triggered by the NT_FACTORY_FINISHED_CURRENT_EVENT message as discussed earlier. The latter notification message is used to implement a notification function that causes the federate to resign from the simulation when the federate reaches the end of the simulation.

3.2 Modeling Capability Extension

Section 3.1 mainly focuses on the question whether *ASAP* fulfills the interoperability requirements. Another very important issue that we need to address is that an enhanced *ASAP* simulator for interoperation should not be more difficult to use than the basic *ASAP*. Bearing this in mind, we preserve the way the *ASAP* is used by introducing two new input data files that define information associated with distributed simulation. Tables 2 and 3 show the information that are needed to use this extension. By doing so, the modelers still build their simulation models in the same way as before. In addition, they just need to supply the data shown in the two tables.

Besides introducing these two data files, we also extend the process route data file such that modelers can define which step of the process route will trigger a lot transfer from one factory to another. Table 4 defines the three new data fields that were added to the process route data file. The condition that triggers lots to be moved from one factory to another can be coded by the modeler using the AP Framework, and a *forward_lot* function that is supplied by our extension. The modeler does not need to know how the lot is forwarded, but just has to indicate the destination federate and exit point to be used to forward the lot. Note that an exit point always corresponds to an entry point in the receiving federate.

Table 2: Distributed Simulation Configuration File

Field	Description
Federation Name	The name of the distributed simulation. It is used for the federate to join the simulation.
Federate Name	The name of this federate.
Fed Filename	The fed file that defines the information to be shared in the distributed simulation.
Number of Federates	Number of federates to join the simulation. Used by the controller federate to kick start the simulation once all federates have joined.
Controller	Indicates if the federate is a controller federate. Only one exists in each simulation.

Table 3: Entry/Exit Points Definition File

Field	Description
Source	Name of source federate.
Destination	Name of destination federate.
Exit Point	Exit point that connect the two federates.
Entry Point	Entry point that connects the two federates.
Delay	Delay in hours for sending a lot from source to destination federate.

Table 4: Extension to Process Route File

Field	Description
Condition	Condition to satisfy for routing the lot.
Destination	Destination federate that receives the lot.
Exit Point	Exit point to be used for movement of the lot.

3.3 The System Architecture

As mentioned in Section 1, we adopted the draft standard defined by the CSPI-PDG that complements the HLA standard in interoperating the *ASAP* simulator. The CSPI-PDG has defined a set of generic interfaces (Wang et al. 2004) that CSPs can call to interact with the RTI of HLA. This reduces the effort required to interoperate CSPs as any CSPs that are compliant to this standard will be able to talk each other, using the same data exchange standard and protocol that is also defined by the CSPI-PDG. Figure 2 shows the system architecture of the enhanced *ASAP* simulator. At the top level, the simulation model is built using the *ASAP* as usual. *ASAP* is extended to invoke the *DSManager* through the generic interface which handles the interaction between *ASAP* and RTI. The details of this interaction are all hidden in the middleware. The middleware also implements the standard data exchange protocols that were defined for alternative reference models in CSPI-PDG. More details of the reference models can be found in CSPI-PDG (2005).

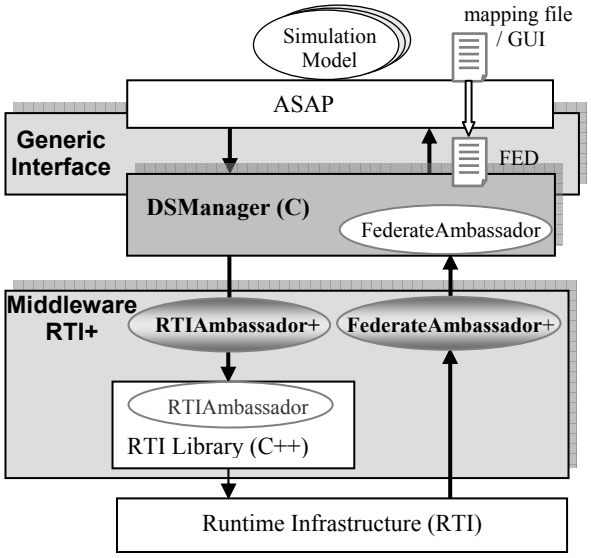

Figure 2: System Architecture

3.4 Time Synchronization Mechanism

Having discussed the details of interoperating the *ASAP* simulator, one very important issue to be resolved in this interoperation exercise is the time synchronization mechanism to ensure efficient execution of the distributed simulation. The most straightforward way of realizing the time synchronization among federates is to request for time advancement for every event that the federate simulates. But this will potentially slow down the simulation significantly. One solution to this problem is to request for time advancement when all local events have a timestamp larger than the last time granted by the RTI, using the timestamp of the earliest event that potentially will trigger external events as the request time.

This approach is valid because local events that do not affect the causality of other federates are not important to the RTI (this statement is only true when we use the *NER*/*NERA* time request). The RTI is only interested in those events that potentially can cause a causality error during a simulation run. Hence, it is safe for federates to issue a *NER* time request with the timestamp of the next event that potentially can trigger an external event. One might argue that an event arriving from other federates potentially will also trigger external events. Issuing a time request without considering these undelivered external events might not be safe. This is not true because the RTI will not give a time grant that is greater than the timestamp of undelivered external events when using the *NER* time request. Hence, this approach of requesting for time advancement using the timestamp of events that potentially trigger external events is still safe. Now the question is how do we identify these potential events?

For the *ASAP* interoperation exercise described in this paper, we can obtain the potential events by looking at the steps that trigger a lot moving from one factory to another. For example, when a lot might be forwarded at step *n*, a lot being scheduled at step *n-1* potentially will trigger an external event after t_{n-1} time units where t_{n-1} is the processing time of the lot at step *n-1*. We can keep track of all the forwarding events that are associated with step *n-1* in a potential event list. Whenever a time request needs to be issued, we will use the timestamp of the next event in the potential event list as the requesting time.

```
1.  While not end of simulation
2.       EventTime = time of next event to be
       processed
3.    If (EventTime > TimeGranted)
4.      If (PotentialEvents.is_empty())
5.          ReqTime = TimeGranted + TimeStep
6.      Else
7.          CurrEvent = PotentialEvents.pop()
8.          ReqTime = timestamp of CurrEvent
9.      End if
10.         TimeGranted = ask for time advance to
            ReqTime from RTI
11.       End if
12. End of loop
```

Figure 3: Optimized Time Synchronization Mechanism

Just using the timestamp of the next event in the potential event list for the time request is not a complete solution. There will be situations in which no activities happen at step *n-1*, and the potential event list will be empty. In such cases, what is the request time that we should use? Can we issue a time request to the end of simulation time? Issuing a time request to the end of simulation time is not valid as some lots that are waiting at step *n-2* or earlier might eventually arrive at step *n-1*, which potentially will

trigger an external event. When an external event is sent, its timestamp must be at least the last granted time plus lookahead. Lookahead is defined as the minimum time delay at which an external event will be sent from the last granted time. A solution to this problem is to switch the time synchronization mechanism to a time step approach, whereby the request time is incremented with a time period from the last granted time. This raises the question of what is the right time period to be used to ensure efficiency of simulation and preserve the causality constraint of the simulation.

The best time period that can be used for this time synchronization mechanism is the processing time of the lot at step *n-1*. If there is more than one step that potentially will trigger a lot transfer, the minimum processing time of all the lots at steps *n-1* can be used. This minimum processing time is safe because any events that are being scheduled during this time period will only generate the potential event at time at least $t+t_{n-1}$, where t is the timestamp of the scheduled event. In the worst case, a lot is moved from step *n-2* to step *n-1* at the last granted time and is scheduled onto the workstation at step *n-1* immediately. The earliest that an external event could be triggered will then be t_{n-1} later, which falls into the next time advance request period. Figure 3 gives a code outline of this optimized time synchronization mechanism that is implemented into the customization module, hidden from the modeler.

4 BORDERLESS FAB SIMULATION MODEL

The Borderless Fab concept is a means of sharing capacity across wafer fabrication plants that are within close proximity (Lendermann et al. 2004). Having a means of sharing capacity across factories is particularly important in the semiconductor industry as equipment is typically expensive. The capital cost of a typical wafer fabrication plant is approximately US$3.5 billion. This sharing of capacity can help factories to handle some unplanned situations more effectively. Examples of unplanned situations are: a sudden surge in demand of certain product types or an unscheduled breakdown of a bottleneck machine. It is also useful in scenarios where specialized equipment that is not fully utilized in one factory can be shared with another instead of buying additional equipment in other factories. Hence, the Borderless Fab concept, if successfully implemented, can help semiconductor companies save millions of dollars in equipment investment, and assists them to handle unplanned situations more effectively.

In this paper, a Borderless Fab model that comprises of two wafer fabs was built using *ASAP*. The two wafer fabs have similar capabilities of producing ten wafer product types (0.35 micron technology logic devices and antifuse gate devices) but with different capacities. The process flows of the wafer products considered range from 200 to 300 steps. A total of 73 workstation types were modeled in each fab, including the downtime behavior of each workstation. Some example of workstations in the wafer fab are: wet benches, furnaces, steppers, implanters and metrology tools.

This Borderless Fab model was used to evaluate different strategies of moving lots from one fab to another. One of the strategies that was evaluated was that lots are moved from one fab to another when the lot of a defined product type reaches a bottleneck machine. Variations of this strategy will be to allow the lots to continue on at the receiving fab, or be moved back to the original factory upon completion of the operation. The model can also be used to decide what is the optimal number of lots that need to be collected before they are moved to another fab. Intuitively, moving one lot at a time is a waste of resources. This model was also used to study the impact of introducing a new product type to a fab that is already at the peak of its capacity. We showed that moving lots of the new product from one factory to another helps us keep the overall capacity of the factory in good shape (Lendermann et al. 2004).

The Borderless Fab concept is a good application of distributed simulation technology. With an interoperable *ASAP* simulator, the two wafer fab models can now be executed in two separate computers, interacting with each other through the RTI, rather than being combined into a single *ASAP* model. The process of integrating the two models into one manually is tedious as we need to worry about the naming of resources, parts, and orders (just to name a few) to ensure that there are no name clashes between the models. Besides that whenever any of the fab models is modified, we will need to redo the integration exercise. With an interoperable *ASAP*, any modification to the fab model is absolutely transparent to the other fab models.

5 EXPERIMENTAL RESULTS

The performance of the enhanced *ASAP* was evaluated using the borderless fab model described in Section 4. Both fabs produce ten wafer product types, with approximately 8500 wafer lots being released into each fab in a year of simulation. We chose a scenario that involves movement of lots in both directions, with a delay of 3 hours for transferring the lots. Lots belonging to a specific product are moved from Fab A to Fab B at a bottleneck workstation, and then continue in Fab B until they reach another bottleneck workstation, when they are moved back to Fab A. We compare the execution time of simulating this scenario as a single model, as a distributed model with the basic time synchronization mechanism, and as a distributed model with the optimized time synchronization mechanism. The experiments were conducted using Pentium IV 1.7 GHz computers, with 1 GBytes of memory. For the distributed

simulation cases, a computer is allocated exclusively to run the server process of the RTI. The computers are interconnected by a 100 Mbps Ethernet.

Table 5 compares the execution time collected for the three scenarios. The execution time collected is the average of five runs to reduce the error introduced by network traffic, especially in the distributed simulation cases. As shown in the table, the performance of the enhanced *ASAP* with the optimized time synchronization mechanism is comparable to the sequential simulation. But the performance using the basic time synchronization mechanism is ten times worse relative to the sequential simulation. The reason for this is that the basic time synchronization mechanism issued approximately 2 million time requests throughout the simulation execution (approximately 2 million events were simulated). In contrast the optimized time synchronization mechanism issued only 4892 and 8533 time requests for Fab A and Fab B respectively. Thus the latter approach reduces the overhead of the distributed simulation significantly. The reason for the difference in the number of time requests issued by Fab A and Fab B is that the time step being used is 91.0 minutes at Fab A and 52.4 minutes at Fab B, which are the respective processing times of step *n-1* where *n* is the step at which lots potentially will be moved from one fab to another.

Table 5: Performance Comparison of Sequential vs Distributed Simulation using *ASAP*

	Sequential	Distributed	
		Basic	**Optimized**
Execution Time	13.0 minutes	133.8 minutes	13.5 minutes
No of Time Requests	**Fab A**	2090819	4892
	Fab B	2053403	8533

These experimental results are encouraging as they show that by distributing the simulation models into two computers, we manage an execution time that is comparable to running the simulation as one single model. The benefits of distributed simulation will become more significant when a larger model is used as distributed simulation provides better scalability. In addition to that, models are much easier to maintain and enhance.

6 CONCLUSIONS AND FUTURE WORK

We have outlined five requirements that CSPs need to fulfill before interoperability can be achieved using the HLA standard. The *ASAP* simulator fulfills all the five requirements, and was successfully made interoperable through the adoption of the complementary standards provided by the CSPI-PDG. The enhanced version of the *ASAP* still preserves the way models are built and the details on interoperation are transparent to the modeler. An important lesson that we have learnt from this exercise is that it is critical to spend some effort in optimizing the time synchronization mechanism during the process of interoperating CSPs. Using the basic time synchronization mechanism provided by the HLA will typically result in an inefficient simulation execution. This was shown in the experiments that we did on a Borderless Fab model, where the basic distributed simulation runs ten times slower than the sequential simulation. On the other hand, the distributed simulation using the optimized time synchronization mechanism was able to achieve an execution time that is comparable to the sequential simulation.

There is still some work that can be done in enhancing the time synchronization mechanism. Further experiments also need to be conducted to evaluate the impact of parameters such as lookahead, and the processing step of the process route for monitoring potential events, to the performance of the simulation. On the interoperability of CSPs, we will next attempt to interoperate *AutoMod* with *ASAP*. This is an important exercise as in 300 mm wafer fabrication plants, *AutoMod* is typically used to model the material handling aspect of the factory, while *ASAP* is used to model the process flow. Having the ability to interoperate these two CSPs allow us to easily integrate the two models to perform *what-if* analysis on the interdependency between the material handling and process flows.

REFERENCES

Brooks Automation. 2001. AutoSched AP Customization Guide v 7.0.

CSPI-PDG. 2005. http://www.cspif.com [accessed March 20, 2005].

Gan, B.P., L. Liu, S. Jain, S.J. Turner, W. Cai, and W.J. Hsu. 2000. Distributed Supply Chain Simulation Across the Enterprise Boundaries. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J.A. Joines, R.R. Barton, K. Kang, and P.A. Fishwick, 1245-1251.

IEEE 1516 2000. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA), New York, NY: Institute of Electrical and Electronics Engineers.

Lendermann, P., N. Julka, B.P. Gan, D. Chen, L.F. McGinnis, and J.P. McGinnis. 2003. Distributed Supply Chain Simulation as a Decision Support Tool for the Semiconductor Industry. *Simulation.* 79. pp. 126-138.

Lendermann, P., B.P. Gan, Y.L. Loh, H.K. Tan, S.K. Lieu, L.F. McGinnis, and J.W. Fowler. 2004. Analysis of a Borderless Fab Scenario in a Distributed Simulation Testbed, *Proceedings of the 2004 Winter Simulation Conference*, Washington DC, USA, December 5-8, 2004, pp.1896-1901.

Strassburger, S., T. Schulze and U. Klein. 1999. Migration of HLA into Civil Domains. *SIMULATION*. 73(5): 296-303.

Taylor, S.J.E., S.J. Turner and M.Y.H. Low. 2005. The COTS Simulation Interoperability Product Development Group. *Proc. 2005 European Interoperability Workshop*. Simulation Interoperability Standards Organization, Institute for Simulation and Training, Florida, 05E-SIW-056.

Wang, X.G., S.J. Turner, M.Y.H. Low and B.P. Gan. 2004. A Generic Architecture for the Integration of COTS Packages with the HLA, *UK Operational Research Society Simulation Workshop*, Birmingham, UK, Mar. 23-24, pp. 225-233.

Wang, X., S.J. Turner, S.J.E. Taylor, M.Y.H. Low, B.P. Gan. 2005. A COTS Simulation Package Emulator (CSPE) for Investigating COTS Simulation Package Interoperability, to appear in *Proc. 2005 Winter Simulation Conference,* Orlando, USA, December 4-7.

AUTHOR BIOGRAPHICS

BOON PING GAN is a Research Engineer with the D-SIMLAB Programme at the Singapore Institute of Manufacturing Technology. The focus of his research is on the application of distributed simulation technology for supply chain simulation. He received a Bachelor of Applied Science in Computer Engineering and a Master of Applied Science from Nanyang Technological University of Singapore in 1995 and 1998, respectively. His research interests are parallel and distributed simulation, parallel programs scheduling, and application of genetic algorithms. His email address is bpgan@SIMTec.a-star.edu.sg.

MALCOLOM YOKE HEAN LOW is a Research Engineer with the D-SIMLAB Programme at the Singapore Institute of Manufacturing Technology. He received his doctorate from Oxford University in 2002. His research interests are in the areas of adaptive tuning and load-balancing for parallel and distributed simulation systems, and the application of multi-agent technology in supply chain logistics coordination. His e-mail address is yhlow@SIMTech.a-star.edu.sg.

PETER LENDERMANN is a Senior Scientist and Head of the D-SIMLAB Programme at Singapore Institute of Manufacturing Technology (SIMTech). Previously he was a Managing Consultant with agiplan in Germany where his focus was on the areas of supply chain management and production planning. He also worked as a Research Associate at the European Laboratory for Particle Physics CERN in Geneva (Switzerland) and Nagoya University (Japan). He obtained a Diploma in Physics from the University of Munich (Germany), a Doctorate in Applied Physics from Humboldt-University in Berlin (Germany) and a Master in International Economics and Management from Bocconi-University in Milan (Italy). His research interests include modelling and analysis of manufacturing and logistics systems as well as distributed simulation. His email address is petrel@SIMTech.a-star.edu.sg.

XIAOGUANG WANG is currently a Ph.D student at School of Computer Engineering (SCE), Nanyang Technological University, Singapore. She received her B.Sc in Computer Science form Nanjing University of Aeronautics and Astronautics, China in 1997. Her research interests lie in Distributed Simulation and the High Level Architecture. Her e-mail address is xgwang@pmail.ntu.edu.sg.

STEPHEN JOHN TURNER joined Nanyang Technological University (Singapore) in 1999 and is currently an Associate Professor in the School of Computer Engineering and Director of the Parallel and Distributed Computing Centre. Previously, he was a Senior Lecturer in Computer Science at Exeter University (UK). He received his MA in Mathematics and Computer Science from Cambridge University (UK) and his MSc and PhD in Computer Science from Manchester University (UK). His current research interests include: parallel and distributed simulation, distributed virtual environments, grid computing and multiagent systems. His e-mail address is assjturner@ntu.edu.sg.

SIMON J E TAYLOR is a Senior Lecturer in the School of Information Systems, Computing and Mathematics, and is a member of the Centre for Applied Simulation Modeling, both at Brunel University, UK. He is also a Visiting Associate Professor at the Parallel and Distributed Computing Centre at Nanyang Technological University in Singapore. He is also Information Director of ACM SIGSIM, ACM SIGSIM PADS Liaison Officer and Chair of the Simulation Study Group of the UK Operational Research Society. He is a steering committee member of PADS and general co-chair of the UK Simulation Workshop Series. His main research interests are distributed simulation and applications of ICT to simulation modeling. His email address is simon.taylor@brunel.ac.uk.