An Investigation into Time Management in COTS-based Distributed Simulation using HLA

H E N R I K   Å H L A N D E R

An Investigation into Time Management in COTS-based Distributed Simulation using HLA

Henrik Åhlander

Master of Science Thesis
Performed at Brunel University, London, UK

August 2004

Examiner

Prof. Rassul Ayani
Microelectronics and Information Technology
Royal Institute of Technology

Supervisor

Dr. Simon J. E. Taylor
Department of Information Systems and Computing
Brunel University

Abstract

This report investigates time management in COTS based distributed simulation using HLA. Discussions are limited to discrete event simulation and conservative algorithms. The COTS Simulation Package Emulator (CSPE) is used as an experimental tool. Its architecture is modified to use HLA as middleware.

A time management algorithm is designed and implemented in Java for an asynchronous entity passing using the non-persistent object type interactions and using the two different time management methods TimeAdvanceRequest (TAR) and NextEventRequest (NER).

Experiments are carried out on an isolated local area network with seven computers. The HLA RTI 1.3-NG Version 5 is used. Three variables, external/internal event ratio, workload and lookahead, are varied in different experiments and tested in four federate configurations. A middleware based on the Chandy-Misra-Bryant (CMB) algorithm is also tested on the same hardware and under the same circumstances.

The results are compared and analysed. Both TAR and NER have the same results in all experiments and the analysis shows that these methods are actually doing the same in the developed algorithm. Their results are found to be equal or faster than CMB in all federate configurations except the configuration that doesn't have any feedback loop, in which CMB is a little bit faster than HLA.

Acknowledgements

First of all, I want to thank my supervisor Dr Simon J. E. Taylor at the Department of Information Systems and Computing, Brunel University for his tremendous support during my work with the thesis. He has encouraged me and given me advice during my five months in England. I also want to thank my examiner Prof. Rassul Ayani at the Department of Microelectronics and Information Technology, KTH for his support.

I would like to express my gratitude to Brunel University in London for letting me come and stay at the university. I want to give a special thank to Navonil Mustafee for taking time to explain the CSPE architecture and for helping me with the verification of the experiments. I also want to thank the PhD students in the Attic for the good time we had together.

Thanks to Jacqueline Brodie, Mattias Krantz, my brother Magnus Åhlander, and my sister Anna Åhlander for your help with reading the report and all the valuable comments you have given me.

Table of Contents

List of Figures

List of Tables

1   Introduction

The aim of this thesis is to investigate time management in Commercial Off The Shelf (COTS) based distributed simulations using HLA. This chapter is divided into five sections. The first section will give a short introduction. The second will motivate why the aim is valuable. The third will describe the objectives that have been identified to fulfil the aim. The fourth will describe how they will be carried out. The last section will show the organisation of this report.

1.1   Preliminaries

Banks et al. (2001) define a simulation as "the imitation of the operation of a real-world process or system over time" (p. 3). Computer simulations are useful when you want to know something about how a real-world system works without actually building the system. They are today used in many different areas, from manufacturing and health care to business and military applications.

COTS simulation packages (CSPs) can be used to build simulations models (Swain 2003). These packages are often very user friendly with graphical user interfaces for building and visualization of the model and with tools for statistical analysis and reporting.

Distributed simulation is concerned with the execution of simulation on computers that are geographically apart (Fujimoto 1999).

The High Level Architecture (IEEE 1516.2000) is an architecture developed for reusability and interoperability of simulations. It was first developed by the Defense Modeling and Simulation Office (DMSO) in the United States for military applications but has become a de-facto standard for distributed simulation.

There have been many attempts to connect COTS simulations packages to each other but the many approaches are currently not compatible (Taylor, et al., 2003). There are even differences between the approaches that uses HLA. To unify research and development the HLA-CSP Interoperability Forum (HLA-CSPIF) has been established. The goal of the forum is to inform and to create standards.

Even though some COTS simulations packages support the possibility to link externally written code in C++, VBA or Java to the simulation models, many of the internal functions and variables needed to create a distributed simulation are not available to the externally linked modules (Taylor, et al., 2003).

The COTS simulation packages and models involved in a distributed simulation must agree on the format and manner data is passed between the models. The representation of an object and its attributes must be equal at the sending and the receiving models and they must also agree what mechanism for object transfer that should be used since several exist. (Taylor, et al., 2003)

1.2 Motivation

The overhead in a distributed simulation can be split into three parts, see Figure 1-1.

The first part is the overhead in the CSP execution. This overhead is difficult to reduce without access to the source code and must be reduced by the CSP manufactures.

The second part is the overhead for the ordering of events and time management between the models involved in the distributed simulation. This could be reduced using smarter algorithms.

The last part is the overhead for the network and the hardware that is used. Faster networks, network protocols and hardware could speed up the simulation.

CSP execution
Event-ordering and time management
Network and hardware

Figure 1-1. Overhead in a distributed simulation divided into three parts.

As we can see, the ordering of events and time management is important for the overall performance of the simulation. HLA supports several methods for this management. It would be beneficial to see how these methods can be used in a COTS based distributed simulation and how well they perform in comparison to other types of middleware.

1.3 Aim and Objectives

The aim of this thesis is to investigate time management in COTS based distributed simulations using HLA. To achieve the aim six objectives have been defined. They and their justifications (in italics) are presented below:

1. Perform a literature review

By reading relevant papers I will get background and better understanding of the problem.

2. Create a framework in which time management in COTS based distributed simulations using HLA can be investigated

By creating a framework based on the literature survey it will be possible to identify which parts to implement later.

3. Implement interesting areas of the framework

By implementing interesting areas of the framework it will be possible to do experiments and get results.

4. Carry out experiments and evaluate the results

By doing experiments and evaluating the results it will be possible to find out which mechanisms are most efficient.

5. Propose efficient mechanisms

By proposing the mechanisms that are found to be efficient the result of the work can be beneficial for others.

6. Disseminate the research findings.

By presenting the research and the results in a report and in a presentation others will be able to utilize the results in their research.

1.4   Research Approach

For each objective a method has been defined:

1. Perform a literature review

Relevant papers concerning simulation, distributed simulation and HLA will be read.

2. Create a framework in which time management in COTS based distributed simulations using HLA can be investigated

Identify and explain which mechanisms for time management exist based on the literature survey. Some interesting methods will be chosen for implementation and experiments.

3. Implement interesting areas of the framework

Implementation will be made using the latest version available of Java to make it compatible with the work earlier made at Brunel University by Mustafee (2003). HLA RTI 1.3-NG Version 5 will be used since this is the RTI version the university has access to.

4. Carry out experiments and evaluate the results

The benchmark proposed in Mustafee (2003) will be used since this is a desire from my supervisor. The benchmark is specially developed for distributed simulations. Experiments will be made on machines with equal hardware connected to each other over a separated local area network.

5. Propose efficient mechanisms

Efficient mechanisms will be proposed and compared to the performance of another middleware based on the Chandy-Misra-Bryant algorithm.

6. Disseminate the research findings

The work and the results will be presented in a report. A presentation will be held in Sweden.

1.5   Organisation of the Report

The remaining six chapters of this report are each connected to one of the objectives identified above. Each chapter is outlined below:

Chapter 2 gives a background to simulation, COTS packages, distributed simulation and HLA. The purpose of this chapter is to present the relevant knowledge needed to understand the report.

Chapter 3 offers a deeper understanding of the problem to be solved and why it is interesting to solve it. Different methods to solve the problems are discussed.

Chapter 4 describes the design and implementation in general for the possible different methods. Details will be presented for the method that will be implemented.

Chapter 5 describes the benchmark used and how the experimentation was done. The equipment and programs used in the experimentation will be explained. The results will be shown in graphs together with the results of the middleware using the Chandy-Misra-Bryant algorithm.

Chapter 6 analyses the results and discusses advantages and disadvantages of the experimental tool that has been used and HLA as middleware for COTS simulation packages.

Chapter 7 summarizes the report and presents what conclusions that can be made from the analysis of the results. Future work will also be discussed.

2 Background

This chapter will give a background to simulation, COTS packages, distributed simulation and HLA. The purpose of this chapter is to give relevant knowledge needed to understand the report.

2.1 Simulation

2.1.1 Introduction

As already noted in chapter one, Banks et al. (2001) define a simulation as "the imitation of the operation of a real-world process or system over time". Simulations are useful when you want to know something about how a system works without actually building the system.

This can increase knowledge of the system and save money since it is possible to test different approaches and parameters to find the best solution(s). Sometimes is it also dangerous to build the real system, as in the case of nuclear and weapon system. Simulations are today used in many different areas, from manufacturing and health care to business and military applications.

It is not always good to do a simulation. It can, for example, cost more money than it saves since simulation modelling and analysis requires special training and takes time. An analytical solution to the problem might also be possible and easier to do.

A simulation model breaks down into certain components. The objects in a system are called *entities* and its properties are called *attributes*. An *activity* is a time period with a specified length. All variables that are needed to describe the system at any time are called the *state variables* and define the *state* of the system. An *event* is an occurrence that may change the state of the system and occurs at an instant of time. (Banks, et al., 2001)

As an example of this, if a factory production chain is being studied, different parts might enter the factory, being put together and then exit the factory as finished products. These parts can be some of the entities in the model. When a part is being processed in a workstation this is an activity that takes a certain time. The number of available workstations can be one of the state variables of the system. A simulation can be a good idea if the company is considering buying a new workstation and wants to know how much it will improve the factory's total performance. This is good to know before the decision to buy that workstation is taken.

Simulations can be built using different approaches but common for most of them is that they have a *simulation executive*, a *simulation clock* and an *event list*. The simulation executive has the overall control of the system. The simulation clock keeps track of the current simulation time, which determines what events that should be executed. The event list contains all known future events, often ordered by their time stamp.

2.1.2 Discrete Event-based Simulations

Real-world systems can usually be divided into *continuous* and *discrete systems* (Law and Kelton, 2000). In a continuous system the state of all variables is changed continuously over time. For example, the level of water behind a dam and the temperature in an electric component.

In a discrete system the state of all variables is only changed at specific points in time. For instance, the case of the numbers of customers waiting in a queue and the numbers of calls going on in a telephone network. Simulations of discrete systems are called *discrete event-based simulations*.

This report will limit its discussions to discrete event-based simulations.

2.1.3 Execution Mechanisms

Several different mechanisms exist for the simulation execution. The most common in discrete event-based simulations is *the three phase method,* but there are also different activity based and event based methods. The three phase method is the only method that will be described here, see Figure 2-1. The advantages of this method are that it is a safe and effective method that reduces risk of error, especially in large and complex models. (Paul and Balmer, 1998)

```
        ┌─────────────────────────────────┐
        │  Start simulation, initialization │
        └─────────────────────────────────┘
                        │
                        ▼
        ┌─────────────────────────────────┐◄──────┐
        │      A. Move simulation clock    │       │
        └─────────────────────────────────┘       │
                        │                          │
                        ▼                          │
        ┌─────────────────────────────────┐       │
        │      B. Execute bound events     │       │ No
        └─────────────────────────────────┘       │
                        │                          │
                        ▼                          │
        ┌─────────────────────────────────┐       │
        │   C. Execute conditional events  │       │
        └─────────────────────────────────┘       │
                        │                          │
                        ▼                          │
        ┌─────────────────────────────────┐       │
        │        Interrupt or finish?      │───────┘
        └─────────────────────────────────┘
                        │  Yes
                        ▼
        ┌─────────────────────────────────┐
        │      Stop simulation, report     │
        └─────────────────────────────────┘
```
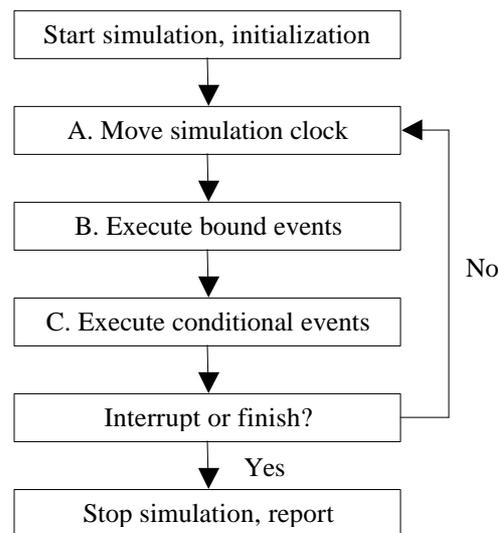
Figure 2-1. The Three Phase Method
(Paul and Balmer, 1998).

In the first phase (A) is the simulation clock is changed to the time of the next event in the event list. Only events can change the state of the system and since there were no events between the old time and the new time nothing could have happened with the simulation there. In the second phase (B) all events at the new time are executed. In the third and last phase (C) all conditional events are tested to see if any of these can be executed. An example of a conditional event can be an entity waiting for a workstation to be available.

6

2.2 Distributed Simulation

2.2.1 Introduction

Distributed simulation is concerned with the execution of simulation on computers that are geographically apart (Fujimoto 1999). For example, it can be motivated to distribute the simulation if a simulation model or some information used in the simulation (e.g. databases) only exist on a specific location. It might also be motivated if you want to use many models that already exist, maybe built using different CSPs, to build a bigger one.

If factories are used as an example again, a distributed simulation can be useful if the company wants to simulate a whole production chain where many factories are involved. Suppose some of these factories are owned by other companies that do not want to share information about how their factories are built. By using a distributed simulation the factories' simulation models can be kept secret at computers located at the different companies. The only data that is shared over the network is entities arriving and leaving the factories.

An important distinction between a sequential and a distributed simulation is that the state variables and the system objects cannot be shared in a distributed simulation as they can be in a simulation running on just one computer. This is also the case with the simulation clock and the event list. How data is shared and what data that is shared between the computers will dramatically affect the over-all performance of the distributed simulation.

2.2.2 Parallel Simulation

Parallel simulation, an area closely related to distributed simulation, concerns the execution of simulations on a tightly coupled computer system, e.g. a supercomputer or a shared memory multiprocessor (Fujimoto 2003). Parallel simulation can be used to reduce the length of the simulation execution time by letting more processors work or to enable larger simulations since they may not be enough memory on one single machine.

Distributed and parallel simulation have historically been differentiated as two different areas but with new computer paradigms such as clusters of workstations and grid computing there is no longer a clear border between these areas (Fujimoto 2003). This report, however, limits its discussions to distributed simulations on computers that are geographically apart and connected to each other over a network.

2.2.3 Time Management

Events sent in a distributed simulation can be classified as *time stamp ordered* (TSO) events or *receive order* (RO) events among others. TSO events must be executed at specific time stamps while RO events are executed in the order they are received.

To ensure that events are processed in the correct order and to make sure that a repeated simulation with the same input produces the same result, synchronisation between the computers involved in a distributed simulation is needed. This synchronisation is called *time management*.

Time management algorithms can be classified as *conservative* or *optimistic*. Conservative algorithms always makes sure events are "safe" to execute before they are executed. An event is said to be safe when it can be guaranteed no other events will arrive with a smaller time stamp. All events are always executed in the correct order.

To know which events that are safe to execute each model, below referred as logical process (LP), must know at what time stamps all other models earliest can produce new events. To be able to do that the *lookahead* values are used. Fujimoto (1999) defines lookahead as: "If LP is at simulation time T, and it can guarantee that any message it will send in the future will have a time stamp of at least T+L regardless of what message it may later receive, the LP is said to have a lookahead of L". A model's lookahead value is set depending on the structure of the model and must be well known by the other simulation models before the distributed simulation may start.

Optimistic algorithms allow events to be executed out of order, but if this occurs a recovery process will start to roll back time and to reset the state variables to their values at the time where the wrong event was executed. This means an optimistic algorithm must save information for this recovery during the execution. The two main problems with this approach are I/O operations that cannot be undone and memory resources.

Whether conservative or optimistic algorithms perform better, depends on the application (Fujimoto 2003). This report limits its discussions to conservative algorithms.

2.2.4 The Chandy-Misra-Bryant (CMB) algorithm

Synchronization problems and solutions were first developed in the late 1970's. Bryant (1977) and Chandy and Misra (1978) developed a conservative algorithm that has been referred as the *Chandy-Misra-Bryant (CMB) algorithm.*

If a distributed simulation uses this algorithm, each LP establishes a direct link to all other LPs it want to be able to send messages to. All messages must be sent with non-decreasing time stamps and all LPs must have one buffer for each incoming link. Figure 2-2 illustrates this with an example. LP A receives messages from the source. It processes them and then sends them to either LP B or C. They are then sent to LP D before they leave to model through the sink.
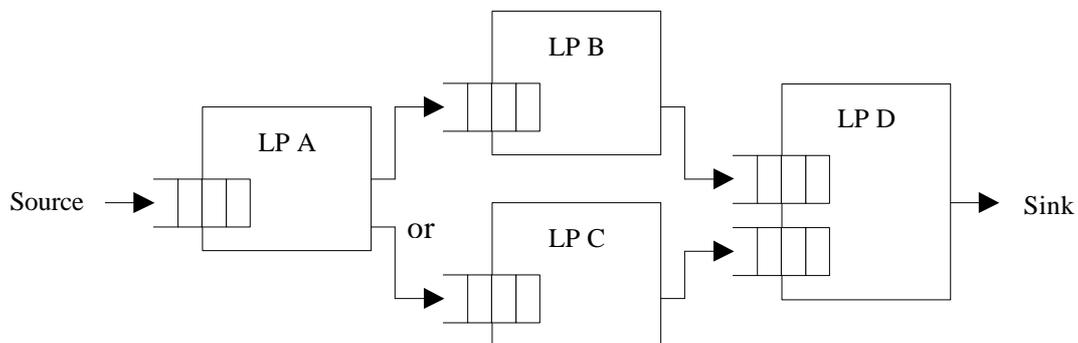
Figure 2-2. Model using the Chandy-Misra-Bryant algorithm

To know what message a LP should execute next, it scans all its buffers for the message with the lowest time-stamp. If any buffer is empty it must wait until it receives a message from that link since that message might have a lower time stamp than any in the other buffers. As soon as the LP knows what message it should execute next, it moves its simulation clock forward to this time, remove the message from the buffer and execute it.

In the example above, LP A, B and C only have one buffer each, which mean they will not have any problem with deciding what message to execute. If they have any messages in their buffers they will execute the one with the lowest time stamp. LP D has two buffers since it can receive messages from both LP B and C. It cannot proceed until it has received a message from both B and C.

One problem with the simplest version of the CMB algorithm is that deadlock situations can appear. Suppose LP A chooses to send all messages to LP B and none to LP C. In this case LP C will never send anything to LP D. If that is the case, a deadlock situation has occurred since LP D is waiting until it has received a message from both B and C. To avoid situations like this *null messages* can be used. A null message is an empty message with only a time-stamp.

There are several ways to use null messages. One simple solution is to send a null message to all outgoing links every time the simulation clock has changed in a LP. This will allow other LPs to proceed. In our case, LP A would also send a null message to LP C when it is sending something to LP B. As soon as LP C executes that null messages it would send a null message to LP D.

The problem with a lot of null messages being used is that they might slow down the overall performance of the simulation. More optimised algorithms exist that uses fewer null messages (Fujimoto 1990). Other solutions are constructed to detect and recover from deadlock situations (Chandy and Misra 1981).

The CMB algorithm will be further studied in Section 6.3.2.

2.2.5 HLA

An architecture developed for the reusability and interoperability of simulations is the High Level Architecture (IEEE 1516.2000). It was originally created by the Defense Modeling and Simulation Office in United States for military applications but has during the latest few years become a de-facto standard for distributed simulation.

The HLA consists of a set of rules and an interface specification but does not prescribe any specific implementation and computer language. Below follows a short introduction to HLA, its terms and components.

The Run-Time Infrastructure

All LPs that are involved in the distributed simulation are called *federates*. Federates are connected to each other in a *federation* through the *Run-Time Infrastructure* (RTI). The RTI makes sure simulation time advances correctly in every federate and also handles object updates and message passing between federates.

The RTI is divided into three components. The RTI Executive (RtiExec) creates and manages multiple federations within a network. For every new federation a Federation Executive (FedExec) is created that manages joining and resigning of federates. The RTI Library (libRTI) is built-in in all federates and provides HLA services to federates. (DMSO)

The RTI Library has two classes, the RTI Ambassador and the Federation Ambassador, that are used for communication between the federate and the RTI. The RTI Ambassador is used when the federate wants to use a service provided by the RTI. This is done with function calls. The Federation Ambassador, an abstract class that must be implemented in federates' code, is used for the call-back functions each federate is obliged to provide. This is illustrated in Figure 2-3.
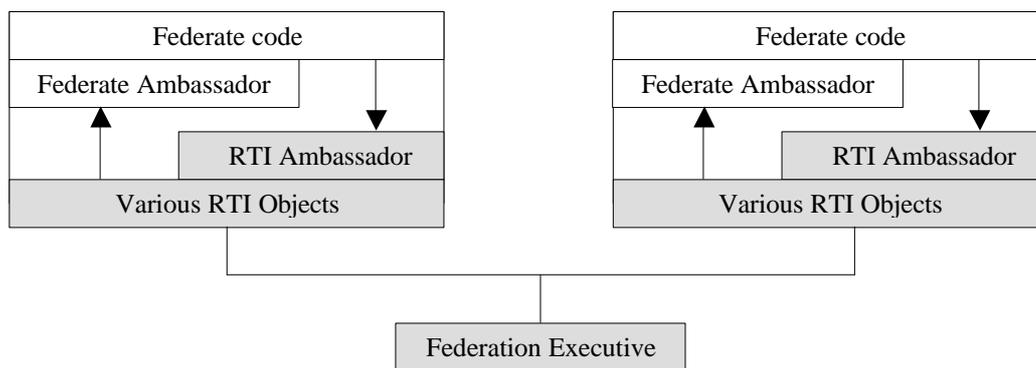
Figure 2-3. Different RTI components in a federation.

Object Management

HLA supports an advanced object management. All objects that can exist in the federation are declared in the FED file. After a federate has joined a federation it informs what objects and attributes it can publish (i.e., generate) and what objects and attributes it wants to subscribe to (i.e., receive).

Objects can be wholly owned by one federate or shared between many federates. If an object is shared, only one federate can own a specific attribute at any time. The owner of an attribute is responsible for updates of the values. An attribute can be *static* or *dynamic*. A static attribute is owned by the same federate throughout its life time and no other federate can update its value. The ownership of a dynamic attribute can be moved from one federate to another using the *push* and *pull* methods.

The push method is used when a federate informs RTI that it does not want to own the object any more. RTI will then inform all other federates that this object is available and those who are interested can request ownership.

The pull method is used when a federate asks RTI if it can receive ownership of an object owned by another federate. The RTI will then ask the current owner or owners, if they can release their ownership. After this is done the RTI will inform the requester that it has gained ownership.

Interactions are a special type of objects that can be sent between federates. They are non-persistent. After an interaction has been received by the receiving federate it is

removed from the RTI. It is not possible for a federate to publish or subscribe only some of the object's attributes. They have to choose between "all" or "nothing".

Time Management

The RTI handles the time management within the federation. Different federates in a federation can use different time management policies. The time in a federate always moves forward but the current time in all federates may not be the same.

A *regulating* federate is a federate that regulates the time for federates that are *constrained*. A federate can be regulating and/or constrained or neither regulating nor constrained. A regulating federate may associate its activities (e.g. updating objects and sending interaction) with a specific time stamp. A constrained federate subscribing these objects and interactions will receive them at the specified times.

This can be illustrated with an example. Suppose we have four federates in the distributed simulation, see Figure 2-4. The first one is only regulating, the second one is only constrained, the third is both regulating and constrained and the last one is neither regulating nor constrained. The two regulating federates (1 and 3) may not generate any TSO events with a time stamp lower than their current time + their lookahead. The two constrained (2 and 3) may not advance further than the *Lower Bound Time Stamp* (LBTS). The LBTS is the earliest time a regulating can generate an event. Federate D may advance to any time.
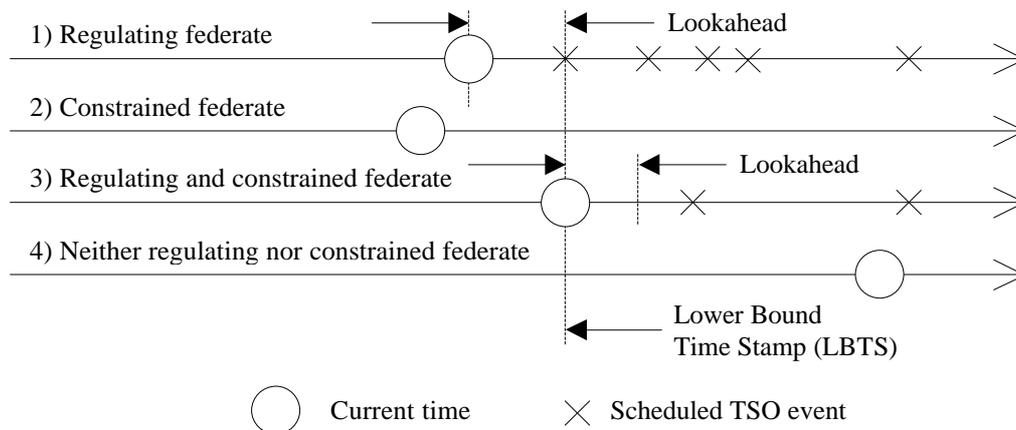
Figure 2-4. Example with regulated and constrained federates.
The federates time axes are shown together with their current time and scheduled events.

All federates that are regulating or constrained must continually request new times from the RTI. A call-back will occur to the federate ambassador when the request is granted. A federate that is only regulating will get this call-back directly while a federate that is constrained will have to wait until the RTI can guarantee that no regulating federate will send anything with a time stamp smaller than the time that will be granted.

The HLA mainly has three different methods that can be used for time requests. The *timeAdvanceRequest* method will grant a time equal to the time requested, as soon as the RTI can guarantee that all TSO events with a lower or equal time stamp to the time requested has been delivered to the federate. The *nextEventRequest* method will

grant a time equal to the lowest of the requested time and the time stamp of the next TSO event. The call-back will occur as soon as all events with a time stamp equal to the time that will be granted has been delivered. The *flushQueueRequest* method is used for optimistic simulations and will not be discussed in this report.

All federates must specify their lookahead values (see section 2.2.3) when they become regulating. When a regulating federate requests a new time with the time-AdvanceRequest method it promises that it will not send any time stamp ordered events with a time stamp lower than the requested time + lookahead. If the federate instead requests the new time with the nextEventRequest method it promises that it will not send any time stamp ordered events with a time stamp lower than the granted time + lookahead. A regulating federate can have a zero-lookahead but this is not desirable since constrained federates have to wait more and it can also invoke design problems.

Table 2-1 shows a summary of the methods that can be used for time requests.

Method	*The time that will be granted is...*	*The federate promise it will not generate any TSO events before...*
timeAdvanceRequest	the requested time	requested time + lookahead
nextEventRequest	what is lowest of the requested time and the next TSO event.	granted time + lookahead

Table 2-1. Summary of methods for time requests.

2.3 COTS Distributed Simulation

2.3.1 COTS Simulation Packages

Commercial Off the Shelf (COTS) simulation packages (CSPs) can be used to build and execute computer simulations. There are over twenty packages (Swain 2001), e.g. Arena, Automod, Sigma, Simul8 and Witness.

Typically they allow the user to build, visualize, save and reuse simulation models, often using a user-friendly graphical user interface. They may also have tools for statistical analysis and reporting. Furthermore, they usually have built-in support for various random distributions that can be used for arrival and service times of entities.

2.3.2 Access Internal Data Problems

Before data can be transferred from one model to another, the data must be accessed. The problem is that CSPs hide access to internal functions and variables that is needed in the distributed simulation. While almost all CSPs have some possibility to access the internal data there is no standardized approach. (Boer and Verbraeck 2003; Taylor, et al., 2003).

Some CSPs support the possibility to link an external program (e.g. Excel and Visual Basic) to stop and start the simulation, to introduce entities or to alter a parameter in the model. Other CSPs use DLL "plug-ins" programmed in for example C++ or Java to do this. The CSPs can also have a COM interface or a dedicated interface library that can be used by an external program. (Taylor, et al., 2003).

2.3.3 Data Types and Attribute Names Problems

Which data types CSPs support may differ from package to package. Some CSPs support many attribute types such as string, real, integer, etc. (e.g. eM-Plant), while others only support the type real (e.g. Arena) (Boer and Verbraeck 2003). Even the names of an entity's attributes may be different in two models.

If two models created in different CSPs should be linked together in a distributed simulation there is a need to translate data when it is passed from one model to another. By using a *wrapper* it is possible to map original attribute values to new translated values (Boer and Verbraeck 2003). The original and new attribute values are saved in a temporal instance table.

Figure 2-5 shows an architecture for attribute type inconsistency with two simulation models. Each model has a wrapper with a table of mapped attributes.

Mapping attribute types

Temporal Entity Instance

Attribute	Attr. Value	Mapping Attr. Value
colour	blue	0
..	..	..
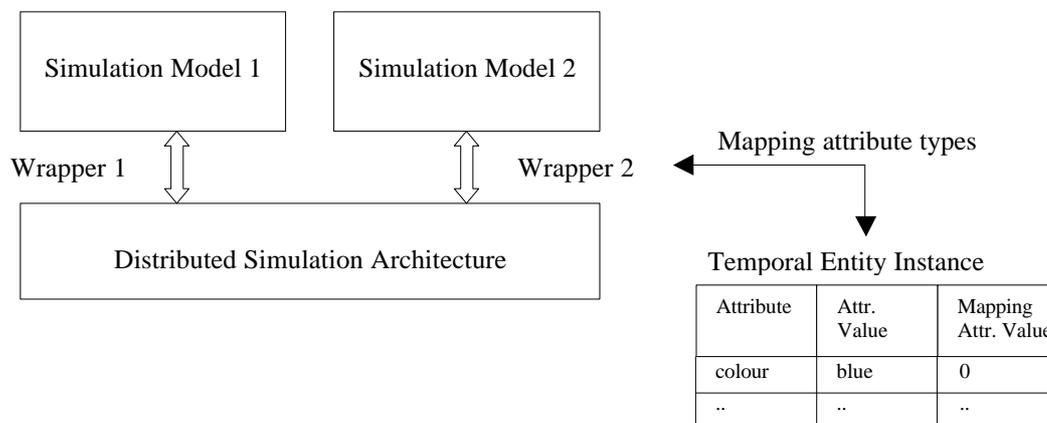
Figure 2-5. Architecture for Attribute Type Inconsistency
(Boer and Verbraeck 2003)

Suppose, as an example, Simulation Model 1 produces a car entity with the attribute colour set to 'blue' and send this entity to Simulation Model 2. The second model is limited compared to the first one in the way that it only supports attribute types that are numbers. Wrapper 2 must translate the colour to a number. It checks in its temporal instance table if it has ever received an entity with this colour and what number is has been mapped to. If not, a new number is mapped to this colour and the old and new attribute values are saved in the table. If entities later are passed back to the first model their attributes will first be translated back to their original values.

2.3.4 Time Management Problems

A sequential simulation, executing on one computer, often uses the three-phase algorithm and moves its clock to the time of the next scheduled event in the A phase, as seen in section 2.1.3. In a distributed simulation, however, all federates have their own current simulation time and events passed between federates.

Events that are scheduled by a COTS simulation package's simulation executive are called *internal events*. Time stamped event messages that are received from another model are called *external events*. Internal events are ordered by the package's event list (see section 2.1.1) whereas external events are ordered by the time management

middleware, for example by the RTI if HLA is used as middleware. (Taylor, Sharpe and Ladbrook 2003)

The problem is how the COTS simulation package should determine the next event to process. Should it be the next internal one taken from the event list or the next external one offered from the middleware? Four different approaches will be described in section 3.2.

3   Method

This chapter will present the problem to be solved in detail and why it is interesting to solve it. Different methods that can be used will be discussed.

3.1   Analysis of CSP Interoperability

Even though HLA has become a de facto standard for distributed simulation there is still a need to specify the format and manner objects are passed between the CSP's involved in the simulation. The HLA-CSP Interoperability Forum has therefore defined six *reference models* (Taylor 2003) that can be used to illustrate this:

Type I. Asynchronous Entity Passing
Type II. Synchronous Entity Passing
Type III. Shared Resources
Type IV. Shared Events
Type V. Shared Data Structure
Type VI. Shared Conveyor

To illustrate the CSP Interoperability problems that exist some of the reference models will be described below. Type I and II will be described in section 3.1.1 and Type III will be described in section 3.1.2.

3.1.1   Asynchronous and synchronous entity passing

If there is no intermediate or direct feedback when an entity is passed from one model to another this is called *asynchronous entity passing*. Figure 3-1 shows two factories F1 and F2 that each consists of an arrival source So_i, a queue Q_i, a workstation W_i, a resource R_i and an exit sink Si_i (where i is the factory identifier).
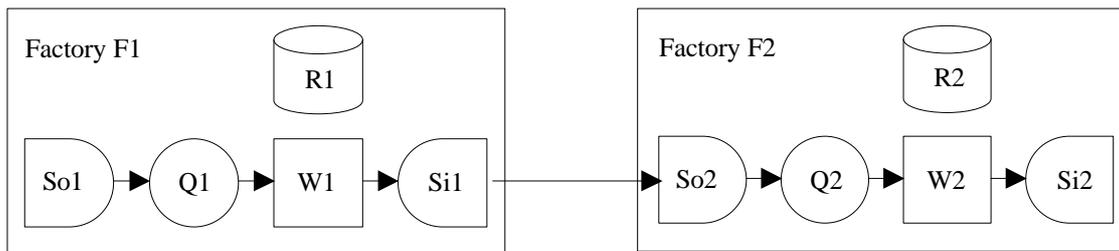
Figure 3-1. Example of distributed simulation with an asynchronous entity passing.
Adopted from (Taylor 2003)

When parts arrive from the arrival source they are put in the queue to wait for the workstation to be available. When the workstation is free a new part will be loaded from the queue (if any is in the queue), and then processed. If the workstation is broken down, a repairman must fix it before the processing can continue. The repairman is depicted with the resource. When the workstation has finished the part will exit the factory via the exit sink and instantly arrive at the arrival source of the next factory.

The federate containing factory F1 must publish the parts leaving the model in a manner and format so that the federate containing factory F2 can understand. The object and attribute names and the data format of all attributes must be identical.

If instead *synchronous entity passing* is used, the models that are involved in the simulation must be synchronized with each other. In the example shown above, this is for example needed if the receiving factory F2 has a bounded queue Q2. If the queue is full, factory F1 must delay the passing of new entities. To know when factory F1 is allowed to send entities the two models need to be synchronized.

3.1.2 Shared Resources

Another form of interoperability between CSP's can be two models that share a resource. Figure 3-2 shows two factories sharing one resource R, for example one repairman that serves both workstations in the two factories. He can only fix one workstation at the same time.
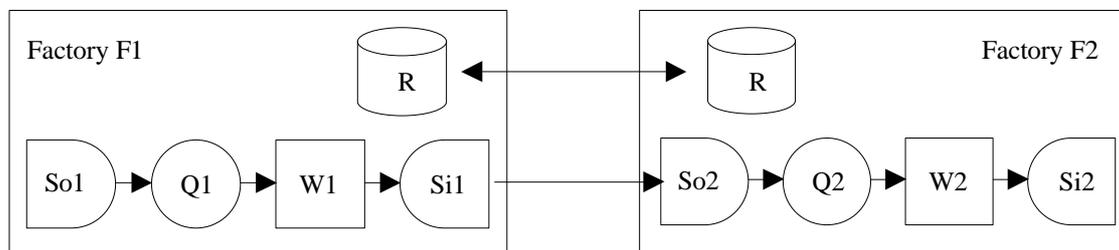
Figure 3-2. Example of distributed simulation with a shared resource.
Adopted from (Taylor 2003)

3.2   Time management algorithms

As seen in section 2.3.4 it is a problem how a COTS simulation package should determine the next event to process. This problem concerns the ordering of external and internal events. Four different approaches will be described below, based on the discussion by Taylor, Sharpe and Ladbrook (2003). All approaches are based on the conservative time management method.

3.2.1 Event List Externalisation (EE)

A simple solution is to treat all events as external events. Each event that is scheduled within the simulation package is externalised and sent through the middleware to itself. The middleware will in this way always offer the next event. The problem with this approach is that it will need redevelopment of the COTS simulation packages.

3.2.2 Permission Request (PR)

In this approach the simulation executive asks the middleware for permission to do the A phase in the three phase algorithm. Phase A moves the simulation clock to the time of the next scheduled event (see also section 2.1.3). The middleware will then answer by either (a) granting permission to advance, (b) passing an event with a timestamp or (c) requesting the simulation executive to wait.

If the middleware is sure that no external event will arrive to this model with a time stamp lower than the time stamp of the next internal event it will grant permission to

advance. If it knows that there is such an external event, it will pass this event to the model and the simulation clock will be moved to the time of this event and Phases B and C will be performed as normal. If the middleware cannot be sure whether an external event will arrive, it will request the simulation executive to wait until further notice.

3.2.3 Incremental Advance (IA)

If it is not possible to obtain the next event time from the event list, the time must be advanced by the smallest possible time unit of the federate. At each new time any internal events are first executed. The simulation executive will then ask the middleware if there is any safe external event to execute.

Even in this approach, three answers are possible. If there is an external event and the time stamp of this is higher than the current simulation time, the simulation executed will be allowed to do another cycle. If there is an external event with a time stamp equal to the current simulation time, this will be passed to the federate. If the middleware cannot identify the next safe external event, the simulation executive will be requested to wait until further notice.

3.2.4 External Control (EC)

In the last approach the control of the time advancement is moved from the simulation package to the middleware. The middleware can order the simulation executive to (a) advance to a given time, (b) advance to a given time and then execute a new external event, and (c) wait. As with incremental advance there is no need to obtain the next event in the event list.

If the middleware has found it safe to advance to a certain time, it will inform the simulation executive about that time. Any internal events with a time stamp lower than or equal to this time will be executed as normal. After that the simulation executive will stop executing until it receives a new order.

If the middleware has instead identified a new external event it will pass the event and its time stamp to the simulation executive. Any internal events with a time stamp less than this time will first be executed. After that the external event and any other internal event scheduled at this time will be executed. The simulation will then stop until new orders arrive.

3.2.5 Discussion

Advantages and disadvantages of the four algorithms are discussed by Taylor, Sharpe and Ladbrook (2003). They suggest considering two factors in the discussion of which algorithm that is the best: technological intervention (affects the implementation cost) and performance. Below is a summary of their discussion about the different algorithms.

Event list externalisation can be immediately discounted since this will mean too many changes of the COTS simulations packages. The event lists are implemented in various ways to improve efficiency and this approach will also be too package specific.

Permission request and incremental advance both require an interaction with the middleware before each time advance. External control needs less interaction as it can execute many events before it has reached the time it has been instructed to advance to. Both permission request and external control require that new events can be placed in the event list.

Incremental advance adds a major overhead of step by step time advance and has no advantage compared to permission request, which except from the time-step part, would have a similar implementation.

Permission request has been compared against external control in a performance test (Taylor, et al., 2002). The results shows that external control is much faster than permission request if the ratio of internal to external events is high, i.e. few external events. They perform equally for a lower ratio.

The conclusion of the discussion is that external control appears to be the best option both in terms of technological intervention and performance.

3.3 The CSP Emulator (CSPE)

The CSP Emulator (CSPE) is a tool developed to investigate how different middleware perform in a COTS based distributed simulation (Mustafee 2003). This section will describe what CSPE is and what it can be used for.

3.3.1 Architecture

A distributed COTS simulation can be divided into three different parts, see Figure 3-3: The COTS simulation package with its model, the distributed simulation middleware and a CSP handler that communicates with both the CSP and the middleware.
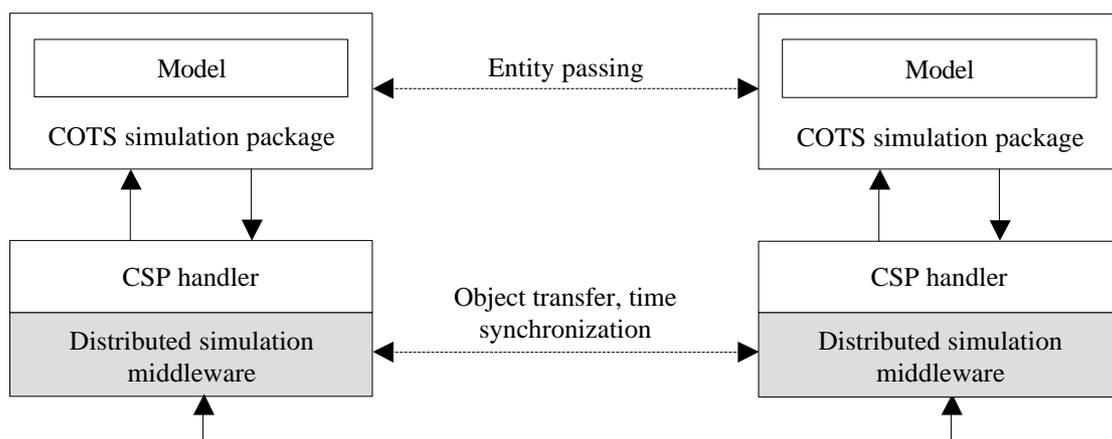
Figure 3-3. A general overview of a distributed COTS simulation.
The simulation is divided into three different parts: CSP, CSP handler and middleware.

The CSP handler is needed to create a common interface for all CSPs in a distributed simulation. The different packages today have various solutions for how external code can interact with the simulation model (OLE Automation, COM, Active X interfaces

18

etc.). It would be preferable if all kinds of middleware could communicate with the CSP in the same way.

The CSPE consists of a simulation executive, an event list, a simulation clock and an event generator. CSPE is deterministic, which means that there is no randomisation within the simulation that is performed. CSPE is also limited to only simulate the behaviour of one assembly line.

The assembly line uses the *First Come First Served Pipeline Workflow Queuing Model*, see Figure 3-4, and consists of a variable number of machines that all have variable setup and processing times. The machine queues have unbounded buffers and the machines start to work as soon as there is an entity waiting in their queue. The configuration of the machines is specified in a file.
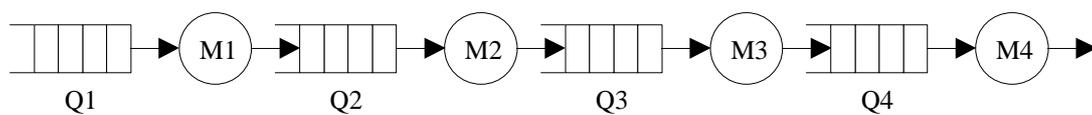
Figure 3-4. First Come First Served Pipeline Workflow Queuing Model.

Mustafee's implementation of CSPE is programmed in Java. CSPE communicates with the CSPE handler through sockets connections, see Figure 3-5. The CSPE listens to port X02 and the handler listens to port X01 (where X is a federate number). The socket connections are setup before the simulation is started.
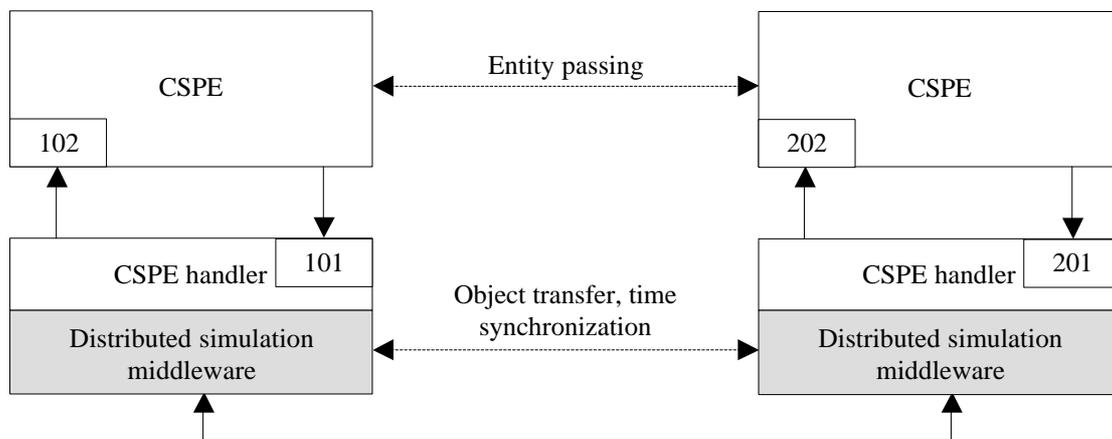
Figure 3-5. An overview of a distributed simulation using CSPE.
The simulation is divided into three different parts: CSPE, CSPE handler and middleware.

3.3.2 API Calls for CSPE Handler

CSPE uses external control (EC), see section 3.2.4, as synchronizing algorithm. It has three API calls for input messages to the CSPE and three API calls for output messages from the CSPE.

The API calls used for input messages to the CSPE are advance(time), advance(time, entity) and start(). See Table 3-1. The first one is used if the middleware has determined a safe time for the CSPE to advance to. Any event in the internal event list with a time stamp lower or equal to this time will be executed. The second one is used

if the middleware has determined a safe external event which is passed to the CSPE. Any internal event in the event list with a time stamp lower or equal to this time will be executed before the external event is executed. The last API call is used to signal the start of the simulation.

advance(time)	Safe for the CSPE to advance to this time.
advance(time, entity)	Safe for the CSPE to advance to this time. Then executed the entity being passed.
start()	Signals the start of simulation.

Table 3-1. Summary of API calls used by CSPE handler for input messages.

The API calls used for output messages from the CSPE are output(time), output(time, entity) and terminate(), see Table 3-2. The first one is used for the CSPE to inform the middleware of its current time. The second one is used to send an entity to another federate and the last one is used to inform the middleware that the CSPE has executed a certain amount of entities specified at start-up.

output(time)	The current simulation time of CSPE.
output(time, entity)	An entity for transfer to another federate and the time when it should arrive.[1]
terminate()	Signals the end of simulation.

Table 3-2. Summary of API calls used by CSPE handler for output messages.

The variable entity is as a string containing the following data (separated with an "*"):
 • Names of the federates to send the entity to
 • Names of all federates that has processed this entity
 • Entity name
 • Event type

3.4   Methodology

As seen in section 1.2, the ordering of events and time management is important for the overall performance of a distributed simulation. The overhead of the CSP execution is not possible to reduce without redevelopment of the CSP software. Better network and hardware can be used for better performance, but of more interest is to investigate how the overhead of the middleware could be reduced.

By using CSPE it will be possible to concentrate this work to how HLA can be used for time management in a COTS based distributed simulation. CSPE is created to be a benchmarking tool with possibilities to use different federate configurations and different variable settings. It has support for the external control synchronization algorithm and Reference Model I (for asynchronous entity transfers).

---

1 The version of CSPE that is specified in (Mustafee 2003) has a slightly different use of output(time, entity). The time is there the current simulation time. The time when the entity should arrive is then calculated by the handler by taking the current time and add the federate's lookahead. This is however a limited method that has been changed in later versions.

The fact that CSPE only supports one synchronization algorithm and one reference model can of course been seen as a limitation and a disadvantage for the use of CSPE. External control has however been found to be the best synchronization algorithm, as seen in section 3.2.5. It would also be interesting to investigate the performance of HLA in all reference models but this is not possible within the time space of this work

An advantage of CSPE is that it during the simulation saves information that makes it possible to trace an entity to a particular position of a particular workstation queue through its lifetime. This could be useful in the verification of the middleware, to see that object and time management works correctly.

To investigate time management in COTS distributed simulation using HLA and CSPE I will first transform the CSPE architecture to an architecture that is using HLA. I will then choose which methods that are best for object management. An algorithm for the time management will be designed.

Implementation of the design will be made using the latest version available of Java, that is 1.4.2, and the HLA RTI 1.3-NG Version 5. Experiments using the benchmark proposed in Mustafee (2003) will be used and the result will be compared to the performance of another middleware based on the Chandy-Misra-Bryant algorithm. Analysis of the results will show if HLA is good for COTS distributed simulation.

4   Design and Implementation

This chapter will describe the design and implementation in general for the different methods discussed in the previous chapter. Details will be given for the method that will be implemented.

4.1   CSPE-HLA Architecture

Figure 4-1 shows the architecture of a distributed simulation of CSPE using HLA as middleware (CSPE-HLA). CSPE-HLA consists of two parts, the CSP handler and the HLA code, that both are integrated into the same program. Communication between CSPE and CSPE handler is done with socket connections.

Figure 4-1. An overview of a distributed COTS simulation using HLA.
The simulation is divided into three parts: CSP, CSP handler and HLA middleware.

For the HLA integration version 5 of the RTI 1.3-NG has been used. It supports for both C++ and Java. The federate part of the HLA code is unique for each program using HLA and has to be programmed while the RTI part is general and is included as a Java or C++ library.

4.2   Object management

4.2.1   Object Update or Interactions?

As mentioned in the section 2.2.5, different methods exist to send object updates between federates in HLA: dynamic object update with push and pull methods, static object update and the non-persistent method with interactions.

It has by experimentation been shown that sending interactions is eventually more efficient for passing entities between federates than updating a static attribute value and that both those methods are faster than the dynamic update techniques (Yen 2003). Since I cannot see any other advantages for any of the other methods in the use with CSPE-HLA I have chosen to use interactions for entity passing in CSPE-HLA.

4.2.2 The Entity Transfer Specification

A specification for how entities could be transferred between COTS simulatation packages using interactions has been proposed by HLA-CSPIF (Taylor, Turner, Low 2003). The specification first makes some definitions before the interaction classes use is described. A summary of the specification is given below.

An entity has a *name* and zero or more *attributes* and is defined as follows.

entity = {entityName, attributes*}

An entity that is sent from one model to another has a *source*, the model the entity leaves, and a *destination*, the model at which it arrives. When an entity leaves a CSP the CSP must be able to provide the following information to the CSP handler:

output(entity, time, source, destination)

When the CSP handler at the receiving federate passes the entity to the CSP it must provide the following information to the CSP:

input(entity, time, source)

Time is defined as the time when an entity exits a source model and instantaneously arrives at the destination model. It is assumed that both models use time in the same way (considering resolution etc.). Source and destination are used to determine the appropriate entry point at the destination model.

The specification suggests an interaction class hierarchy with three levels, see Figure 4-2. Each destination has one interaction class named *transferEntityToFedDest* where "transferEntity" is unique for each type of entity and "FedDest" is the name of the destination. Each of these classes also has one subclass for each federate that can send entities to this federate, named *transferEntityFedSoToFedDest*, where "FedSo" is the name of the source. There is also one super class, named *transferEntity*, that a federate can subscribe to receive all entities (for purposes of monitoring, visualization, etc.).

During initializing a federate will indicate what federates it is capable of sending entities to by publishing various transerEntityFedSoToFedDest interaction classes. It will also subscribe to all transferEntityToFedDest interaction classes to indicate it is capable of receiving entity from other federates.

When a CSP later wants to send an entity to another federate the CSP handler will use the correct interaction class for this transfer.
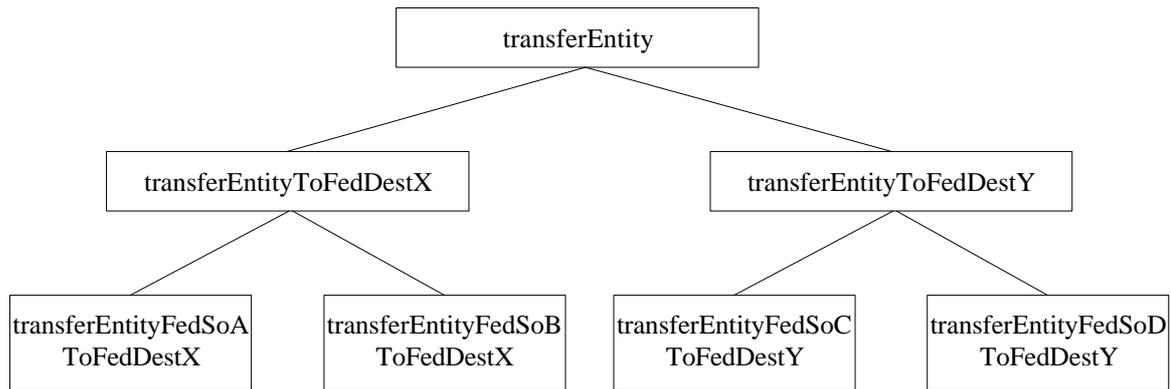
Figure 4-2. **Interaction Class Hierarchy.**
Adopted from (Taylor, Turner, Low 2003).

4.2.3 Design

Even though the full Transfer Entity Specification provides a good solution for how entities can be transferred between COTS simulation packages I have found the class hierarchy and interaction use to be unnecessary complex for the entity transfers needed by CSPE. Instead I have used a simpler approach to pass entities.

Only one interaction class exists for each federate named after the federate. A federate that wants to receive entities from other federates subscribes to the interaction with its name at initialisation. All federates that want to send entities to this federate publishes that interaction. See Figure 4-3.
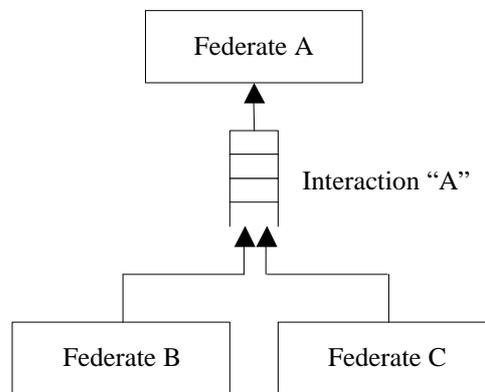
Figure 4-3. Two federates sending entities to one another using interactions.
Federate B and C want to send entities to Federate A and are publishing the interaction class named "A". Federate A is subscribing to this class.

The interaction classes have one parameter, named *message*, that is used for the entity (source and destination included) in the same data format that is used by CSPE handler for the output and advance methods, see section 3.3.2.

4.2.4 Implementation

CSPE and CSPE-CMB uses a *Federate Definition File* to specify what other federates a federate is connected to. Each federate has its own file. The file consists of a list of federates that can send entities to the federate and a list of federates the federate can

send entities to. For compatibility reasons with CSPE and CSPE-CMB, the same file format is used even for CSPE-HLA.

A federate that wants to use an interaction class to send or receive objects must use the interaction class's handle that is a unique number. The handle is requested from the RTI by calling the method *getInteractionClassHandle(name)* where *name* is the name of the interaction class. CSPE-HLA stores all handles to classes it publishes in a hash table with the class names as keys for fast access since they are needed every time an entity is sent.

To publish and subscribe to an interaction class a federate uses the methods *publishInteractionClass(handle)* and *subscribeInteractionClass(handle)*. To send and receive objects the methods *sendInteraction(handle, parameters, time, tag)* and *receiveInteraction(handle, parameters, time, tag, eventRetractionHandle)* are used. *handle* is the interaction class's handle, *parameters* is the parameters of the interaction class (in this case only *message*, see section 0), *time* is the time stamp of the object, *tag* is used for user-specified messages (not used in CSPE-HLA) and *eventRetractionHandle* is a unique identity for each TSO event in the federation (used in optimistic simulations for the retraction of objects, but not used in CSPE-HLA).

4.3   Time management

4.3.1   Which of the Time Advance Methods To Use?

As mentioned in section 2.2.5, HLA has mainly three methods for time advancement: *timeAdvanceRequest*, *nextEventRequest* and *flushQueueRequest*. There are also two variants, *timeAdvanceRequestAvailable* and *nextEventRequestAvailable*, that are used for federations with zero-lookahead but that is out of the scope for this thesis. This section will discuss what methods that can be used for the time synchronization algorithms Permission Request and External Control.

The method flushQueueRequest can directly be sorted out for all algorithms, at least for the approaches described in section 3.2, since it doesn't guarantee that it will deliver all events in time stamp order. When flushQueueRequest is invoked, all events that exist in the receive queue will be delivered to the federate. If this method should be used with any of the synchronization algorithms, they must first be modified to be able to pass external events without giving the simulation executive permission to advance to the time of the passed event. Except from what events that are released to the federate this method is similar to the timeAdvanceRequest method and will not be discussed further.

The methods timeAdvanceRequest and nextEventRequest are very similar except from what the federate that uses them promises. Recall from section 2.2.5 and Table 2-1 that a federate that calls timeAdvanceRequest promises it will not generate any TSO events with a time stamp lower than the requested time + lookahead. A federate that calls nextEventRequest promises it will not generate any TSO events with a time stamp lower than the granted time + lookahead.

For Permission Request the method timeAdvanceRequest cannot be used. The reason is that by calling this method, events that are buffered in the receive queue can be

delivered to the federate. Suppose one of these events has a time stamp, t_{event}, that is smaller than the time that is requested by the simulation executive, $t_{requested}$, (equal to the time of the next internal event). If this event, when it is executed, generates a new external event this new external event can be sent with a time stamp not smaller than t_{event} + lookahead. This time can however be smaller than the time the federate had promised not to generate any TSO events before, that is $t_{requested}$ + lookahead for timeAdvanceRequest.

For External Control both timeAdvanceRequest and nextEventRequest can be used.

4.3.2 Time Update Cycle

This section will propose an approach how CSPE-HLA can use External Control for time and object management when CSPE uses HLA as middleware. The approach can be described as a cycle that goes on and on until the simulation is ended. Figure 4-4 shows an algorithm for this and the approach is also illustrated with a sequence diagram, see Figure 4-5.

```
While simulation not ended do
      Query minimum next event time from RTI
      Pass time to CSPE if not equal to last time sent
      Receive events from CSPE and pass to RTI
      Wait until new simulation time is received from CSPE
      Request RTI to advance to new time
      Receive events from RTI and pass to CSPE
      Wait until time advance is granted
End while
```

Figure 4-4. Algorithm for CSPE-HLA.

The time update cycle starts with that the CSPE-HLA middleware asks RTI for the minimum time when a message could arrive to this federate, using the method *queryMinNextEventTime()*. The answer will depend on the current time and lookahead values of the other federates and whether any messages are queued in the receive-buffer. CSPE-HLA will inform CSPE what simulation time it could advance to since it can be guaranteed that no messages will arrive earlier than this time by calling *advance(time)*.

CSPE will now execute all events in its event list that has a time stamp lower or equal to the received time. If outgoing messages are generated these will be passed to CSPE-HLA by calling *output(time, entity)*, which, in turn, passes them on using interactions with the method *sendInteraction(entity, time)*. After each time the simulation clock is changed in CSPE it will inform CSPE-HLA about its new time by calling *output(time)*. After CSPE has executed all events it could, it will inform CSPE-HLA this by returning the time that it was allowed to advance to.

CSPE-HLA will request this new time from the RTI with either *timeAdvanceRequest(time)* or *nextEventRequest(time)*. If any messages are queued in the receive-buffer with a time equal to this time they will be received with the method *receiveInteraction(entity, time)* and passed to CSPE using *advance(time, entity)*. After all messages have been received and RTI knows that no more messages will come at the requested time, the new time will be granted with *timeAdvanceGrant()* and *advance(time)*.

A new time update cycle now begins with CSPE-HLA asking for the minimum time when a message could arrive.
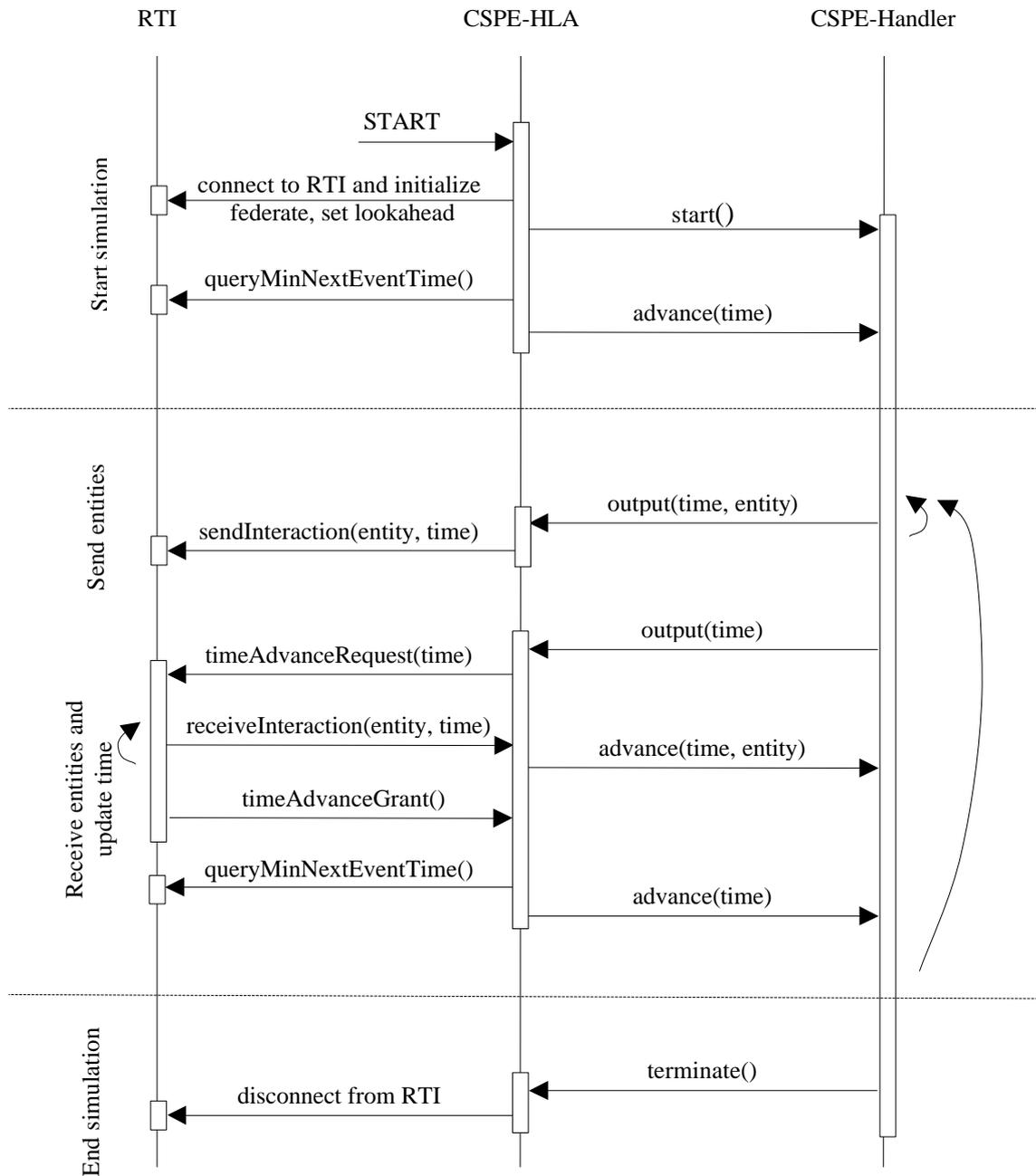
Figure 4-5. Sequence diagram of the time update cycle in CSPE-HLA.

Pseudo Code

Below follows a pseudo code implementation of the time update cycle in CSPE-HLA:

Double lastCSPETime = 0.0
Double lastGrantedTime = 0.0
Double lastTimeSentToCSPE = 0.0
Boolean simulationEnded = false

Method run()
    Connect to RTI and set lookahead, timeRegulated and timeConstrained. Connect to CSPE.
    Start simulation by calling start() in Handler
    While simulationEnded == false **Do**
        Query minimum next event time from RTI by calling queryMinNextEventTime(). Set minNextEventTime = received time
        If minNextEventTime > lastTimeSentToCSPE **Then**
            Let CSPE advance to the new time by calling advance(minNextEventTime)
        End If
        While lastCSPETime <= lastGrantedTime **And** simulationEnded == false **Do**
            Sleep thread for a short time
        End While
        If simulationEnded == false **Then**
            Ask RTI to advance to lastCSPETime by calling timeAdvanceRequest(lastCSPETime) in RTI and await callback to timeAdvanceGrant. Set lastGrantedTime = time granted.
        End If
    End While
    Disconnect from federation and CSPE
End Method

Method output(time)
    Current simulation time is received from CSPE-Handler. Set lastCSPETime = time
End Method

Method output(time, entity)
    Send entity as a TSO event by calling sendInteraction(..). The time is the time the event should be received.
End Method

Method terminate()
    Set simulationEnded = true
End Method

Method receiveInteraction(time, entity)
    Send entity to CSPE by calling advance(time, entity) in CSPE-Handler.
    Set lastTimeSentToCSPE = time
End Method

4.4 Code Implementation

4.4.1 Class Structure

Figure 4-6 shows a class structure for CSPE-HLA. To make it easier to understand, the classes are put in the same way as the different parts of the overview of a distributed COTS simulation using HLA in Figure 4-6.

CspeHla is a super class that different HLA approaches can extend. It contains all methods and variables that the different approaches have in common. I have implemented two approaches that, except from the what time advance method they use, are very similar. *CspeHlaTar* uses TimeAdvanceRequest and *CspeHlaNer* uses NextEventRequest.
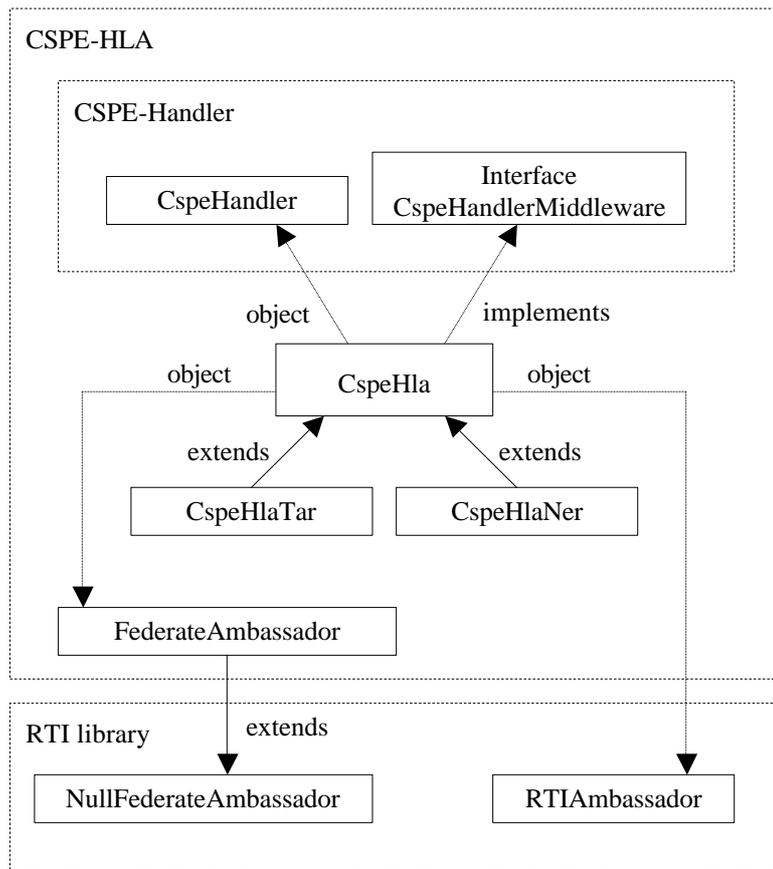
Figure 4-6. Class Structure for CSPE-HLA.

4.4.2 CSPE-Handler

For compatibility with CSPE and CSPE-CMB I have based CSPE-Handler on code from the implementation in (Mustafee 2003). The main change in the new version of CSPE-Handler is that the handler and the middleware have been completely separated. The handler's API calls for output messages from CSPE has been put in an interface that classes that wants to use the handler must implement.

The new version of CSPE-Handler consists of the following classes: *CspeHandler* (main class), *CspeHanderMiddleware* (interface mentioned above), *CspeHandlerClient* (used to send messages to CSPE), *CspeHandlerServer* and *CspeHandlerServerThread* (both used to receive messages from CSPE).

4.4.3 RTI

CspeHla uses *RTIAmbassador* in the RTI library as an object to send messages to the RTI. When the RTI wants to do a call-back, for example to say that a new time is granted or to pass a new interaction, it makes a function call to the *FederateAmbassador*. To have support for API calls the RTI can do it therefore extends the *NullFederateAmbassador* found in the RTI library. Only the methods that the middleware wants to use are overridden in FederateAmbassador.

5 Experimentation

This chapter will describe the benchmark that was used and how the experimentation was done. The equipment and programs that were used in the experimentation will be explained. The results will be shown in graphs together with the results of the middleware using the Chandy-Misra-Bryant algorithm.

5.1 Benchmark

The benchmark proposed by Mustafee (2003) has been used. It uses three different experiments that are performed using four different federate configurations. The federate configurations will be described below in section 5.1.1 and the experiments will be described in section 5.1.2.

5.1.1 Federate Configurations

Pipeline

The pipeline federate configuration is the simplest of the configurations. It consists of six federates connected to each other as a pipeline, see Figure 5-1. Entities are generated in federate A (source) and are then passed in one direction through all federates until they finally are removed after been processed in federate F (sink). This can for example be a model of production chain.

Q-W-Q-W..	→	Q-W-Q-W..	→	Q-W-Q-W..	→	Q-W-Q-W..	→	Q-W-Q-W..	→	Q-W-Q-W..

Federate A   Federate B   Federate C   Federate D   Federate E   Federate F
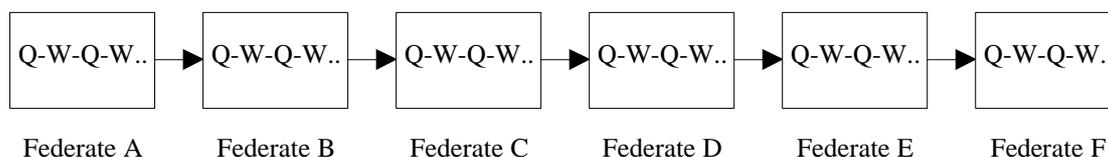
Figure 5-1. Pipeline Federation Configuration.
Adopted from (Mustafee 2003)

Local Feedback

The local feedback federate configuration is almost designed as the pipeline federate configuration but makes it possible for entities to be returned to the previous federate, see Figure 5-2. This can be a model of a production chain where entities can be returned if they are found to be incorrect.

Federate A   Federate B   Federate C   Federate D   Federate E   Federate F
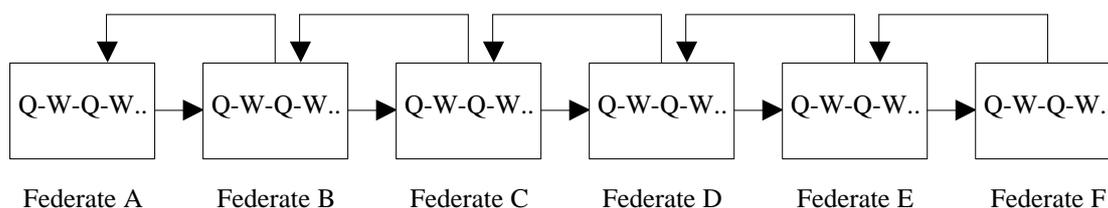
Figure 5-2. Local Feedback Federation Configuration.
Adopted from (Mustafee 2003)

In this benchmark half of the entities are returned. All entities are also forwarded to the next federate to avoid a situation where federates in the beginning of the chain

stop the simulation before later federates has received the number of entities that has been specified.

Client Server

The client server federate configuration consists of the five federates A-E (sources) that generates entities and then passes them to federate F (sink), see Figure 5-3. Federate F will return half of the entities to the federate that created them, a form of local feedback, and remove the rest of them.
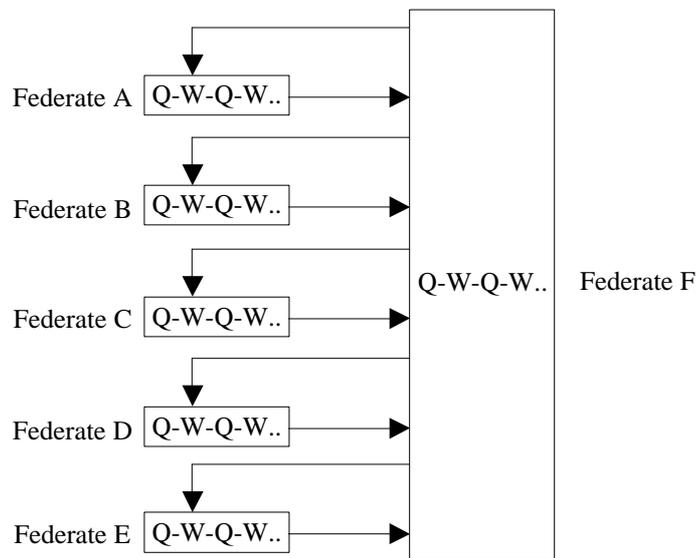
Figure 5-3. Client Server Federation Configuration.

Fully Interconnected

The fully interconnected federate configuration consists of six federates (sources) that each generates new entities, processes, and passes them in a round-robin fashion to other federates, see Figure 5-4. Every federate can send and receive entities to and from every other federate.
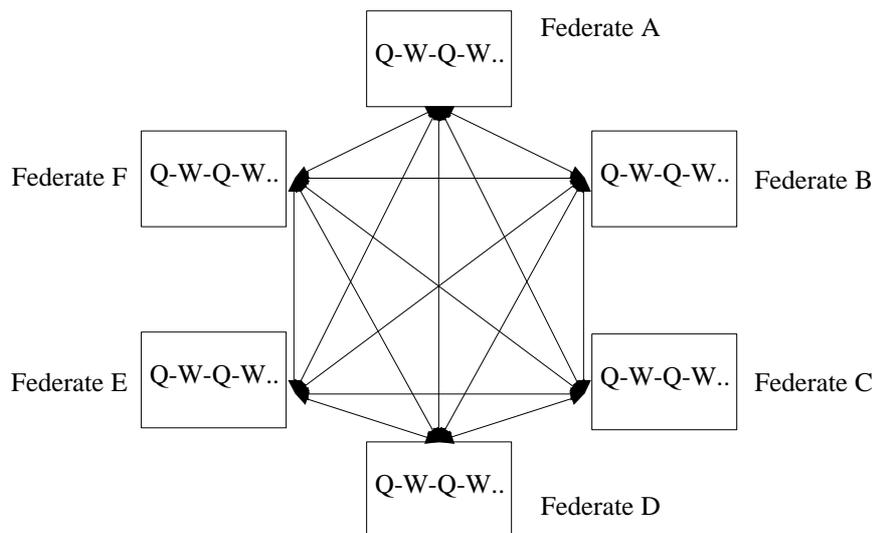
Figure 5-4. Fully Interconnected Federate Configuration.

5.1.2 Experiments

The benchmark has three experiments: variable internal/external ratio, variable workload and variable lookahead.

Variable External/internal Event Ratio

As seen in section 3.2, the ratio of external and internal events can be important as it for some time synchronization approaches can affect the performance for distributed simulations. It would therefore be interesting to see how different middleware perform at different external/internal event ratios.

The test varies the number of federates in the pipeline workflow each CSPE simulates to change the external/internal event ratio. This method to change the ratio is easy to implement but the drawback is that the experiments execute different amount of events in total and cannot easily be compared with each other. There are, however, no problems to compare the results from one experiment with one middleware with the results from the same experiment with other types of middleware.

Table 5-1 shows the setup for the experiments with variable external/internal ratio.

Experiment	Entities	Federates	External events	Internal events	External/ internal ratio	Machine Setup time	Machine Processing time	Lookahead
1	1000	1	1	1	1	5	5	10
2	1000	2	1	2	0.5	5	5	10
3	1000	5	1	5	0.2	5	5	10
4	1000	10	1	10	0.1	5	5	10
5	1000	20	1	20	0.05	5	5	10

Table 5-1. Variable external/internal ratio.

Variable Workload

This test varies the workload by varying the number of entities each federate must process. Table 5-2 shows the setup for the experiments with variable workload.

Experiment	Entities	Federates	External events	Internal events	External/ internal ratio	Machine Setup time	Machine Processing time	Lookahead
1	1	5	1	5	0.2	5	5	10
2	10	5	1	5	0.2	5	5	10
3	100	5	1	5	0.2	5	5	10
4	250	5	1	5	0.2	5	5	10
5	500	5	1	5	0.2	5	5	10
6	1000	5	1	5	0.2	5	5	10

Table 5-2. Variable workload.

Variable Lookahead

The lookahead federates can affect how far in time other federates can proceed before they must wait for each other, see also section 2.2.5 and 4.3. This can affect the

overall performance of the distributed simulation and it is therefore interesting to see how different types of middleware perform with different lookahead values.

This test varies the lookahead each federate has. Table 5-3 shows the setup for the experiments with variable lookahead.

Experiment	Entities	Federates	External events	Internal events	External/ internal ratio	Machine Setup time	Machine Processing time	Lookahead
1	1000	5	1	5	0.2	5	5	2
2	1000	5	1	5	0.2	5	5	4
3	1000	5	1	5	0.2	5	5	6
4	1000	5	1	5	0.2	5	5	8
5	1000	5	1	5	0.2	5	5	10

Table 5-3. Variable lookahead.

5.2 Test Environment

5.2.1 Computers and Network

The performance tests have been carried out on seven computers connected through a isolated local area network. Six computers run one federate each and the last computer run the RTI Executive (only for HLA tests). All computers also run programs used for test automation, see section 5.2.2 for details. Version 5 of RTI 1.3-NG has been used. The speed of the network was 10 Mbit.

Setup for the 6 computers running federates:
Intel Pentium III 650 MHz, 256 mb RAM
Windows 2000/Windows XP

Setup for the computer running the RTI executive:
Intel Pentium III 950 MHz, 256 mb RAM
Windows XP

5.2.2 Automatic Tests

Before each test software had to initialised and started with correct settings. This setup took approximately 1-3 minutes. Each test had an execution time between 1 second and 16 minutes depending on the settings. If every time advance algorithm, federate configuration and variable setting should be executed one single time almost 200 tests were needed. To be able to do all tests, I had to develop software that automates the tests.

The software, named PerformanceTests, consists of six servers, one for each federate machine, and one client. The servers listen for incoming socket connections. The client connects to each machine and transmits the commands needed to start CSPE and CSPE-HLA/CSPE-CMB with the correct settings for a specific test. In the code of the client it is possible to set some variables to choose what tests that should be performed.

Performance Evaluator, a program that measures the simulation execution time developed by Mustafee (2003), has been integrated into the client of PerformanceTests. PerformanceTests gets all the results from the Performance Evaluator and saves them in a comma-separated log file that easily can be imported into Microsoft Excel.

5.2.3 Program Execution Order

The servers of PerformanceTests are first started at all federate machines. Then the client is started. It connects to all servers. For HLA tests, it will also launch RTI Executive and wait for it to initialise. The client sends out commands how to start CSPE and CSPE-CMB/CSPE-HLA to all servers.

After all programs has initialised and established the right connections to each other, every federate has informed the Performance Evaluator that they were ready. The Performance Evaluator has then started a timer and informed all federates to start the simulation. When all federates has reported back that they have finished the simulation the timer has been stopped. The time has been saved in a log file.

After both CSPE and CSPE-HLA/CSPE-CMB have exited on a federate machine the server will report back to the client. The client will either exit or start a new test by sending news command to the servers. Figure 5-5 shows the execution order of PerformanceTests.
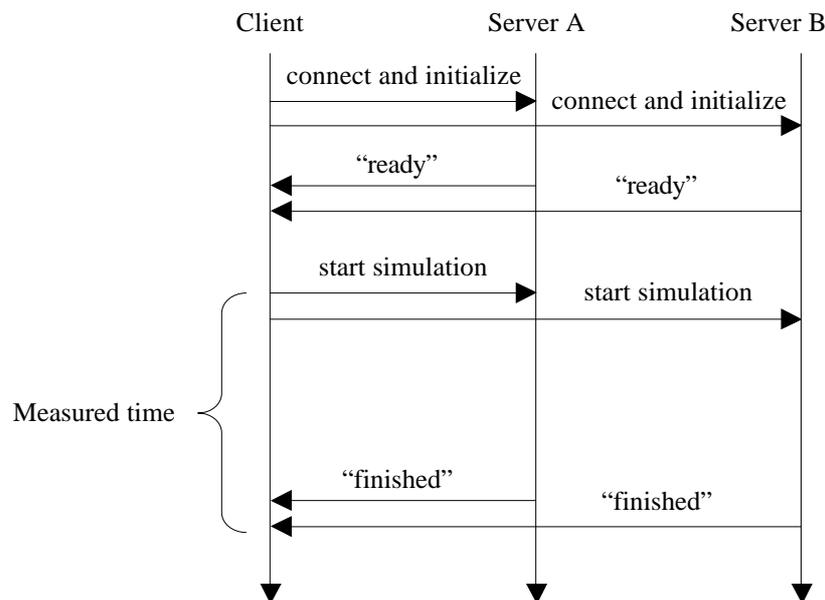
Figure 5-5. Execution order of PerformanceTests

5.3   Results

Below follow the results of the experiments performed in the form of graphs. For comparison the results from the Chandy-Misra-Bryant algorithm are also illustrated in all graphs. They were obtained using the same hardware. Every different test was run three times. Since the tests were run on an isolated network the deviation was very small. The values presented in the graphs are the mean values.

34
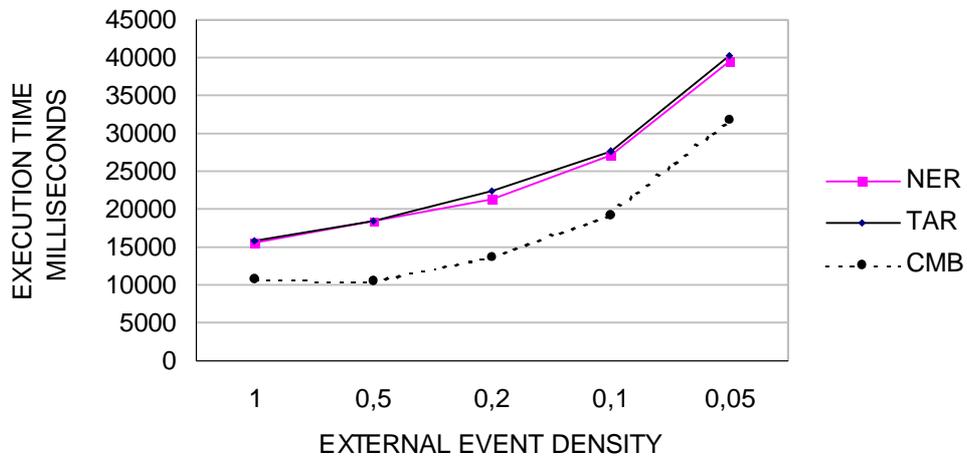
5.3.1 Pipeline

Figure 5-6. Pipeline: Variable External / Internal Event Ratio
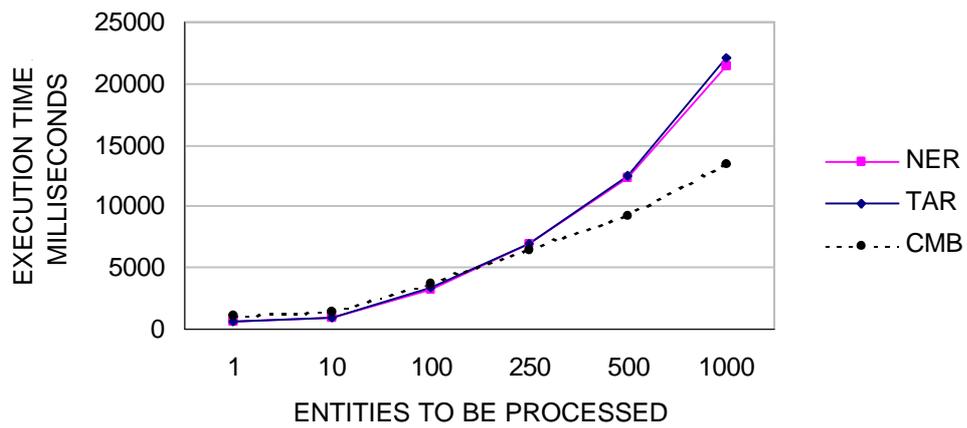
Figure 5-7. Pipeline: Variable Workload
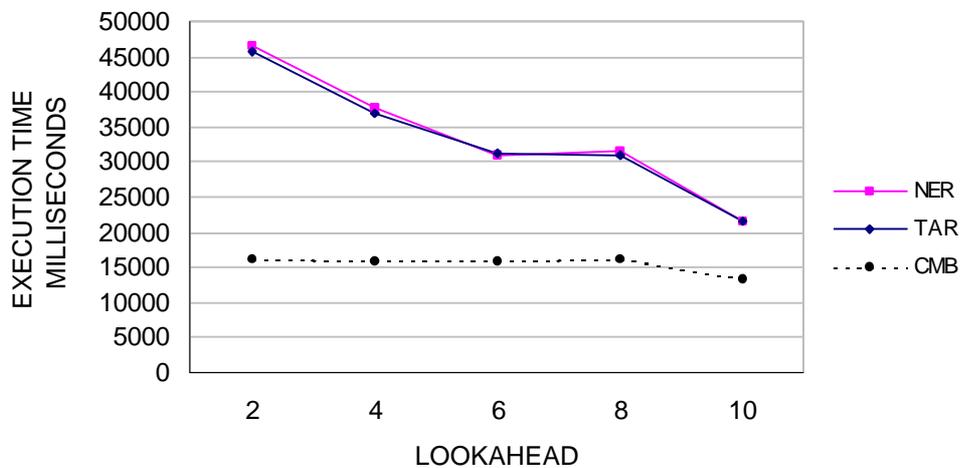
Figure 5-8. Pipeline: Variable Lookahead

5.3.2 Local Feedback
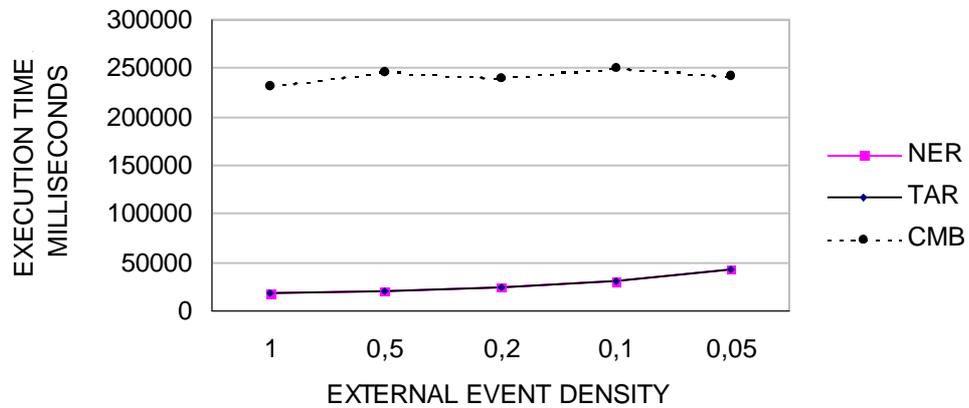
Figure 5-9. Local Feedback: Variable External / Internal Event Ratio
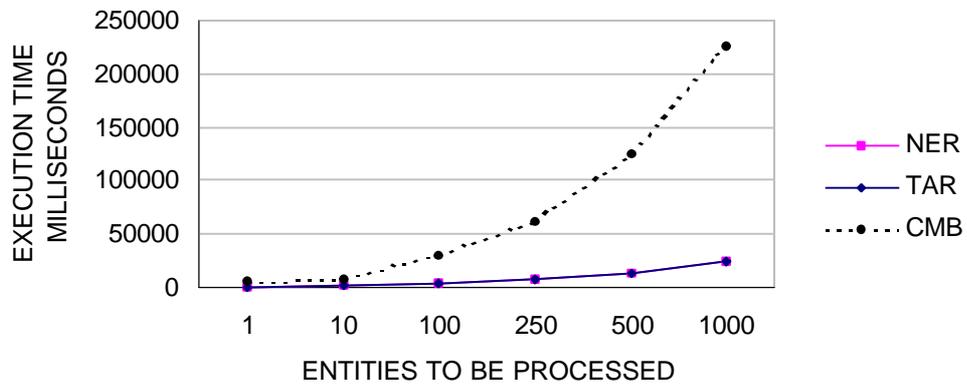
Figure 5-10. Local Feedback: Variable Workload
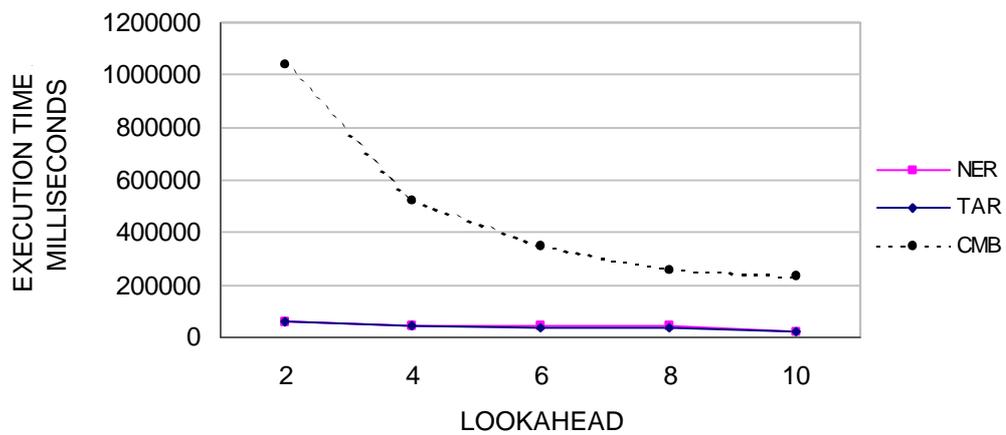
Figure 5-11. Local Feedback: Variable Lookahead

5.3.3 Client Server
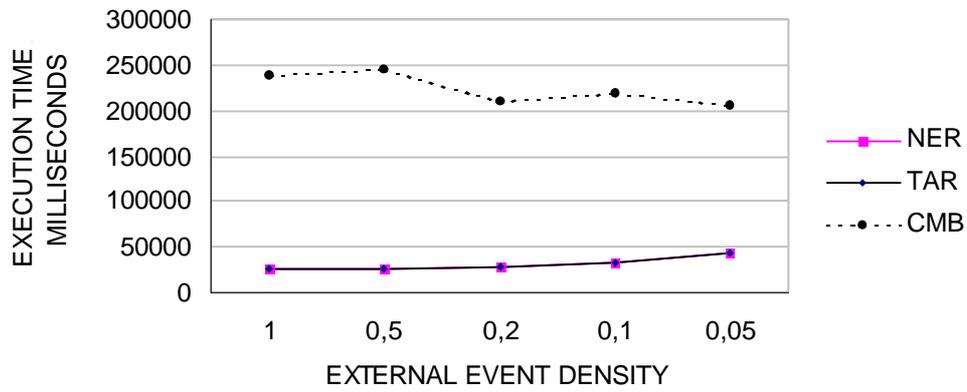
Figure 5-12. Client Server: Variable External / Internal Event Ratio
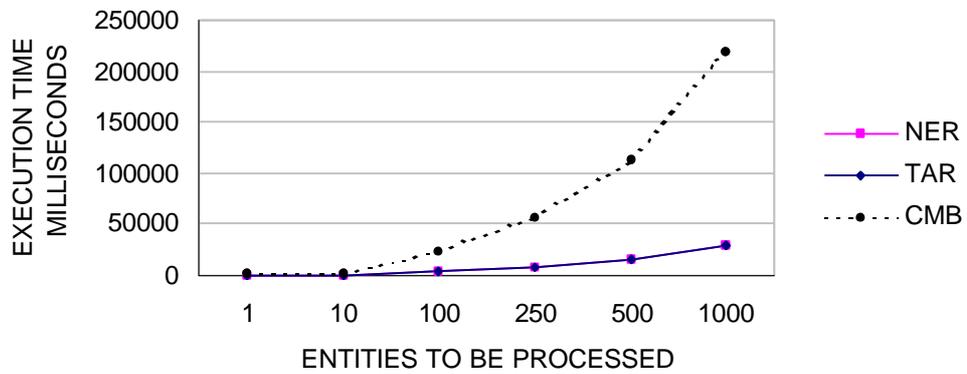
Figure 5-13. Client Server: Variable Workload
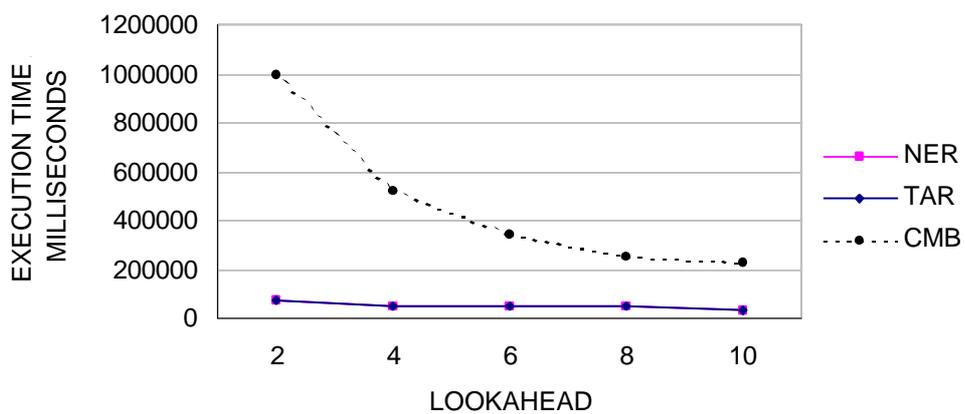
Figure 5-14. Client Server: Variable Lookahead

5.3.4 Fully Interconnected
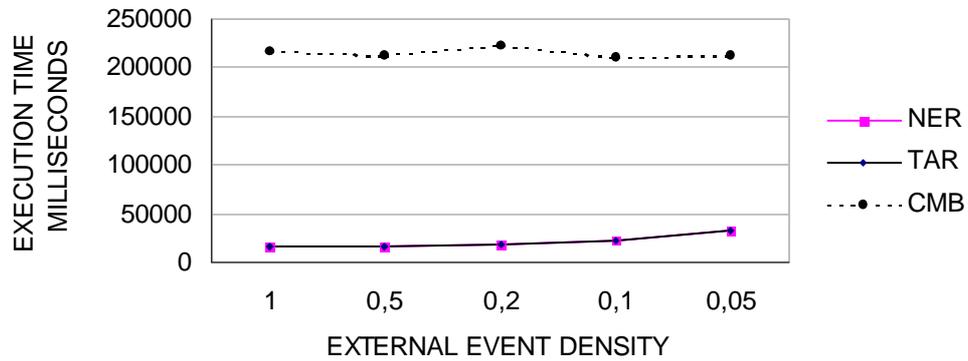
Figure 5-15. Fully Interconnected: Variable External / Internal Event Ratio
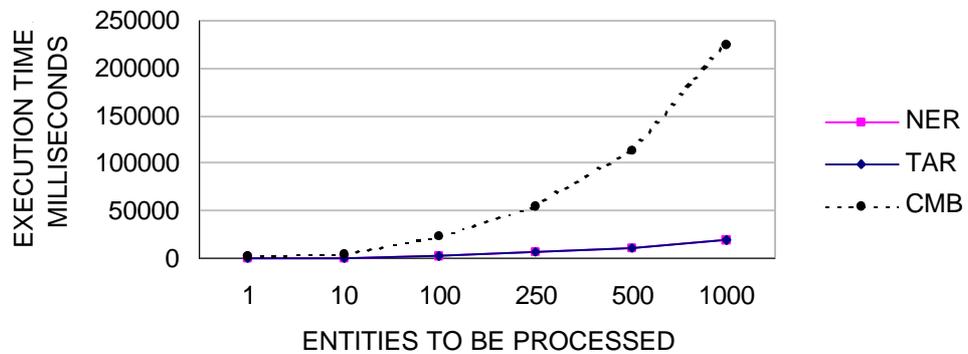
Figure 5-16. Fully Interconnected: Variable Workload
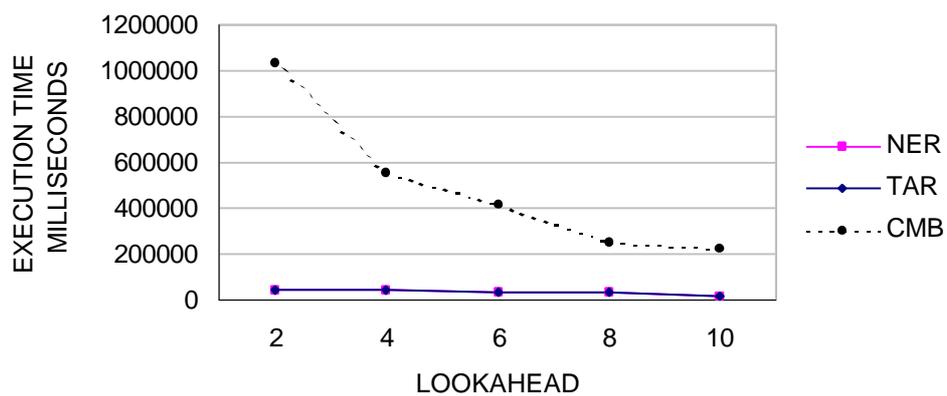
Figure 5-17. Fully Interconnected: Variable Lookahead

6 Analysis

This chapter will analyse the results and discuss advantages and disadvantages of using CSPE as an experimental tool and HLA as middleware for CSPs.

6.1 Different CMB and RTI Implementations

When we analyse the results we must consider, as seen in Section 2.2.5, that HLA does not prescribe any specific implementation. In this report we have used a HLA RTI 1.3-NG Version 5 developed by DMSO for the experiments. Other implementations of RTI exist and how they perform is of course not possible to discuss using the results in this report.

We must also consider, as seen in Section 2.2.4, that several versions of the CMB algorithm exist. The version that has been used in the experiments uses null messages to avoid deadlock situations. Null messages are sent to all outgoing links of a federate after an event has been executed. However, it is a little bit optimised since it never sends a null message to an outgoing link if a message or a null message with the same time stamp already has been sent through that link.

When NER, TAR, HLA and CMB are mentioned in the following sections, we mean the versions that have been used in the experiments.

6.2 Similar Results for NER and TAR

The first we can note is that both NextEventRequest (NER) and TimeAdvance-Request (TAR) in every experiment give almost the same results. Any differences can in fact depend on normal divergences in the measurement.

The similar results can be explained if we study the algorithms that have been used, see section 4.3.2. The time update cycle in a federate always starts with a query to the RTI for the minimum next event time. This is the time when an event earliest can arrive to this federate. The time is passed to the simulation executive, the simulation advances and a new current simulation time is returned. The returned time is used to request a new time from the RTI. This value is always equal or lower than the time of the next event and NER and TAR will therefore execute in the same way.

Other algorithms may give other results. See also the discussion in section 7.2. The results from the two methods are below mentioned as HLA.

6.3 Differences Between HLA and CMB

HLA is faster than or equal to CMB in all configurations except the Pipeline. HLA has more or less similar results for the all configurations while CMB is much slower on Local Feedback, Client Server and Fully Interconnected compared to its results on the Pipeline.

6.3.1 Centralised and Decentralised Approaches

The implementation of DMSO's RTI is not well known but from the results it is reasonable to believe that at least the time advance procedure use a centralised approach since the different federate configurations show so similar results.

CMB is totally decentralised. Every federate knows which other federates that can send messages to it and will stop the execution until it has received a message from each of these federates. To avoid deadlock situations a lot of null messages are sent to other federates to inform about the current simulation time.

The Pipeline is the only configuration where there is no feedback loop. All entities are generated in federate A, passed through all federates in one direction and then finally removed in federate F. In this configuration every federate only has to wait for one other federate and as soon a new message arrives it can be processed.

6.3.2 Comparison of Feedback Loop Handling Between CMB and HLA

In this section we will see how feedback loop is handled by CMB and HLA. We will use an example to show the differences in the way they work.

Example with CMB

Figure 6-1 shows the initial state for a feedback loop configuration with the three federates A, B and C. Federate A is the source and federate C is the sink. The current simulation time is 0 in all federates (shown inside each federate's square). No messages are stored in any of the receive queues (shown with N/A in the figure). All federates has a process time and lookahead of 10 time units. Federate B and C does not have any internal events scheduled.
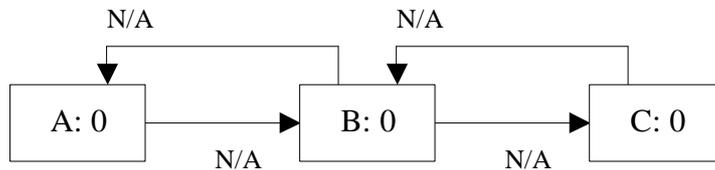
Figure 6-1. Initial state for a feedback loop configuration.
Three federates A, B and C. The current simulation time is shown inside each federate's square.

At time 0 federate A will start to process entity E1. Federate A will now send a message to federate B that it can start process entity E1. This message must be sent now since the lookahead is equal to the process time (10 time units).

Since the receive queues at federate B and C are empty they will send null messages to federate A and C to avoid a possible deadlock. These messages will arrive at time 10. Both federate A and C can only receive messages from federate B and can therefore safely advance to time 10 when the null messages are received. Figure 6-2 shows the current state.
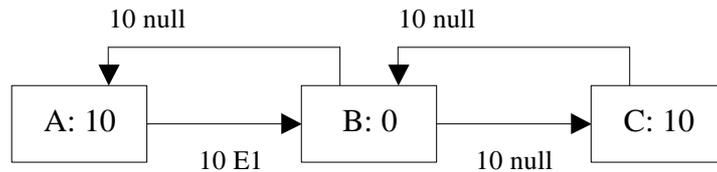
Figure 6-2. State after two null messages and one entity have been sent.

Federate A can now stop the processing of entity E1 and start with entity E2 instead. By the same reason as before federate A must immediately send a message with entity E2 to federate B that will arrive at time 20.

When federate B has received the null message from federate C it knows it is safe to process the message from federate A received at time 10. Federate B will now start to process entity E1 and the current simulation time in B will advance to 10. This will result in a new null message sent to federate A and a message with entity E1 to federate C. Both these message will arrive at time 20.

When federate C receives the first null message from federate B it will send a new null message to federate B that will arrive at time 20.

Figure 6-3 shows the current state.
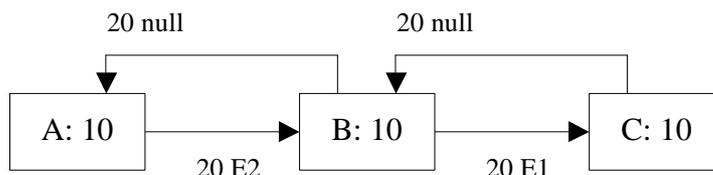
Figure 6-3. State after two null messages and two entities have been sent.

Considering this example it is not difficult to see that an increasing number of number of federates or a smaller lookahead value will increase the number of deadlock situations and null messages needed in the system.

Example with HLA

The same model is used in this example with HLA as middleware. No null messages are sent when HLA is used. Each of the federates must instead frequently ask RTI for time advancements. Figure 6-4 shows the initial state. The current simulation time is 0 in all federates.
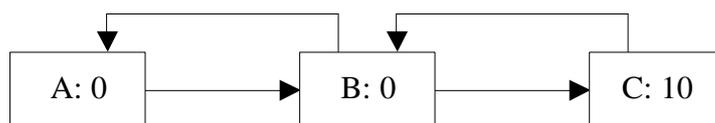
Figure 6-4. Initial state for a feedback loop configuration.
Three federates A, B and C. The current simulation time is shown inside each federate's square.

All federates have declared their lookahead when they initialised with RTI. As seen in the algorithm from Chapter 4, see Figure 4-4, the first step is in the time update cycle

to query RTI for the time when an event earliest can arrive to the federate. Since the lookahead is 10, all federates will get 10 as response. All federates are now allowed to advance to this time and perform any internal events with a time stamp lower or equal to this time.

Federate A is the only federate with an internal event at this stage. At time 0 it will start to process entity E1. It will also send this entity to federate B with an interaction. The interaction will be received at time 10. At time 10 it will stop the processing of entity E1 and start with E2. E2 will also be sent to federate B. It will arrive at time 20. Figure 6-5 shows the current state.
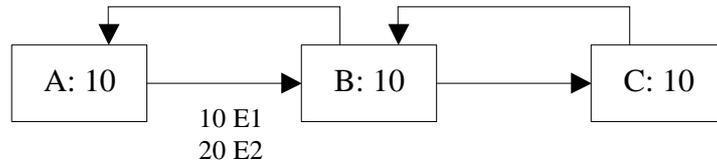
Figure 6-5. State after one entity has been sent.

The next step is to request RTI to advance the time. Each federate will request the time they were allowed to advanced to, in this case 10 for all. All federates requests will be granted as soon as RTI knows it is safe to advance to the requested times and each federate has received any external events waiting in the receive queue with a time equal to the requested time. Federate B will receive the first interaction with entity E1 but entity E2 will remain in the queue. It will directly send entity E1 to federate C to be received at time 20.

All federates will now once again query the RTI for the minimum time of the next event. Since all federates have requested time 10 this mean no events can arrive earlier than at time 20 because of the lookahead of 10.

As can be seen, there is no need for null messages but the federates must wait for each other when they make a time advance request. More federates mean more federates to wait for and smaller lookahead values means more loops in the time update cycle. How the federates interconnected is, however, not that important which the similar results for HLA from the different federate configurations show.

7 Conclusion

This chapter will summarize the report and present what conclusions that can be made from the analysis of the results. Future work will also be discussed.

7.1 Summary

The aim of the work was to investigate time management in COTS based distributed simulation using HLA. Discussions were limited to discrete event simulation and conservative algorithms.

The CSPE "experimental methodology" was derived from Mustafee (2003). It was found to be a useful tool for the performance testing of different types of middleware. The CSPE architecture was modified to use HLA as middleware. A time update cycle algorithm was designed and implemented in Java. Version 5 of RTI 1.3-NG developed by DMSO was used.

Reference Model I (Taylor 2003) with asynchronous entity passing was implemented using the non-persistent object type interactions and the two different time management methods TimeAdvanceRequest (TAR) and NextEventRequest (NER).

The experiments were carried out on an isolated local area network with seven computers. The three variables external/internal event ratio, workload and lookahead was varied in different experiments and tested in four federate configurations. The Chandy-Misra-Bryant (CMB) middleware derived from Mustafee (2003) was also tested on the same hardware and under the same circumstances.

A special program was developed to automate the testing since there were many tests to perform. With five-six variable settings in three different experiments, four federate configurations and three types of middleware there was almost 200 tests that all needed to be run several times.

The results were compared and analysed. Both TAR and NER gave the same results and the analysis showed that these methods were actually doing the same in the algorithm used. Their results, mentioned as just HLA now, were found to be equal or faster than CMB in the three federate configurations with feedback loop. CMB was found to be a little bit faster than HLA in the Pipeline federate configuration.

7.2 Conclusions

This section will discuss the insights I gained about HLA, CMB and CSPE during the work.

7.2.1 HLA

I have found HLA to be very useful for distributed simulation. In fact it can be used for other forms of distributed computing as well with its advanced object and time management, its support for other features such as synchronization barriers and data distribution management.

HLA has become a de-facto standard for distributed simulation. A lot of research is carried out on how to use HLA in different areas and applications. As seen in section 2.2.5, it doesn't prescribe any specific implementation and computer language. This means different techniques can be used to implement the distributed services as long as the developers follow the HLA rules and interface specification. This construction and the fact that many are using the standard make HLA future-safe for new computer languages and new distributed algorithms.

The weak side of HLA is that it is very complex even if you often only need to use parts of all methods defined in the API. When I implemented support for Reference Model I with unbounded buffers (described in section 3.1) I only had to use a limited number of HLA's all methods. This complexity can indicate that HLA is very "heavyweight" standard with a lot of unnecessary functions for COTS distributed simulations.

The experiments in this report show, however, that the performance of HLA is comparable with the "lightweight" middleware CMB and also in models with feedback loops it is much faster. It is also not unlikely that the implementation of more complicated reference models will need to use more of HLA's methods.

The complexity makes the learning curve high and it takes a long time before you have learned everything.

7.2.2 DMSO RTI 1.3-NG Version 5

The RTI version I used in my experiments seemed to be buggy. The RTI software could deadlock, and it often did, when federates were resigning from a federation. I spent many days to find a solution to this problem. Finally I had to implement a feature in the automatic test program that killed processes at the client and the servers if the RTI had deadlocked.

Another bad thing with the RTI version I used was a method called *tick()* that is not a part of the HLA standard. It must, however, be called frequently to give process time to the RTI ambassador. The method can be called with or without two arguments that specifies the lower and upper bound of the time being allocated to tick(). How tick() is used in the program was found to be extremely important for the overall performance. Not well-chosen values could mean double execution time or even worse. The values I found to be best and that I used in my experiments was tick(0.0, 0.0).

7.2.3 CMB

The Chandy-Misra-Bryant (CMB) algorithm was fast for the federate configuration without feedback loop but slow for the other configurations tested. The algorithm is simple and easy to understand. The CMB middleware doesn't have any built-in support for any advanced distributed computing methods, only message passing, and is alone not developed enough for future interoperability issues.

7.2.4 CSPE

I found the CSPE "experimental methodology" to be very useful for distributed simulation experiments. CSPE makes it possible to investigate different kinds of middleware and vary federate configurations and variable settings. The architecture consist of the three parts CSPE, CSPE handler and the middleware. This dividing

separates the simulation's internal implementation from the middleware in a very good way. The handler also has a good API for communication with CSPE.

Since the structure of CSPE is well known (see Mustafee 2003), it is possible to implement more functionality to investigate new COTS interoperability problems. For example, if you want to do experiments on bounded buffers, you can implement support for this into CSPE, add new function calls to the API and then develop middleware that can provide this functionality. You will have full control of all parts and it will be possible to try different solutions.

Without a tool such as CSPE, distributed simulation experiments with COTS simulation packages would be limited to the API calls that the software developers implement in their packages. It is also reasonable to believe that it will take a lot more time to try different solutions if real CSP packages would be used in the experiments.

CSPE however, needs further development to support more reference models when these will be designed, implemented and tested. The version of CSPE I used (and the only one that currently exists) can also be improved a little bit in future versions if the number of classes is reduced.

7.3   Future Research and Development

7.3.1 Other RTI Versions

It is difficult to compare HLA to other middleware when the implementation is not standardized and can differ in different RTI implementations. It would therefore be interesting to compare the results I got using DMSO RTI 1.3-NG Version 5 with results from experiments, carried out using the same circumstances, with other RTI implementations. It would also be interesting to compare these results to a simple centralized approach since the results is similar for the different federate configurations.

7.3.2 Combination of External Control and Permission Request

In the end of my work I found that it is probably possible to combine the external control (EC) and permission request (PR) synchronization algorithms in a new algorithm.

EC was used in this work to control CSPE. The middleware defined a safe time that the simulation executive was allowed to advance to. When this time was reached, it returned the time to the middleware to let it know the current simulation time. This time was then used to request time advancement from the RTI. If, instead, the time of the next scheduled internal event should be returned, it sometimes would be possible to request a higher time and in this way speed up the simulation since other federate may have to wait less.

This is, however, only possible if the NextEventRequest method is used since the federate in this case only promise that it will not send any external events earlier than the granted time + lookahead, see section 2.2.5. The granted time will be equal what is lowest of the time of the next event that will received and the requested time.

This is a form of PR since CSPE requests to advance to its next internal event. If no external event arrives it will be allowed to do that. If a higher safe time can be defined by the middleware CSPE will be allowed to advance to this higher time instead.

Figure 7-1 shows the sequence diagram for the time update cycle of this algorithm.
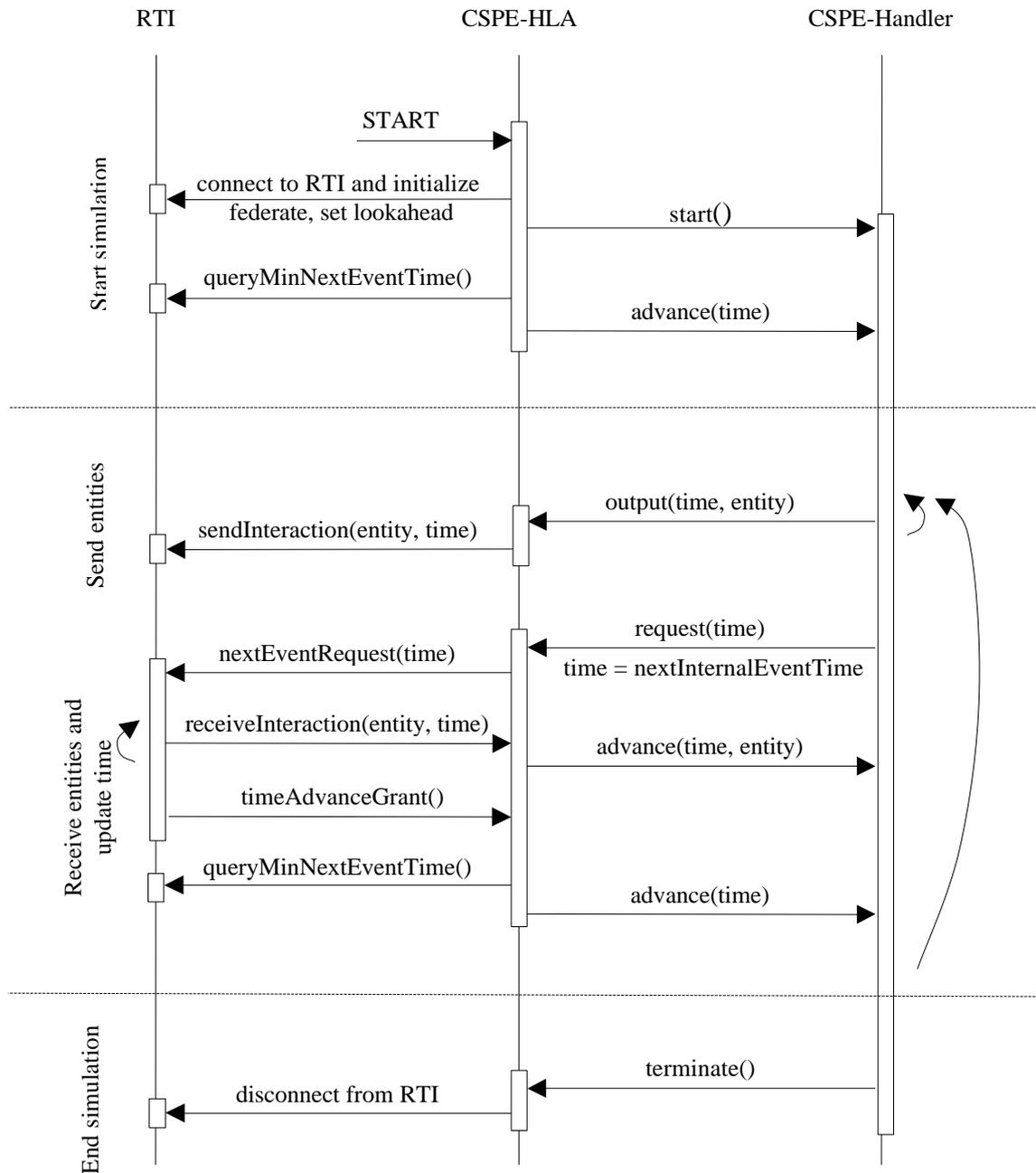
Figure 7-1. Sequence diagram of the time update cycle of the EC/PR algorithm

7.3.3 Other reference models

In this work only Reference Model I was implemented and performance tested. Other more reference models, for example bounded buffers, can demand more of the middleware and maybe give other results. It would be interesting to investigate how HLA can be used with these reference models.

I think it is difficult or maybe even impossible to implement Reference Model II (synchronous entity passing with bounded buffer) in a middleware that is using HLA, if the distributed simulation should be conservative and if the lookahead value should not be set to 0. The receiver of an entity must be able to respond that the buffer is full and that the entity cannot be sent yet. This response must, however, arrive at the same time as the entity is sent if the sending federate should not start processing the next entity. This demands a lookahead of 0.

An alternative to a lookahead of 0 is maybe that sender always is aware of the status of the receiver's buffer. This assumes that processing times at the receiver is known and that only one federate can send entities to the receiving federate. This alternative seems to be very complicated and a lookahead of 0 is presumably the only real solution. The problem with a lookahead of 0 is that is will probably give bad overall performance.

An optimistic approach would not have this problem. In any cases more reference models must be investigated to be able to answer which middleware that is best for COTS based distributed simulations.

CSPE needs further development if more reference models should be implemented and tested. It would also be interesting to see if the middleware performance changes if the processing times of the workstations are randomised. Today all processing times are fixed values that can be specified.

References

Banks, J., J. S. Carson II, B. L. Nelson, and D. Nicol (2001). Discrete-Event System Simulation, 3d ed., Prentice-Hall, New Jersey. 3-22.

Boer, C. A., and A. Verbraeck (2003). Distributed Simulation with COTS Simulations Packages. In Proceedings of the 2003 Winter Simulation Conference. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (eds.). Institute for Electrical and Electronics Engineers, New Jersey. 829-837.

Bryant, R. E. (1977). Simulation of packet communications architecture computer systems. MIT-LCS-TR-188.

Chandy, K. M. and J. Misra (1978). Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. IEEE Transactions on Software Engineering SE-5(5). 440-452.

Chandy, K. M. and Misra, J. (1981). Asynchronous Distributed Simulation via a Sequence of Parallel Computations. Communications of the ACM. 24(11). 198 - 206.

DMSO. RTI 1.3-Next Generation Programmer's Guide Version 5. Department of Defense. Defense Modeling and Simulation Office.

Fujimoto, R. M. (1990). Parallel Discrete Event Simulation. Communications of the ACM. 33(10). 30-53.

Fujimoto, R. M. (1999). Parallel and Distributed Simulation. In Proceedings of the 1999 Winter Simulation Conference. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans (eds.). Association for Computing Machinery Press, New York, NY. 122-131.

Fujimoto, R. M. (2003). Distributed Simulation Systems. In Proceedings of the 2003 Winter Simulation Conference. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (eds.). Institute for Electrical and Electronics Engineers, New Jersey. 124-134.

Law, A. M., and W. D. Kelton (2000). Simulation Modeling and Analysis, 3d ed., McGraw-Hill, New York.

Mustafee, N. (2003). Performance Evaluation of Interoperability Methods for Distributed Simulation. Department of Information Systems and Computing, Brunel University, England.

Paul, R. J. and D. W. Balmer (1998). Simulation Modelling. Brunel University, England. 1-25, 102-108

Ryde, M. D. and S. J. E. Taylor (2003). Issues using COTS Simulation Software Packages for the Interoperation of Models. In Proceedings of the 2003 Winter Simulation Conference. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (eds.). Institute for Electrical and Electronics Engineers, New Jersey. 772-777.

Swain, J. J. (2001). Power tools for visualization and decision making: 2001 simulation software survey. OR/MS Today 28(1). Baltimore, Maryland: Institute for Operations Research and Management Science.

Taylor, S. J. E., R. Sudra, T. Janahan, G. Tan and J. Ladbrook (2002). GRIDS-SCF: An Infrastructure for Distributed Supply Chain Simulation. In SIMULATION, Vol. 78, Issue 5, May 2002. The Society for Modeling and Simulation International. 312-320.

Taylor, S. J. E., S. J. Turner and M. Y. H. Low (2004). A Proposal for an Entity Transfer Specification Standard for COTS Simulation Package Interoperation. European Simulation Interoperability Workshop 2004. 04E-SIW-081

Taylor, S. J. E., J. Sharpe and J. Ladbrook (2003). Time Management Issues in COTS Distributed Simulation: A Case Study. In Proceedings of the 2003 Winter Simulation Conference. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (eds.). Institute for Electrical and Electronics Engineers, New Jersey. 838-846.

Taylor, S. J. E. (2003). HLA-CSPIF: The High Level Architecture – COTS Simulation Package Interoperation Forum. In Proceedings of the Fall 2003 Simulation Interoperability Workshop. Simulation Interoperability Standards Organisation, Institute for Simulation and Training. Florida. 03F-SIW-126.

Taylor, S., B. Gan, S. Straßburger, A. Verbraeck (2003). HLA-CSPIF Panel on Commercial Off-the-Shelf Distributed Simulation. In Proceedings of the 2003 Winter Simulation Conference. S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice (eds.). Institute for Electrical and Electronics Engineers, New Jersey. 881-887.

Appendix A – Table of Acronyms

API	Application Programming Interface
CMB	Chandy-Misra-Byrant
COTS	Commercial Off the Shelf
CSP	COTS Simulation Package
CSPE	COTS Simulation Package Emulator
CSPE-CMB	CSPE using CMB for time management
CSPE-HLA	CSPE using HLA for time management
DLL	Dynamic Link Library
DMSO	Defense Modeling and Simulation Office in US
EC	External Control
EE	External List Externalisation
FED	Federation Execution Details
FedExec	Federation Executive
HLA	High Level Architecture
HLA-CSPIF	HLA-CSP Interoperability Forum
IA	Incremental Advance
IEEE	Institute of Electrical and Electronics Engineers
LBTS	Lower Bound Time Stamp
libRTI	RTI Library
LP	Logical Process
NER	Next Event Request
PR	Permission Request
RO	Receive Order
RTI	Run-Time Infrastructure
RtiExec	RTI Executive
TAR	Time Advance Request
TSO	Time Stamp Order