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

TIME MANAGEMENT ISSUES IN COTS DISTRIBUTED SIMULATION: A CASE STUDY

Simon J.E. Taylor
Jon Sharpe

Centre for Applied Simulation Modelling
Department of Information Systems and Computing
Brunel University
Uxbridge, Middlesex, UB8 3PH, ENGLAND

John Ladbrook

Dunton Engineering Centre
Ford Motor Company
Laindon, Basildon, Essex, SS15 6EE, ENGLAND

ABSTRACT

Commercial off-the-shelf (COTS) simulation packages are widely used in industry. Several international groups are currently investigating techniques to integrate distributed simulation facilities in these packages. Through the use of a case study developed with the Ford Motor Company, this paper investigates time management issues in COTS simulation packages. Time management is classified on the basis of the ordering of events that are externally produced to a federate and the ordering of these with events that occur within a COTS simulation package federate. Several approaches to the latter are discussed and one approach is presented as the most effective. Finally the paper presents a bounded buffer problem and proposes the classification of information sharing with respect to the certification of solution.

1 INTRODUCTION

In 2002 the Commercial Off-The-Shelf (COTS) Simulation Package Interoperability Forum (CSPIF, www.cspif.com) was founded (an offshoot of the GROUPSIM collaborative simulation modelling forum www.groupsim.com) The forum was created to organise research into the development of COTS distributed simulation tools for those who practice simulation with COTS discrete event simulation packages (Arena, Extend, Simul8, Taylor, Witness, etc.).

The main aim of this work is to provide industry with a business benefit from distributed simulation while minimising the cost of the use of this technique. For example, *transparency* (technological intervention) and *usability* are of key importance. Simulation modelling is already a costly technique and any additional major cost is undesirable. It may be argued that for distributed simulation to be utilised then it must be low cost and easy to use. The technology should therefore be transparent and useable, fully integrated into the COTS simulation package that the simulation modeller is using.

There are many aspects to this work (some of which were debated in Taylor et al. (2002a). This paper reports on some experiences the authors have had in the development of a distributed simulation of a automotive engine production line that is being developed in collaboration with Ford Motor Company. Specifically, these are derived from the time management of the COTS simulation package WITNESS (produced by Lanner, www.lanner.com). The discussion of this link is the subject of a future paper. Our contribution in this paper is presented so as to be generally useful to others who are attempting to produce COTS distributed simulation tools and applications. The paper is structured as follows. Section 2 presents our case study that has been developed with the Ford Motor Company. Section 3 introduces the approach that has been taken to interoperate the models in the automotive engine production line. Section 4 presents some approaches to COTS simulation package time management and one approach that has been developed on the basis of the analysis of information exchanged between models. Section 5 discusses these with respect to transparency and performance. Section 6 introduces an interesting challenge that we are currently investigating and a possible solution. Section 7 concludes the paper with a short discussion of future research.

2 THE AUTOMOTIVE ENGINE PRODUCTION CASE STUDY

The automotive engine production case study has been developed in conjunction with the Ford Motor Company. It has been created to investigate (in the public domain) a representative situation where distributed simulation is required. Each model is created by a team in a particular COTS simulation package. When a production problem is to be analysed, the models are linked together to form a distributed simulation. The consideration of alternatives to this are outside the scope of this paper (although the con-

venience of linking heterogeneous packages with dedicated data sources (through Excel for example) through distributed simulation middleware might be one convincing, but flawed argument!) This is an interesting case study as it is challenging in terms of distributed simulation and is relevant to the practice of simulation modelling.

The background to this problem is as follows. The production of an engine is a complex problem involving the manufacture and assembly of a wide variety of components into several possible engine types (different capacities, fuel injection options, etc.). The requirement for different engines is determined by orders from the customer. The demand for the different parts is ultimately derived from this, but, given that it takes time to machine the different possible parts, and production lines need to be reconfigured for each part production, buffer stock is required to keep the engine assembly process from waiting. The problem that needs to be addressed is how much of each manufactured part must be kept in the buffer. Figure 1 shows the automotive engine production model. The system is split into two parts – the production of engine parts and the assembly of these parts into an engine. The machining area has five lines (head, camshaft, con-rod, crank, and block) that can produce a varied mix of parts. Each line has a series of machining operations with buffers, machines and resources. The lines feed into the assembly area. Heads and camshafts are assembled in the cylinder head assembly area. The con-rods feed into the piston and con-rod assembly area. Outputs from these two assemblies and the remainder of the machining lines feed into the main assembly line on which the engines are built through a combination of automatic and manual operations. When finished the engine assembly passes through hot test and after test dress operations before being shipped onto the car plant. In our case study, each machine line is a different model (five of them) and the assembly line is represented as a single model. Each production model produces a series of entities that are destined to arrive in the input buffers that exist for each line. Entities will arrive in each buffer at times determined by each production

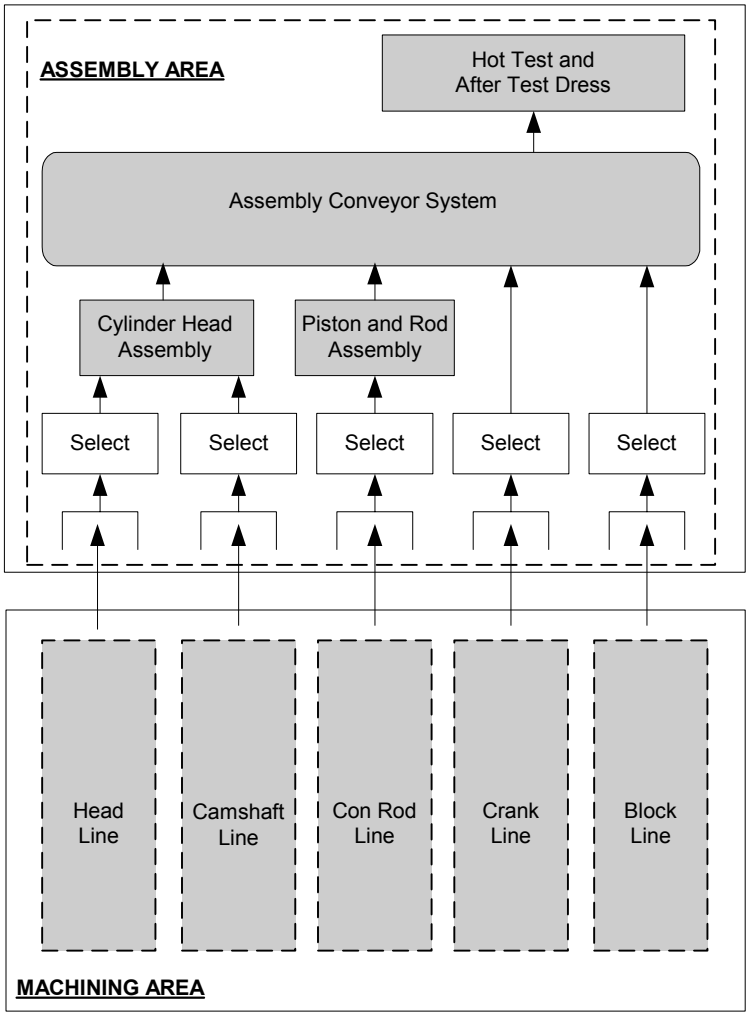

Figure 1: Automotive Engine Production Model

line and are picked by various select operations. The case study has appeared in a slightly different form in Taylor et al. (2001) and Taylor et al. (2002b).

3 DISTRIBUTED SIMULATION

Although in some ways distributed simulation has become synonymous with the High Level Architecture, a discussion of distributed simulation issues often becomes one of "how to use" the HLA's Object Model Template (OMT) and Runtime Infrastructure (RTI) (particularly the DMSO RTI). While this in itself is important it can cloud more fundamental issues. Some HLA terminology is convenient. A model is resident in a COTS simulation package. We shall use federate to refer to the model/package combination and federation to refer to the complete distributed simulation. To keep in line with distributed systems terminology we shall use the term middleware to refer to the computer software that enables federates to communicate (see Boer et al. (2002) and Taylor et al. (2000) for discussion of non-RTI middleware approaches that are an alternative to the HLA RTI approach). Figure 2 shows one possible layered middleware. The COTS SP (simulation package) Interface layer provides interfacing services with the COTS SP (in section 4 we discuss some approaches to time management through this layer). The Presentation layer provides translation services between the various data needs of each COTS simulation package. Time management, data distribution and network services are provided by each other layer in turn.

Our case study has six models that reside in six COTS simulation packages. Our distributed simulation federation therefore has six federates as shown in grey in figure 1. As can be seen, there are essentially two types of federate: five producers and one consumer. A producer federate simulates the production of engine parts from one of the machine lines. The consumer federate simulates the assembly of engines from the engine parts supplied by the machine lines. Information exchanged between the two types of federate is simply the engine part entities. This is represented by the exchange of timestamp event messages. In our case study, the machine lines and the assembly lines are tightly coupled, i.e. any entity that leaves a machine line will immediately arrive at the relevant buffer in the assembly line. In a machine line model, when an engine part entity p1 begins the last machining operation an event e1 representing the end of that operation will be scheduled at t1. When the simulation executive advances to time t1, e1 will be executed. This models the end of the machine operation and the movement of the engine part p1 to the exit point of the machine line. As the machine and assembly lines are tightly coupled, logically as soon as p1 departs the machine line it will immediately arrive in the appropriate buffer of the assembly line, i.e. the end of machining operation event e1 models both the end of the machining operation in the machine line and the arrival of p1 in the assembly line buffer at t1.

To represent this logical interaction, the producer federate must develop an event message, i.e. if we have an event e1 that models the end of machining of an entity part p1 and its arrival at the assembly federate at t1, then we may represent the timestamped event message as *timeadv(Entity, Time)* or *timeadv(p1,t1)* Note that in an actual distributed simulation the sender and receiver federate information will also be included for use by the distributed simulation middleware. Note also that engine part entity p1 will have attributes. The discussion of attributes in distributed simulation and their type compatibility is outside the scope of this paper.

Our producer federates will therefore produce a series of event messages that logically represent the arrival of a series of engine part entities at scheduled points in simulation time. The next section will consider the consumer federate and time management issues of external and internal events in COTS simulation packages.

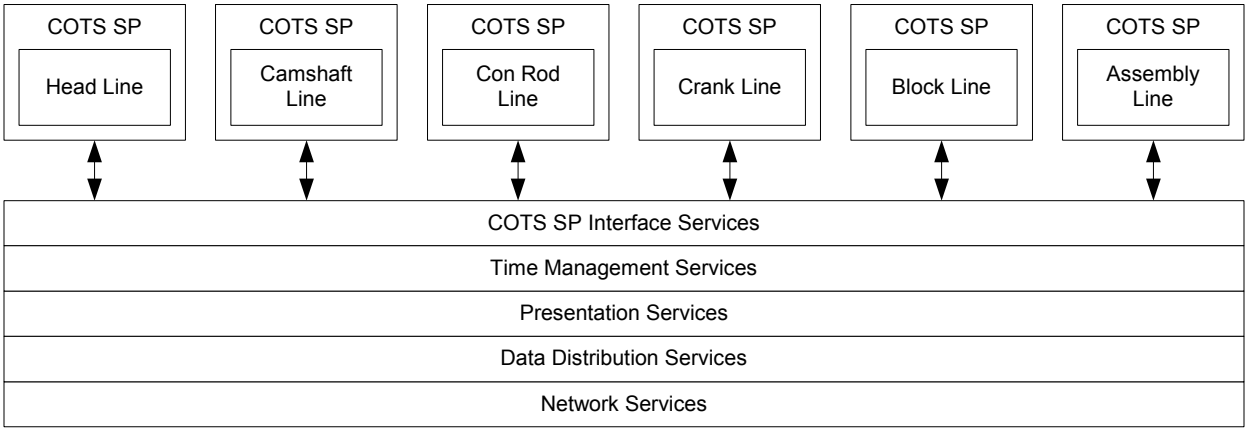

Figure 2: Distributed Simulation Middleware

4 TIME MANAGEMENT ISSUES

In our case study distributed simulation we have two types of federate: a producer (a production line) and a consumer (the assembly line). As has been discussed, the producer federates simply perform the simulation of their particular production line and produce engine part entities. These become timestamped event messages that the distributed simulation middleware must transport to the assembly line federate. Apart from the minor point of producing timestamped event messages, there are no time management issues in the producer federates.

The consumer federate and the distributed simulation middleware must deal with two time management issues: the ordering of timestamped event messages (external events) and the ordering of these with events scheduled by the simulation executive of the COTS simulation package as it simulates the assembly line model (internal events). In figure 2, these are dealt with by the Time Management Layer and the COTS SP Interface Layer respectively. We shall deal with each management issue in turn. We shall base the discussion around the conservative time management method.

4.1 Time Management of External Events

The time management layer service organises external events in the assembly line federate. Following the conservative method, a link exists between each producer federate and the assembly federate. It is assumed that timestamped event messages will be sent in ascending timestamp order (a valid assumption if (a) the sending federate performs its simulation correctly and (b) if there is only one exit point from the model being simulated). If this is the case then timestamped messages will enter a link buffer in correct order as well. Each link buffer has a link clock. The value of the link clock is either the timestamp of the message at the head of the link buffer or, if the buffer is empty, the timestamp of the last message to be held in the buffer. External events are ordered for introduction to the federate by repeatedly identifying the link buffer with the earliest clock. If the link buffer is empty, the time management service waits for another event message to arrive (as it cannot guarantee safety). If the link buffer has an event message, it will remove the event message from the link buffer and pass it to the interface service (the role of the presentation layer is outside of this discussion). We now address how we synchronise external events with internal events.

4.2 Time Management of External/Internal Events

The problem of managing external and internal events is an interesting one. As discussed above, the time management of external events follows a familiar algorithm to identify the next external event. However, the management of external and internal events presents a challenge.

A COTS simulation package typically possesses a simulation executive, an event list, a clock, a simulation state and a number of event routines (this is a gross simplification as these packages have many variants of this). The simulation state and the event routines are derived from the model that is implemented in the package. Initialising the simulation, events are placed on the event list (typically modelling entities arriving in the model). If we assume that the simulation executive uses some form of the three phase approach (TPA), the simulation first advances clock time to the time of the next event (the A Phase) and then executes that event (the B Phase) according to the event's routine (the event code). This may result in a change in the simulation state and the scheduling of new events on the event list. The simulation execute then determines if the changed state has enabled any conditional events (the C Phase). If any have been enabled, these events are executed in some priority order and any changes to the simulation state or event list are made. The simulation executive then makes a new cycle of the three phases.

The problem of time management of external and internal events is therefore as follows. If a federate consisting of a COTS simulation package and its model has an event list that contains internal events, and the time management middleware has ordered timestamped event messages, how can the simulation executive of the COTS simulation package determine the next event to process? Is the next event an internal one taken from the event list or an external one represented by the timestamped event message offered from the middleware? In our layered middleware, the Time Management Service identifies and offers the next external event and the COTS SP Interface Service controls the synchronisation of this with the package's event list. Several approaches to this synchronisation are possible.

4.2.1 Event List Externalisation

A simple solution to this is to remove the event list from the COTS simulation package and treat it as if it another link buffer. Events scheduled within the simulation package are externalised and placed on the link buffer representing the event list. The next event is determined by using the same conservative time management algorithm as described above.

The main problem to this approach is that it would take much redevelopment of a COTS simulation package for this to be accomplished. As one of the underlying themes of this work is to reduce the cost of technological intervention, this approach would be prohibitively expensive.

4.2.2 Permission Request

In this approach, the TPA is modified to request permission from the COTS SP Interface service. Prior to phase A, time advance, the modified form of the TPA asks permission to advance from the service. The service would respond by either (a) granting permission to advance, (b) passing an event with a timestamp, or (c) requesting the simulation executive to wait. In the case of (a), the timestamp of the next external event is greater than the scheduled time of the next (internal) event, the TPA would execute phase A by advancing to the time of the next event and then perform phases B and C as normal before making a new cycle of the modified TPA. If the timestamp of the next external event is less than the scheduled time of the next (internal) event, the middleware would convert the timestamped event message to a form that can be placed on the federate's event list. The TPA would then carry on by executing phase A, i.e. advancing to the time of the newly scheduled event. Phases B and C would be executed as normal. If the service could not determine the earliest safe timestamped message (as in the case when there is an empty link buffer), when the TPA next asked permission it would be requested to wait (as in case (c)). The TPA would be suspended until the time management service indicated a change of circumstances.

4.2.3 Incremental Advance

In this, rather than controlling the advancement of time though the TPA, this assumes that is it not possible to obtain access to the "next event time." Here the we must advance time by the smallest possible time unit of the federate. At each time advance any internal events are executed automatically by the TPA. When this has been accomplished a request is made to the COTS SP Interface service to determine if there is a new safe external event. Again three possibilities exist. If the service is aware of the next safe external event, and the timestamp of this greater than the next incremented time, the federate is allowed to make another cycle (a). If the timestamp of the next external event is equal to the next incremented time, the external event will be introduced for execution at the next incremented time (b). Finally, if the service cannot identify the next safe external event the incremental time advance will be halted until a new message arrives (c).

4.2.4 External Control

An alternative to making the TPA request permission is to effectively make the federate a slave of the COTS SP Interface service. The service first determines the course of action and then externally controls the behaviour of the federate's time advancement. Depending on the status of the link buffers, the service may make the federate (a) advance to a given time, (b) advance to a given time and then execute a new (external) event, or (c) do nothing. In the case of (a) the service has determined that it is safe for the federate to advance to a given time. The federate cycles through the TPA, advancing time until this "safe" time. If the service has identified a new safe external event, it orders the federate to advance until the timestamp of the event message and then introduces the new (external) event to the federate to be processed (as in (b)). The TPA method used by the federate must be modified appropriately either by re-executing the C Phase or by introducing the event just prior to its timestamp. The decision depends on event priority and actual package implementation. For example, if the C Phase cannot be manually executed, then a possible approach would be to stop the simulation at the smallest timestamp increment possible (i.e. after the completion of the C Phase) and then introduce the event for the next cycle of the TPA. Unfortunately this might be necessary due to the way in which a COTS simulation package is coded. For (c), if the service cannot achieve either (a) or (b) it must do nothing until a new timestamped message arrives. In this case the federate can do nothing either as it cannot determine the next safe event to execute.

4.2.5 Summary

In this section we have defined four strategies for the time management of external and internal events for COTS simulation package federates. We have attempted to described them "neutrally," i.e. not in terms of the HLA. The reason for this is that it is currently unclear how these might be interfaced directly, or indirectly via an ambassador, to the HLA time management services. Another reason for this is that it is hoped that COTS simulation package vendors and potential end users of distributed simulation might find this "technology neutral" discussion of relevance (as yet far more people understand the TPA than the HLA). A future paper will address this interfacing.

We will now discuss the implications of each of the strategies.

5 DISCUSSION

The previous section introduced four strategies for the time management of external and internal events for federated COTS simulation packages. Of these, which should be used? Two factors need consideration: technological intervention and performance. Technological intervention is important as this will ultimately determine the cost to the vendor (and therefore the end user!) in terms of COTS simulation package modification (we do not consider methodological implications here). Performance is important as the time management of external events is already a performance overhead that does not need adding to by further synchronisation cost.

The first approach of event list externalisation can be immediately discounted as the cost to the vendor of the intervention would be prohibitive. COTS simulation packages tend to use the notion of an event list but implement it in various ways to improve efficiency. Removing the event list is not a case of just "unbolting" the data structure and modifying the TPA (or similar) code. Adoption of this technique is too costly and, in cases where it might be possible, too package specific. The other approaches are based on a well defined relationship between the federate, the package that it encapsulates and the COTS SP Interface service. Most packages have the ability to make certain features accessible. For example, some allow the use of an external program (typically Excel or some Visual Basic application) to stop and start the run control, to introduce entities, or to alter a parameter in the model. Some use DLL "plug-ins" to do this, while others have a COM interface or a dedicated interface library. While virtually all have some interfacing ability, there is no notion of a "standard" interface. What do the three remaining time management approaches require in terms of an interface and package modifications?

In terms of time advance, permission request and incremental advance require that there is some modification to the simulation executive. Permission request requires that prior to time advance there is interaction with the Interface service. This will result in either a normal time advance, a new event being added to the event list and then a time advance or nothing. Incremental advance requires this and the additional timestep feature (which some packages do support). External control requires no such modification to the simulation executive as the time management service controls time advancement through an external interface. This approach, however, still requires the modification of the package to allow a new event to be placed on the event list. From a simple analysis all three approaches require a modified interface so that there is interaction with the time management service. Permission request and incremental advance both require interaction from the package to the Interface service and vice versa. External control is one directional as it instructs the package what to do - the next event time derived from an external event is derived from the time management service. In terms of performance, external control may well be the best option as it allows the package to advance with the least interaction. Additionally, it overcomes the need for the next event time. Incremental advance also overcomes this need but adds the major overhead of step by step time advance. Intuitively, external control also out performs permission request as many internal events can be processed without the overhead of a permission request protocol. Figure 3 shows this. This is an example taken from a comparative performance evaluation of time management for the assembly line federate. Permission request is compared again external control. As can be seen as the ratio of internal to external events per external event processed increases, the performance of the federate also increases. This is possibly an obvious result but one that is worth underlining in view of current practices.

External control appears to be the best option both in terms of technological intervention and performance. It does still have the problem of entering the new external

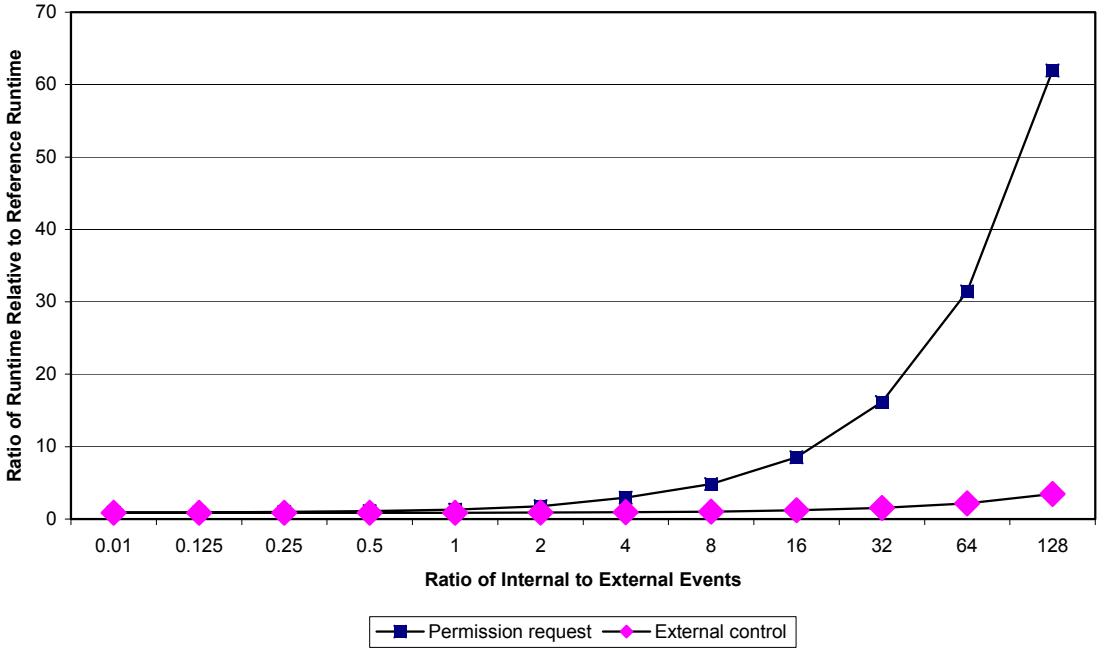

Figure 3: Relative Performance of Two Time Management Strategies

event into the event list so that it can be executed correctly. Fortunately, analysis of the interaction between the federates suggests a possible solution to this problem. Information exchanged between the federates in our case study is based on the movement of entities between the producing machine lines and the consuming assembly line. The external events introduced into the assembly line federate are not general – they relate specifically to the arrival of entities in one of the assembly line buffers. Although ideally the introduction of such an arrival event should be correctly placed in the event list, it is possible to avoid this intervention. Instead, we modify the model state (the contents of the particular buffer) at the appropriate time to reflect the arrival of a given entity.

This approach still has problems. Using the external control approach as an example, let us assume that we have advanced the federate to a given safe time. At this point, from an analysis of our link buffers, we determine our next external event. Within the federate our TPA simulation executive is ready to begin a new cycle. If we instruct the federate to advance to the timestamp of the external event, given that we are attempting to avoid any modification to the simulation executive, the TPA for that timestamp will execute without the introduction of our entity. If, however, it is possible to execute the A phase, introduce the entity into the appropriate buffer (effectively forcing the external event to be executed in the B phase) and then cycling the TPA to correctly execute the C Phase, then (given event priority is observed) the external event can be executed without need for modification to the C Phase. Other approaches are possible – this is presented here to suggest that the "event list" or "next event time" problems can be avoided with a minor modification to the TPA (or equivalent).

Before finishing this discussion of time management issues in COTS distributed simulation, we would like to draw the reader's attention to an associated problem. This is the *bounded buffer* problem.

6 THE BOUNDED BUFFER PROBLEM

In the previous section we outlined an approach to external/internal event time management that is good both in terms of technological intervention and relative performance. In this penultimate section we present a new problem based on our case study that presents a further challenge to distributed simulation. This is the bounded buffer problem.

In the case study presented in this paper, our original challenge was to create a distributed simulation that could be implemented using existing COTS simulation packages (in this case WITNESS). Our approach was made on the assumption that the input buffers to the assembly line were infinite. This was a valid assumption at the time as the end user understood the limitations of the approach – the approach still allowed some problems to be analysed but not all. Further discussion with the end user identified that ideally the input buffers be limited in space, i.e. they could only take a certain amount of entities. The input buffers are bounded.

Normally, when a machine produces a part, the part is removed from the machine and placed in the following buffer. If the following buffer is full, then the machine cannot give up the part and must block until it can pass on the part. A blocked machine cannot accept new work and will therefore cause its own buffer to fill. This in turn could cause another machine to block, etc. This bounded behaviour is important to model as it can effect the overall performance of a production line.

In terms of distributed simulation the implications of this are as follows. Figure 4 shows an example of this. A producer federate produces entities in workstation W1z. When these leave W1z they will instantly arrive in buffer B2a to be processed by workstation W2a. When B2a becomes full, W1z must keep hold of the next entity it processes until B2a has space. This happens when W2a processes one of its entities. When W2a does this W1z can immediately release its entity and begin work on another from its own buffer. Some kind of protocol is required so that this can be represented in such a way that the producer federate can carry the simulation of the rest of its model while W1z is blocked, i.e. the consumer federate must be able to inform the producer when to block and when to unblock.As a step towards a general solution to this, we present the following algorithms for discussion (algorithm 1 and 2 show this from the consumer and producer federate respectively). Better, more efficient forms of this are possible and performance evaluation of these approaches with respect to our case study are underway. In the following, InputBuffer and OutputWorkstation refer the input buffer (B2a) and output workstation (W1z) of the consumer and producer respectively.

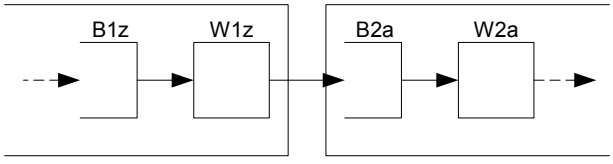

Figure 4: Bounded Buffer Problem

The message pull(Amount) sent from consumer to producer demands a certain amount of entities to be produced. The amount is determined by the space that the InputBuffer has at the decision simulation time. The implication is that while there is space, the producer is left to produce event messages until the InputBuffer may be full. The consumer advances time as discussed in section 4 (received as timeadv(Entity, Time)).

When the InputBuffer is full, adv(endNextEndProcTime(InputWorkstation)) is sent from the consumer to the producer to identify the next possible time that the InputBuffer could have space. The producer receives this as

```
while not end of simulation do
    //wait for instructions from Consumer
    get InputMsg;
    //The Consumer will either demand entities
    //or will give a safe Time to advance to
    if InputMsg = adv(Time) then
        //Simulate until Time.  When the
        //OutputWorkstation finishes an Entity
        //it will block until Time.  If
        //OutputWorkstation is blocked at Time
        //the Producer will unblock the
        //Workstation and send the Entity to
        //the Consumer at Time.  If not it will
        //advance until the next Time it
        //produces an Entity and then send it.
        send timeadv(Entity, Time)
    else //pull(Amount)
        simulate until Amount entities have
            been produced
    endif
endwhile
```

Algorithm 1: Consumer Federate

```
while not end of simulation do
    Amount = remainingAmount(InputBuffer);
    //If there is no Amount left then find
    //out when there will be and order
    //the Producer to simulate until that Time
    if Amount = 0 then
        //Get the Time at which an Entity
        //leaves the InputBuffer and send
        //it to the Producer.  Producer will
        //simulate until that Time.  The
        //Producer will block the
        //OutputWorkstation when it finishes
        //processing its next Entity.  At
        //Time, If the OutputWorkstation has
        //an Entity, the Producer will send it
        //to the Consumer.  If not, the
        //Producer will send the next Time
        //when it produces an Entity and then
        //wait
        send adv(endNextEndProcTime
            (InputWorkstation)) to Producer;
        //Wait for the response
        get InputMsg(Entity, Time);
        timeadv(Entity, Time);
        //Producer has produced an Entity.
        //Advance Time and add Entity to
        //buffer.
        //If Time = endNextEndProcTime
        //InputWorkstation will end
        //simultaneously and take an Entity
        //from InputBuffer.
    else
        send pull(Amount) to Producer
        repeat
            get InputMsg;
            timeadv(Entity, Time)
            //Producer has produced an Entity
            //Cycle the TPA as normal to
            //consume it
        until Amount entities received
    endif
endwhile
```

Algorithm 2: Producer Federate

adv(Time). It advances time to Time, simulating the production line as appropriate. If during this simulation, OutputWorkstation begins work on an entity, the workstation will block until Time and then send timeadv(Entity, Time). If not, the producer continues simulation past Time until OutputWorkstation finishes working on an entity. When it does, timeadv(Entity, Time) is sent to the consumer. The algorithm is expressed for the consumer and producer as shown in algorithms 1 and 2.

7 CONCLUSION

On the basis of a real world case study, this paper has discussed issues concerning time management that are unique to the distributed simulation of COTS simulation packages. It has also presented a "challenge" that must be overcome for an approach with wider applicability. The main point to this paper is that the interrelationships between models in such a federation may be exploited to provide workable solutions to distributed simulations of this type. It also shows that different approaches are possible given the requirements of the simulation (i.e. unbounded and bounded behaviour). From this it is trivial to argue that there might be other information sharing needs that are either known but not needed or unknown (and therefore cannot be planned for).

The reason why the two problems have been presented separately is to show that they can be considered in isolation. One makes supply chain simulation problems possible, the other production line problems. This is one of the major goals of HLA-CSPIF – to separate out requirements of distributed simulation. By doing this it is possible that practical solutions to distributed simulation are possible which do not need to be held back by other as yet unsolved problems. Whether certification is the key it is difficult to say. Steps are in progress to develop reference object models in conjunction with SISO from which "standard" approaches can be identified.

ACKNOWLEDGMENTS

The authors would like to take the opportunity to thank the reviewers of this paper who have provided stimulating comments. We have attempted to address all comments where possible and, where not, will follow these in future work (as possible collaboration)!

REFERENCES

Boer, C.A., A. Verbraeck and H.P.M. Veeke. 2002. Distributed Simulation of Complex Systems: Application in Container Handling. In *Proceedings of SISO European Simulation Interoperability Workshop*. Simulation Interoperability Standards Organisation, Orlando, Florida.

Sudra R., S.J.E Taylor and T. Janahan. 2000. Distributed Supply Chain Management in GRIDS. In *Proceedings of the 2000 Winter Simulation Conference*, J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds. Association for Computing Machinery Press, New York, NY. Association for Computing Machinery Press, New York, NY. 356-361.

Taylor, S.J.E., R. Sudra, T. Janahan, G. Tan and J. Ladbrook. 2001. Towards COTS Distributed Simulation Using GRIDS. In *Proceedings of the 2001 Winter Simulation Conference*, B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds. Association for Computing Machinery Press, New York, NY. 1372-1379.

Taylor, S.J.E., A. Bruzzone, R. Fujimoto, B.P. Gan, S. Strassburger and R.J. Paul. 2002a. Distributed Simulation and Industry: Potentials and Pitfalls. In *Proceedings of the 2002 Winter Simulation Conference*, , *E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, eds.* Association for Computing Machinery Press, New York, NY. 688-694.

Taylor, S.J.E., R. Sudra, T. Janahan, G. Tan and J. Ladbrook 2002b. GRIDS-SCS: An Infrastructure for Distributed Supply Chain Simulation. *SIMULATION*. 78(5): 312-320.

AUTHOR BIOGRAPHIES

SIMON J.E. TAYLOR is the 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, DS-RT and general co-chair of the UK Simulation Workshop Series. He currently leads the international COTS simulation package interoperability forum (HLA-CSPIF) through SISO (www.cspif.com). He is a Senior Lecturer in the Department of Information Systems and Computing and is a member of the Centre for Applied Simulation Modelling, both at Brunel University, UK. He was previously part of the Centre for Parallel Computing at the University of Westminster. He has an undergraduate degree in Industrial Studies (Sheffield Hallam), a M.Sc. in Computing Studies (Sheffield Hallam) and a Ph.D. in Parallel and Distributed Simulation (Leeds Metropolitan). His main research interests are distributed simulation and applications of simulation health care. He is also a member of the Purple Theatre Company. His email and web addresses are <simon.taylor@brunel.ac.uk>.

JON SHARPE is a Research Associate in the Centre of Applied Simulation Modelling. He is currently working on a distributed simulation infrastructure as part of on-going research work with Dr Taylor. His is also a member of the Purple Theatre Company.

JOHN LADBROOK has worked for Ford Motor Company since 1968 where his current position is Simulation Technical Specialist. In 1998 after 4 years research into modelling breakdowns he gained an M.Phil (Eng.) with the University of Birmingham. In his time at Ford, he has served his apprenticeship, worked in Thames Foundry Quality Control before training to be an Industrial Engineer. Since 1982 he has used and promoted the use of Discrete Event Simulation. In this role he has been responsible for sponsoring many projects with various universities. For the past seven years, he has been Chairman of the Witness Automotive Special Interest Group. His email address is <jladbroo@ford.com>.