AN ANALYSIS OF INTERNAL/EXTERNAL EVENT ORDERING STRATEGIES FOR COTS DISTRIBUTED SIMULATION

Simon J. E. Taylor
Navonil Mustafee
Centre for Applied Simulation Modelling
Department of Information Systems and Computing
Brunel University
UB8 3PH, Uxbridge, England
E-mail: simon.taylor@brunel.ac.uk

KEYWORDS
COTS Simulation Packages, Distributed Simulation, Discrete Event Simulation.

ABSTRACT

Distributed simulation is a technique that is used to link together several models so that they can work together (or interoperate) as a single model. The High Level Architecture (HLA) (IEEE 1516.2000) is the *de facto* standard that defines the technology for this interoperation. The creation of a distributed simulation of models developed in COTS Simulation Packages (CSPs) is of interest. The motivation is to attempt to reduce lead times of simulation projects by reusing models that have already been developed. This paper discusses one of the issues involved in distributed simulation with CSPs. This is the issue of synchronising data sent between models with the simulation of a model by a CSP, the so-called *external/internal event ordering* problem. The motivation is that the particular algorithm employed can represent a significant overhead on performance.

INTRODUCTION

Distributed simulation is a technique that is used to link together several models so that they can work together (or interoperate) as a single model. The High Level Architecture (HLA) (IEEE 1516.2000) is the *de facto* standard that defines the technology for this interoperation. Models, or federates, interoperate together to form a federation. Interoperation takes the form of organised communication of data specified in a Federate Object Model, using tables derived from the Object Model Template (IEEE 2000b), via supporting communication technology called a Runtime Infrastructure (RTI) (as defined by IEEE 2000a). Currently the HLA is most widely used in real-time simulation of defence related training problems.

In many areas of industry, COTS Simulation Packages (CSPs) are used to model systems in diverse domains such as commerce, defence, health and manufacturing. A CSP is a generic term that refers to a computer simulation package that is a visual interactive modelling environment that helps simulationists to build models, perform experiments, visualise and report during simulation projects. Although not exclusively, they are typically based on some variant of the discrete event simulation paradigm, i.e. models change state at discrete points in time by scheduled or conditional events and typically represent entities or objects (documents, patients, parts, trains, etc.) in some form that pass through networks of queues and workstations (work queuing at a desk in an office, patients waiting to see a doctor, parts buffered for machining, trains waiting at a station, etc.) Generally, each package has a range of basic model elements (queue, workstation, resource, source, sink, etc.) and advanced model element (conveyor, shift worker, warehouse, etc.) that are used to build a model via a drag and drop visual interface. Each model element can be modified as is required, either by a menu system or by a package programming language, to better represent the system being studied (for example the queuing logic of a queue or the behaviour of a resource). Entities or objects can be represented and differentiated by attributes. Terminology between packages differs as there is no generally recognized naming convention.

The creation of a distributed simulation of models developed in CSPs is of interest. The motivation is to attempt to reduce lead times of simulation projects by reusing models that have already been developed. For example Boer, et al. (2002) discuss the use of distributed simulation to simulate container handling at a port, while Sudra, et al. (2000) and Taylor, et al. (2002) discuss how distributed simulation can facilitate the modelling of supply chains and problems in the automotive industry. A factor that distinguishes this work from other research in distributed simulation is that interoperation must not only take place between models but also the CSPs in which the models reside.
This paper discusses one of the issues involved in distributed simulation with CSPs. This is the issue of synchronising data sent between models with the simulation of a model by a CSP, the so-called *external/internal event ordering* problem. This is of interest as the particular algorithm employed can represent a significant overhead on performance. The paper is structured as follows in section 2 we describe the external/internal event ordering problem in more

Proceedings 15th European Simulation Symposium
Alexander Verbraeck, Vlatka Hlupic (Eds.)

ISBN 3-936150-28-1 (book) / 3-936150-29-X (CD)

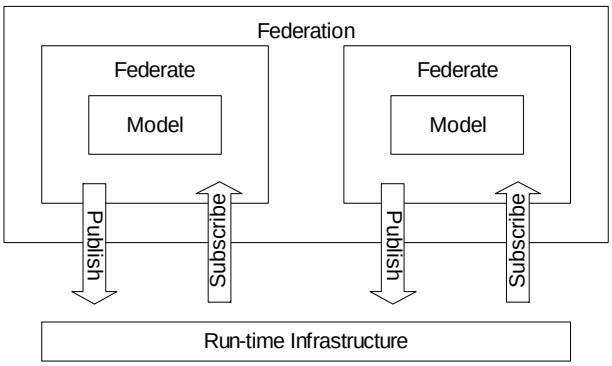

Figure 1: A Distributed Simulation Federation

detail. Section 3 introduces four algorithms that can be used for this problem. Section 4 presents some results. Section 5 ends the paper with some conclusions and further work.

THE EXTERNAL/INTERNAL EVENT ORDERING PROBLEM

The general problem of external/internal event ordering can be described as follows. Generally speaking, in a federates exchange information to perform a simulation of a particular system. Initially this is done on the basis of publication of information of interest and subscription to information of interest (publish-subscribe). A run-time infrastructure performs the actual exchange of information. Each federate performs the simulation of a particular model. The information exchanged between federates is therefore dependent on the models being simulated. Figure 1 shows a general distributed simulation federate.

In distributed simulation with CSPs, a federate contains a CSP which in turn contains a model. Data is generally on the basis of timestamped event messages. Event messages arriving at a federate from another are notionally organised by some causal ordering protocol and are introduced to the CSP and then the model being simulated by the CSP in timestamp order. These *external* events are ordered with the CSP/model's *internal* events according to some algorithm. Figure 2 shows these relationships.

To investigate the implications of this problem in a CSP

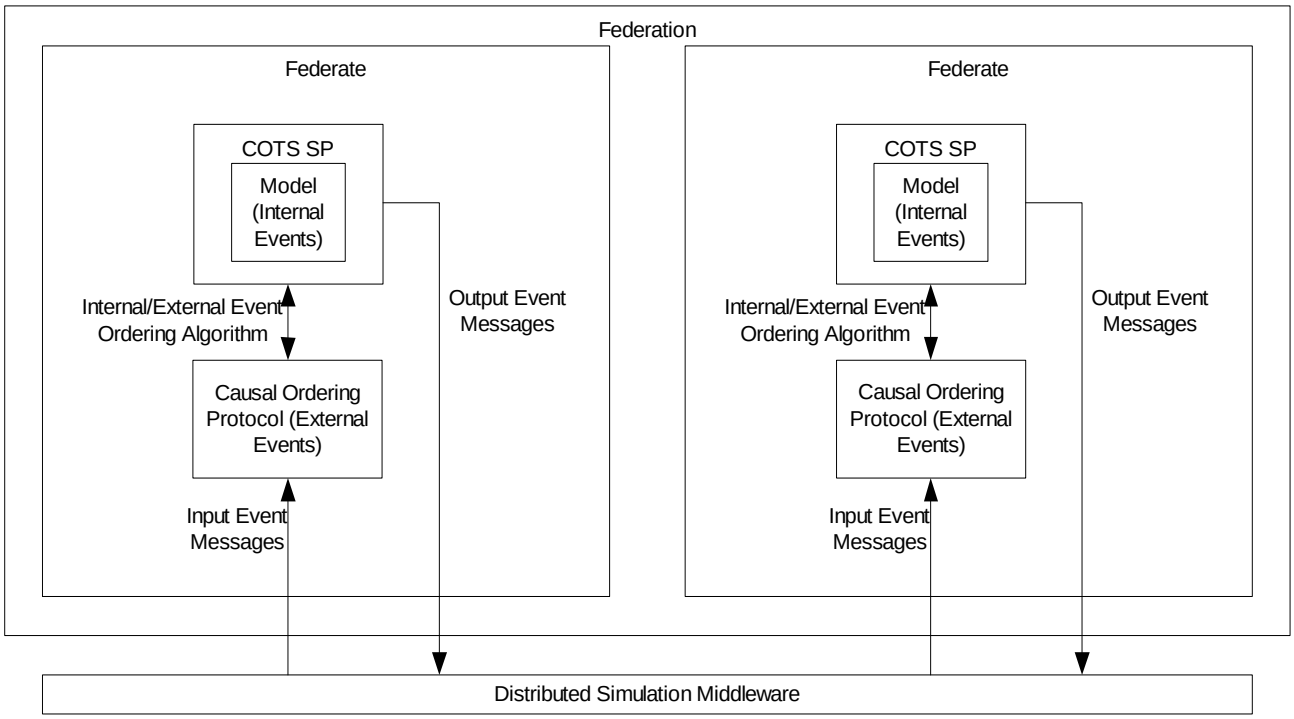

Figure 2: Distributed Discrete Event Simulation

federate, consider the following. A CSP typically possesses a simulation executive, an event list, a clock, a model state and a number of event routines (this is a gross simplification as these packages have many variants of this). The model state and the event routines represent the state of the model at a particular time and he logic by which it changes and are derived from the model that is implemented in the package (and therefore represent the model). Initialising the simulation, events are placed on the event list (typically modelling entities arriving in the model, e.g. raw materials arriving in a factory). 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 its event routine. This may result in a change in the simulation state, the scheduling of new events on the event list and the sending of new timestamped event messages. The simulation executive 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 again may result in a change in the simulation state, the scheduling of new events on the event list and the sending of new timestamped event messages. The simulation executive then makes a new cycle of the three phases. Algorithm 1 describes this (note that we assume that *Update event list*, *Update simulation state*, and *Send event messages* are conditional on the results of the B/C Phase).

```
while not terminated do
        Advance to time of next event (A Phase)
        Execute event (B Phase)
                Update event list
                Update simulation state
                Send event messages
        Test conditional events (C Phase)
                Update event list
                Update simulation state
                Send event messages
endwhile
```

Algorithm 1 The Three Phase Approach

The problem of external/internal event ordering is therefore as follows. If a federate consisting of a CSP and its model has an event list that contains internal events, and a causality ordering protocol has ordered the external events messages arriving from other CSPs, how can the simulation executive of the CSP 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 by the protocol? In the next section we review several algorithms that could be used to perform this ordering.

EXTERNAL/INTERNAL EVENT ORDERING ALGORITHMS

There are several possible algorithms that can be used to order external events with internal events. Each algorithm is defined on the basis of a relationship between a modified from of the CSP TPA and an external body that orders external events via a causality ordering protocol (the *external event manager* EEM). Each of these with their assumptions are now discussed.

Event List Externalisation

A simple solution to this is to remove the event list from the CSP and treat all events as external events. Events scheduled within the simulation package are externalised and ordered with the external events, i.e. any *internal* event becomes an event message. *Get next external event* represents the action of taking the next external event that has been identified by the EEM and introducing it to the CSP. Algorithm 2 describes this.

```
while not terminated do
        Get next external event
        Advance to time of next event (A Phase)
        Execute event (B Phase)
                Update simulation state
                Send event messages
        Test conditional events (C Phase)
                Update simulation state
                Send event messages
endwhile
```

Algorithm 2: Event List Externalisation

Permission request

In this approach, the CSP's TPA is modified to request permission from the EEM. Prior to the A phase, time advance, the modified form of the TPA asks permission from the EEM to advance to the time of the next event on its event list by performing *Request (permission (Next_Event_Time))*. This sends the time of the next event *Next_Event_Time* to the EEM. The TPA would then wait until the EEM responds with a *Reply Message* where *Message* can be *advance(Time)*, *event(Time)* or *wait*. The actions dictated by the reply from the EEM are either (a) to grant permission to advance to a given time *Time* by message *advance(Time)*, (b) to pass a timestamped external event *event* with timestamp *Time* by message *event(Time)*, or (c) to request the simulation executive to wait by message *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 therefore 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 (b), the external event would be passed to the simulation executive. 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 EEM could not determine the earliest safe timestamped message (as is possible with causal ordering protocols), when the TPA next asked permission it would be requested to wait (as in case (c)). The TPA would then be suspended until the EEM indicated a change of circumstances. Algorithm 3 describes this.

```
while not terminated do
        Request (permission(Next_Event_Time))
        if Reply (advance(Time)) then
                Advance to time of next event
                        (A Phase)
                Execute event (B Phase)
                        Update event list
                        Update simulation state
                        Send event messages
                Test conditional events (C Phase)
                        Update event list
                        Update simulation state
                        Send event messages
        else if Reply (event(Time)) then
                Advance to Time (Modified A Phase)
                Execute event (B Phase)
                        Update event list
                        Update simulation state
                        Send event messages
                Test conditional events (C Phase)
                        Update event list
                        Update simulation state
                        Send event messages
        else if Reply (wait)
                wait until notified
        endif
endwhile
```

Algorithm 3 Permission Request

Incremental advance

Rather than controlling the advancement of time in the CSP though the TPA, this algorithm assumes that is it not possible to obtain access to the "next event time." Here we must advance time by the smallest possible time unit of the CSP. Before each time advance the TPA performs *Request permission(Time_Increment)*. This sends the time of the next time increment *Time_Increment* to the EEM. The TPA must then wait until the EEM responds with a *Reply Message*, where *Message* can be *granted*, *event* or *wait*. The actions dictated by the reply from the EEM are either (a) to grant permission to advance by a single time increment by message *granted*, (b) to pass a timestamped external event *event* and to grant the time increment advance by message *event*, or (c) to request the simulation executive to wait by message *wait*.

The consequence of these messages are that if the EEM is aware of the next safe external event, and the timestamp of this greater than the next incremented time, the CSP is allowed to make another incremented advance (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 and the TPA is a allowed to make another incremented advance (b). Finally, if the EEM cannot identify the next safe external event the incremental time advance will be halted until a new message arrives (c).

```
while not terminated do
        Request (permission(Time Increment))
        if Reply (granted) then
                Advance by Time Increment
                        (A Phase)
                if next event then
                        Execute event (B Phase)
                        Update event list
                        Update simulation state
                        Send event messages
                        Test conditional events
                                (C Phase)
                        Update event list
                        Update simulation state
                        Send event messages
                endif
        else if Reply (event) then
                Execute event (B Phase)
                        Update event list
                        Update simulation state
                        Send event messages
                Test conditional events (C Phase)
                        Update event list
                        Update simulation state
                        Send event messages
        else if Reply (wait)
                wait until notified
        endif
endwhile
```

Algorithm 4 Incremental Advance

External control

An alternative to making the TPA request permission is to effectively make the CSP a slave of the EEM. The EEM first determines the course of action and then externally controls the behaviour of the CSP's time advancement. Depending on the status of the external events, the EEM may makes the CSP wait on *Wait (instruction)*. The possible values of *instruction* are *advance(Time)* and *event(Time)*. When the value of *instruction* is instantiated, the modified TPA may, if *instruction* equals *advance(Time)* execute as normal until *Next_Event_Time* is greater than *Time* (a), or if *instruction* is *event(Time)* execute as normal until *Next_Event_Time* is greater than *Time* and then execute the new event *event* (b). In the case of (a) the EEM has determined that it is safe for the CSP to advance to a given time. The CSP cycles through the TPA,

advancing time until this "safe" time. If the EEM has identified a new safe external event, it orders the CSP to advance until the timestamp of the event message and then introduces the new (external) event to the CSP to be processed (as in (b)). If neither is the case then the CSP waits until an instruction is sent by the EEM, i.e. the EEM cannot identify a safe course of action.

```
while not terminated do
        Wait (instruction)
        if instruction = advance(Time) then
                while Next_Event_Time < Time do
                Advance to time of next event
                        (A Phase)
                Execute event (B Phase)
                Update event list
                Update simulation state
                Send event messages
                Test conditional events (C Phase)
                Update event list
                Update simulation state
                Send event messages
                endwhile
        else if instruction = event(Time) then
                insert event(Time) in event list
                while Next_Event_Time <=
                        event(Time)
                Advance to time of next event
                        (A Phase)
                Execute event (B Phase)
                Update event list
                Update simulation state
                Send event messages
                Test conditional events (C Phase)
                Update event list
                Update simulation state
                Send event messages
                endwhile
        endif
endwhile
```

Algorithm 5 External Control

Summary

This section has introduced four algorithms to solve the external/internal event ordering problem. The next section reports on some results obtained from experimentation with programs based on the algorithms.

EXPERIMENTS

As the purpose of the experiments are to investigate the external/internal event ordering problem, not external event ordering, we shall assume that there is no wait time, i.e. all external events have been produced and the EEM has ordered them. This is an artificial but valid assumption as the results of these experiments will give us a base line on performance which will degrade under conditions where algorithms are forced into their wait state caused by the EEM being under populated with event messages (or other). Timestamps of internal and external events were arbitrarily selected to suit the experiments and are deterministic. Experiments were be performed on the basis of event density, *D*, which is the ratio of the number of external events *X* to the number of internal events *I*, i.e. defined as $D = X/I$. The number of internal events were held constant at 1000. The processing time for an event executed by the artificial CSP was held at 5ms. The data points on the graph are for an average based on 10 runs. The data points on the graph are for values of D against average time to process 1000 internal event messages. 0.001, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 3, 4, 5. The program was implemented as a client-server system in Java under Microsoft Windows 2000 using sockets. The computer was an Intel Pentium III processor 744 MhZ with 256Mb RAM running Windows 2000. Figure 3 shows these results. Note that the results for the incremental advance algorithm are excluded as they are a magnitude greater than results for the other algorithms.

CONCLUSIONS

Of our four algorithms, *external control* appears to be the "winner." *Event list externalisation* and *permission request* give similar results, while *incremental advance* gives a magnitude worse performance. This is unsurprising as *external control* allows the CSP to proceed with the least interaction. However, the selection of the "best" solution to our problem cannot be made just on performance. The problem faced by interoperating CSPs is that many of the features that are required for the ordering of external and internal events are sometimes not obtainable. For example, some CSPs have COM controls that make all data structures (including the event list) easily accessible. Others have little in the way of accessibility – even the time of the next event is hidden. For example, even though *external control* may appear to be the best ordering algorithm, only *incremental advance* may be possible as there is no method of advancing the simulation clock to a given timestamp. Further work will investigate this problem of compatibility.

It is hoped that the work presented in this paper will stimulate other external/internal event ordering algorithms. This ordering represents a major performance overhead and attempts to reduce this overhead can only make distributed simulation a more attractive possibility.

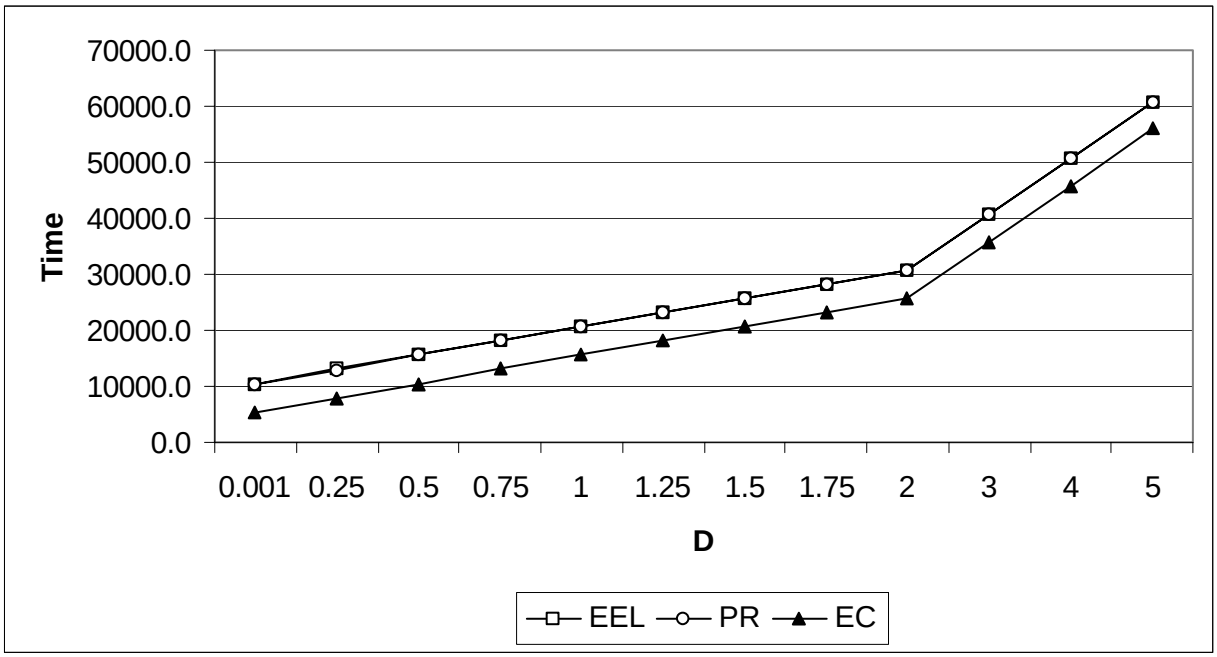

Figure 3: Comparison of External/Internal Event Ordering Strategies

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.

IEEE 2000a. IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification. IEEE Std 1516.1-2000. IEEE Computer Society, New York, NY.

IEEE 2000b. IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT) Specification. IEEE Std 1516.2-2000. IEEE Computer Society, New York, NY.

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

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. 1372-1379. Association for Computing Machinery Press, New York, NY.

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, San Diego, CA. 688-694. Association for Computing Machinery Press, New York, NY.

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.