Fault-Tolerance in HLA-Based Distributed Simulations

Martin Eklöf

A dissertation submitted to
the Royal Institute of Technology
in partial fulfillment of the requirements for
the degree of Licentiate of Philosophy

Department of Electronic, Computer & Software Systems

TRITA-ICT/ECS AVH 06:03
ISSN 1653-6363
ISRN KTH/ICT/ECS AVH-06/03--SE

Abstract

Successful integration of simulations within the Network-Based Defence (NBD), specifically use of simulations within Command and Control (C2) environments, enforces a number of requirements. Simulations must be reliable and be able to respond in a timely manner. Otherwise the commander will have no confidence in using simulation as a tool. An important aspect of these requirements is the provision of fault-tolerant simulations in which failures are detected and resolved in a consistent manner. Given the distributed nature of many military simulations systems, services for fault-tolerance in distributed simulations are desirable. The main architecture for distributed simulations within the military domain, the High Level Architecture (HLA), does not provide support for development of fault-tolerant simulations.

A common approach for fault-tolerance in distributed systems is check-pointing. In this approach, states of the system are persistently stored through-out its operation. In case a failure occurs, the system is restored using a previously saved state. Given the abovementioned shortcomings of the HLA standard this thesis explores development of fault-tolerant mechanisms in the context of the HLA. More specifically, the design, implementation and evaluation of fault-tolerance mechanisms, based on check-pointing, are described and discussed.

Keywords: *HLA, Fault-tolerance, Distributed Simulations, Federate, Federation*

Sammanfattning

Framgångsrikt nyttjande av simulering som ett verktyg inom ramen för det Nätverksbaserade Försvaret (NBF) och ledningssystem innefattar fullgörande av ett antal krav. Simuleringar måste vara tillförlitliga (robusta) och kunna leverera resultat inom givna tidsramar. En viktig aspekt av detta är stöd för feltoleranta simuleringar som inkluderar detektering av fel och återställning av felande komponenter. Många av de simuleringar som återfinns inom det militära är distribuerade till sin natur. Den mest väletablerade standarden för distribuerade simuleringar, High Level Architecture (HLA), stödjer dock inte feltolerans i någon större utsträckning. Därav är det viktigt att utveckla metoder för feltolerans inom ramen för HLA i syfte att på sikt kunna införliva simuleringar i NBF.

En vanligt förekommande metod för feltolerans inom distribuerade system är att kontinuerligt spara ett systems tillstånd under dess exekvering. Om ett fel inträffar återställs systemet genom att utnyttja det senaste sparade tillståndet. Detta arbete ser på möjligheterna att utveckla mekanismer för feltolerans inom HLA genom att utnyttja detta angreppssätt. Arbetet beskriver design, utveckling och analys av en mekanism som möjliggör feltoleranta HLA-baserade simuleringar.

Nyckelord: *HLA, Feltolerans, Distribuerad Simulering, Federat, Federation*

Acknowledgements

This research was funded by the Swedish Defence Research Agency (FOI). It was made possible by support from the head of the Division of Systems Technology, Monica Dahlen, and the head of the Department of Systems Modeling, Farshad Moradi.

First and foremost I would like to express my gratitude to my supervisor at the Royal Institute of Technology (KTH), Professor Rassul Ayani, for his great support and fruitful discussions during these years. I would like to thank my colleague, Jenny Ulriksson, for great support during the process of completing this thesis, and for challenging discussions over the years. Further, I wish to thank the project manager of the NetSim project, Farshad Moradi, for allowing me to carry out my research and for great support during the way. I would also like to thank my opponent, Dr Gary Tan, for his constructive comments on a draft of the thesis.

Furthermore, I would like thank current and past members of the NetSim project, especially Marianela Garcia Lozano and Magnus Sparf, for great collaboration and support. Last but not least, I would like to thank all colleagues at the Department of Systems Modeling for making it a great place to work in.

Contents

Part I – An overview of main issues in fault-tolerant distributed simulations

1. Introduction

1.1 Background

Modeling and Simulation (M&S) in the context of Command and Control (C2) systems and the Network-Based Defense (NBD) provide efficient means for decision support, planning of operations, as well as training. Simulation tools in these settings supply the decision maker with support that enables faster decisions, and also improves the overall quality of decisions made. Thus, in order for simulations to be considered beneficial in the context of C2 systems and the NBD, they must respond in a timely-fashion and at the same time provide reliable results.

Today, methodology for distributed simulations is important in the development of simulation systems. This is motivated by the nature of many of today's simulation models, requiring access to vast processing capacity, and the benefit of simulation decomposition to promote reuse and/or connection of geographically dispersed units. The High Level Architecture (HLA) is the most widely adopted standard for distributed simulations in the defense sector. In HLA a simulation model is decomposed into logical units referred to as federates, whereas the simulation (a set of federates) is referred to as a federation.

1.2 Motivation

The distribution of a simulation system certainly has its merits but will typically lead to a higher failure rate. This is simply due to the fact that the probability for failure increases as the number of machines of the distributed simulation system rises. From the perspective of a decision support system the failure of a critical simulation component is in most cases unacceptable. If time is a constraining factor, rerunning a simulation due to malfunction is not plausible, and an undetected failure may interfere negatively with simulation results, which may bring catastrophic side effects.

Given this, it is crucial to provide services for failure detection and recovery to enable robust execution of simulations, i.e. support for fault-tolerance is required for distributed simulations in the context of C2 systems and the NBD. The HLA standard does not treat fault-tolerance extensively, nor has the research community explored this topic sufficiently. The next generation of the HLA, the HLA Evolved, focuses more on fault-tolerance compared to its predecessors, but the new standard mainly addresses detection of faults. Thus, it is necessary to develop scalable and efficient means of failure recovery in HLA-based distributed simulations.

1.3 Problem formulation

In this thesis we investigate how to design, develop, test and analyze Fault-Tolerance (FT) mechanisms that can be used in HLA-based distributed simulations. In particular, we are interested in FT mechanisms that utilize check-pointing protocols.

We will also investigate the efficiency of the FT mechanisms, since time is essential in many situations where simulation is used as a real-time decision-support tool.

1.4 Thesis Outline

Chapter 2 gives a brief introduction on distributed systems as a basis for understanding terms and acronyms used in later sections and chapters. Furthermore, it provides some general definitions related to fault-tolerance and specifically addresses check-pointing as a basis for leveraging fault-tolerance in distributed systems.

Chapter 3 describes fault-tolerance in the context of distributed simulations. First, a brief introduction to distributed simulations is provided, after which the High Level Architecture (HLA) is described, especially time-management in HLA since this is of importance for the fault-tolerance mechanism described in this thesis. Moreover, this chapter provides some information regarding the next generation of the HLA and the extended fault-tolerance support provided in this version. Finally, some recent work in enabling fault-tolerant HLA-based simulations is described.

Chapter 4 presents the scientific contributions of this thesis. It gives a short description of the context within which fault-tolerant HLA-based simulations has been explored. It then briefly describes the Distributed Resource Management System (DRMS) that provides fault-tolerance services for HLA-based simulations and the fault-tolerance mechanism implemented in this system. Finally, this chapter describes the results and conclusions drawn from the experiments made in order to evaluate the fault-tolerance mechanism.

Chapter 5 discusses some possible extensions to the fault-tolerance mechanism proposed in this thesis and provides some crucial points for further evaluation of the mechanism.

Chapter 6 provides references used in chapter 1 to 5.

1.5 Summary of scientific contribution

The main contributions of this thesis have been published in conference proceedings and in a journal as described below:

I. M. Eklöf, J. Ulriksson & F. Moradi. 2003. *NetSim – A Network Based Environment for Modeling and Simulation*. NATO Modeling and Simulation Group, Symposium on C3I and M&S Interoperability, Antalya, Turkey.

   Summary and contribution: This paper describes development of a common environment for M&S, referred to as NetSim, supporting the Swedish Armed Forces in a C2 context. The author of this thesis contributed to the development of this paper in cooperation with J. Ulriksson and F. Moradi and was primarily responsible for the section on resource management and distributed simulation execution through the DRMS.

II. M. Eklöf, M. Sparf, F. Moradi, & R. Ayani. 2004. *Peer-to-Peer-Based Resource Management in Support of HLA-Based Distributed Simulations*. SIMULATION 80: 181 – 190.

Summary and contribution: This paper describes the initial architecture and implementation of the DRMS. Further, it discusses federate migration that could be utilized for load-balancing purposes or fault-tolerance. The author of this thesis developed the initial ideas and design of the DRMS in cooperation with M. Sparf, F. Moradi and R. Ayani. He was also responsible for implementation of the HLA-related components of the system and main contributor to the paper.

III. M. Eklöf, F. Moradi & R. Ayani. 2005. *A Framework for Fault-Tolerance in HLA-Based Distributed Simulations*. Proceedings of the 2005 Winter Simulation Conference (WinterSim), Orlando, Florida.

Summary and contribution: This paper describes a refined architecture and implementation of the DRMS. Further, a mechanism for fault-tolerance in HLA-based distributed simulations is proposed. The author of this thesis was the main contributor to the development of the refined DRMS and the fault-tolerance mechanism.

IV. M. Eklöf, F. Moradi & R. Ayani. 2006. *Evaluation of a Fault-Tolerance Mechanism for HLA-Based Distributed Simulations*. Proceedings of the 20th Workshop on Parallel and Distributed Simulations (PADS), Singapore.

Summary and contribution: This paper describes an evaluation of the proposed fault-tolerance mechanism in the context of the refined DRMS. The author of this thesis was responsible for conducting the evaluation of the proposed fault-tolerance mechanism and main contributor to the development of this paper.

2. Fault-tolerance

In this chapter a brief introduction to distributed systems and fault-tolerance is given. More specifically, this chapter addresses check-pointing methods as a basis for fault-tolerance in distributed systems.

2.1 Distributed systems

Several definitions of distributed systems exist. In [Tannenbaum & Steen 2002] the following definition is provided:

"A distributed system is a collection of independent computers that appears to its users as a single coherent system"

An important notion of a distributed system is that its hardware, i.e. individual computers of the system, is autonomous. Further, the actual distribution of the system is transparent to its users, meaning that they perceive it as an ordinary, non-distributed system.

Parallel and distributed simulations are often conceptualized as a set of Logical Processes (LPs). In the following sections we use the concept of LPs to describe distributed systems. To deliver functionality, the LPs of a distributed system exchange messages. The message exchange is carried out by means of a communication protocol such as Remote Procedure Call (RPC) or Remote Method Invocation (RMI).

2.2 Fault-tolerance in distributed systems

In a distributed system, a failure is often partial, i.e. one or some of the components of the system fails. A partial failure is most often not critical since the entire system will not be brought down. The failure of an LP may have an impact on the proper operation of other LPs. However, in some cases other LPs may remain unaffected. In contrast to this, a failure in a non-distributed system will often cause malfunction of the entire application.

[Tannenbaum & Steen 2002] defines four aspects that are important in understanding fault-tolerance. First of all, fault-tolerance is strongly associated with something called *dependable systems*, which in turn covers the following features:

1. Availability
2. Reliability
3. Safety
4. Maintainability

Availability refers to the probability that a distributed system is able to deliver its services in an expected way at any given time. Reliability refers to the probability that

the distributed system can deliver its services during a certain time interval. Safety means that a temporary malfunction of the distributed system will not cause disastrous effects. Finally, maintainability describes how easy it will be to repair a distributed system that does not function in an expected way. These definitions give some basic understanding of the requirements that are imposed on a fault-tolerant distributed system.

If a system can not meet its promises, it is said to *fail*. In a service oriented distributed system, this means that when one or more of the individual services cannot perform in an intended manner, it has failed. An *error* is part of a system's state, which may cause a failure in the system. For instance, if data packets are sent over a network, some of these may be corrupted as they reach their destination, potentially causing the receiving component to fail. Finally, the cause of an error is a *fault*. In the case of corrupted data packets, the cause may originate from a bad transmission medium [Tannenbaum & Steen 2002]

Occurring faults are either transient or permanent. Transient faults occur for a limited time interval and are usually caused by some temporary breakdown in parts of the system. Permanent faults are caused by major breakdowns in system components and persist until failed components are fixed or replaced. Generally, development of fault-tolerant services for distributed systems considers permanent faults [Agarwal 2004].

As expected there are numerous potential causes for a failure in a distributed system and these faults will induce different types of failures as well. Based on [Cristian 1991; Hadzilacos & Toueg 1993], [Tannenbaum & Steen 2002] outlined a classification scheme for failures, as shown in table 1.

Table 1. Classification of failures in a distributed system [Tannenbaum & Steen 2002].

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure - Receive omission - Send omission	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure - Value failure - State transition failure	A server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

The purpose of implementing fault-tolerant distributed systems is to avoid failures, even though faults are present. Ideally, a fault-tolerant system should mask the presence of faults. A distributed system comprises several sub-systems, whose failure should not affect the overall system performance [Agarwal 2004].

There is no general method for fault-tolerance in distributed systems, but two reoccurring phases can be identified, error detection and recovery. There are

numerous techniques for recovery in distributed systems. These can be classified into two main categories [Damani & Garg 1998]:

- Replication-based techniques
- Check-pointing-based techniques

In replication-based approaches one or more copies of an LP is maintained in addition to the main LP. In case of failure, one of these replicas will take the failed LP's place. In check-pointing-based approaches, states of individual LPs are saved on a stable storage device. In case of failure, an LP is restarted using the last stable state saved on stable storage. The following sections will describe fault-tolerance techniques based on check-pointing and more specifically address rollback recovery.

2.2.1 Check-pointing techniques

The purpose of fault-tolerance services of a distributed system is to enable recovery of the system to a consistent state in case of failure. If considering a single process, i.e. a uni-processor application, this is fairly simple. The process saves checkpoints on stable storage and recovers, in case of failure, using the latest saved state. However, if a system comprises multiple communicating LPs, then it becomes more complicated. In this case the system state includes the states of all LPs [Agarwal 2004] and it is required to do the recovery based on a consistent system state. [Chandy & Lamport 1985] defines as consistent system state as one where messages received by LPs and reflected in their states are at the same time reflected as sent messages in other LPs states.

A fundamental device used in check-pointing-based approaches is the stable storage. All LPs of a distributed system have access to this device and use it periodically to save check-points. At a minimum, the checkpoint in this case comprises the states of all individual LPs. Given the requirements on the fault-tolerance protocol, the design of the stable storage device will differ. The purpose of the stable storage device is to enable persistent storage of check-points through-out the operation of the system. If only transient faults are tolerated the stable storage could reside in the local context of an LP, e.g. local disk in each host. Given that non-transient faults are tolerated this is not sufficient. Non-transient faults are permanent faults and thus, the stable storage must reside outside of the hosts of the participating LPs, e.g. a replicated file system [Elnozahy et al. 2002].

The saving of checkpoints to stable storage can be accomplished in two different ways; either by coordinated or un-coordinated check-pointing. As the name implies, in coordinated check-pointing, LPs cooperate in producing a snapshot of the system state. In un-coordinated check-pointing, LPs report their states to stable storage individually, which of course will have an effect on how the recovery of a failed LP is accomplished, see next section for details.

2.2.2 Rollback recovery

One way of realizing check-pointing-based recovery is to employ rollback recovery. In this approach, a consistent system state is reached by rolling back participating LPs in time when recovering from a failure. Section 3.1.2 provides more information on

the rollback mechanism. Thus, the fundamental idea of rollback recovery is to bring a system back to a system consistent state in case a fault causes inconsistencies. However, it is not certain that the consistent state is one that occurred prior to the failure. The recovery protocol just assures that it is a state that can occur in a failure free execution [Elnozahy et al. 2002].

Looking at a distributed system as a set of LPs exchanging messages, i.e. a message-passing system, rollback recovery becomes a fairly complicated matter. This is because messages exchanged during execution impose inter-LP dependencies. The effect of this may become evident upon failure and subsequent recovery of an LP. Due to the inter-LP dependencies, LPs that have not failed may also be forced to rollback. This is commonly referred to as *rollback propagation*.

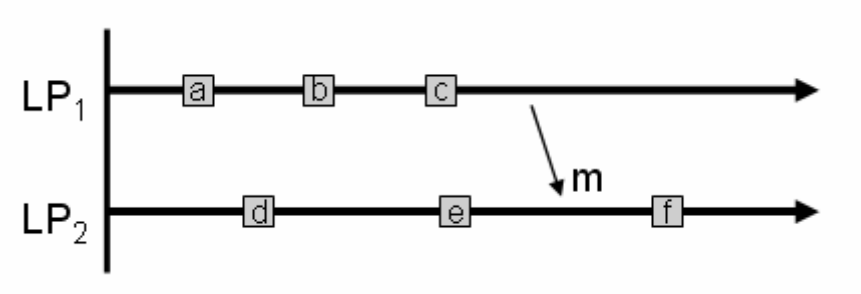

Figure 1. Rollback propagation in case of rollback.

Consider the case depicted in figure 1 where two LPs exchange messages. Grey boxes indicate states of LPs, which are successively saved during the operation of the system. In case LP_1 fails during the operation of the system it will initiate a recovery based on a state that has not recorded the sending of message m. This requires that LP_2 makes a rollback to a state that does not record the receipt of message m. In this case LP_2 uses state e in the rollback procedure. Otherwise the states of the LPs would be inconsistent, i.e. the state of LP_1 (state c) does not record the sending of message m whereas the state of LP_2 (state f) records the receipt of that exact message. In the worst case a system may rollback to the initial state from where the execution began, which is referred to as the *domino effect*.

One way of avoiding rollback propagation, and in the worst case the domino effect, is to use coordinated check-pointing. In this case a system consistent state exist, which can be seen as a lower bound for rollback. Un-coordinated check-pointing does not guarantee absence of rollback propagation or the domino effect but is advantageous in the sense that each LP decides when to take the snapshot. Thus, taking the checkpoint when the state comprises small amounts of data may reduce the communication overhead.

Independent check-pointing

As the name implies, LPs do not cooperate to produce their checkpoints in independent check-pointing. Instead, check-points are established individually by all LPs. LPs maintain two different logs, namely the volatile log and the stable log. The volatile log records check-points of the LP state for each processed event. Periodically, samples are taken from the volatile log and brought to the stable log. In

case of recovery, an LP uses the last saved state in stable log as the current state. Next, the recovered LP sends a message to each neighboring LP, stating the number of messages the recovered LP has sent to the concerned LP in the current state. Then, the neighboring LP checks if the number of messages received from the recovered LP in the current state is greater than the number in the received message. If this is the case the neighboring LP is rolled back to a state where these numbers are equal. The rollback will in turn produce rollback messages for other neighboring LPs. Finally, when the states of all LPs are consistent with the states of neighboring LPs, a globally consistent state is reached [Agarwal 2004].

Coordinated check-pointing

In coordinated check-pointing, LPs cooperate in order to produce a snapshot of the system state. Coordinated check-pointing simplifies the process of LP recovery and avoids the domino effect. A common approach used in coordinated check-pointing is to block the communication while the snapshot is taken. One of the LPs acts as a coordinator broadcasting a request for execution of the check-point procedure. The LPs flush all communication channels and produces a tentative check-point after which an acknowledgement is sent to the coordinator. When all LPs have acknowledged production of a check-point the coordinator sends a commit message, which instructs the LPs to make the tentative check-point the current one, thus removing the old check-point. After this the LPs resumes normal execution. There are also non-blocking coordinated check-pointing schemes available [Elnozahy et al. 2004].

3. Fault-tolerance in distributed simulations

In this chapter a brief introduction to distributed simulations and the HLA is given. Moreover, support for fault-tolerance in HLA is discussed and some related work is presented.

3.1 Distributed simulations

Simulation systems are usually categorized as being either continuous or discrete. In continuous systems state variables change continuously over time, whereas in a discrete system changes occur at certain points in time. The latter is usually referred to as Discrete Event Simulation (DES). There are two main approaches for advancing time in a DES, namely [Moradi and Ayani 2003]:

- Time-stepped approach where the simulation time is advanced by a fixed time interval
- Next-event approach where the simulation time is advanced to the time of the next event

A traditional DES runs on a single processor machine and thus behaves in a sequential manner. However, as modern simulation models require vast amounts of processing capacity a sequential machine will not suffice. To cope with large simulation systems, a number of techniques for parallel and distributed simulations have been developed. Parallel DES (PDES) aims at reducing the time spent on executing a simulation through parallelization of the system. Distributed simulation on the other hand aims at executing several interacting simulation models on a network of computers. The benefit of distributed simulation is increased reuse of simulation models, enhanced interoperability between simulation models and potential for massive scalability [Moradi and Ayani 2003].

When designing a parallel or distributed simulation system it is required to decompose the target system into logical units. These units are usually referred to as Logical Processes (LPs). Depending on the simulation task at hand the decomposition differs, but usually an LP represent a physical process of some kind. The LPs of a simulation system communicates through exchange of time-stamped messages (events). Each LP maintains its own logical time and operates on a list of received events.

3.1.1 The High Level Architecture – HLA

Today, the High Level Architecture (HLA) is the de-facto standard for distributed simulations in the defense domain. HLA was originally developed by the Defense Modeling and Simulation Office (DMSO) to support reuse and interoperability across the large number of simulations developed and maintained by the U.S. Department of

Defense (DoD). The HLA baseline definition was completed in 1996, and since 2000, the HLA is an approved open standard through the Institute of Electrical and Electronic Engineers (IEEE) – IEEE Standard 1516.

An HLA-based simulation is referred to as a federation, whereas individual participating components are called federates. Federates can be of numerous types, ranging from manned simulators to federation support systems. A federation is formed by connecting individual federates to a Run-Time Infrastructure (RTI). The RTI is an implementation of the HLA standard and provides basic services that enable interaction between participating federates [Kuhl et al. 1999]. Figure 2 illustrates a simple federation in which three federates are connected to the RTI and interact through defined services.

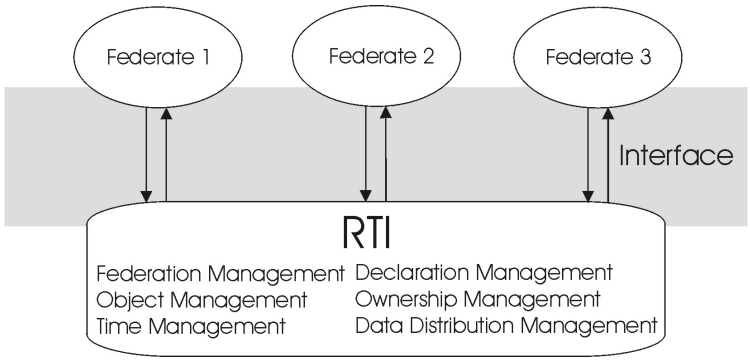

Figure 2. Federate interaction through services provided by the Run-Time Infrastructure (RTI).

The HLA standard comprises three major components; the HLA framework and rules [HLA Framework and Rules 2001], the HLA federate interface specification [HLA Interface Specification 2001] and the HLA Object Model Template [HLA Object Model Template 2001]. Below, these components are briefly described:

- **HLA Framework and Rules**: This document defines the HLA, its components and the responsibilities of federates and federations. To ensure consistency of an HLA federation, two sets of rules must be obeyed. The first set of rules defines that [HLA Framework and Rules 2001]:

    1. Federations shall have an HLA Federation Object Model (FOM), documented in accordance with the HLA Object Model Template (OMT).
    2. In a federation, all simulation-associated object instance representations shall be in the federates, not in the RTI.
    3. During a federation execution, all exchange of FOM data among joined federates shall occur via the RTI.
    4. During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification.
    5. During a federation execution, an instance attribute shall be owned by at most one joined federate at any given time.

The second set of rules defines that [HLA Framework and Rules 2001]:

1. Federates shall have an HLA Simulation Object Model (SOM), documented in accordance with the HLA OMT.
2. Federates shall be able to update and/or reflect any instance attributes and send/or receive interactions, as specified in their SOMs.
3. Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs.
4. Federates shall be able to vary the conditions (e.g. thresholds) under which they provide updates of instance attributes, as specified in their SOMs.
5. Federates shall be able to manage local time in a way that will allow them to coordinate the data exchange with other members of a federation.

- **HLA Federate Interface Specification**: The HLA was defined to provide a common architecture for M&S, integrating various simulations. Thus, HLA relies on a standardized inter-federate interaction API. The HLA federate interface specification document defines this interface [Seiger 2000]. The interface specification defines six basic types of RTI services, these are [HLA Interface Specification 2001]:

  1. **Federation Management** (FM): FM refers to the creation, dynamic control, modification and deletion of a federation execution. Thus, the FM services are used to control federation wide activities during a federation execution.
  2. **Declaration Management** (DM): DM refers to the declaration of individual federates to receive and/or produce certain types of data. The DM services manages the publish/subscribe model for the data exchange within a federation.
  3. **Object Management** (OM): OM refers to registration, modification, and deletion of object instances, but also the sending and receipt of interactions. Thus, OM manages the lifecycle and message passing for object instances.
  4. **Ownership Management** (OSM): OSM refers to the transfer of ownership of object instance attributes between federates. Thus, OSM enables cooperative modeling of a given object instance across a federation.
  5. **Time Management** (TM): TM refers to means of ordering the delivery of messages throughout a federation execution. Thus, TM provides services for coordinating the federate time advancement along the federation time axis.
  6. **Data Distribution Management** (DDM): DDM refers to the reduction of both the transmission and the reception of irrelevant data. Thus, DDM provides services that make the data transmission among federates more efficient.

- **HLA Object Model Template (OMT)**: The OMT defines the format and syntax for representing the information in HLA object models. This includes object, attributes, interactions and parameters. The OM could be seen as a template for documenting information in HLA federations. The OM comprises two different templates, namely the Federation Object Model (FOM) and the Simulation Object Model (SOM). The purpose of the FOM is to define the data exchange in a standardized and common format, for a set of federates of a federation. The SOM specifies what capabilities an individual federate can bring to a federation [HLA Object Model Template 2001].

3.1.2 Time management in HLA

As computers in a distributed simulation do not share a common clock it is required that a virtual time, usually referred to as logical time, is introduced for each member of the simulation. A time synchronization protocol is used to maintain the logical time of members and ensures the causal ordering of events.

Within time-stepped simulations, time is advanced in fixed time steps. A time-stepped federate will use a time step, s, which also represents the federate's lookahead value. Given that the federate is at logical time t, it will produce events having a timestamp of t + s, t + 2s, etc. In figure 3, the evolution of a typical time-stepped simulation is illustrated. The solid line in figure 3 represents the federate's logical time, whereas the dotted line represents the lower bound of timestamp for events that the RTI will accept. The federate performs the cycle of requesting advancement of time (TAR in figure 3) and being granted the requested time (Grant in figure 3) [Kuhl et al. 1999].

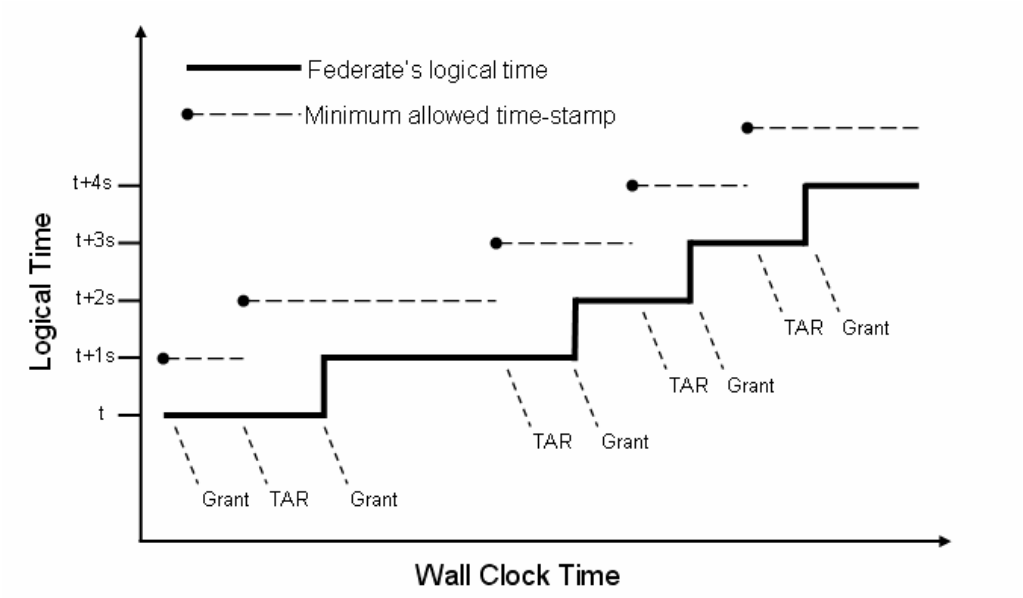

Figure 3. Evolution of time in a time-stepped simulation [reproduced from Kuhl et al. 1999].

Consider the case where the federate makes a request to advance its time to t + 2s. At this point the federate's logical time is at t + s and it must produce events having a time-stamp of at least t + 3s (given by the dotted line). This is because the RTI interpret a time advancement request (TAR) from a federate as a promise saying that it will not produce events earlier than the request time. The federate in the example

will not be granted advancement to t + 2s until all other joined federates have made a time advancement request for this time. The pace at which a time-stepped federation will progress, will of course be dependent on the performance of each federate. A federate, performing extensive computation between the Grant and TAR will slow down the entire federation [Kuhl et al. 1999].

The HLA Time Management Design Document [HLA TM 1996] describes the life of a time-stepped federate in the following way:

```
Become time-regulating and constrained
While federation execution still in progress:
      Compute state of federate at time now.
      Provide any changed information to the RTI.
      Receive all external events in the time step.
      Invoke one of RTI's Time Advance Request with the supplied
      argument
            (now + step)
      Respond to possible RTI requests for Reflect Attribute Value
      and Receive Interaction
      Honour RTI's invocation for Time Advance Grant
```

In this context, time-constrained means that the federate receives TSO (Time Stamp Ordered) events in time stamp order, whereas time-regulating means that the federate is able to send TSO events.

In addition to the time-stepped approach, a federation can utilize the conservative approach. In the conservative case the federation is event-driven. Basically, the federate processes the event with the smallest future logical time, i.e. the federate can not receive events having a time-stamp smaller than this. The federate process events received from other federates and correspondingly advances its logical time according to the time-stamp of these events. Thus, the logical time will not progress in even steps during the simulation execution [Kuhl et al. 1999].

The HLA Time Management Design Document [HLA TM 1996] describes the life of an event-driven federate in the following way:

```
Become time-regulating and constrained
While federation execution still in progress:
      Tslocal = the time stamp of next local event
            Invoke RTI's Next Event Request with the supplied
            argument    Tslocal
            Handle possible RTI requests for Reflect Attribute Values
                  and
                  Receive Interaction by using RTI's Update Attribute
                  Values
                  and/or Send Interaction services.
            Receive RTI's Time Advance Grant
            If (no TSO messages were received since the Next Event
                  Request call)
                  Now = Tslocal
                  Process the next local event notice identified
                  above
            Else
                  Now = time stamp of the received TSO message
```

The time-warp synchronization protocol, proposed by [Jefferson 1985], is the most well known optimistic synchronization protocol. In the time-warp protocol, logical

processes (LPs), or federates, are allowed to process events optimistically. This means that a situation can occur where the time-stamp of a received message is smaller than the time-stamp of a previously processed event; this is called a straggler message. This implies that LPs are also allowed to send messages optimistically. When a straggler message is received, the receiving LP needs to correct its logical time to less or equal the time-stamp of the straggler message, this is called rollback. In the rollback process, events that have been processed, having a greater time-stamp than the straggler message, needs to be unprocessed. Further, additional events, sent to other LPs, generated from processing these events needs to be annihilated. The annihilation of events is accomplished by sending anti-messages to concerned receivers of the original events. Anti-messages will also induce rollback if the time-stamp of the anti-message is smaller than the time-stamp of the latest processed event.

Below, an example of how the time-warp algorithm manages rollback is described. In figure 4, the event queue of an LP is illustrated. Black boxes represent processed events, white boxes unprocessed events, grey boxes output messages, whereas grey circles represent snapshots of the LP state. At the stage illustrated in figure 4, the LP has processed an event for time 15 and saved a snapshot of its state.

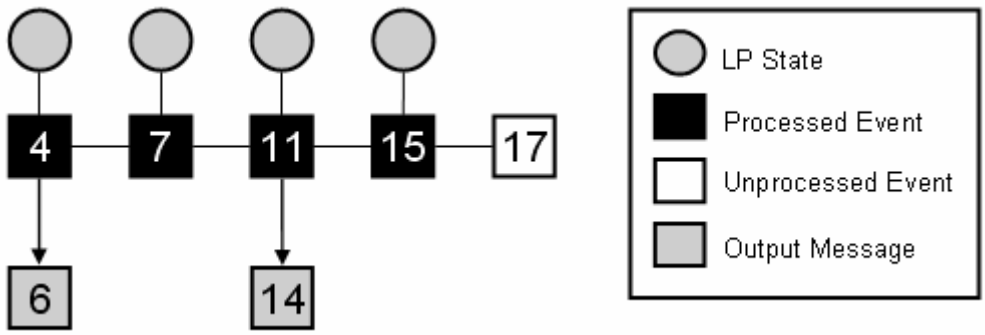

Figure 4. Event queue of a logical process (LP).

Next, a straggler message with time stamp 5 reaches the LP, as illustrated in figure 5. At this stage, the LP must rollback event 7, 11 and 15 since these must be processed after the current straggler message. Thus, the LP restores using the snapshot taken after processing event 4. The states for event 7, 11 and 15 are deleted. Further, the LP must annihilate output message 14 since this was caused by processing event 11, which now has been rolled back (unprocessed). Therefore, the LP sends an anti-message for output message 14.

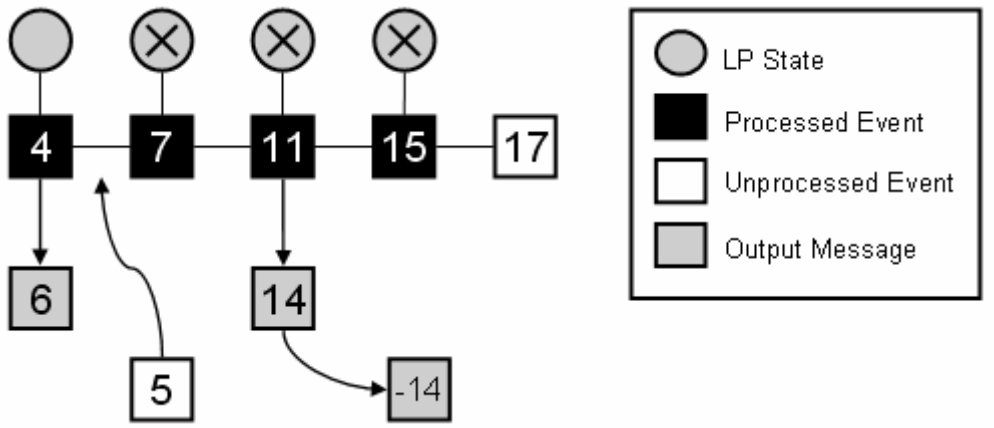

Figure 5. A straggler message reaches the logical process causing rollback.

The event queue of the LP after completion of the rollback is illustrated in figure 6. At this stage the LP resumes execution by processing event 5 (the straggler message). The LP must also manage the case when it is reached by an anti-message from another LP. If an LP receives an anti-message for an event that has not been processed yet it is simply deleted. This occurs if for example the LP in figure 5 receives an anti-message for event 17. However, if the current event already has been processed a rollback is generated. For example, if the LP in figure 5 receives an anti message for event 15, it must un-process and then delete event 15, as well as restore from the snapshot produced after processing event 11.

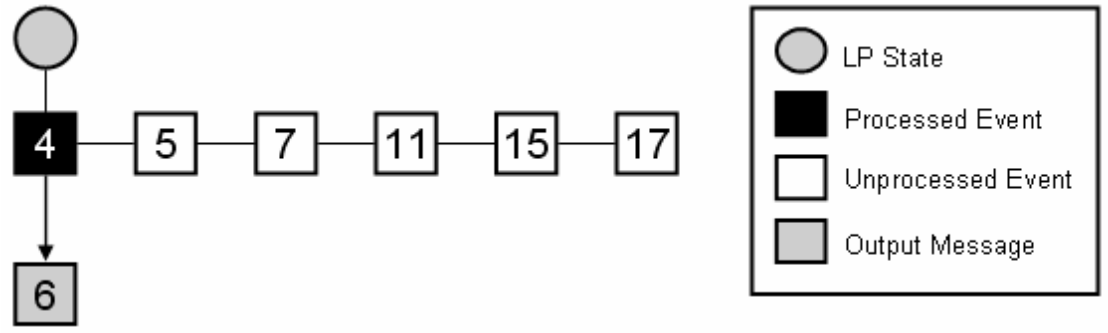

Figure 6. Event queue of logical process following rollback.

The HLA Time Management Design Document [HLA TM 1996] describes the life of a time-warp federate in the following way:

```
Become time-regulating and constrained
GVT = 0.0
flushQueueRequest (min time stamp among local events)
while (GVT < FederateEndTime)
      nextEventTS = min time stamp among local events
      if timeAdvanceGrant (t) has not been invoked
            Allow RTI to deliver events
            Add non-retraction events to message list
            Add retraction events to retraction list
      else
            GVT = t /* Note: this is the federation time */
            fossilCollect (GVT)
            flushQueueRequest (nextEventTS)
      while (message list is not empty)
            if (TS of the head of message list < TS of last processed
```

```
                  event)
                  rollback to TS of the head of message list
            enqueue head of message list into federate's local event
            queue
            remove head of message list
      while (retraction list is not empty)
            find retractionlist head in unprocessed or processed
            message queue of federate
            if the head of retraction list has been processed
                  rollback (head of retraction list)
            delete head of retraction list from federate
      dequeue next event to be processed
      save state
Process event
```

3.2 Supporting fault-tolerance in HLA

The issue of fault-tolerance in distributed systems has been researched extensively and a range of solutions exist today. However, techniques for fault-tolerance in distributed simulations have not been developed at the same pace. Research in this area has been quite sparse to date [Kiesling 2003]. Considering the broadened application of M&S in various domains, this aspect of fault-tolerance certainly needs more coverage. If distributed simulations are incorporated in future military command and control systems, these simulations must be reliable. In a mission critical system, supporting a decision maker in short decision cycles, it is not acceptable to have unreliable simulation components, i.e. components that upon failure do not support recovery and need to be rerun to ensure a consistent execution. Apart from influencing the effectiveness of a simulation execution, failures that are not recovered and managed timely may impact the simulations results negatively.

Today, the support for implementation of fault-tolerant federations, based on the HLA, is weak. If a federate of a federation fails, simply restarting the federate may leave the simulation in an inconsistent state. The only viable option has been to restart the entire application [Damani & Garg 1998], potentially initiating the restarted federation using previously saved states of the individual federates. However, using the save and restore features of the Management Object Model may cause a significant overhead as reported in [Zajac et al. 2003] and [Rycerz et al. 2005]. Moreover, the save facility of the HLA is performed in a local context, meaning that states are not available outside of the node where a federate resides. Also, the HLA provides no means of detecting failures in a federation.

3.2.1 Fault-tolerance in HLA Evolved

Currently, work is carried out to define the next generation of the HLA standard, through the HLA Evolved track. This work is expected to be completed in 2006. An interesting aspect of HLA Evolved is that the issue of fault-tolerance has been covered more extensively, compared to earlier versions of the HLA. In HLA Evolved, a common semantics for failure is defined and mechanisms for fault-detection are provided.

Basically, two additions have been made to the Management Object Model (MOM). These are two interactions named federate lost and disconnected. The purpose of these interactions is to signal the failure of a federate, from the perspective of a federation (federate lost) and from the perspective of a federate (disconnected). Federates

subscribing to the federate lost interaction will be notified by the RTI when a member of the federation is lost (link to the RTI is broken). Subscribing to the disconnected interaction means that when a federate loses its link to the RTI it is notified internally to initiate an attempt to reconnect. When a federate is lost, the RTI has the responsibility to resign on behalf of the failed federate [Möller et al. 2005].

Figure 7 illustrates the life cycle of a federate with respect to faults. In the not connected state in figure 7, the federate will attempt to connect to the RTI using the Connect call. Faults occurring in this state are covered by the fault-tolerance of HLA Evolved. In the next state, the federate is connected. At this stage, faults are managed by HLA Evolved, bringing the federate to the not connected state. Similarly, in the joined state a fault will bring a federate to the not connected state. Figure 7 illustrates the use of the disconnected interaction added to the FOM, which triggers a reattempt to connect and join.

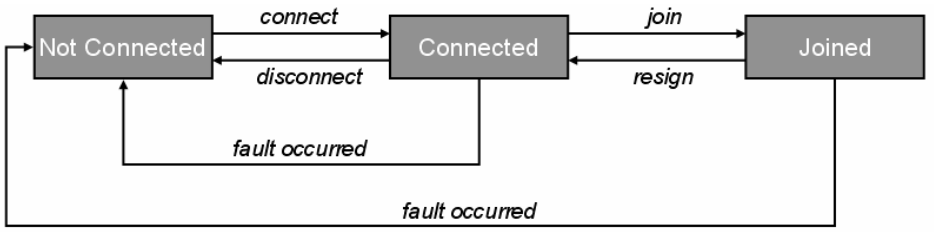

Figure 7. Federate life cycle in the presence of faults [reproduced from Möller at el. 2005].

A number of levels, where faults can occur in a federation, were envisioned when considering fault-tolerance for HLA Evolved. These are [Möller et al. 2005]:

1. **Communications**: Typical faults occur when a cable is disconnected or the link between two remote sites is lost
2. **Computer hardware**: Faults may arise if components of a computer fails, e.g. power supplies malfunctions or hard drives crashes
3. **Operating system**: The host system of a federate freezes or certain components, such as drivers and processes, fail
4. **RTI components**: A failure may result from crashed, or corrupt, RTI components
5. **Federate**: A federate may crash or degrade
6. **Federation**: Faults may occur if federates of a federation do not follow predefined agreements, e.g. interpret data in a unintended manner
7. **Users**: Unintentionally, users may trigger faults on lower levels, or terminate federates at the wrong time.

HLA Evolved will probably ease development of fault-tolerant federations but some crucial aspects are still not covered. In HLA Evolved federates are notified (if desirable) of a failed federate but no mechanism for recovery is provided.

3.2.2 Related work

Even though fault-tolerance in the context of the HLA has not been researched extensively, some researchers address this issue. In [Lüthi and Berchtold 2000] a structured view of fault-tolerance in parallel and distributed simulation is given and

possible solutions are outlined. Several papers address the issue of run-time federate migration, which represent a fundamental function in building an infrastructure for reliable execution of federations. In [Tan et al. 2005] the issue of federate migration is explored. In order to make distributed simulation executions more efficient the workload should be uniformly distributed over available nodes. One way of maintaining the workload distribution over time is to implement run-time federate migration. The paper describes a mechanism that allows migration of federates, executed on nodes with high workload, to nodes having less workload.

In [Bononi, et al. 2003] an adaptive framework, the generic adaptive interaction architecture (GAIA), is outlined that supports the dynamic allocation of model entities to federates in an HLA-based simulation framework. The potential benefit of this framework is the reduction of messages being communicated among separate execution units. This is achieved by a heuristic migration policy that assigns model entities to executing federates as a trade-off between external communication and efficient load balancing. Load balancing is required to avoid the concentration of model entities over a small number of execution units, which would degrade the performance of the simulation. The proposed mechanisms proved beneficial in simulating a prototype mobile wireless system by reducing the percentage of external communication and by enhancing the performance of a worst-case scenario.

In [Cai et al. 2002] an alternative approach to dynamic utilization of resources for the execution of HLA federations is presented. In this case, the framework is based on grid technology, more specifically services of the Globus Grid toolkit. Each federate in the proposed architecture is embedded in a job object that interacts with the RTI and a load management system (LMS). The LMS performs two major tasks through the use of a job management system and a resource management system. These systems carry out load balancing whenever necessary and the discovery of available resources on the grid.

In [Lüthi and Großmann 2001] a resource sharing system (RSS) is presented that uses idle processing capacity in a network of workstations to execute HLA federations. The owners of workstations within a local-area network (LAN) can control the availability of their computers, through a client user interface, for the execution of individual federates of a federation. Computers that are willing to share their resources are registered with the RSS manager that performs elementary load balancing. The RSS is built around a centralized manager that relies on an ftp server for the storage and migration of federates. Currently, there are no extensive fault-tolerance mechanisms included in the RSS implementation, but as this is an important feature of distributed simulations and not well supported in the HLA, the RSS will eventually include functionality for check-pointing and management of replicated federates and fault detection.

In [Berchtold and Hezel 2001] a replication-based concept for fault-tolerant federations is presented, called R-FED. The concept supports both Byzantine and fail-stop failures. In the approach, some FT specific components manages a set of replicas of the federates and detects failures in the federation.

3.3 Summary

For successful integration of simulations within the NBD, and in C2 settings, a number of requirements must be met. Simulations must be reliable and be able to respond in a timely fashion. Otherwise the commander will have no confidence in using simulation as a tool. An important aspect of these requirements is the provision of fault-tolerant simulations in which failures are detected and resolved in a consistent way. Given the distributed nature of many military simulation systems, services for fault-tolerance in distributed simulations are sought. The main architecture for distributed simulations within the military domain, the HLA, does not provide support for development and execution of fault-tolerant simulations. First, mechanisms for detection and signaling of failures within a simulation are required. These features will most likely be part of the next generation HLA, developed within the HLA Evolved track. Second measures for recovery of failed federates, to ensure consistent federation executions, are needed. This aspect is not part of the blueprints for the next generation HLA.

Given the abovementioned shortcomings of the HLA standard, this thesis explores development of fault-tolerant mechanisms for the HLA. Specifically, the thesis addresses recovery in federations synchronized according to the time-warp protocol, which is accomplished through the use of a rollback-recovery scheme.

4. Contribution of the thesis

This chapter contains the main contribution of the thesis. First a common environment for M&S, the NetSim environment, is described briefly as a context and motivation for the development of fault-tolerant distributed simulations. Second, the architecture and implementation of the Distributed Resource Management System (DRMS), enabling fault-tolerant distributed simulations are described. This is followed by a description of the fault-tolerance mechanism implemented within the framework of the DRMS. Finally, results from an evaluation of the proposed fault-tolerance mechanism are outlined and discussed.

4.1 The NetSim environment explained

This section outlines the concept of network-based M&S in general and its role in modern C2 systems. Further, an overview of a network-based M&S environment, called NetSim, supporting computer-based collaborative work, distributed resource sharing and fault-tolerant distributed simulations is given.

4.1.1 M&S in modern C2 systems

Use of M&S in a C2 context provides efficient means for decision support, simulation-based acquisition (SBA), training and planning of operations. The potential gain from coupling C2 and M&S systems have been discussed and studied during recent years, see for example [Tolk 2003] or [Carr 2003], and the HLA has been proposed to bridge the gap that currently exists between the two domains.

Use of M&S in the C2 domain will provide the decision maker with tools that enable fast decisions in short decision cycles, but also that improves the quality of decisions that have been made. However, the application of simulations in these environments will require a high level of interoperability and collaboration between various actors, i.e. systems interoperability and means of collaboration between decision makers, technical staff etc. Given this, it is crucial to provide support for computer-based collaborative work, efficient sharing and use of M&S-related resources through a common framework, built on standards, e.g. standards for distributed simulations (HLA) and common information exchange data models. Also, simulations provided through C2 systems must respond in a timely fashion and ensure high quality output, thus fault-tolerance is crucial in these settings.

When coupling the C2 and M&S domains it is important to consider the last decade's rapid development in web and network technologies. It is of interest to see how web and Internet technologies can facilitate integration and also change the way we model and execute simulations. To explore these aspects the NetSim project was initiated at the Swedish Defence Research Agency (FOI).

4.1.2 Vision of NetSim

The purpose of the NetSim environment is to provide a common M&S environment managing issues of interoperability, availability and reusability of simulation models. The intended environment will cover the complete M&S cycle, from conceptual design to execution of simulations. In the environment, Subject Matter Experts (SMEs), software developers, VV&A agents etc., can meet to share each others resources and expertise, but also to collaborate in real-time. The main areas of consideration for the NetSim environment are computer supported collaborative M&S and distributed resource sharing and use.

The NetSim environment is based on a Service-Oriented Architecture (SOA). Figure 8 illustrates an overview of this architecture. The top layer comprises various M&S-related tools, e.g. tools for composition of simulations by a single user or collaboratively by a group of users. The M&S-tools derive their functionality from various NetSim specific services. These are denoted DRMS, CC and Repository in figure 8. As mentioned before, the DRMS provides services for execution of simulations. The CC (Collaborative Core) provides services to support collaborative work, whereas the Repository provides lookup of available resources within the environment, e.g. simulation models, simulations and computing capacity. The NetSim services are based on various overlay network technologies such as Web Services, Grid Services, peer-to-peer or the HLA RTI. Throughout all layers, a common syntax and semantics is used to ensure interoperability. Further, security is considered an integral part of all layers. The purpose and scope of the NetSim environment is described in greater detail in part II, paper I. Also, [Eklöf et al. 2004] and [Eklöf et al 2005] provide descriptions of the architecture and prototype implementation of the NetSim environment.

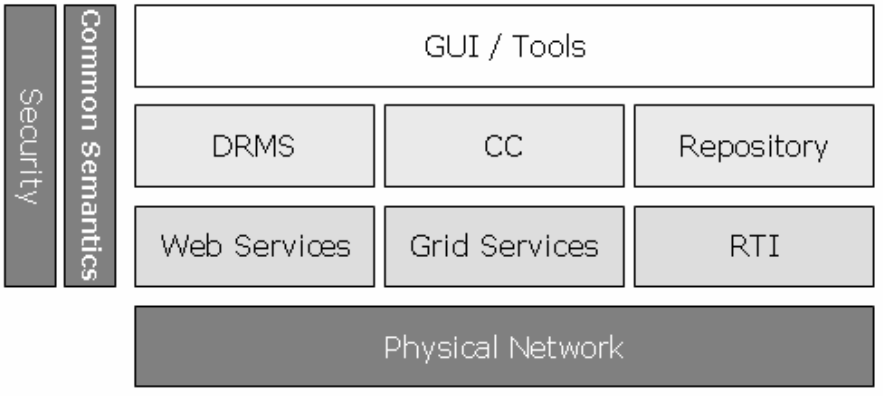

Figure 8. Architecture of the NetSim environment.

4.1.3 Summary

This section addresses several important aspects of enabling the use of M&S in future C2 systems. Among these aspects the provision of fault-tolerant distributed simulations is a key challenge. In a C2 context, simulations must be reliable, i.e. simulations must respond in a timely fashion and must provide reliable outputs, otherwise the decision makers will have no confidence in using them. Today, HLA is the key technology for distributed simulations within the military community and

HLA does not provide fault-tolerant services in its present form. Thus, fault-tolerance mechanisms for HLA must be developed before the full potential of M&S will be seen in the C2 domain.

4.2 The Distributed Resource Management System

In this section the issue of resource management in NetSim is described in greater detail. More specifically the Distributed Resource Management System (DRMS) is described at a conceptual level. A requirement on DRMS is run-time migration of simulation models between host-environments in a network. This function is fundamental for development of fault-tolerance mechanisms in the context of DRMS.

4.2.1 DRMS - An overview

The basic idea behind the DRMS is that users within an M&S community should be able to execute processing intensive simulations regardless of what kind of hardware they possess. To do this, idle processing capacity on a network is utilized to perform the execution on behalf of the users. This feature is transparent to the individual users, which are entirely detached (if desirable) from the process of managing the execution process, i.e. the process of locating, allocating and monitoring computers used for the execution. Owners of desktop computers, willing to contribute their idle processing capacity, download and install an application that under certain circumstances (decided by the desktop owner) shares the processing capacity of its host machine. The desktop owner always has the ability to withdraw shared hardware resources at any time. The DRMS simplifies the process of deploying, configuring and executing federations by enabling the user to manage these tasks from a central location, i.e. the users own desktop computer.

Further, the DRMS supports run-time migration of federates, i.e. transfer of federates between host-environments during the federation execution. This function is used for several purposes:

1. **User-triggered migration**: the owner of a computer, used in a federation execution, chooses to withdraw shared processing capacity, which triggers migration of a federate to a new host environment.
2. **Load-balancing migration**: to gain better performance in the federation execution the most suitable distribution of federates among available computers can be sought. Since conditions may change during a federation execution, migration may be used as basis for a run-time optimization scheme.
3. **Failure migration**: in case a federate, its host environment, or a network connection to a remote site fail, migration is used to restore a federation execution

In the context of the DRMS, user-triggered and failure migrations have been explored. Migration for the purpose of load-balancing has not been covered.

4.2.2 DRMS implementation

Two implementations of the DRMS concept described above have been made. The first implementation is based on peer-to-peer technology and is described in greater detail in part II, paper II. The second implementation is based on web services and is

presented paper III. The following section provides a brief description of the implementations.

In the first version of the DRMS, peer-to-peer technology was used for the implementation, more specifically the JXTA technology. JXTA is an open-source project initiated by Sun Microsystems aiming at providing system and community interoperability, platform independence and technology ubiquity for peer-to-peer solutions. A basic entity in a JXTA-based system is a peer, which can be seen as a virtual communication point. In any peer-to-peer system, a peer constitutes the fundamental processing unit. In JXTA, the notion of a peer group is also important. A peer group assembles multiple peers into a group within which services and resources are shared. In the JXTA implementation of the DRMS two different types of peers are present; one called Manager Resource (MR) and the other Computing Resource (CR). As the name implies, the MR is responsible for managing the execution of a simulation on behalf of a user, or a group of users, whereas the CR is responsible for executing a simulation component, in this particular case an HLA federate.

Given the concept of a Network-Based Defence, where capabilities are exposed as services, the NetSim project adopted a Service Oriented Architecture (SOA). Thus, the architecture and implementation of the DRMS was revised and a web services-based solution was developed. Three main service types were defined, namely Execution, RemoteJob and Repository. The Execution service is equivalent to an MR in the JXTA-based implementation, whereas the RemoteJob service can be seen as a CR. Figure 9 illustrates the interrelations among the services and the service interfaces employed.

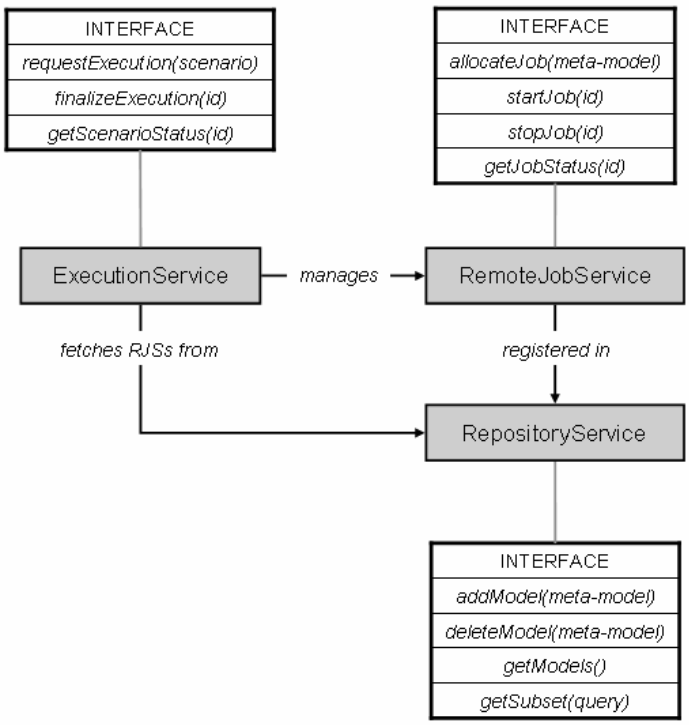

Figure 9. Services and service interfaces of the web services-based implementation.

The Repository service manages meta-data that describes various resources, e.g. simulation models, data and RemoteJob services. When deployed in a host environment, the RemoteJob service registers meta-data in the Repository service. This meta-data describes features such as the hardware and software configuration of the host machine, e.g. the CPU speed and installed Java version. If the RemoteJob service accepts a request from the Execution service to execute a federate, it downloads the required code from an FTP server. The Execution service is used from the NetSim environment to manage the execution of an assembled simulation. First, the Execution service fetches available RemoteJob services from the Repository service. Then the Execution service finds a suitable distribution of the simulation components (federates) given the available RemoteJob services. The distribution is resolved by matching hardware and software requirements of the federates with the features of the RemoteJob services. During a federation execution the Execution service monitors the federation to detect failures and to initiate a recovery mechanism in case a failure occurs.

4.2.3 Summary

The peer-to-peer-based version of the DRMS showed the feasibility of utilizing idle storage and processing capacity in a network of workstations, to execute HLA-based distributed simulations. This work yielded a solution for migration of federates in a federation utilizing a time-stepped synchronization scheme. However, the implementation suffered from two major drawbacks. First, the migration of federates was a coordinated activity, which could not be used for migration in case of failure, i.e. it only supported user-triggered migrations. Second, the implementation only supported time-stepped federations. To overcome these drawbacks and in order to adapt to the service-oriented architecture of NetSim, a web services-based solution was developed. This work showed the feasibility of implementing fault-tolerance in time-warp federations using web services and the RTI communication infrastructure.

4.3 Fault-tolerance enabled DRMS

In this section the fault-tolerance mechanism implemented within the framework of the refined DRMS is described briefly. In part II, paper III and IV, a more elaborate description of this is given.

4.3.1 Fault-tolerance mechanism

To realize reliable execution of time-warp federations, a checkpoint-based rollback recovery protocol was developed. In check-pointing-based approaches, states are reported from individual LPs to what is commonly known as stable storage. When a failure occurs, an LP is restored using the latest saved state on stable storage.

The fault-tolerance mechanism implemented in the DRMS utilizes the RTI communication infrastructure. This means that an extension to the Federation Object Model (FOM) is introduced, which enables check-pointing and federate recovery. Checkpoints are reported to a stable storage component, which is a member of the federation execution. The stable storage component is deployed by the Execution service (described in the previous section), which also comprises a simple component for detection of failures in the federation. However, the failure-detection phase has been treated superficially in this work.

4.3.2 An example of federate recovery

Consider a case where four federates form a simple airport simulation federation. In this simple example only one type of event exists, the time of arrival of an airplane to an airport. The federates represent airports managing arrivals and departures of airplanes. The processing of an event in this case simply means scheduling the arrival of the airplane at a new destination, i.e. in one of the other federates. Thus, in each federate we have an event queue, comprising arriving airplanes. When an event is processed, it will generate a new event (arrival of an airplane) in another federate (airport). The federation is synchronized according to the time-warp protocol (described in section 3.1.2), thus, the federates process events optimistically, and similarly, produce events for other federates optimistically. Given the topology of the federates (airports), which is a fully connected network, the message exchange is fairly homogeneous.

Consider the event queue configuration of four federates of this federation as Global Virtual Time (GVT) equals 10, see table 2. The letters in the event queue column indicate the origin (airport) of events (airplanes), whereas the following numbers indicate the arrival time. In the output column, the letters indicate the destination of the airplanes and the following numbers indicate the arrival time at destinations.

Table 2. Event queues of federates preceding migration.

Airport	Event Queue	Output
A	B12	C15
	B14	D17
B	D9	A12
	C12	A14
	C21	-
C	D11	B12
	A15	B21
D	A8	C11
	A17	-

The process of federate recovery, in case of failure, resembles a rollback to GVT. However, special care is needed since the failed federate is not able to produce anti-messages. In this case, message annihilations must be performed separately by each federate. For instance, consider the case where federate A in table 2 fails. During the absence of federate A (due to migration), federate B, C and D will continue the federation execution. However, as federate A does not report any checkpoints to stable storage during that time, GVT will not be increased during the migration. When federate A has been redeployed in a new host environment and initiated using the latest saved state from stable storage, it requests resend of all messages. This request first triggers annihilation of messages in federate B, C, and D, after which the resend takes place. Given the event queue configuration in table 2, the following actions will be taken:

- *Federate* **C** *annihilates event* **A15**
- *Federate* **C** *sends anti-message for event* **B21**
- *Federate* **B** *annihilates event* **C21**
- *Federate* **D** *annihilates event* **A17**
- *Federate* **B** *resends event* **A12**
- *Federate* **B** *resends event* **A14**

The configuration of the event queues of the federates, post migration, is presented in table 3. At this point the federation is synchronized and can resume normal execution.

Table 3. Event queues of federates following migration.

Airport	Event Queue	Output
A	B12	-
	B14	-
B	D9	A12
	C12	A14
C	D11	B12
D	A8	C11

4.3.2 Summary

The fault-tolerance mechanism of the DRMS uses the RTI as means of communication to enable check pointing and federate recovery. This implies that federates conform to certain requirements in order to be executed within the framework of DRMS. Also, the introduction of fault-tolerance in any kind of system will impose a cost. Regardless of the approach chosen for implementing fault-tolerance, replication-based or check-pointing-based fault-tolerance, the system must cope with increased network traffic and consumption of more hardware resources. The implemented fault-tolerance mechanism is capable of managing non-transient failures, however management of multiple concurrent failures are currently not supported.

4.4 Evaluation of the fault-tolerance approach

In this section, an evaluation of the proposed fault-tolerance mechanism for time-warp federations is briefly presented. The evaluation considers the cost of using fault-tolerant federations, in terms of message overhead caused by the fault-tolerant mechanism. A comparison is made between federation executions, with and without fault-tolerance, when faults are triggered at different stages of the simulation. If the fault-tolerance mechanism is utilized when faults occur, failed federates are recovered and the federation can resume normal execution. In case the fault-tolerance mechanism is not used, a fault causes rerun of the entire federation. Note that the evaluation does not consider required time for a consistent federation execution, but focuses on number of generated messages. For the purpose of evaluating the fault-tolerance mechanism a simple test federation was developed. This test federation consists of four federates, forming a fully connected network. Time synchronization is

carried out by means of the time-warp protocol, and in case of failure, a rollback recovery scheme is employed to restore the federation. In part II, paper IV, a more comprehensive description of the test federation is given along with an in-depth presentation of the evaluation.

4.4.1 Results

Figure 10 shows the difference in total number of generated messages for six different simulation executions, namely:

1. 0 faults – no FT
2. 0 faults – with FT
3. 1 fault – no FT
4. 1 fault – with FT
5. 3 faults – no FT
6. 3 faults – with FT

For the zero faults case, the total number of generated messages is greater if the FT mechanism is implemented. This is due to the extra communication induced by communicating federate states to stable storage. If one fault is introduced during the simulation execution the total number of messages is almost equal, regardless if the FT mechanism is used or not, as illustrated in figure 10. When three faults are triggered during the simulation execution the total number of messages is greater for simulations that do not utilize the FT mechanism, see figure 10.

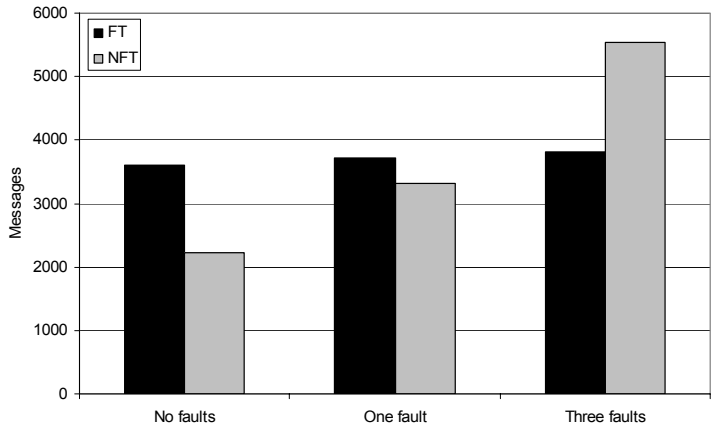

Figure 10. Total number of messages generated for the zero, one and three faults cases, with and without use of the FT mechanism.

Figure 11 provides an alternate view of the experiments. In this case the difference in total number of messages, given a mean failure time of five consecutive intervals of the simulation, between the non FT and FT cases is depicted. The y-axis in figure 11 represents the difference in number of generated messages between the non FT and FT cases, whereas the x-axis represents the logical time interval of the federation executions. As indicated in the figure, the overhead for the FT cases is greater than the extra communication caused by the faults in the non FT cases for the first three intervals. In the fourth interval the total communication cost is almost equal, whereas in the fifth interval the FT case shows better performance over the non FT case.

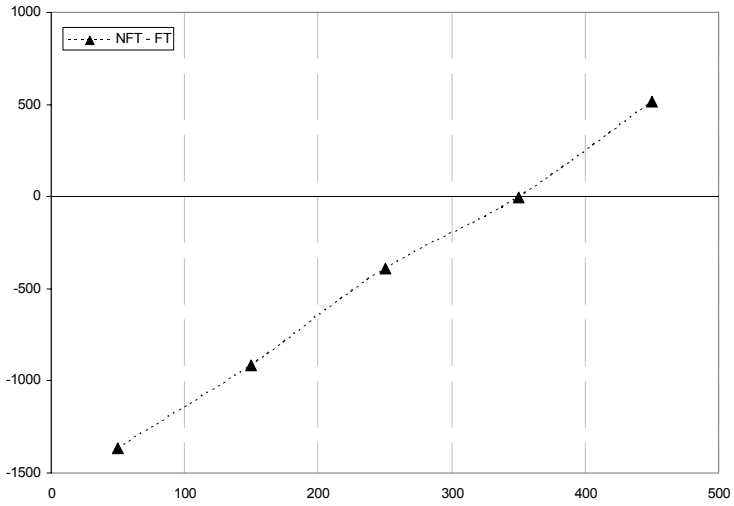

Figure 11. Difference in total number of messages, assuming one fault, between the non FT and FT cases for mean failure times of five intervals.

4.4.2 Analysis of efficiency

In order to analyze the cost, in terms of time required for federation execution, of using the fault-tolerance mechanism, a simple cost model is defined. Consider the worst-case scenario in the one fault case where a federate fails at the end of the federation execution. We define $C(NFT)$ as the cost (required time) for executing the federation when the FT mechanism is not implemented. Let $C(FT)$ be the cost (required time) for executing the federation using the FT mechanism. Further assume that one of the federates of the federation fails during the execution. Then we can define the cost, $C(NFT)$ and $C(FT)$, as:

$$C(NFT) = Mk + R + Nk$$

and

$$C(FT) = Mk + r + Ms + (N - M)(k + s) = Nk + Ns + r$$

where

- N: number of event-messages executed until the end of a simulation
- M: number of event messages executed from the beginning of a simulation until a federate fails
- k: average cost of processing one event-message
- s: amortized cost of saving one message on the stable storage
- R: cost of reinitiating the simulation in NFT case
- r: cost of resuming (restoring) a failed federate in the FT case

As mentioned above, the worst case performance, assuming one fault, equals the case when a federate fails at the end of the federation execution. Thus, in this case M will

approach N ($M = N$). Under this assumption we can compare $C(NFT)$ and $C(FT)$ in the following way:

$$\frac{C(NFT)}{C(FT)} = \frac{Nk + R + Nk}{Nk + Ns + r} = \frac{Nk\left(2 + \frac{R}{Nk}\right)}{Nk\left(1 + \frac{s}{k} + \frac{r}{Nk}\right)} = \frac{2 + \frac{R}{Nk}}{1 + \frac{s}{k} + \frac{r}{Nk}}$$

If we assume that the cost of restarting the federation in the NFT case and the cost of resuming a federate in the FT case is negligible compared to the cost of processing event-messages or saving states on stable storage then:

$$\frac{R}{Nk} = \frac{r}{Nk} = 0$$

Under such an assumption:

$$\frac{C(NFT)}{C(FT)} = 2 \Big/ \left(1 + \frac{s}{k}\right)$$

Given the above formula we can describe the efficiency of an FT federation. The formula indicates that the efficiency depends on how expensive the amortized cost of saving messages on the stable storage is in relation to the cost of processing event-messages. Figure 12 illustrates the efficiency, $C(NFT)/C(FT)$, as a function of *s/k* provided that *k* is constant.

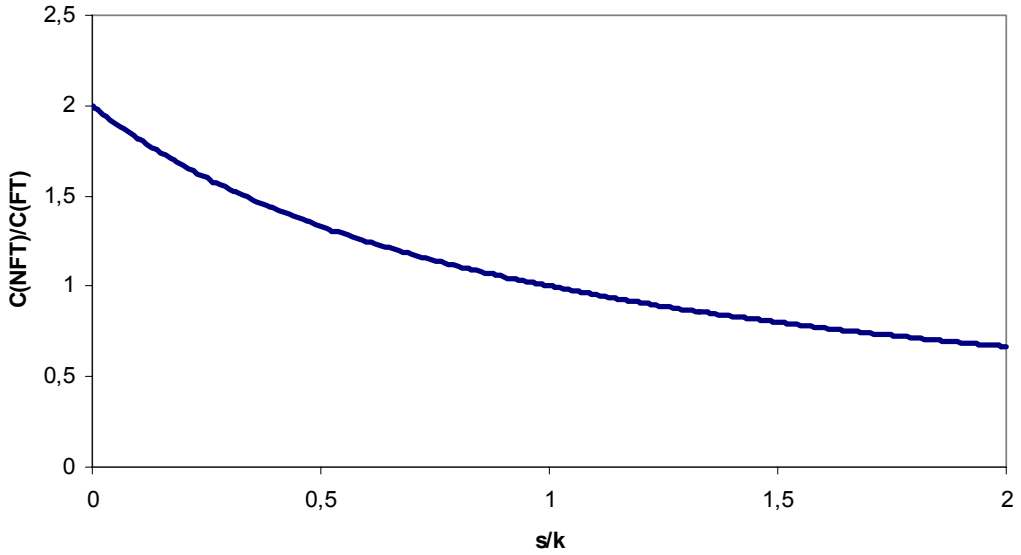

Figure 12. Efficiency of simulation executions, *C(NFT)/C(FT)*, as a function of *s/k*.

The amortized cost of saving messages on the stable storage depends on how frequently states are saved, i.e. the state-saving interval employed. However, the state-

saving interval is coupled to the cost of restoring a failed federate (*r*). In case states are saved infrequently, a failed federate may be forced to rollback to a point in time far back in its past. This in turn may cause rollback of other federates, which all in all increases the cost of restoring the failed federate. Thus, it is of interest to find a suitable state-saving interval that take into account the communication overhead, caused by the FT mechanism, and the cost of restoring a failed federate.

Furthermore, the efficiency, or *s*, is coupled to the hardware configuration of the stable storage device. A slow stable storage device will of course degrade the efficiency of an FT simulation. The *s* parameter also embeds the impact of sizes of states. Federates that require large allocations of memory for representation of their state will have a large impact on the efficiency. Large states will require more bandwidth and more time to save and extract. In this context it would be valuable to investigate approaches that take into account when during the life-time of a federate states are small. By avoiding saving of states that are large, the efficacy of an FT simulation can be improved.

4.4.3 Summary

The overhead caused by the FT mechanism is justifiable, especially when considering the worst-case scenario for federate failure times. In the worst-case a federate fails near the end of the federation execution, which in the non FT case will double the number of messages required for a fault-free execution. The performance of the FT mechanism (as indicated in figure 11), is coupled to the total number of event-messages of the federation. An increase of the amount of event-messages will yield better performance for the FT-mechanism since the number of checkpoint messages will remain the same. Moreover, if additional faults are introduced during the federation execution the benefit of the FT-mechanism will become even more apparent.

As the cost-model and figure 12 indicate, the cost of using the stable storage device has essential impact on the efficiency of FT federations. In this context, it is of importance to determine which state-saving interval to use and to provide a fast and reliable implementation of the stable storage component. The inclusion of FT mechanisms in distributed simulations will always impose a cost. However, this cost can be reduced by careful design of algorithms and use of high-performance hardware.

5. Future work

In order to verify the proposed fault-tolerance mechanism, it should be evaluated in the context of a real simulation system. The federation used in the tests is rather simple. Above all, the exchange of messages in this federation is far more homogeneous compared with most federations. Thus, the fault-tolerance mechanism should be tested in various types of federations, considering the pattern of the message exchange, to get an estimate of its suitability under different conditions. Also, different types of federates (federations) will require different amounts of payload in their checkpoints, i.e. federates require different amounts of data for representation of a state. Given this, it would be of interest to investigate the impact of sizes of checkpoints on the performance of the fault-tolerance mechanism.

Furthermore, the design of the current fault-tolerance mechanism does not consider multiple concurrent failures. Given that more than one federate fails at the same time, the mechanism in its present form does not guarantee that the restoration will bring back the federation to a consistent state. Thus, it would be of interest to extend the current mechanism to handle these situations as well.

Currently the fault-tolerance mechanism is integrated in the code of the federates that will form a robust federation. In order to simplify development of fault-tolerant federations some kind of middle-ware system should be developed. This middle-ware should abstract fault-tolerance aspects away from the developer. However, the design of such middle-ware means that the fault-tolerance mechanism needs to be generalized. Currently, the mechanism is targeting federations synchronized according to the time-warp protocol. A middle-ware for fault-tolerance should provide services for other types of time-synchronization schemes, as well as federations comprising a mixture of federate types.

6. References

Agarwal, V. 2004. *Fault-tolerance in distributed systems*, http://www.cse.iitk.ac.in/report-repository/2004/Y4111065-CS697.pdf, accessed 2006-02-14.

Berchtold, C., and M. Hezel. 2001. *An architecture for fault-tolerant HLA-based simulation*, Proceedings of the 15th European Simulation Multiconference, Prague, Czech Republic.

Bononi, L., G. D'Angelo, and L. Donatiello. 2003. *HLA-based adaptive distributed simulation of wireless mobile systems*. Proceedings of the 17thWorkshop on Parallel and Distributed Simulation, San Diego, California.

Cai, W., S. Turner, and H. Zhao. 2002. *A load management system for running HLA-based distributed simulations over the grid*, Proceedings of the 6th IEEE International Workshop on Distributed Simulation and Real-Time Applications, Fort Worth, Texas.

Carr, F. 2003. *C4ISR/Sim technical reference model applicability to NATO interoperability*, NATO Modeling and Simulation Group, Symposium on C3I and M&S Interoperability, Antalya, Turkey.

Chandy, M., and L. Lamport. 1985. *Distributed snapshots: Determining global states of distributed systems*, ACM Transactions on Computing Systems 31, 1, 63-75.

Cristian, F. 1991. *Understanding fault-tolerant distributed systems*, Commun. ACM 34(2), 56-78.

Damani, P., and K. Garg. 1998. *Fault-tolerant distributed simulation*, Proceedings of the 12th Workshop on Parallel and Distributed Simulation.

Eklöf, M., M. Garcia Lozano, F. Moradi, D. Nordqvist, M. Sparf, and J. Ulriksson. 2004. *Network-based modeling and simulation – Architecture*, Methodology report FOI-R—1439—SE.

Eklöf, M., M. Garcia Lozano, F. Moradi, D. Nordqvist, and J. Ulriksson. 2006. *Network-based modeling and simulation – Prototype*, User report, FOI-R—1767—SE.

Elnozahy, E., L. Alvisi, Y-M. Wang, and D. Johnson. 2002. *A Survey of Rollback-Recovery Protocols in Message-Passing Systems*, ACM Computing Surveys. Volume 34, Number 3.

Hadzilacos, V., and S. Toueg. 1993. *Fault-tolerant broadcast and related problems*, In: Distributed Systems, Ed. S. Mullender, Addison-Wesley, Wokingham, UK.

HLA Framework and Rules. 2001. *IEEE standard for modelling and simulation (M&S) high level architecture (HLA): framework and rules*. New York.

HLA Interface Specification. 2001. *IEEE standard for modelling and simulation (M&S) high level architecture (HLA): federate interface specification*. New York.

HLA Object Model Teamplate. 2001. *IEEE standard for modelling and simulation (M&S) high level architecture (HLA): Object Model Template (OMT) specification*. New York.

HLA TM. 1996. *HLA Time Management: Design Document Version 1.0*, available online http://www.static.cc.gatech.edu/computing/pads/PAPERS/HLA-TM-1.0.pdf, accessed 2006-02-15.

Jefferson, D. 1985. *Virtual Time*, ACM Transactions on Programming Languages and Systems 7(3), 404-425.

Kiesling, T. 2003. *Fault-tolerant distributed simulation: A position paper*, available online http://fakinf.informatik.unibw-muenchen.de/~tkiesling/documents/ftdsposition-paper.pdf, accessed 2005-03-21.

Kuhl, F., R.Weatherly, and J. Dahman. 1999. *Creating computer simulation systems: An introduction to the high level architecture*. Prentice Hall, Upper Saddle River, NJ.

Lüthi, J., and C. Berchtold. 2000. *Concepts for dependable distributed discrete event simulation*. Proceedings of the 14th International European Simulation Multi-Conference, Ghent, Belgium.

Lüthi, J., and S. Großmann. 2001. *The resource sharing system: Dynamic federate mapping for HLA-based distributed simulation*. Proceedings of the 15th Workshop on Parallel and Distributed Simulation, Lake Arrowhead, California.

Moradi, F., and R. Ayani. 2003. *Parallel and distributed simulation*, In: Applied system simulation, Eds. M.S. Obaidat, and G.I. Papadimitriou, Kluwer Academic Publishers, Dordrecht, Netherlands.

Möller, B., M. Karlsson, and B. Löfstrand. 2005. *Developing fault tolerant federations using HLA evolved*, Proceedings of the 2005 Spring Simulation Interoperability Workshop. San Diego, California.

Rycerz, K., M. Bubak, M. Malawski, and P. Sloot. 2005. *A framework for HLA-based interactive simulations on the grid*, SIMULATION, 81(1), 67-76.

Seiger, T. 2000. *Time management in High Level Architecture based distributed simulation*, FOA-R—00-01434-202—SE.

Tolk, A. 2003. *Overview of recent findings of the study groups of the simulation interoperability workshop delaing with C3I and M&S interoperability*, NATO Modeling and Simulation Group, Symposium on C3I and M&S Interoperability, Antalya, Turkey.

Tan, G., A. Persson, and R. Ayani. 2005. *HLA federate migration*, Proceedings of the 38th Annual Simulation Symposium.

Tanenbaum, A. S., and M. Steen. 2002. *Distributed Systems: Principles and Paradigms*. Prentice-Hall.

Zajac, K., M. Bubak, M. Malawski, and P. Sloot. 2003. *Towards a grid management system for HLA-based interactive simulations*, Proceedings of the 7th International Symposium on Distributed Simulation and Real-Time Applications, Delft, The Netherlands.

Part II – Published papers

I. NetSim – A Network Based Environment for Modeling and Simulation

M. Eklöf, J. Ulriksson & F. Moradi
NATO Modeling and Simulation Group, Symposium on C3I and M&S Interoperability, Antalya, Turkey, 2003.

Abstract

Modelling and Simulation (M&S) is a powerful tool that is used to support training and analysis of military operations, development of military concepts and gradually, it is becoming an integral part of modern C3I systems. As the web has evolved, new ways of carrying out modelling and simulation and realizing C3I systems have emerged. These achievements address some of the research issues considered vital for future development of the M&S/C3I domain. Firstly, web related technologies provide means of overcoming the interoperability barriers, for example through standardized data exchange formats (such as XML), platform independent software (for example Java) and shared knowledge of a domain (semantics). Secondly, networked environments offer ways of setting up virtual organisations, sharing common goals and interests, to efficiently collaborate in problem solving. Finally, computer networks promote efficient sharing of resources, which for example could increase the reuse of existing models or utilize idle processing capacity of computers.

At the Swedish Defence Research Agency (FOI) there is ongoing research, targeting the role of network/web based technologies in M&S, to support defence communities in their work. Our vision comprises an environment supporting the entire M&S-process, including conceptualization, scenario definition, design, development and execution. All these tasks should be maintained by a framework for collaboration, which lets users; developers, analysts, administrators etc, jointly work on a project. During the first phase of this research focus has been on efficient resource sharing and means of collaboration. Through experimental research and implementation of a prototype (NetSim), methods and techniques have been identified to form a framework for collaborative work, resource management and distributed execution.

Following current trends within development of networked applications, decentralized (Peer-to-Peer) solutions were of primary focus when implementing the prototype. Based on the open source Peer-to-Peer platform JXTA, two distinct components of our envisioned system were implemented, namely; a decentralized resource management system deploying a network of workstation for execution of HLA federations and a collaborative environment for joint modelling of federations. Our results show that the utilization of Peer-to-Peer concepts for resource sharing and collaboration are favourable in terms of scalability, robustness and fault tolerance. The technology allows formation of virtual organisations without the need of intermediate resources like centralized and powerful servers. However, some aspects of our implementation temporarily rely on central control, thereby diminishing the benefits of the

Peer-to-Peer paradigm. Future research will therefore address distributed algorithms for synchronisation of collaborative work and a more flexible and extendable approach to resource management. Furthermore, as many studies have pointed out before, one of the great challenges of any type of Peer-to-Peer system is discovery and matching of resources. This is an area that deserves great attention when planning for the next generation C3I/M&S tools.

1. Introduction

C3I systems of the future Network-Centric Defence are dependant of the network and require interoperability between different components of the system. During the past decades the Modelling and Simulation (M&S) community, particularly the area of distributed simulation, has explored the possibility of coupling live and simulated systems in joint exercises and hence addressed the interoperability issues from many perspectives. Therefore, many of the challenges that future C3I systems are facing have already been dealt with by the M&S community.

M&S as an integrated part of C3I systems could provide means for decision support, simulation based acquisition, planning, training, etc. Furthermore, M&S could be employed as a tool for development of C3I systems, e.g. for studies, test and verification. The mutual benefit of a close collaboration between C3I and M&S systems has been identified and discussed during recent years [1] and the High Level Architecture (HLA) has been suggested as a mean to interface and increase interoperability between the two systems.

The High Level Architecture (HLA) is an IEEE[1] standardized architecture (HLA 1516), that provides means of connecting independently developed components (federates) to form simulations (federations). A simulation is formed by connecting individually developed components to a Run-Time Infrastructure (RTI), which implements the HLA standard. The RTI resembles a distributed operating system for simulations by providing services that enable interaction between participating components [2].

Integrated computer based decision support tools have also been identified as an important part of future C3I systems [3]. The fundamental idea is to make decisions faster and at the same time improve the quality of the decisions made. Tools that are accomplishing this are generally based on simulation systems, which often require interoperability and collaboration between different actors, such as decision-makers, field commanders, and technical staff etc. To realize these ideas efforts have been made within the area of computer based collaboration, enabling sharing of various resources, work areas, tools and environments. These techniques will not act as a substitute for real human-human work, but can be used for bridging distances and increasing and facilitating cooperation.

An evolving technology that could provide a fundament for modern C3I systems is Peer-to-Peer (P2P) [4, 5]. The technology offers advantages such as live peer interaction and collaboration, ad-hoc networking and robust and fault-tolerant systems through redundant application and communication paths. P2P-technologies aim at

[1] The Institute of Electrical and Electronics Engineers

utilization of resources at the edges of Internet as opposed to the traditional client-server model. P2P can be seen as an alternative network architecture that does not exclude, but does not naturally build upon centralized solutions [6]. Essentially, P2P is about community and mutual sharing of resources, by organizing nodes into groups sharing common interests and goals.

The aim of this paper is to give an overview of work related to web/network based modelling and simulation carried out at the Department of Systems Modelling, Swedish Defence Research Agency. Moreover it provides an insight in ongoing research in this area and its relation with integrated M&S/C3I systems.

2. Background

2.1 Interoperability

Connecting systems of various types developed for different purposes, during different technological eras and for different platforms, inflicts major difficulties, especially in terms of interoperability. It is required that systems are capable of communicating between them, but also that the communication is semantically and syntactically agreed upon. If these basic requirements are not met, systems may interoperate for the wrong reasons. The problem of interoperability is important to address in the management of highly distributed systems like distributed simulations and C3I systems. The issue of interoperability has been of major concern within the modelling and simulation community for some time now. Already during the 80's, efforts were made to standardize distributed simulations to facilitate interoperability among simulators, simulation models etc. The efforts made and experience gained in this forum, are definitely worth considering when planning for and developing integrated future C3I – M&S systems. Moreover, the rapid development of web/network related technologies brings new possibilities for overcoming the interoperability barriers and problems related to availability and management of resources. For example, through new ways of exchanging data (XML), distributing resources (P2P and Grid computing), and assuring semantic and syntactic correctness (Semantic Web initiative[2]).

2.2 The NetSim project

In 2001 a project was initiated at the Department of Systems Modelling, FOI, with the goals to manage and facilitate some of the issues concerning simulation interoperability, availability and reusability. The project was called WebSim, and focused on the area of web-based M&S. It produced interesting result regarding adapting legacy simulations to the web, and also concerning implementing HLA federations for web-based composition and execution. Some of the work was reported in [7].

As a follow-up to WebSim the NetSim project was formed. NetSim is a shortening of the full name *A Network Based Environment for Modelling and Simulation*. The new project does not reject the ideas of web-based M&S, but instead extends the concept. The main directions are to investigate decentralized solutions for M&S in general,

[2] The semantic web is a web of machine-readable information [8].

both web-based solutions and other possibilities. For this cause a prototype environment is being developed. It is intended to provide functionality and tools for the complete M&S life cycle, all the way from design and simulation modelling, to execution and documentation. The NetSim environment shall also provide access to distributed resources such as simulation models, various data, or even CPU usage, as aid for M&S activities.

NetSim is intended for use within several different defence systems. It will support computer-based collaborative work, such as shared work areas and means of communication. This means that NetSim will not only work as a common platform for M&S, but also as a place of meeting other (M&S) people. In the computer environment, software developers, Subject Matter Experts (SMEs), soldiers and VV&A people may meet, and use and share each others' resources and expertise. In modern and future defence systems M&S is used as technical aid for decision making, Simulation Based Acquisition (SBA), logistics planning and military training etc. Hence, NetSim could constitute an excellent tool for those systems and activities, and for activities where M&S is not currently used, but would be beneficial if made possible. An example of this is the support for mobile clients such as PocketPCs. This allows people on the move to interact with the environment. Hence a soldier may receive direct access to data and information about supplies and routes, and may collaboratively plan or decide what forthcoming actions to take. This also means that dynamic and actual data can be transmitted back to the base, and more optimized and well-planned decisions can be made easily. The NetSim environment will allow people (nodes) within a network to collaborate through their computers, which makes it easier to create a common picture of the situation/problem to handle, and supplied direct contact between all actors concerned. It hereby allows immediate access to the competence and expertise needed.

2.3 The NetSim environment prototype

At present a NetSim prototype is implemented. The prototype is not yet complete, but it demonstrates useful functionality and what the environment can be used for if employed in defence systems. It is implemented as a lightweight Java application, in which a user retrieves access to M&S tools and distributed resources within a local network. In order to let various kinds of users access NetSim, who may be located in different computer environments, a set of requirements were identified and followed during the design phase. These declare that the implementation should be:

- Flexible – Supporting different users with varying computer capacity and properties to utilize the system
- Scalable – In critical situations the number of users must not affect the system capacity
- Platform independent – As much as possible the implementation should be kept platform independent
- Technology independent – The result of our work is primarily the concepts being designed, not the implementation. Hence the solution is kept as technology independent as possible
- Extensible – The infrastructure must allow for further integration of new systems and functionality

All simulation modelling and execution is today performed according to the HLA for purposes of project directives. The system is based on a distributed infrastructure implemented with P2P technology, see 2.5 – 2.6, which provides means for resource sharing and distributed computing among others. It allows users to search for, locate and access distributed resources and users in the network. Resources may in this case be anything from simulation models to CPU usage. Within NetSim only a few simple tools are currently provided, such as a text chat, an application for managing resources, and a graphical modelling tool, in which a user may compose HLA federations out of HLA federates residing within the network. A snapshot of the graphical M&S tool is shown in figure 2.1. Users can also run and view the composed HLA federations from the environment. The execution is performed transparently to the user, within the P2P network, through efficient utilization of idle processing capacity in desktop computers.

2.4 Areas of research

When designing NetSim, we identified some areas of significance to networked environments and network based M&S in particular. We decided to focus on a few, which are:

- Component-based M&S – Allowing reusable, easily distributed simulations
- Standards and techniques for M&S – Distributed, reusable simulations set high requirements on interoperability
- Thin clients – Involving not only PCs and web-based clients in the M&S system, but also PocketPCs and others
- Collaborative environments – Environments that provide a common picture of the problem to solve. Involving problems of maintaining consistency and control within the collaboration group
- Resource utilization – Efficient ways of utilizing distributed resources, through efficient resource description and allocation

Of the issues listed above, the last two have been the key issues in previous and current work. The collaborative work is described in chapter 3, and the resource utilization in chapter 4. More technical descriptions of these have been presented in two papers [9, 10].

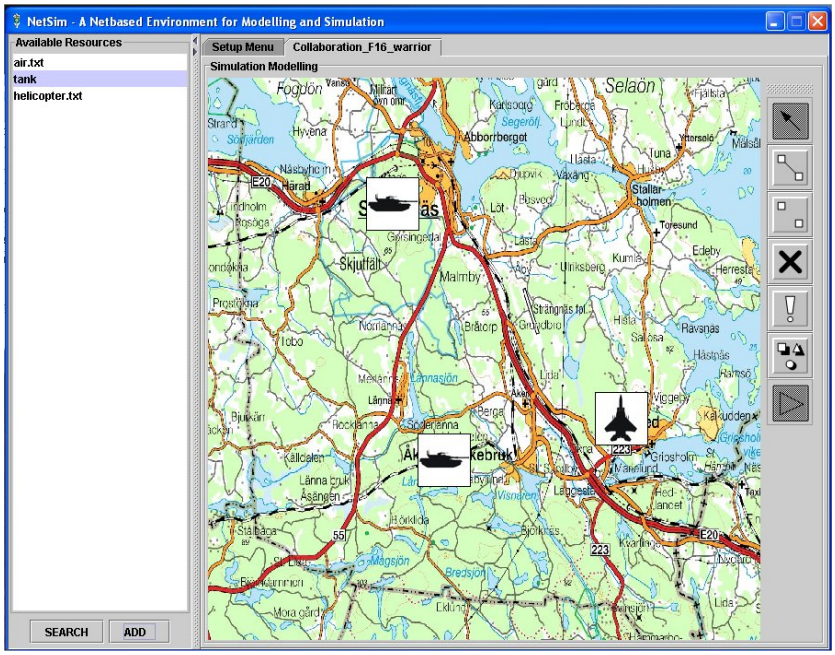

Figure 2.1: Snapshot of graphical modelling tool in NetSim. Users may (collaboratively or not) compose HLA federations out of HLA components (federates) residing in the network.

2.5 Peer-to-Peer based resource sharing

Peer-to-Peer (P2P) as a concept for computer communication is nothing new. In the early sixties, the pioneers of ARPANET formulated their vision of a future computer network comprising host-to-host capabilities. In their vision all connected nodes were equal in terms of functionality and could access resources from any other computer on the network [11]. These early ideas have not greatly influenced how the Internet is used today. The dominating architecture is the client-server model, where resources of various kinds tend to accumulate at dedicated centers. Large parts of the Internet remain unused, as network traffic around certain spots shows increasing activity. However, in the past years ideas and technologies have been put forth that promote the idea of distributing resources through use of P2P technologies. The distribution of resources is advantageous from many aspects; it reduces the occurrence of bottlenecks, minimizes possible system downtime and increases system availability and robustness etc.

P2P has definitely made a great impact on how ordinary desktop computers may communicate and exchange various resources. This is especially true for the so called file sharing applications, although they are heavily questioned in terms of legal property rights. However, some more academic projects have successfully confirmed the strength of P2P for distributed computing. The Intel Philanthropic P2P program has demonstrated utilization of idle processing capacity in desktop computers to solve problems within the medical domain [12], whereas the SETI@Home project represents a successful P2P model for distributed computing, used for processing of radio astronomic data [13].

2.6 JXTA

JXTA is an open-source P2P project, initiated by Sun Microsystems in 2000, providing a standardized and platform independent P2P platform [14]. The system is

based on XML[3] messaging through employment of six protocols. Any piece of Internet connected hardware implementing these protocols, or a subset of them, can participate in a JXTA network. Nodes on the JXTA network are called peers. Peers form peergroups, based on common interests and goals, within which the participants share resources [11]. The JXTA platform provides a rich set of P2P features, thus simplifying the development of distributed systems.

3. Computer supported collaborative M&S

3.1 Computer-based collaboration and M&S

Since the science and hype of Virtual Reality (VR) broke through, huge interest and activities have been conducted within the field. Despite the interest and future-thinking about the area, 3D virtual worlds have not yet reached into our offices and everyday lives. VR is instead a part of the larger field of using computers to support human-human collaboration, an area which has gained far more usage than VR in itself, due to its availability and range of technical possibilities. Groupware, videoconferencing and shared project areas are just a few of the kind of products used for these purposes. Computer-based collaboration can assist in joining people and organizations in the same environment, allowing people to share not only resources but also work areas, tools and environments. Though computer-realized collaboration may never represent a substitute for real human-human work, it can be used for bridging distances and increasing and facilitating cooperation.

These advantages are applicable within other areas as well. If considering collaborative M&S, sometimes referred to as CMAS, it could help joining people like software engineers, VV&A expertise and others in a common computer environment. With CMAS a project team could cooperate on M&S problems, with immediate support from SMEs, and with the customer supervising the activities, no matter if they are located on the same place or not. This improves and assures quality of work and enhances work efficiency. Within the defence in general, computer-based collaboration is a very interesting issue, since often military personnel are spread over long distances. A key feature here is the possibilities of increasing the availability of competence and expertise, an issue which could be of considerable importance within critical systems for C3I and other domains.

3.2 Infrastructure for CMAS in NetSim

One of the main goals of NetSim is to provide support for collaborative work. If complying with the definition of Collaborative Virtual Environments (CVEs) as described in [16], a CVE shall provide shared information, tools and communication access, and need not provide a 3D visual environment. The work of NetSim focus on constructing an every-day used defence environment, for practical collaborative M&S, which is why we dismissed the thoughts of flashy 3D worlds and emphasized on reducing complexity and enhancing availability instead. This increases the possibilities of integrating the infrastructure in already existing systems, such as for example C3I systems. Thus a flexible and lightweight infrastructure for CMAS was

[3] *The Extensible Mark-up Language (XML) is a mark-up language designed to describe and encapsulate data. It has become a major technique for exchanging data in heterogeneous environments, since it provides a platform and programming language neutral data format [15].*

designed. A first prototype has been implemented and integrated within the NetSim environment. It mainly constitutes a middle layer, between a Java GUI[4] and the JXTA P2P network, and is based on two components. The first component is a core, the Collaboration Core (CC), containing functionality for collaborative work, implemented using P2P technology. The second is an HLA coupling that provides means of collaborative simulation, and is implemented on top of the CC. These are further described below, and were describes in detail in [9].

3.2.1 Collaboration Core

The CC allows users to collaborate on M&S activities, or any kind of activities that the environment supports. Currently it provides, among others, functionality for creating, searching for and joining collaboration groups[5]. The collaboration groups are based on the concept of JXTA peer groups, and are used for grouping collaboration participants and for utilizing group functionality and mechanisms for managing collaboration (described in 3.3). A group provides group specific settings and information, and specific services and tools that may be used (collaboratively) within that specific group. The tools can be anything from communication means, such as text chats and web-cams, to advanced M&S or other tools. The infrastructure allows for further extension and integration of tools but, as mentioned before, it is currently just a prototype. The tool that is demonstrated and used today is a simple shared graphical tool, in which users may collaboratively compose HLA federations out of HLA federates. Users cooperate through using communication tools such as chats and web cams. In the current application, a chat is provided within the application and a web cam is used externally. The CC allows participants to share views, meaning that all see the same thing at the same time within a work area or tool. Hence, they see the same actions and changes at the same time, similar to sharing tools over a network, but in a decentralized way through P2P technology (JXTA).

3.2.2 HLA coupling

The second prototype component is the HLA coupling that supplies the user(s) with a graphical interface in which the result of the distributed simulation is visualized. All participants in a group see the simulation and the same states of the simulation at the same time. This feature is implemented using the HLA framework rather P2P technology, for reasons discussed below. Users may stop, play and step-forward the simulation, and all group participants receive the same new states if the simulation is changed or interacted with. This demonstrative collaborative simulation can be of considerable use for military planning, distance education, strategy demonstration, or when using M&S as basis for decision support etc.

3.3 Challenging issues and implemented solutions

During the work some challenging issues were identified that exist within collaborative systems, and which are naturally common for most distributed systems [17], such as mechanisms for maintaining information consistency and fault tolerance. It is challenging to synchronize and coordinate actions and interactions within a group, i.e. to guarantee that all participants have the same common picture at the same time, and that no changes or actions on the same object collide. Another issue was the

[4] *GUI – Graphical User Interface, the visual application in which the user interacts with the environment.*
[5] *A user can be a member of any number of groups.*

(collaborative) simulation, since different platforms demand various solutions for the same visualization. This requires generic interfaces for simulation and tools, which comply with the computer capacity and properties in use, problems that are not fully addressed in current work. Moreover network properties may constitute a problem due to delays and overhead, another issue not covered yet. Challenging issues that are currently considered are presented in brief below.

3.3.1 General infrastructure for collaboration

Designing an infrastructure for collaborative work in an efficient way, which is extensible for integration of new functionality, is not an easy thing. A usual procedure is to employ a server-centric solution, where the server (or central computer) propagates screen-dumps[6] of a shared work area (may be a tool) to all participants. This is used by products such as VNC[7] and NetMeeting[8] and can be easily applied but is inefficient due to overhead among other things. Our implementation (the CC) provides a more optimized solution that is principally distributed and that considers changes in objects' states, and transmits the state changes only, to all participants. On the other hand our solution sets high requirements for integration of new tools into the infrastructure, which may have to be generalized in future work.

The distributed infrastructure was implemented using built-in group functionality in the P2P framework JXTA [14]. A new kind of group was created, the CollaborationGroup, which is an implementation that extends the group concept and includes services for group functionality etc. When a new group for collaboration is created, a new such group is started, instead of an ordinary PeerGroup. When joining a collaboration group users receive a handle to a shared communication channel. Thus, all actions produced within a group are propagated and managed along this channel.

3.3.2 Coordinating participant actions

Collaborative modelling requires real-time interaction. The actions produced must be coordinated and synchronized among the participants. For this we applied a coordinator-based scheme, which is a widely used solution to the problem. A coordinator represents the node through which all actions are passed and coordinated. When a user wishes to perform an action on an object in the shared area, it requests the coordinator for permission. If no other user wants to act upon the same object, the request is processed immediately and all users receive the new shared state, without causality errors or action conflicts. This results in a temporary centralized solution, which is easily implemented but not optimal if a lot of information has to be coordinated. Coordination and synchronization would rather be managed in a distributed fashion (more complex). Our solution also brings that client synchronization is managed immediately, rather than when it is necessarily needed, i.e. when users need to have the same views. If all participants' views are coordinated only when needed, a better and more scalable implementation would be achieved. Thus, these two issues are of concern for current and future work.

---

[6] *Screen-dump = a momentary image taken of the screen.*
[7] *VNC – Virtual Network Computing is a free product for sharing work areas [18].*
[8] *NetMeeting is a product that allows projects to use shared work areas and tools [19].*

3.3.3 Synchronizing collaborative simulation

Collaborative simulation does not require such frequent coordination of interactions as other M&S activities do, since the user interactions during simulation are most likely simple ones such as stopping or pausing the simulation. Thus coordination is not an issue here. In contrast, a high amount of simulation information needs to be transmitted to and synchronized between the users, in order to guarantee that all users see the same state of the simulation at all times. This means that it may not be possible for the information to be continuously synchronized due to the overhead and time delays it may cause. In current implementation, the clients' views are synchronized continuously (synchronously), and would preferably be exchanged with more efficient solutions. The issue of effective synchronization was highlighted in previous work [9], and is an important part of current work (discussed below).

3.4 Synchronizing collaboration participants using HLA

When implementing functionality for collaborative simulation, the design choice was made to use HLA instead of JXTA for synchronizing participant visualization and user interaction in the simulation. One of the reasons was that HLA provides excellent functionality for time management and means for federation synchronization [20]. The HLA coupling was implemented as a layer on top of the CC, as mentioned above. Each user application comprises HLA functionality, which acts as an HLA federate, called the Visualizer federate. The Visualizer subscribes to the simulation objects and attributes necessary for visualizing the federation, in order to illustrate the accurate simulation result in the client's window. The Visualizer federates, i.e. the clients' views, are synchronized using HLA built-in mechanisms. User interactions with the simulation are handled through the Visualizer and are forwarded to the rest of the federation. An HLAManager[9] reflects the interaction event, and makes sure the proper action is processed, as for example pausing the federation or stepping it forward. Federations used today are based on time-stepped federates and the synchronization is performed synchronously in fixed time intervals. This is neither efficient nor scalable, since it results in a great number of synchronization points, no matter if anything important has occurred or not. This may in turn cause time delays and unnecessary overhead.

In order to investigate more efficient ways of synchronizing the federation for our purposes, current work emphasizes on facilitating the use of time management (TM) in HLA. A middle layer on top of the HLA/RTI is being designed and implementation of it has been initiated, which is somewhat similar to approaches made such as [21] and [22]. The layer is included and utilized in each federate, and comprises functionality for TM and various synchronization protocols[10]. A schematic view of this is presented in figure 3.1. Protocols that are intended are first of all simple solutions for synchronous and conservative simulation, but optimistic protocols are also considered. The layer is designed to relieve the simulation developer from some of the HLA specific logic. It will also provide various ways of synchronizing federations and estimating performance, which allows flexibility of synchronization

[9] *The HLAManager is used for facilitating some functionality within a federation, such as controlling synchronization and managing the federation. This is not crucial and everything may be carried out within each federate instead, but it facilitates federation development.*

[10] *The middle layer is nothing necessary, but it facilitates working procedures, user flexibility and provides extra functionality.*

protocols. It is intended to support us in evaluating and implementing efficient synchronization for the collaborative infrastructure, but can be of use within other areas as well.

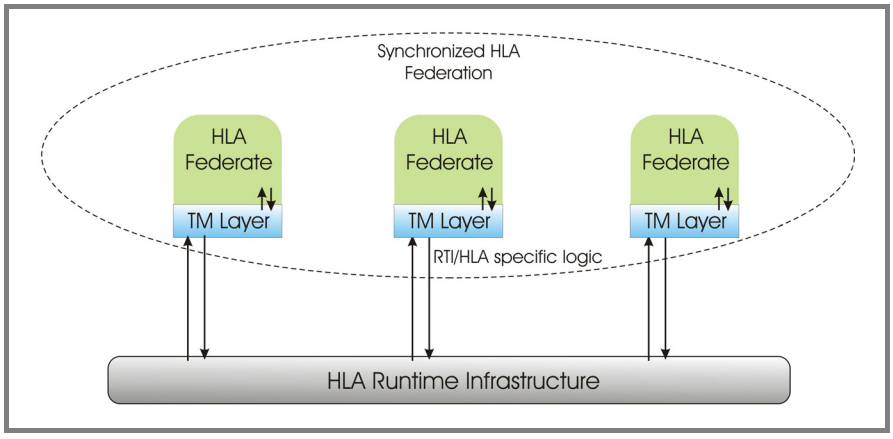

Figure 3.1: Schematic picture of middle layer for RTI/HLA specific logic.

3.5 Conclusions

Applying JXTA P2P functionality for coordinating and supporting collaborative simulation modelling turned out to be a good solution. The concept of JXTA peer groups and functionality for groups such as membership, authentication, group services etc. was very beneficial within this context. The group concept was extended, and gave us the result desired. But JXTA was by the time of work not a fully complete technology[11], and was not very easily managed.

For the collaborative modelling a centralized coordinator-scheme was used. Since this solution is not scalable or efficient, it can be done more efficiently in a decentralized way. This issue is considered in current work. Also, the design to use optimized information flows (instead of screen-dumps) proved to be good. For synchronizing and visualizing the collaborative simulation the HLA framework was applied, something that proved useful but needs to be extended regarding flexibility and ease-of-use. Synchronizing the federation synchronously showed non-efficient, and an alternative solution is currently designed which constitutes a flexible middle layer between HLA and the federates. During the work it was pointed out that CMAS can be of considerable use within the defence, such as distance education and military planning. This holds for activities that use M&S as basis for decision support and situations when presence of SME:s may not be physically possible, but highly desirable etc.

4. Resource management

4.1 Resource utilization for distributed simulations

As part of the NetSim environment, a module for execution of HLA federations has been developed based on the JXTA P2P platform, described in section 2.6. The main idea of this module, the Distributed Resource Management System (DRMS), is to utilize idle processing capacity in a network of workstations for distributed

[11] *The latest version of JXTA is 2.0, a version which the authors of this paper have no practical experience of yet.*

simulations. Furthermore, it should provide a distributed repository for storage of simulation components and associated documentation. Other projects have explored these possibilities, see for example [23], but then often based on the client server model for management of resources and storage of simulation components.

The basic idea of the DRMS is that desktop owners within an organisation download and install a small client that under certain circumstances share resources with other connected nodes. There are currently three levels of involvements for connected nodes. First, a node may share computing capacity for execution of HLA federations, referred to as a *computing resource* in the following text. Second, a node can be part of the distributed repository for sharing of content (HLA federates, documentation etc.). Finally, a node may share both computing capacity and content. The desktop owner always has the option to withdraw its involvement by changing a switch in the user interface or by closing the client. Therefore, the availability of resources on the network is expected to change fast and unpredictably in an Ad-Hoc manner. To comply with this the system includes mechanisms for migration, or movement, of federates between available computing resources during a federation execution. Furthermore, the dynamic characteristics of the network calls for redundancy (replication) in storage of simulation components to gain access to the same set of federates at all times. However, this part of the problem has not yet been fully addressed in the current implementation.

A major aspect to consider when implementing any type of P2P based system is discovery and matching of resources. The first problem relates to the basic strategy used to discover the presence of other nodes/resources on the network. Another problem to handle is how to identify those resources that match certain requirements. The JXTA platform, and thus the DRMS, supports three different mechanisms for identifying nodes/resources, these are [24]:

- *No discovery* – using this approach, nodes rely on a cache of previously located advertisements that describe the features of resources. This is implemented by broadcasting advertisements from nodes at regular time intervals
- *Direct discovery* – in this case the nodes do not publish any advertisements until they are asked to do so, i.e. until a consumer of resources broadcasts a resource request on the network. This strategy is often referred to as flooding
- *Indirect discovery* – using this approach all nodes publish their advertisements to a centralized catalogue. The consumer node locates resources by requesting the catalogue. However, when a consumer has identified a producer, the communication is performed directly between involved parties

We have not yet performed any measures of the performance of these three approaches, but this will be addressed in future work, where also the new JXTA 2.0 release will be taken into consideration. When suitable resources have been identified by consumer peers, the requirements of the requests have to be matched against the features of available resources. At present, the implementation includes simple

mechanisms for this activity. First, the SOMs[12] of selected federates are automatically matched to assure simulation interoperability. Then federates are mapped to available computing resources i.e. nodes among the list of available resources are selected and assigned jobs to execute the federates. The advertisements of computing resources contain node specific information, for instance running hardware, software etc. Using this information, nodes running the fastest processors are chosen to execute federates. For a more technical description of the DRMS see [10].

The present approach of matching simulation components to form simulations and mapping individual components to computing resources is rather rudimentary. There is a need to enable matching of simulation components, not only at the architectural level (matching of SOMs), but also at a higher level. Furthermore, it is also important to describe a simulation component in terms of its requirements on the execution environment, and likewise describe the features that a computing resource provides for a simulation component. This calls for a better way of managing meta-descriptions of resources within the NetSim environment, to facilitate efficient searching, matching and execution.

4.2 Describing resources

This section gives an overview of ongoing and future research topics, aimed at extending the support for metadata in the NetSim environment. The employment of meta-descriptions of resources within the NetSim environment is especially pronounced during three activities; searching for simulation components, matching simulation components and during execution of simulations. The role of metadata also differs greatly between these activities, which will be explained below. However, there are no solid boundaries between the uses of metadata in these activities. Certain types of metadata may be applicable in all three cases.

4.2.1 Searching for components

In this activity the user/users of the NetSim environment searches for available simulation components or previously assembled simulations. The basic requirement on metadata supporting this process is a well defined class structure, identifying subclass/superclass relations. This enables simple queries like “all airplanes” or “all fixed-wing aircrafts”, which yields all components which are subclasses of airplane or fixed-wing aircraft respectively. However, note that this classification is not equal to the implementation related object class structure. Furthermore, the components should be described in terms of a system-of-systems view where, if applicable, a component’s relations with other components are defined. For instance describing that system A and system B may integrate to form the superior system C. Finally, the metadata infrastructure should support descriptions of roles or capabilities, which enable searches in the form of “air based transportation” or “underwater surveillance”.

4.2.2 Matching components

This activity represents the attempt to compose simulations out of a number of components, i.e. component based development. This area is sometimes labeled

[12] *SOM is the short term for ”Simulation Object Model”, and is the documentation and definition of a federate’s all characteristics and possible interactions etc.*

composability and has been investigated extensively to promote model reuse and interoperability. According to [25] composability is:

"the ability to combine and recombine components into different simulation systems for different purposes."

Irrespective of matching at the simulation architectural level, for instance matching of SOMs in the case of HLA, an environment supporting component based simulation development should include extensive metadata. This is to guarantee the composition of valid simulations at all levels. [26] outlines some of the fundamental requirements on metadata to support composability, namely;

Information about the model as a software component:

- Programming language
- Communication protocol
- Location of component

Information about the model as a simulation component:

- Spatial resolution
- Aggregation
- Temporal resolution
- Fidelity
- Required services

4.2.3 Executing simulation

This final activity involves mapping simulation components to computing resources prior to and during federation execution, i.e. assign jobs to various nodes in the network. Metadata that should support this process include running hardware and software on the computing resources and a set of requirements imposed by the simulation components. These requirements consist of information such as; what platform is needed to run the component? Is a specific runtime-environment needed to run the component? How computing intensive is the component? etc. Note that this process is not only required prior to the execution. Since the allocation of computing resources is not static, it is necessary to perform rescheduling from time to time.

4.2.4 Metadata framework

In order to create a foundation (or framework) for metadata, to support resource consuming systems (M&S and C3I systems) in various ways, a number of components are required:

- *Meta-language* – formal semantics and syntax, expressing shared and common understanding of a domain
- *Metadata repository* – supporting uploading/downloading of metadata through standardized protocols (http, SOAP etc.), consistency checking and version control
- *Query language* – supporting complex queries on the metadata

Figure 4.1 outlines a conceptual view of a framework for metadata supporting efficient searching, matching and execution within the NetSim environment. The central component of this system is a repository where the descriptions of resources on the network, simulation components, data and computing resources are stored. The resources should be annotated according to a standard for data representation augmented with the shared knowledge of the domain. Activities within the NetSim environment are then supported by extraction of meta-descriptions from the repository, followed by semi-automated reasoning using the knowledge expressed by these descriptions

There are a number of efforts that could provide a basis for our envisioned metadata framework, for instance the Resource Description Framework (RDF) [27], the Web Ontology Language (OWL) [28] promoted by the W3C [27] or the DAML-S initiative, supporting semantic mark-up of Web Services [29]. These approaches support the creation of specialized schemas, to represent the knowledge within a domain, which are used to describe various resources on the Web. There are also several efforts within the semantic web research community that build on these concepts to provide frameworks for meta-data driven solutions. Work has been carried out to support RDF based metadata in JXTA, including query, replication, mapping and annotation services [30]. Several other projects have constructed dedicated RDF databases with support for various RDF query languages, see for example [31] and [32]. The features of these approaches are diverse, ranging from stand-alone to distributed databases or P2P-style systems.

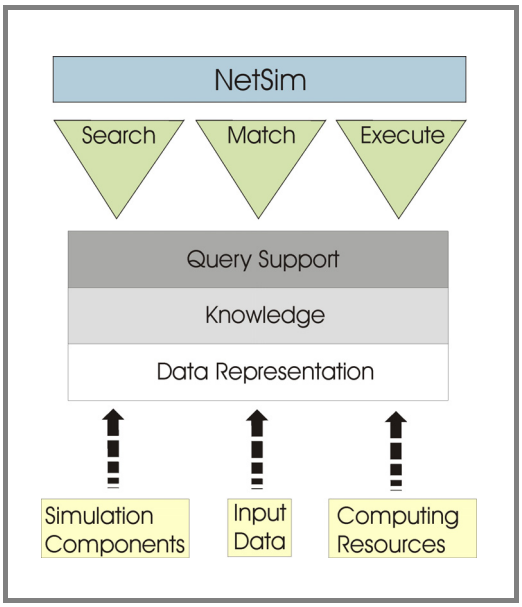

Figure 4.1. Conceptual view of a proposed framework for metadata enabling efficient searching, matching and execution within the NetSim environment.

4.3 Conclusions

From our experiences developing the DRMS we consider the lack of supporting metadata within the NetSim environment of major concern. The support for meta-description of resources within JXTA in general is relatively weak, mainly keyword based searches of resource advertisements. We envision a layer on-top of JXTA

supporting more complex descriptions of resources derived from a shared view. The meta-data layer should support the users of NetSim in simplifying identification and matching of resources as well as for optimization of the federation execution. We consider that the work carried out within the semantic web community is of great interest in this respect. Concepts from this area could be applied to model knowledge and provide extensive meta-descriptions of resources to enable automatic/semiautomatic localization, selection, composition and execution of various resources.

5. Summary & conclusions

At the Department of Systems Modelling, Swedish Defence Research Agency, ongoing research is targeting the role of network/web based technologies in M&S, to support defence communities in their work. During the first phase of this research, focus has been on efficient resource sharing and means of computer collaboration. A prototype, named NetSim, has been implemented to investigate and demonstrate these issues, based on the open-source Peer-to-Peer platform JXTA. The NetSim prototype allows people at disperse locations to collaborate in creating various HLA simulations in a component-based manner. Executions of assembled simulations utilize idle processing capacity of desktops currently connected to the system. As the NetSim is based on Peer-to-Peer concepts, and not dependant on a single server or desktop machine within the network, the system is to some extent more robust and fault-tolerant than a client-server solution. However the synchronization of collaboration participants is partly based on centralized control, which proved non-efficient and non-scalable. JXTA was at the time of implementation not a fully mature technology, which affected the overall performance of NetSim to some extent. For example, it lacks support for extensive meta-descriptions of available resources on the network. However it should be pointed out that JXTA provides a rich set of P2P features, suitable for implementation of distributed systems such as NetSim. Future research will address distributed algorithms for synchronisation of collaborative work and a more flexible and extendable approach to resource management.

6. References

[1] D. H. Timian, et. al., *Report out of the C4I study group*, Simulation Interoperability Workshop, FSIW00, 17-20 September 2000, Orlando, Florida, USA.

[2] F. Kuhl, R. Weatherly, J. Dahman, *Creating Computer Simulation Systems*, Prentice Hall, Upper Saddle River, US, 1999.

[3] A. Tolk, Klaus Schiefenbusch, *Decision Support Systems-Concepts and Effects*, AORS XVIII Special, Session 2: C4ISR and Information Warfare, Fort Lee, Virginia, October 1999.

[4] G. Allerbo, *Ledningssystem bygga med Peer-To-Peer Teknik*, CIMI 2003, 20-22 May 2003, Enköping, Sweden.

[5] B. Mattiasson, T. Andreasson, *Peer-To-Peer som en bas för NFB*, CIMI 2003, 20-22 May 2003, Enköping, Sweden.

[6] D. Barkai, *An Introduction to Peer-to-Peer Computing*, Intel Developer UPDATE Magazine, http://developer.intel.com/update/issue/idu0201.htm, February 2001.

[7] J. Ulriksson, F. Moradi, O. Svensson, *A Web-based Environment for building Distributed Simulations*. Proceedings of the European Simulation Interoperability Workshop, London, Great Britain, 2002.

[8] T. Berners-Lee, J. Hendler, O. Lassila, *The Semantic Web*, Scientific American, May 2001.

[9] J. Ulriksson, F. Moradi, R. Ayani, *Collaborative Modelling and Simulation in a Distributed Environment*. Proceedings of the European Simulation Interoperability Workshop, Stockholm, Sweden, 2003.

[10] M. Eklöf, F. Moradi, M. Sparf, *HLA Federation Management in a Peer-to-Peer Environment*. Proceedings of the European Simulation Interoperability Workshop, Stockholm, Sweden, 2003.

[11] J. Gradecki, Mastering JXTA: *Building Java Peer-to-Peer applications*. 2002, ISBN 0-471-25084-8, John Wiley & Sons.

[12] Intel Philanthropic P2P Program, http://www.intel.com/cure/.

[13] The Search for Extraterrestrial Intelligence, http://setiathome.ssl.berkley.edu/.

[14] Project JXTA, http://www.jxta.org.

[15] S. St.Laurent, *XML A Primer*, M&T Books, New York, US, 2001.

[16] E. Churchill, D. Snowdon, A. Munro, *Collaborative Virtual Environments – Digital Places and Spaces for Interaction*. Springer-Verlag 2001.

[17] R. Fujimoto, *Parallel and Distributed Simulation Systems*. John Wiley & Sons Inc, January 2000, ISBN: 0471183830.

[18] VNC – Virtual Network Computing. URL: http:/www.vnc.org.

[19] Microsoft's NetMeeting. URL: http://www.microsoft.com/windows/netmeeting.

[20] R. Fujimoto, *Time Management in the High Level Architecture*. Simulation, Vol. 71, No. 6, pp. 388-400, December 1998.

[21] S. Turner, W. Cai, J. Chen, *A Middleware Approach to Causal Order Delivery in Distributed Simulations*. Proceedings of the EuroSIW, Stockholm, Sweden, June 2003.

[22] T. McLean, R. Fujimoto, B. Fitzgibbons, *Middleware for Real-Time Distributed Simulations*.

[23] J. Lüthi, S. Großmann, *The Resource Sharing System: Dynamic Federate Mapping for HLA-based Distributed Simulation*, Proceedings of the Fifteenth Workshop on Parallel and Distributed Simulation, PADS 2001, Lake Arrowhead, US, May 2001.

[24] B.J Wilson, *JXTA*, New Riders Publishing, http://www.brendonwilson.com/projects/jxta/.

[25] M. D. Petty, E. W. Weisel, *A Formal Basis for a Theory of Semantic Composability*.2003 Simulation Interoperability Workshop, Orlando FL, March 30-April 4 2003, Paper 03S-SIW-054.

[26] R. H. Pollack, R. Baldwin, J. R. Neyer, D. F. Perme, *Requirements for Composing Simulations: A Use-Case Approach*. Simulation Interoperability Workshop, 2002.

[27] W3C, RDF, http://www.w3c.org/RDF/.

[28] OWL, Web Ontology Language, http://www.w3.org/2001/sw/webont.

[29] DAML-S, Darpa Agent Mark-up Language, http://www.daml.org/.

[30] JXTA Edutella, RDF-based metadata infrastructure for P2P applications, http://edutella.jxta.org/servlets/ProjectHome.

[31] HP Labs semantic web research, http://www.hpl.hp.com/semweb/index.html.

[32] Sesame, open-source RDF repository, http://sesame.aidministrator.nl/.

II. Peer-to-Peer-Based Resource Management in Support of HLA-Based Distributed Simulations

M. Eklöf, M. Sparf, F. Moradi, & R. Ayani
SIMULATION 80: 181 – 190, 2004.

Abstract

In recent years the concept of peer-to-peer computing has gained renewed interest for sharing of resources within and between organisations or individuals. By sharing resources, cost related to investments of hardware and development of software is reduced. Reusability and availability of simulation components have been a long-term goal of the Modelling and Simulation (M&S) community. Much has been done on the simulation architectural level to promote this, but still the simulation components are not widely shared on a regular basis.

This paper describes a Decentralized Resource Management System, DRMS, which utilizes a network of workstations for execution and storage of HLA (High Level Architecture) federations/federates in a peer-to-peer environment. The implementation of DRMS is based on the open source project JXTA, which represents an attempt to standardize the peer-to-peer domain. DRMS is part of a web based simulation environment supporting collaborative design, development and execution of HLA federations. The overall aim of this work is to evaluate the possibilities of using peer-to-peer technology for increasing the reuse and availability of simulation components within the defence M&S community. More specifically it addresses the necessary adjustments of simulation components (federates) in order to conform to the requirements of the DRMS.

This study shows that JXTA could provide the foundation for a distributed system that increases the possibilities for reuse of simulation components. However, in order to achieve this, the individual simulation components accessible from the environment have to conform to requirements that the DRMS dictates. This includes methods for capturing the internal state of a federate and restoring from a previously saved state. These features are required to migrate federates between host environments as nodes disconnect. Furthermore, the experiences from using the JXTA platform revealed that it should be extended to include better support for describing resources at a meta-level, to facilitate better localisation, selection and matching of simulation components. Finally, none of the employed mechanisms for discovery of nodes on the DRMS network are considered satisfactory on its own. Preferably, a more complex mechanism is needed to assure scalability and robustness of the system. Fortunately, the JXTA project has evolved in this direction by the new release of the platform (JXTA 2.0) that extends the current indirect (rendezvous) discovery concept.

1. Introduction

Peer-to-Peer (P2P) and Grid technologies provide means of sharing resources among users in a network. A mutual collection of resources constitutes a major benefit for organisations and inter-organisation cooperation, both economically as well as academically. By sharing hardware, software and data within a community, utilization of existing resources is improved, which could reduce costs related to hardware investments, software development, data preparation etc. The concept of sharing strongly facilitates the potential of reusing developed software components and gathered data by making these available to a larger group of people.

The software development community in general and the modelling and simulation community in particular, have put considerable amounts of efforts into development of technologies to support reuse of software components at the inter-component communication level (interoperability). Within the domain of modelling and simulation, standards have evolved supporting reusability and distribution of simulation components, i.e. DIS (Distributed Interactive Simulation) and later HLA (High Level Architecture). The HLA standard provides methods to connect simulation systems built for separate purposes, implemented during different technological eras, produced by diverse vendors and developed for various platforms through services provided by a Run-Time Infrastructure (RTI) [1]. However, even though software components do conform to the HLA standard, it does not necessarily mean that they are being reused.

The Swedish Defence is a strong supporter of HLA and has selected it as the major architecture for distributed simulations. HLA-based simulations (federations) and simulation components (federates) are developed in various contexts. However, the developed simulation software and experiences from using these are rarely shared within the defence community, at least not in an efficient and systematic fashion.

Peer-to-Peer and Grid technologies represent interesting platforms for managing available resource more economically and efficiently within the defence. An infrastructure for sharing resources related to M&S activities would allow users/projects to share their own work, and the work of others, in an efficient way. Moreover, it could provide a pool of processing capacity enabling single users to execute computing intensive tasks from their own machines.

Some approaches for managing resources related to distributed simulations have previously been reported. In [2] a Resource Sharing System (RSS) is presented that utilizes idle processing capacity in a network of workstation to execute HLA federations. The owners of workstations within a LAN can control the availability of their computers, through a client user interface, for execution of individual federates of a federation. Computers that are willing to share their resources are registered with the RSS manager that performs elementary load balancing. The RSS is built around a centralized manager that relies on an ftp-server for storage and migration of federates. Currently there is no extensive fault tolerance mechanisms included in the RSS implementation, but as this is an important feature of distributed simulations, and not well supported in the HLA, the RSS is planed to include functionality for check-pointing and management of replicated federates and fault detection. In [3] an

alternative approach to dynamic utilization of resources for execution of HLA federations is presented. In this case the framework is based on Grid technologies, more specifically services of the Globus Grid toolkit [4]. Each federate in the proposed architecture is embedded in a job object that interacts with the RTI and a Load Management System (LMS). The LMS performs two major tasks through use of a job management system and a resource management system. These systems carry out load balancing whenever necessary and discovery of available resources on the Grid. In [5] an adaptive framework, Generic Adaptive Interaction Architecture (GAIA), is outlined which supports dynamic allocation of model entities to federates in an HLA-based simulation framework. The potential benefit of this framework is the reduction of messages being communicated among separate execution units. This is achieved by a heuristic migration policy that assigns model entities to executing federates as a trade-off between external communication and effective load-balancing. Load-balancing is required to avoid concentration of model entities over a small number of execution units, which would degrade the performance of the simulation. The proposed mechanisms proved beneficial in simulating a prototype mobile wireless system by reducing the percentage of external communication and by enhancing the performance of a worst-case scenario.

The purpose of this work is to investigate how simulations based on HLA can be managed in a peer-to-peer environment. More specifically, how an infrastructure based on peer-to-peer technology may facilitate issues not currently supported by the HLA standard and the implemented RTIs, such as sharing of M&S related resources, fault-tolerance and load balancing. During the first stage of the work, efforts have been made to implement a system for efficient sharing of resources, such as sharing of simulation components (federates) and idle computing capacity. Currently, no advanced mechanisms for load balancing, such as those described in [3] and [5], have been considered. Moreover, techniques for increased fault-tolerance have not yet been implemented. However, as the current work addresses sharing of computing capacity in a dynamic network environment, the issue of federate migration has been of major concern. As pointed out in [6], [2] and [5] the HLA/RTI does not include support for migration of federates, which, among other things, is required to implement load balancing or increasing the fault-tolerance of a system. The problems relates to the fact that the HLA and its implementations does not support transferring of a federate's internal state. The functionality developed in this context could serve as a foundation when considering load balancing and fault-tolerance in the next stage of development.

An important aspect of our work was to investigate the constraints put on simulations and individual simulation components if managed and executed within a peer-to-peer framework. It was observed that federates of a federation require some additional functionality to enable migration imposed by the dynamic availability of individual execution hosts. The level of adjustment for federate developers, in order to conform to these requirements and to participate in an M&S-related community for resource sharing, is of particular interest. The issues were addressed by using a common platform for peer-to-peer computing, namely the JXTA peer-to-peer platform [7].

The rest of this paper is organized as follows. Section 2 introduces the technologies and tools employed in this study. In section 3 the architecture of the implementation,

the Distributed Resource Management System (DRMS), is outlined, whereas section 4 describes some issues of the current implementation along with a presentation of an example of use. Section 5 discusses the feasibility of utilizing JXTA for managing HLA-based simulations and section 6 concludes our work.

2. Technologies & Tools

2.1 JXTA

As the popularity in adopting peer-to-peer technologies increases, efforts to create viable infrastructures for peer-to-peer systems are continuously put forth, for example JINI [8] and Gnutella [9]. Among the promising proposals for a general and standardized peer-to-peer architecture, JXTA has emerged as a strong contender. JXTA is short for Juxtapose, as in "side by side". JXTA is an open source project initiated by Sun Microsystems in 2000, aiming at providing system and community interoperability, platform independence and technology ubiquity for peer-to-peer solutions [7].

At the core JXTA comprises a set of APIs and XML-based protocols, providing basic elements for peer-to-peer computing. JXTA employs traditional peer-to-peer concepts, but also introduces some new aspects. The following section briefly outlines the most important notions of a JXTA-based system, according to [10], [11] and [12].

Peer – A peer is a virtual communications point. In any peer-to-peer-based solution, a peer constitutes the fundamental processing unit. A peer does not represent a single user, since a user may be in possession of multiple peers, for instance on their phone, home/office computer or other device.

Peergroup – A peergroup is used to group peers and give access to specific services that are only available to group members. Before a peer can start communicating with other peers, it must belong to a peergroup. However, a peer is not limited to join only one peergroup. Peergroups are meant to divide the peer-to-peer network into groups with common goals and interests.

Endpoints and Pipes – The endpoint is the basic addressing method used to communicate between peers in a JXTA-based system. A simple example of this is an IP address and port, which can be used to open a communication stream to other peers. However, JXTA places a layer on top of streams called pipes. If the endpoint is the basic addressing method, then the pipe is the basic communication tool that JXTA peers use to communicate with each other.

Messages – A message is a package that JXTA application developers use to send data through pipes. This can be done in two different ways, XML or binary.

Advertisements – An advertisement is a special document that announces the presence of JXTA resources. A resource can in itself be anything used by a peer, for example other JXTA peers, peergroups, pipes and services provided by JXTA peers.

Services – A service provides the capability to perform "useful work" on remote peers. This might include transferring of a file, performing a calculation and so on.

Identifiers – JXTA offers a number of different identifiers, for example large, unique identifiers, names or URLs. These are used to assure that each resource is uniquely identifiable within the JXTA-network.

2.2 HLA

The High Level Architecture (HLA) is a standardised architecture providing means of connecting independently developed components to form a simulation. In this context, components refer to simulation models, tools etc. that are reusable, i.e. they are not developed for a single simulation [13].

A HLA-based simulation is referred to as federation, while individual participating components are called federates. Federates can be of numerous types, ranging from manned simulators to federation support systems. A federation is formed by connecting individual federates to a Run-Time Infrastructure (RTI). The RTI resembles a distributed operating system for federations, by providing basic services that enables interaction between participating federates [1]. Figure 1 illustrates a simple federation, where three federates are connected to the RTI and interact through the services defined by the interface and supplied by the RTI.

It is essential to provide a concise and rigorous description of an object model template in order to define the interface between federates and the RTI and the service that the RTI provides. In HLA a Simulation Object Model (SOM) documents characteristics of a federate to facilitate its reuse. However, this description is not of concern to the RTI, which instead relies on the Federation Object Model (FOM). The FOM describes the inter-federate communication by defining a federation's object classes and interaction classes [14].

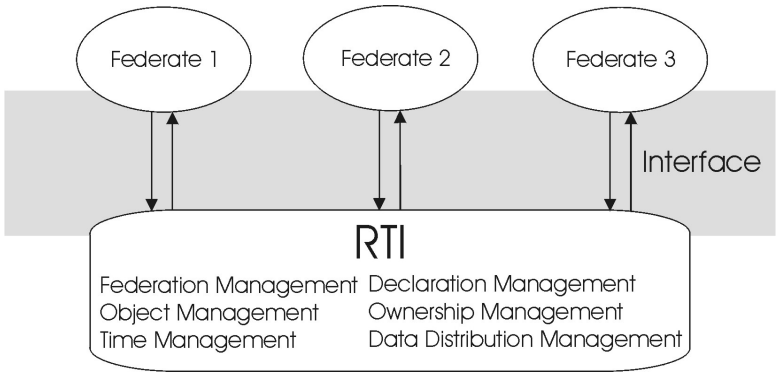

Figure 1. Federate interaction through services provided by the RTI.

The HLA was originally developed by the Defence Modelling and Simulation Office (DMSO), to support reuse and interoperability across the large number of simulations developed and maintained by the U.S. Department of Defence (DoD). The HLA Baseline Definition was completed on August 21, 1996 and since September 2000 the

HLA is an approved open standard through the Institute of Electrical and Electronic Engineers (IEEE) - IEEE Standard 1516.

2.3 JAVA

Java was used as a common platform when implementing the DRMS. Thus, the Java binding of the JXTA protocols provided the fundamental peer-to-peer functionality, while a Java implementation of HLA, pRTI, was employed for federation execution [15]. Java was chosen to offer a platform independent solution, suitable for integration with web-based applications.

3. DRMS Architecture

3.1 The NetSim Environment

The DRMS provides services for federation execution as part of a network based simulation environment, called NetSim, being developed at the Department of Systems Modelling, Swedish Defence Research Agency [16]. NetSim supports collaborative simulation development and execution through a web-based interface, accessible within an organization. Figure 2 outlines the major layers of the NetSim environment. The top layer comprises various NetSim tools dedicated to M&S related tasks. At present a simple composition tool (CT), for development of HLA federations, has been developed. The CT supports composition of a federation by a single user, or collaborative development of federations by a number of users. The NetSim tools derive their functionality from the NetSim services layer. This layer consists of a Collaboration Core (CC) providing services in support of collaborative work [17], a Content Management System (CMS) for sharing of content and the DRMS that provide services for sharing computing capacity. The DRMS manages the execution of simulations composed in the CT by utilizing idle processing capacity on the network. The NetSim services layer is based on fundamental services provided by the overlay network services layer. In this layer JXTA provides fundamental peer-to-peer functionality, whereas the services provided by an RTI are used for execution of assembled federations. Beneath all this is a physical network, which in our case is a Local Area Network (LAN). The features provided through the NetSim environment are transparent to the users, who experience the NetSim tools as being locally installed applications.

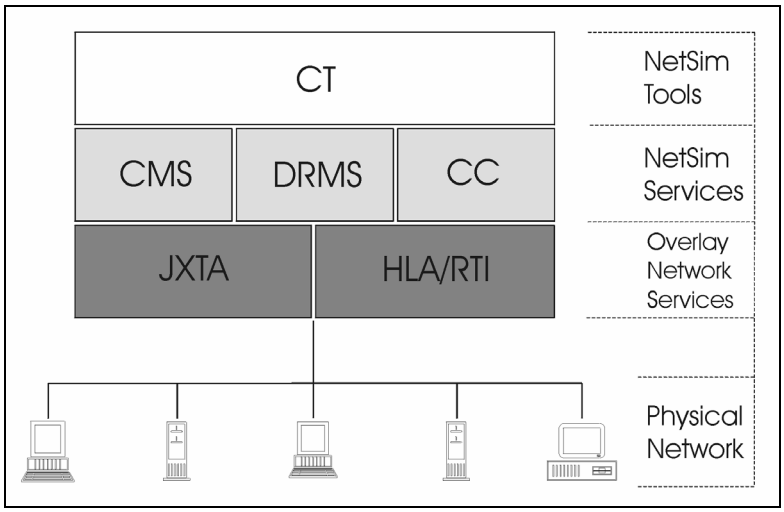

Figure 2. Conceptual view of the NetSim architecture.

3.2 System Overview

The basic idea behind the DRMS is that users within an M&S community download and install a lightweight client. The concept mimics that of SETI@Home [18] where users download and install a small application that under certain circumstances shares the resources of the computer with others. A DRMS-client provides functionality for sharing of idle processing capacity of its host computer. The owner of the individual desktops determines under which circumstances the sharing of processing capacity should take place, by changing a switch in the user interface of the DRMS-client. Before implementing the DRMS-client some fundamental features of the system were defined. A DRMS-client should support:

- Dynamic utilization of idle processing capacity of its host environment, for execution of simulations based on the HLA standard
- Execution requests from the NetSim tools layer, at present execution requests from the CT
- Control of a simulation (federation) from the NetSim tools layer, at present federation control by the CT

Moreover, services for sharing of content have been added to the DRMS-client through incorporation of CMS services, thus enabling:

- Storage of simulation components (federates) and documentation in a distributed fashion
- Content requests from the NetSim tools layer, at present content requests from the CT
- Federate migration

A dynamic utilization of processing capacity on a network means that allocation of resources is not static. It is not possible to reserve a resource for a given amount of time, since individual nodes connect and disconnect in an Ad-Hoc manner. Because of this, the DRMS must include support for migration of federates between host environments during simulation execution. The management of resources is handled

by a Resource Manager (described in section 4.3), which keeps track of available resources prior to, and during a simulation, assigns federates to available nodes and coordinates the migration (transfer) of federates between nodes. The Resource Manager utilizes functionalities of an HLA Manager (described in section 4.2) in coordinating an executing federation. The functions of this component are also employed by the CT, giving control of the federation to the user/users of the system. This includes operations for pausing, stepping and stopping the federation execution.

The storage of simulation components and documentation is managed by a distributed repository (described in section 4.1) through services provided by the CMS. This repository contains all simulation components available for users of the CT and provides means of sharing content among participating nodes. This is a vital feature since it supplies the mechanisms to transfer a component from its original location to a host environment for execution.

4. DRMS Implementation

4.1 Distributed Repository

A core component of DRMS is the distributed repository. This repository contains all simulation components and their associated documentation that are accessible for users of the NetSim environment (the Composition Tool at present). All participating peers in the network are capable of sharing new files, which means that users have the option to distribute new federates as they are developed. An important feature of the repository becomes evident prior to and during simulation execution, namely the downloading and uploading of files. When a peer has been assigned the task of executing a federate, the files representing that federate are downloaded from the common file area, provided that the files does not already reside in the local directory of the peer. Further more, during migration of federates throughout simulation execution, the current status of a federate is uploaded to the common file area, from where it is retrieved by the peer expected to continue the federate execution.

The file library of DRMS is based on the JXTA Content Management System, CMS, which provides a framework for sharing/exchanging content among JXTA peers. More specifically, the component supplies means of hosting, locating and retrieving of content [10]. However, the search and file transfer mechanisms of CMS are fairly simple. For example, the association of meta-data with shared federates and the capability to query this information is somewhat limited. The protocol for querying the repository is CMS specific and based on XML.

4.2 HLA Manager

The purpose of the HLA Manager is to allow the Resource Manager to control the federation and give control of a federation to the users of NetSim. This includes functionality to start simulation, control the simulation to wall clock time ratio (the upper limit depends on the individual capacity of participating workstations) and termination of a simulation. The HLA Manager is a federate that is part of an executing federation.

The HLA Manager currently supports conservative time management. This means that all federates are both time-constrained and time-regulating. Thus, the advancement of time in a simulation is carried out according to the following principle: All joined federates calculate their internal state for the current time-step. During this phase the HLA Manager does nothing but make a request for advancement of time to the RTI. After federates are done calculating their current state, they also make a request for advancement of time to the RTI. When all federates have made this request, RTI grants the advancement of time.

The HLA Manager has the possibility to control the federation execution by not requesting a time advancement, which will temporarily freeze the federation. Further more, the HLA Manager has the option to control the simulation to wall clock time ratio (speed of execution) by waiting a variable amount of time between its requests for time advancement. The conservative time management strategy was mainly chosen to ease implementation of the DRMS during the first stage of its development. We also intend to consider optimistic time-warp in near future.

4.3 Resource Manager

For the purpose of utilizing workstations in the execution of federations, it is necessary to discover available resources on the network. DRMS defines two different types of resources; Manager Resource (MR) and Computing Resource (CR). All participating peers on the DRMS (JXTA) network share a common functionality, which means that all peers are capable of taking the role as MR and/or CR. As an execution request reaches the DRMS from the NetSim environment, an arbitrary peer is assigned the task of managing the simulation. Management in this context refers to resource discovery and allocation, as well as HLA related tasks. The task of the CRs is simply to execute those federates assigned to them by the Resource Manager.

One of the great challenges in peer-to-peer computing is the discovery of resources, mainly because of the lack of a central point of control [19]. According to Wilson [12], JXTA provides three basic ways for a peer to discover an advertisement that describes the presence of a resource:

1. No discovery – a peer can rely on a cache of previously discovered advertisements to supply information about resources, as an alternative to actively searching for advertisements on the network. To implement this, the resources (peers) broadcast advertisements in the peer-to-peer system about their presence and abilities at a regular time interval. The advertisements have a time-to-live value. If the peer doesn't send a new updated advertisement by the time of expiration, it is no longer considered a participating resource.

2. Direct discovery – the resource does not publish any advertisement until it is asked to do so, that is to say until a consumer broadcasts a request for resources over the peer-to-peer network. If a peer fulfils the requirements of a request, it responds by sending back the advertisement that represents the matching resource. This kind of resource discovery is often referred to as flooding and is used to some extent in the Gnutella protocol [9].

3. Indirect discovery – the resource publishes its advertisement to a centralized catalogue, a rendezvous, which a consumer accesses to identify suitable resources. The rendezvous responds with a list of previously cached advertisements that fulfil the requirements of a request. As an extended option, it could further propagate a request to other rendezvous (that option is not investigated in this work).

In all three approaches, once a resource is found the communication is performed directly, without intermediating resources between the consumer and the resource. After this point all approaches behave in a similar manner and all resources are considered to be one overlay network hops away. This characteristic constitutes the difference between a peer-to-peer solution and a client-server solution; once the resource discovery is done, the communication is performed directly between peers in the peer-to-peer approach and not with the server as an intermediate channel as in the client-server approach. In order to identify the most efficient discovery mechanism all three approaches described above were included in the DRMS. Some empirical studies were carried out to verify theoretical assumptions made concerning the effectiveness of the discovery mechanisms, as reported in [20].

4.4 Federate Requirements

Due to the dynamic characteristics of the underlying JXTA network of the DRMS, it is necessary to adapt individual federates to certain requirements. Resources used for execution of federates are not statically allocated, since desktop owners always have the possibility to withdraw their capacity at any time. This means that it is required to include mechanisms for transfer of individual federates between hosts at simulation run-time. To implement this, federates are obligated to include functionality for capturing its current state before it is transferred to a new host environment.

Two approaches for managing the transfer of federates were tested. Both methods require that federates include methods for saving the current state of the simulation to a file and restoring from a previously saved state. This functionality is internal to each federate, but the DRMS currently dictates that the current state is saved to an ASCII file. First of all the save/restore services provided by HLA were considered. The state of a single federate of a federation could in this approach be saved by using a user supplied tag of the requestFederationSave call as the name of the federate to be migrated. This is similar to the method used in [2]. The second approach introduces two new interaction classes to the FOM to help manage the transfer of federates. A transfer using this approach is accomplished through the following principle:

- A resigning peer informs the Resource Manager of its intension to leave the DRMS network
- The Resource Manager uses the HLA Manager to send a “sleep” interaction, supplying the name of the concerned federate, to the federation
- The concerned federate saves its current state to a file and resigns from the federation
- The federate is started in a new host environment (peer) and initiates using the saved state

- When initialisation of the federate is complete it sends an “awake” interaction to the HLA Manager, which continues the advancement of the simulation

Using either of these approaches it is easy to commit a transfer of a federate between two nodes since the simulation time is always shared by all federates. When the HLA Manager is notified of an initiated transfer it simply stops advancing time, which will temporarily “freeze” the complete federation until the HLA Manager starts advancing the time once again.

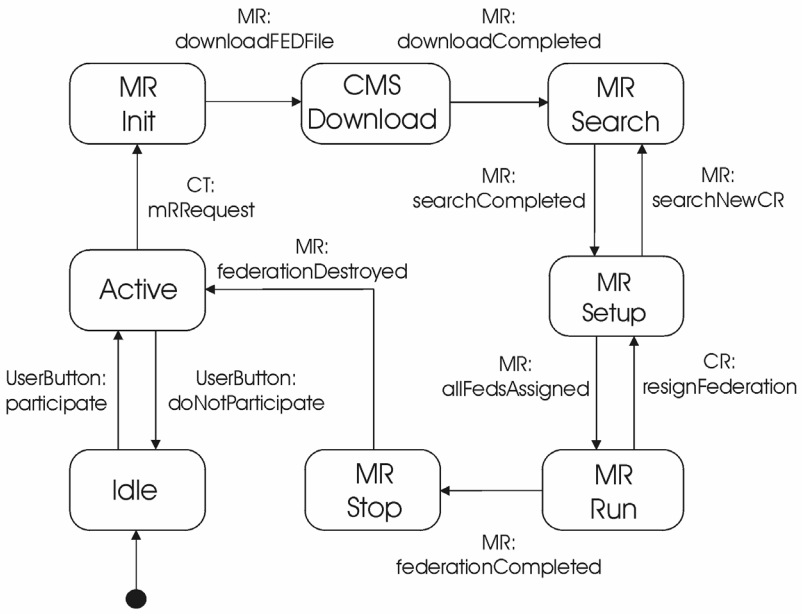

Figure 3. DRMS-client acting as an MR (Manager) during federation execution

4.5 Federation Execution

This section gives an in-depth description of the elements of a federation execution within the DRMS. The example is based on the special interaction classes introduced (sleep and awake) for managing federate migration, as described in section 4.4. The federation execution is explained in terms of two state charts depicting the successive states of a DRMS-client (JXTA peer) acting as an MR and CR respectively. The scenario presupposes the existence of a Composition Tool (CT), representing the graphical user interface in which a federation is composed by a user. Moreover, it is assumed that the CT has uploaded a valid FED-file for the federation that will be executed within the DRMS environment, to the distributed repository (CMS). The CT is in possession of a JXTA peer acting as a bridge between the DRMS environment and the CT. However, this peer is not assigned any tasks related to the execution of the composed federation, since the goal is to make the load of the CT client computer as small as possible.

Figure 3 describes the states of a DRMS-client acting as an MR during a federation execution. When a DRMS-client is started on a computer it is by default in the Idle state. This means that it is inactive and does not share any resource of its host environment. If the owner of the computer chooses to share the resources of the machine, by changing a switch in the graphical user interface, a participate event is

triggered and the DRMS-client enters the Active state. While in this state, the DRMS-client re-enters the Idle state if the owner of the computer triggers a doNotParticipate event.

When the composition of a federation has been completed in the CT, its JXTA peer searches for available DRMS-clients and selects one of them to perform the MR task. A DRMS-client enters the MR Init state when it is reached by an mRRequest from the CT. The selected DRMS-client, referred to as MR, then enters the CMS Download state where the FED-file required to run the composed federation is downloaded from the CMS. When the download has been completed the MR enters the MR Search state where one of the discovery mechanisms, described in section 4.3, is used to locate other available DRMS-clients. After the completion of the search process the MR enters the MR Setup state. In this state, those federates that are part of the composed federation are assigned to the identified DRMS-clients. Moreover, the MR starts an HLA Manager and through its interface, creates and joins a federation using the downloaded FED-file. When all federates have been assigned to host-environments, the MR sends an execute request to the concerned CRs and then enters the MR Run state. In this state the federation is executed under supervision of the MR, through its HLA Manager.

If one of the CRs makes a request for the release of its shared resources, the DRMS-client is notified by the concerned CR through a resignFederation request. The MR then re-enters the MR Setup state and tries to assign the concerned federate to other available DRMS-client. If needed, the MR re-enters the MR Search state to search for newly connected DRMS-clients. The re-configuration of the federation also requires that the MR sends a sleep interaction to the concerned federate by using the HLA Manager. This interaction instructs the federate to save its state and then to resign from the federation execution. After resigning from the federation, the MR instructs the resigning CR to upload the produced status file to the CMS. When the federate has been initiated in a new host environment using the previously saved state, fetched from the CMS, it sends an awake interaction to the MR. This tells the MR that all federates are assigned a host environment and connected to the federation, which transfers the MR to the MR Run state once again. When the end of the simulation is reached the MR enters the MR Stop state. In this state the MR resigns from the federation execution and then destroys the federation. After successful destruction of the federation the MR enters the Active state and thus becomes an ordinary DRMS-client, ready to serve the role of MR or CR.

Figure 4 describes the states of a DRMS-client acting as CR during a federation execution. As mentioned above, the DRMS-client is by default in the Idle state when started and will upon request from the owner of the computer, where the client resides, share the resources of the machine and enter the Active state. When an MR has located available DRMS-clients as described above, it sends a cRRequest to the concerned DRMS-clients. If a DRMS-client accepts this request it will share the computing capacity of its host environment for execution of the federates, thus entering the CR Init state. The CR starts downloading files required to execute the federates that it has been assigned when reached by a downloadFederate request from the MR. When all CRs have completed downloading the required federates the MR

triggers an execute request which transfers a CR to the CR Execute state. Upon successful completion of the federation execution the CR is released from its duties by a makeAvailable request from the MR and thereby re-enters the Active state.

However, if the owner of the computer where the CR resides chooses to withdraw its resources during the federation execution, the doNotParticipate event is triggered and the CR enters the CR Leave state. In this state the CR notifies the MR of its intention to stop sharing its resources by sending a resignFederation request to the MR. The MR responds by sending back an uploadStatusFile request thereby bringing the CR to the CMS Upload state. This request is triggered when the MR is informed by its HLA Manager that the concerned federate has resigned from the federation. During the CMS Upload state the CR uploads the status file produced by the federate upon receiving the sleep interaction and then notifies the MR when the operation is completed. Finally the CR enters the Idle state when the MR triggers a leaveP2PNet request.

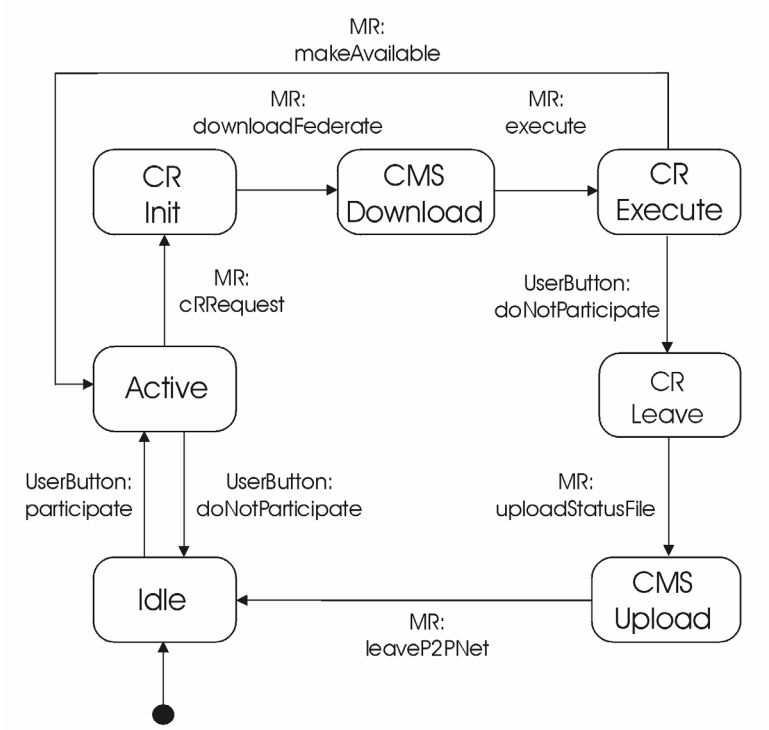

Figure 4. DRMS-client acting as a CR during federation execution.

5. Results & Discussion

There are two fundamental areas of use for peer-to-peer based M&S in respect to reusability and availability. First, the technology could be utilized for sharing of simulation components and composing simulations, thereby facilitating reuse and availability at the simplest level. Distributed storage of content, which the peer-to-peer paradigm promotes, is advantageous from many aspects. The responsibility of managing the repository of simulation components does not fall on a particular part of an organisation, as would be the effect of a centralized repository. Those interested in participating and using the M&S related resources are collectively responsible for managing the repository, for instance by requiring that individual contributors manage

their own content (simulation models). Moreover, certain constraints related to storage and use of an individual simulation model could also be satisfied. This could be beneficial when required that a simulation model is stored and executed within a dedicated part of an organisation's network or within the local network of an external organisation. In this case only the description of a simulation components and its interface is publicly available. If carefully designed, a distributed repository could also be more resilient to failures. Since the repository has no single point of failure, a basic level of service can always be upheld. However, a drawback of using this model for storage of simulation components is the difficulty of assuring the quality of models being uploaded. From a Verification, Validation & Accreditation (VV&A) perspective, a user may not have confident in the models developed by others. Also, the documentation of individual simulation models needs strict rules and regulations, for instances by constructing descriptions from templates derived from an M&S-related ontology, as will be discussed in section 5.2.

To increase the potential for reuse further and to make simulation models available to a larger audience within the defence M&S community, sharing of computing capacity is of great interest. The basic idea is to execute the simulation components, which are parts of a simulation, where computing capacity is available. This will provide users within an M&S community access to computer power otherwise not available for them, thereby enabling execution of computing intensive simulations initiated by simple lightweight clients. As stated in [3] there is no built in support for automatic setup of distributed simulations in the HLA standard and its implementations, which is required to implement dynamic utilization of computing capacity within a network. To provide the capability to initiate and configure a federate remotely an external infrastructure is needed, which the DRMS is an example of. In the first prototypical implementation of the DRMS only federates that are of the type "computable objects" are supported. This means that the federates have no graphical user interface and do not require any intervention by an operator at run-time. However, this is inevitable since a federate is executed at an arbitrary node within the network that is not known in advance. Moreover, a federate might be migrated between host environments during federation execution. The DRMS supports pre-run-time set-up of federates by providing configuration files that are downloaded together with the federates from the CMS. Future plans for further development of the DRMS includes additional types of federates, for instance federates that are bound to a specific machine.

5.1 HLA

The level of adjustment for federate developers in order to conform to the requirements of the DRMS is fairly light at present. The individual federates must include methods for capturing the current state of the simulation and also for restoring from a previously saved state. In the current implementation this means that a federate produces a simple ASCII file that can be used by all federates of that particular type when restoring. To enable the transfer of federates during federation execution, an HLA-based notification mechanism is needed. Using the save/restore services of the HLA proved sufficient, but caused too much overhead in terms of messages being generated. Due to the fact that all federates are time stepped (a limitation in itself) and thereby easily controlled by the manager, it is feasible to temporarily stop the federation execution and through the special interaction classes introduced, save and

restore a federate's state. This is a fairly light requirement introducing a simple extension of the FOM and save/restore features of a federate's internal state. However, this requirement is also coupled with a limitation of the current system, the lack of support for different time-management schemes

At present the DRMS only supports federates that are time constrained and time regulating, which makes it easy to implement federate migration during federation execution. Using this approach for time management means that the simulation time is shared by all federates at all times, i.e the logical time of the individual federates are always synchronized. However, it is not satisfying to put the requirement of conservative time management on each federate that are accessible from the NetSim environment. Therefore, research is being conducted to include support for optimistic time management schemes within the DRMS. The desired functionality is similar to the approaches described in [21] and [22] where a middle layer is introduced to enable check-pointing and rollback management in HLA-based simulations. Using this kind of capability for federation managed within a DRMS context, would allow bringing optimistic federates to a shared simulation time before committing a transfer, thus relaxing the requirement for execution of only time constrained and time regulating federates within the DRMS. Check-pointing and rollback management are crucial for enabling migration of federates in a dynamic network environment if optimistic time synchronisation is used. Furthermore, these facilities could be utilized for introducing fault-tolerance mechanisms in the architecture.

5.2 JXTA

Using the services provided by the CMS for management of content, in our case simulation components and documentation, revealed some limitations. The possibility to provide rich descriptions of resources is limited, thereby inhibiting efficient search for simulation components and composition of simulations within the NetSim environment. In spite of these shortcomings, CMS provided the necessary functions to fulfil the requirements on DRMS at this stage, and above all, minimizes the coding efforts. Furthermore, the functionality of CMS is expected to improve with time and moreover, the source code is publicly available offering the possibility of further refinement.

The current problem relates to the simple features of the CMS where tagging of content is limited to simple keyword-based mapping. In the future, the services related to the distributed model repository within JXTA, should maintain descriptions of resources that are based on shared concepts. Problems will certainly arise giving the individual developers the power to meta-describe their simulation components in any way possible. To fully exploit the potential in a system like NetSim, the tagging of resources must follow strict rules through shared semantics and syntax. The vocabulary used when describing resources should be formalized in an ontology, whose knowledge is accessible by all parts of the NetSim environment. Introducing ontology-based descriptions of simulation components would facilitate search for simulation components and matching of the components when composing a simulation. Furthermore, it could make the distribution of simulation components to available host environments more efficient. This could be accomplished by matching descriptions of the execution environment, required by each component, with

descriptions of the execution environment provided by each host. Work has been carried out to support ontology-based resource description within JXTA, see for example [23] where the Resource Description Framework (RDF) is integrated with JXTA or [24] where the Darpa Agent Mark-up Language (DAML) is used as a foundation for describing and reasoning about resources. We believe that an ontology-based approach for describing resources, within an environment such as the DRMS, would make the processes of searching, selecting and matching simulation components as well as executing of the simulations more efficient.

Our basic empirical work, regarding the three implemented basic discovery approaches in peer-to-peer systems, shows the most essential differences between them and guides the future developer to some extent in choosing one. Given the conclusions made in [20] and the general guidelines concerning centralized and decentralized systems reported in [25], the following features characterize the resource discovery mechanisms in JXTA. A centralized approach, such as the indirect discovery mechanism, has limited scale, but that limit is easy to understand and measure. In contrast to this the direct discovery and no discovery approaches offer the opportunity for massive scalability, a phenomenon that in practice can be hard to achieve. This due to the amount of traffic generated to keep the system up-to-date (no discovery) and the traffic needed to perform a search for resources (direct discovery).

In the indirect discovery approach, which can be seen as a centralized way of finding resources, we have a theoretical weak point in relation to scalability. However, in practice it has been shown that even a somewhat modest computer running a Web server can easily handle hundreds of thousands of visitors a day. Computers are fast and, in practice, they often meet predefined requirements; but one cannot dismiss the fact that scale is clearly limited by the capacity of the centralized resource. Unlike other complex topologies, scalability is easy to measure in this case and more capacity can be added to the centralized resource as the system grows. When it comes to the direct discovery and no discovery approaches we consider a decentralized system. In these systems the scalability is much harder to evaluate. In theory, the system should become more capable as more peers are added. In reality, the addition of peers increases the amount of overhead communication needed to keep the system coherent at a fast rate [25]. When it comes to scalability in a decentralized system the answer is that a decentralized approach may be more scalable than a centralized one, depending on the routing algorithms used to keep the system up-to-date.

We believe that one has to find a more advanced solution to the resource discovery problem if the system grows too much in number of connected resources. This problem has been addressed by the release of the new platform (JXTA 2.0), which extends the in-direct (rendezvous) discovery concept.

6. Conclusions

The Distributed Resource Management System (DRMS) utilize storage capacity and processing capability in a network of workstations to manage and execute simulations based on the High Level Architecture (HLA). The overall goal of this system is to promote reuse and availability of simulation components (federates) within the defence M&S community. The DRMS is based on the JXTA peer-to-peer platform,

which represent an attempt to standardize the peer-to-peer domain. The use of idle processing capacity in a network exposes some requirements on simulation components that are part of the environment. Due to the dynamic characteristics of the underlying peer-to-peer network, mechanisms for migrating simulation components between host environments during simulation execution are needed. In reality this calls for methods to capture the internal state of an individual simulation component before it is transferred to a new host environment. Further more, a simulation component should include support for restoring from a previously saved state following a completed transfer. Finally, simulation components are required to declare interest for two additional interactions, introduced to notify the simulation of an imminent or completed migration. The use of JXTA for managing resources, idle processing capacity and storage, proved advantageous in general. However, a limitation of the current JXTA platform is lack of support for extensive meta-descriptions of content. This is required to enable efficient localisation, selection and composition of simulation components. Hence, this part of JXTA needs to be improved further to suit our particular needs.

7. References

[1] F. Kuhl, R. Weatherly, J. Dahman, Creating Computer Simulation Systems – An Introduction to the High Level Architecture, Prentice Hall, Upper Sadle River, USA, 1999.

[2] J. Lüthi, S. Großmann, The Resource Sharing System: Dynamic Federate Mapping for HLA-based Distributed Simulation, Proceedings: The Fifteenth Workshop on Parallel and Distributed Simulation, PADS 2001, Lake Arrowhead, USA, 2001.

[3] W. Cai, S. Turner, H. Zhao, A Load Management System for Running HLA-based Distributed Simulations over the Grid, Proceedings of the IEEE International Symposium on Distributed Simulation and Real Time Applications, 2002.

[4] Open Source Community, Globus, http://www.globus.org/, 2004.

[5] L. Bononi, G. D'Angelo, L. Donatiello, HLA-based Adaptive Distributed Simulation of Wireless Mobile Systems, Proceedings of the 17th Workshop on Parallel and Distributed Simulation (PADS), San Diego, USA, 2003.

[6] M. Myjak, S. Sharp, W. Shu, W. Wei, J. Riehl, D. Berkley, P. Nguyen, S Camplin, M. Roche, Implementing object transfer in HLA, Proceedings of Simulation Interoperability Workshop, Orlando, USA, 1999.

[7] Open Source Community, JXTA, http://www.jxta.org/, 2004.

[8] Open Source Community, Jini, http://www.jini.org, 2004.

[9] Gnutella Community, http://www.gnutella.com, 2004.

[10] D. Brookshier, D. Govoni, N. Krishnan, JXTA: Java P2P Programming, SAMS, Indianapolis, USA, 2002.

[11] S. Oaks, B. Traversat, L. Gong, JXTA in a Nutshell, O´Reilly & Associates, Inc., 2002.

[12] B.J. Wilson, JXTA, New Riders Publishing, http://www.brendonwilson.com/projects/jxta/, 2004.

[13] J. Dahman, The Departement of Defence High Level Architecture, Proceedings of the 1997 Winter Simulation Conference, 1997.

[14] IEEE Computer Society, IEEE Standard for Modelling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification, IEEE, inc., New York, USA, 2001.

[15] Pitch, provider of pRTI, http://www.pitch.se, 2004.

[16] M. Eklof, J. Ulriksson, F. Moradi, NetSim – An Environment for Network Based Modelling and Simulation, Symposium on C3I and M&S Interoperability (MSG-22), NATO Modeling and Simulation Group, 2003.

[17] J. Ulriksson, F. Moradi, R. Ayani, Collaborative Modelling and Simulation in a Distributed Environment, Proceedings of European Simulation Interoperability Workshop, Stockholm, 2003.

[18] The Search for Extraterrestrial Intelligence, http://setiathome.ssl.berkeley.edu/, 2004.

[19] D. De Roure, N. Jennings, N. Shadbolt, Research Agenda for the Semantic Grid: A Future e-Science Infrastructure, Report commissioned for EPSRC/DTI Core e-Science Programme, http://www.semanticgrid.org/v1.9/semgrid.pdf, 2001.

[20] M. Sparf, A Comparison of Peer-to-Peer Resource Discovery Strategies in Distributed Simulation, MSc. Thesis, Dept. of Computer and Systems Sciences, Stockholm University, No. 03-86 DSV-SU/KTH, 2003.

[21] C. Maziero, F. Vardanega, Generic Rollback Manager for Optimistic HLA Simulations, Transactions of the Society for Computer Simulation, v.18, n.2, p.56 – 61, San Diego, USA, 2001.

[22] J. Huang, M. Tung, K. Wang, L. Hui, M. Lee, J. Wu, S. Wai, Smart Time Management – The Unified Time Management Mechanism, Proceeding of the European Simulation Interoperability Workshop, Stockholm, 2003.

[23] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, T. Risch, EDUTELLA: A P2P Networking Infrastructure Based on RDF, Proceedings of the Eleventh International World Wide Web Conference, Honolulu, USA, 2002.

[24] D. Elenius, Service Discovery in Peer-to-Peer Networks, MSc. Thesis, Dept. of Computer and Information Science, Linköping Institute of Technology, LiTH-IDA-EX--03/060--SE, 2003.

[25] N. Minar, Distributed Systems Topologies: Part2, The O'Reilly Network, http://www.openp2p.com/pub/a/p2p/2002/01/08/p2p_topologies_pt2.html, 2002.

III. A Framework for Fault-Tolerance in HLA-Based Distributed Simulations

M. Eklöf, F. Moradi & R. Ayani
Proceedings of the 2005 Winter Simulation Conference (WinterSim)
Orlando, Florida, 2005.

Abstract

The widespread use of simulation in future military systems depends, among others, on the degree of reuse and availability of simulation models. Simulation support in such systems must also cope with failure in software or hardware. Research in fault-tolerant distributed simulation, especially in the context of the High Level Architecture (HLA), has been quite sparse. Nor does the HLA standard itself cover fault-tolerance extensively. This paper describes a framework, named Distributed Resource Management System (DRMS), for robust execution of federations. The implementation of the framework is based on Web Services and Semantic Web technology, and provides fundamental services and a consistent mechanism for description of resources managed by the environment. To evaluate the proposed framework, a federation has been developed that utilizes time-warp mechanism for synchronization. In this paper, we describe our approach to fault tolerance and give an example to illustrate how DRMS behaves when it faces faulty federates.

1. Introduction

Simulation models are increasingly being used as integral parts of modern military command and control and decision support systems. The nature of many of today's simulation models, in terms of processing capacity required for execution or decomposition to promote reuse and/or connection of geographically dispersed units, shows the importance of methodology for distributed simulation. In this context the High Level Architecture (HLA) is a widely used standard for distributed simulations. In HLA, a simulation is referred to as federation, whereas an individual simulation component is referred to as a federate. The decomposition of a simulation system certainly has its merits, but will typically lead to a higher failure rate (Kiesling 2003). In the perspective of a military decision support system the failure of a critical simulation component is often unacceptable, especially when time is a constraining factor. Thus, support for fault-tolerant distributed simulation is crucial in such systems. Thus, we need some mechanisms for detecting errors in the simulation execution, as well as measures for restoring an erroneous federation execution.

The HLA provides basic functionality for restoring an unsuccessful simulation execution, through the save and restore features of the federation management services. However, no means of detecting an error, or automatically restoring an

erroneous simulation execution, are defined by the HLA. Also, the save and restore facilities are used in the local scope of a federate, meaning that a saved state is not automatically distributed outside the node where the federate resides. To cope with an unreliable host environment of a federate, in terms of hardware crashes or lost network connections, it is necessary to enable state saving at a "global" level and resumption of a federate execution in a new host environment. These functions are the fundament for what is usually referred to as federate migration, i.e. the transfer of federates between different host environments.

In our previous work we explored the possibilities for migration of federates using the peer-to-peer-based Distributed Resource Management System (Eklöf, Sparf, and Moradi 2004). However, in our previous work the decision to migrate a federate was based on the willingness of workstation owners to share their computing capacity for simulation execution. Thus, the system did not consider detection of a failure and migrations of federates were based upon user requests.

In this paper we present a revised architecture of the DRMS and outline a partial implementation of the concept, based on Web Services. More specifically, this paper will address a mechanism for fault-tolerance in time-warp based federations. The fault-tolerance mechanism does not consider software errors, in terms of a simulation model producing erroneous result, but handles cases where the host environment of a federate crashes, the federate itself crashes for some reason or the federate's link to the RTI is lost. Moreover, we assume that the federates executed within the scope of DRMS are transferable, meaning that they are not bound to a specific piece of hardware and can easily be migrated between different host environments.

2. Background

As computers in a distributed simulation do not share a common clock it is required that a virtual time, usually referred to as logical time, is introduced for each member of the simulation. A time synchronization protocol is used to maintain the logical time of members and ensures the causal ordering of events.

2.1 Optimistic Synchronization in HLA

The time-warp approach to synchronization, proposed by Jefferson (1985), is the most well known optimistic synchronization protocol. In the time-warp protocol, logical processes (LPs) are allowed to process events optimistically, which means that events may arrive that have a smaller timestamp than previously processed events. This implies that LPs are also permitted to send messages optimistically, which means that sent messages could later be cancelled. The cancellation is performed by sending anti-messages to the receivers of the original events. An inevitable aspect of the time-warp protocol is the ability of an individual LP to restore to a previous state in its past. This process is referred to as rollback. Rollback is triggered if an LP receives a message in its past, or if a processed event is annihilated by an anti-message.

The time-warp protocol has also been utilized in HLA-based distributed simulation. The bulk of research in this area addresses development of middleware that will shield the developer of a federation from the often complex task of implementing time-warp. In (Wang et al. 2004) the issue of time-warp is investigated in the context of

integrating COTS simulation packages and the HLA. A middleware for management of the rollback mechanism is presented and evaluated. In (Yan, Sun, and Zhong 2003) a time management meta-level is introduced between the RTI and individual federates. This layer uses a computational reflection technique to free the developer of an optimistic federate from the complex task of implementing the roll-back mechanism. Huang et al. (2003) describes the addition of a middle layer, referred to as the Smart Time Management (STM), which aims at unifying various time management schemes, such as time-stepped, event-driven and optimistic time advancement approaches in the HLA. In (Vardanega and Maziero 2001) a generic rollback manager for optimistic HLA simulations, based on computational reflection techniques, is presented. The manager implements state saving and manages rollback for optimistic federates.

2.2 Fault-Tolerance in HLA

A distributed simulation, or distributed system for that matter, has a higher failure rate than a simulation, or system, executed on a single machine. However, a failure in a distributed simulation is often partial, that is, one of the components of the system fails. The failure may, or may not, affect other components of the system. In the past, several techniques for fault-tolerance in distributed systems have been developed. These techniques can be classified into two main categories; replication-based and check-pointing-based (Damani and Garg 1998). In replication-based approaches one or more copies of an LP is maintained in addition to the main LP. In case of failure, one of these replicas will take the failed LP's place. In check-pointing-based approaches, states of the individual LPs are saved on stable storage. In case of failure, an LP is restarted using the last stable state saved on stable storage.

According to Kiesling (2003) research in fault-tolerant distributed simulation has been quite sparse. Work in application of fault-tolerance techniques in the context of HLA is even scarcer. However, there is some work that aims in this direction. Lüthi and Berchtold (2000) provide a structured view of fault-tolerance in parallel and distributed simulations and possible solutions are presented. In (Lüthi and Großmann 2001) a Resource Sharing System (RSS) is presented that in a future extension could serve as the basis for fault-detection, check-pointing and replication of federates. In (Berchtold and Hezel 2001) a concept, named R-FED (Replica Federate), in support of fault-tolerant HLA federations is presented. As the name implies, the approach is based on replication of individual federates in a federation. Several papers address the issue of federate migration, which is an important cornerstone in designing an infrastructure for fault-tolerant distributed simulation, see for example (Eklöf, Sparf, and Moradi 2004; Tan, Persson, and Ayani 2004; Bononi, D'Angelo, and Donatiello 2003; Cai, Turner, and Zhao 2002; Lüthi and Großmann 2001). However, these papers usually address federate migration in the context of load-balancing and do not explicitly address fault-tolerance.

The present version of HLA is IEEE 1516-2000. Currently, work is carried out to define the next version of HLA, through the HLA Evolved (Möller, Karlsson, and Löfstrand 2005). By the end of 2005, or early 2006, this work is expected to be complete. An interesting aspect of HLA Evolved is that fault-tolerance has been given more focus than before. HLA Evolved will provide a common semantics for failure

and mechanisms for fault-detection. At the core, two additions have been made to the Management Object Model (MOM), namely federate lost and disconnected. These interactions provide the basic mechanisms for signaling a fault from the context of a federation, through federate lost, and from the perspective of a federate, through disconnected. Upon failure, the RTI has the responsibility to do resign on behalf of the lost federate using the Automatic Resign Directive. This line of development is important for future realization of fault-tolerant distributed simulations, based on the HLA.

3. Distributed Resource Management System – DRMS

In the following section the DRMS is presented in the context of network-based M&S. Next, the mechanism for fault-tolerance implemented in DRMS, to support robust execution of time-warp based federations, is described.

3.1 Network-Based M&S

The DRMS provides computing capacity for reliable execution of simulations and is an essential part of a network-based modeling and simulation environment, referred to as NetSim, being developed at the Swedish Defense Research Agency (Eklöf, Ulriksson, and Moradi 2003). NetSim supports collaborative simulation development and execution within and between organizations and will thus promote increased use and reuse of simulation models and also lead to increased quality of work in the M&S development process. Figure 1 presents an overview of the service-oriented architecture of NetSim. The uppermost layer comprises various NetSim tools, dedicated to M&S-related tasks, for instance tools for composition of federations by a single user, or collaborative development of federations by a number of users. The NetSim tools derive their functionality from NetSim specific services, denoted DRMS, CC and Repository in Figure 1. As stated above the DRMS provides computing capacity for reliable execution of simulations. The CC (Collaboration Core) provides services for collaborative work, whereas the Repository provides services for look-up of available resources on a network. The NetSim specific services are based on various overlay network service technologies, such as Web Services, Grid Services and the HLA RTI. These are just examples of network technologies that could be deployed to achieve the goals of the NetSim environment. Throughout all layers in Figure 1, a common syntax and semantics for description of resources is used to promote interoperability. Moreover, security is considered an integral part of all layers.

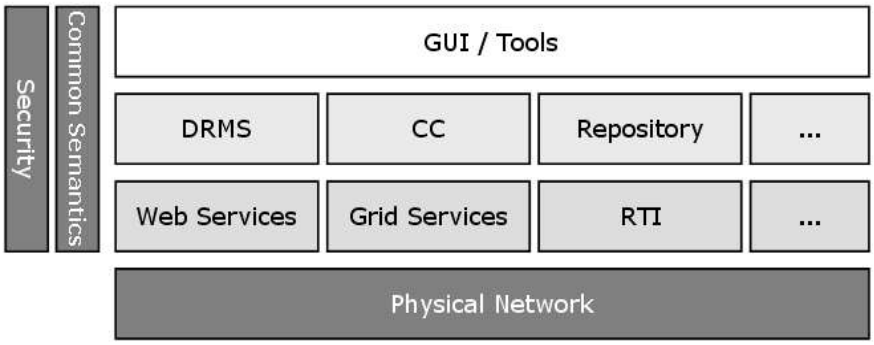

Figure 1: Architecture of NetSim.

3.2 DRMS Concept

DRMS comprises two basic service types, namely a worker service and a coordinator service. A worker is responsible for execution of one or more jobs, whereas a coordinator is responsible for the coordination of one or more workers in managing a batch of jobs. In addition to these basic services, the DRMS is dependant on a repository service. A repository is used by a worker to advertise its presence on the network and also its availability for execution of various jobs. Furthermore, the repository is used by a coordinator for localization of available workers. A repository also contains advertisements of other resources available on the network and is therefore used as entry point when worker services fetch resource files and executable code.

3.3 Fault-Tolerance Approach

The main idea of our approach to fault-tolerance in time-warp based federations is to use a check-pointing mechanism to enable restoration of a federation upon failure. The check-pointing is done by means of the RTI communication infrastructure, utilizing an extension to the Federation Object Model (FOM). In this context, a checkpoint (CP) represents the state of a federate at a specific point in time, for example through a vector of state variables. The check-points are saved in a stable storage component, which is also a member of the federation execution. An important feature of the check-pointing is to make sure that the individual state represents a federate at a point in time that could not suffer from rollback. This means that the federate must report checkpoints to stable storage, which represent the state of the federate at a point in time that is less than the smallest timestamp of a message that could ever be delivered to the federate. In this way, it will always be safe to use the check-point for restoration. The state-saving is not synchronized throughout the federation, but federates report their states to stable storage individually. The mechanism for check-pointing is illustrated in Figure 2.

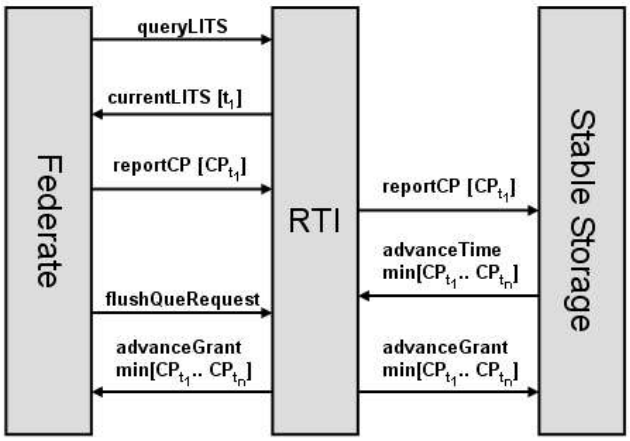

Figure 2: Check-Pointing Mechanism in the DRMS.

First, the concerned federate uses the queryLITS (Least Incoming Time-Stamp) method of its RTI ambassador to extract the timestamp of the next TSO (Time-Stamp Order) message that it may have to process. The federate uses this value to produce a checkpoint that could not be invalidated in the future. The checkpoint can not be can-

celled since the federate can never be roll-backed prior to this time and thus, it represents a safe state of the federate. The LITS is used as timestamp when reporting the check-point to the stable storage. When the stable storage receives a checkpoint, it calculates the minimum timestamp of the checkpoints that have most recently been reported from each federate in the federation. This minimum time-stamp is then used by the stable storage for requesting advancement of time. Upon requesting flush of the RTI queues, the individual federates will receive time advancement grants based on the timestamps of the supplied checkpoints. The granted time represents the GVT (Global Virtual Time) of the federation. Note that this time does not represent the actual (local) time of a federate. GVT is the boundary up to which the simulation execution is regarded as complete by all participants and is used to perform garbage collection of saved states to free memory space.

The purpose of allowing the stable storage to control advancement of GVT is crucial for migration purposes. During the migration of a federate the GVT must not be advanced beyond the time-stamp of the checkpoint that the migrating federate will rely upon for its restoration. The mechanism described above will make sure that this will not occur.

3.4 Migration of Federates

Below, the process of migrating a federate upon failure is described. When an individual federate is deployed in a new host environment, the startup scheme differs slightly from the normal case. This process is illustrated in Figure 3.

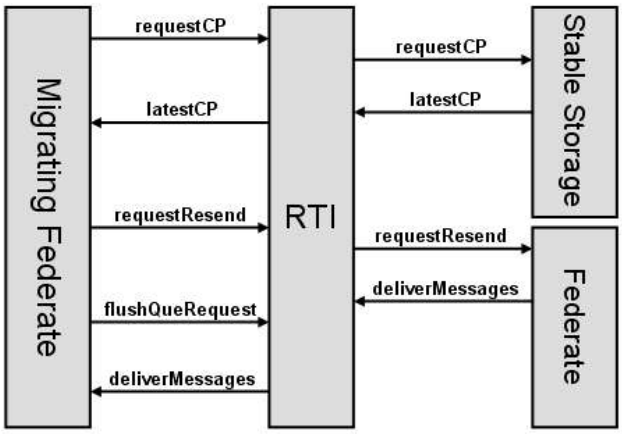

Figure 3: Federate Migration Process.

Initially, the federate requests the most up-to-date checkpoint of its state from stable storage. The federate restores its state based on this checkpoint. Then the federate makes a request to all participants to resend all messages whose timestamp is greater than GVT. Federates that have produced messages to the concerned federate, resend these messages. When the migrated federate requests flush of the RTI queues, it will receive the missing messages and can then resume the execution.

The described mechanism requires additional customization of participating federates and introduction of four supplementary interactions to the FOM, namely reportCP,

requestCP, latestCP and requestResend. Extra interactions required by the fault-tolerance mechanism are outlined in Table 1.

Table 1: Interactions Added to the FOM to Support the Fault-Tolerance Mechanism (P = Publish, S = Subscribe).

Interaction	Description	Federate	Stable Storage
reportCP	Reports checkpoint to Stable Storage	P	S
requestCP	Requests latest checkpoint from stable storage	P	S
latestCP	Delivers latest checkpoint to migrated federate	S	P
requestResend	Requests resend of messages from GVT	P, S	-

3.5 Services and Ontology

The DRMS concept presented in section 3.2 has been partially implemented. The implementation is based on Web Services, the Axis platform (Saleem 2004), and Semantic Web technology, through use of the Jena toolkit (McBride 2002). In the following section the implementation is described briefly. The following components of the implementation are described:

- DRMS ontology
- RemoteJobService
- ResourceRepositoryService
- ExecutionService.

To enable uniform and semantically rich descriptions of resources within the environment, a DRMS ontology is used. In near future, this ontology will be aligned with a general NetSim ontology that is currently under development. The DRMS ontology comprises constructs for description of simulation models and computing resources. The main purpose of the ontology is to promote a shared view of information throughout the environment and facilitate localization and matching of resources. The chosen language for its representation is the Web Ontology Language (OWL) (McGuniess and van Harmelen 2004). The expressiveness of OWL is sufficient for representation of information required by the DRMS. The language also enables inference over information, which is used to match resources in the implementation.

The RemoteJobService implements a worker as described in section 3.2. When deployed on a workstation, this service announces its presence on the network by registering an announcement in a repository. The announcement is represented by a meta-model, defining the features of the RemoteJobService's host environment. This includes aspects such as the workstation's hardware configuration, OS type and version etc. The meta-model is an instance based on the DRMS ontology. Table 2 outlines the service interface of the RemoteJobService.

The ResourceRepositoryService is a simple implementation of a repository as described in section 3.2. This service supports storage of meta-models, such as the meta-model describing the RemoteJobService's host environment. The interface of the ResourceRepositoryService includes methods for registering, deletion and lookup of meta-models. The lookup can either respond with the entire content of the ResourceRepositoryService, or a subset of registered meta-models, defined by a search query. Table 2 outlines the service interface of the ResourceRepository-Service.

Table 2: Service Interfaces of the DRMS Implementation.

Service	Method
RemoteJobService	allocateJob(Meta-model)
	startJob(Id)
	stopJob(Id)
	getJobStatus(Id)
ResourceRepositoryService	addModel(Meta-model)
	deleteModel(Meta-model)
	getModels()
	getSubset(Query)
ExecutionService	requestExecution(Scenario)
	finalizeExecution(Id)
	getScenarioStatus(Id)

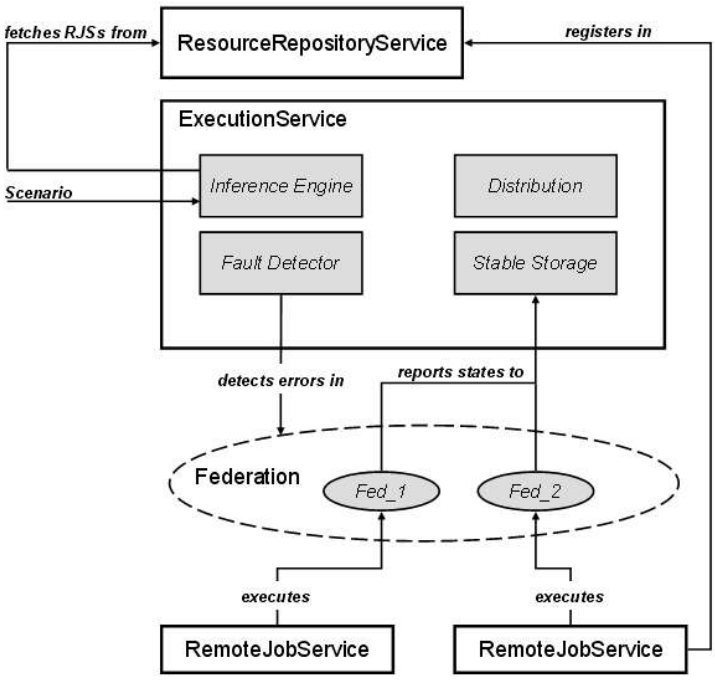

Figure 4: Interrelation of DRMS Services.

The ExecutionService is an implementation of a coordinator as described in section 3.2. An ExecutionService is utilized by the NetSim environment when a single user, or group, requests execution of a scenario (federation). The main tasks of the

ExecutionService are to automatically setup a federation and to monitor the federation execution. Table 2 outlines the service interface of the ExecutionService. Figure 4 gives a schematic view of the interrelation of DRMS services.

When the NetSim environment requests execution of a federation, it feeds the ExecutionService with a scenario description. The scenario description comprises meta-models for all federates that are part of the federation. In order to distribute the federates in the federation, to suitable nodes in the network, the ExecutionService fetches meta-models, representing RemoteJobServices, from the ResourceRepositoryService. Next, the ExecutionService determines a suitable distribution of federates, by matching meta-models of the federates with meta-models of available RemoteJobServices. The matching procedure utilizes an inference engine and a set of pre-defined rules to find a suitable distribution of federates over available RemoteJobService nodes. If the allocation of one or more federates is accepted by a RemoteJobService, it starts downloading the required executable code and possible resource files. The URLs to these files are defined in the meta-models representing the federates in question. When the download process is completed, the ExecutionService signals start-up of the federation to concerned RemoteJobServices.

To enable fault-tolerant execution of the federation the ExecutionService comprises a stable storage and a fault detector component. These components are members of concerned federation through a common federate. The stable storage stores checkpoints reported from federates in the federation, whereas the fault detector detects failed federates in the federation and initiates preventive measures to resolve these errors. The error detector detects the failure of a federate by means of the HLAfederate object of the MOM, which is deleted if the link to the RTI is broken. As an additional measure the error detector calculates the time passed from the last reported checkpoint and if this value exceeds a pre-defined time, the federate is not longer considered active. When a federate crashes, or its network connection is simply lost, the fault-detector initiates redistribution of the lost component in the inference engine. The inference engine finds a new host environment for the federate under consideration, given the requirements of the federate as defined by its meta-model, and allocates the job to the RemoteJobService node.

3.6 An Example

Below, we look at an example to illustrate how the DRMS handles the occurrence of faults in a federation. In order to test the proposed fault-tolerance approach, a simple time-warp federation has been developed. This federation consists of four federates, which form a fully connected net-work, i.e. each federate is able to send messages to all other federates. In the test federation, processing of a message simply means updating a statistics object that describes the message exchange during a federation execution. Each federate randomly schedules events. This means that at random points in time, a federate sends a message to a randomly selected joined federate. The federates are initiated using disparate random seeds, causing the event scheduling to be based on different random streams within each federate. The federates process and produce events optimistically, thus when a message is received in a federate's past, a rollback is triggered. Similarly, when an anti-message is received that will annihilate an already processed message, a rollback is also triggered. The rollback mechanism

relies on a record of locally saved check-points. Advancement of GVT is used to garbage collect the record.

Consider a federation comprising four federates, labeled A, B, C and D. Table 3 describes the state of the federates, in terms of their message queues, as GVT equals 10. Grey cells represent messages that have been processed by the federates, whereas the white cells represent unprocessed messages.

The process of federate migration, in case of failure, resembles a rollback to GVT. However, in this case special attention is required since the action is not coordinated. In an ordinary rollback the concerned federates send anti-messages to annihilate invalid messages. In case of failure this is not possible and must be handled separately be each federate. For instance, consider the case when federate A in Table 3 crashes. When federate A is absent, federates B, C and D continue their execution. However, since no check-points are reported from federate A, the stable storage will not request time advancement greater than 10. When the fault detector has detected the lost federate, and a new host environment has been identified by the inference engine, the failed federate is deployed at a new node. Next, the migrated federate fetches the latest saved state in stable storage, as defined in section 3.4. When the federate has re-stored using the state from stable storage, it requests resend of messages. This request also means that the non-migrated federates must annihilate messages received from federate A. In this case federate C must annihilate message A15 and federate D message A17. This will trigger retraction of message C21 in federate B, but no rollback will be initiated since the message has not been processed yet.

When the potential message annihilation is finalized federate B, C and D resend messages destined for federate A, whose time-stamp is greater than GVT. In this case, given that the queue configurations do not change during migration, federate B resends message A12 and A14. Next, federate A reschedules the received events and the federation resumes normal execution.

Table 3: Message Queues in Federates of Test Federation when GVT Equals 10. Grey Cells Represent Processed Messages, whereas White Cells Represent Unprocessed Messages.

Federate	IN	OUT
A	B12	C15
	B14	D17
B	D9	A12
	C12	A14
	C21	-
C	D11	B12
	A15	B21
D	A8	C11
	A17	-

4. Discussion

As modeling and simulation is integrated in various environments and used as a tool in the decision process, the requirements on the supporting infrastructure will be high. An important aspect in this is to enable fault-tolerant distributed simulation, since this ensures a robust execution environment that can respond to user needs in a timely fashion. The de facto standard for distributed simulation, the HLA, which is widely used throughout the military domain, does not treat fault-tolerance extensively. Nor has this topic been treated by the research community comprehensively. Given this, it is crucial to develop efficient and scalable methods for fault-tolerance in HLA-based distributed simulation.

Our approach is based on implementing fault-tolerance mechanisms within the framework of the HLA, i.e. communication related to the fault-tolerance mechanism is sent over the RTI. This of course implies that individual federates conform to the requirements imposed by the fault-tolerance mechanism, in terms of publishing and subscribing to the interactions defined in Table 1. Currently, these aspects must be implemented by each federate individually. In the long run, it is desirable to implement these features in a generic fashion, through some kind of middleware system, to simplify the deployment of federates within the DRMS.

Introducing fault-tolerance mechanisms in M&S infrastructures will impose a cost. Regardless of the approach taken, replication-based or check-pointing-based fault tolerance, the infrastructure must cope with increased network traffic and consumption of more hardware resources. Thus, it is important to evaluate the cost of having fault-tolerant simulations to determine when the approach is beneficial. Furthermore, aspects of fault-tolerance should be considered in the early phases of the FEDEP process. For instance, it is important to determine what levels of fault-tolerance are required by different components of the simulation in the context of what degree of degradation is acceptable for a given target (Möller, Karlsson, and Löfstrand 2005).

The proposition for a next version of the HLA standard, the HLA Evolved, will simplify the process of developing fault-tolerant federations. In this standard, a common semantics for failure and mechanisms for fault detection are provided. Still, there are others issues to resolve as well. Given the failure of a critical component in a federation, whose original host environment is not accessible for its restoration, a mechanism for deployment of the component in a new host environment is required. This can be solved through replication of the component, or through utilization of check-pointing, at a global level. Our work shows that it is feasible to use a check-pointing-based scheme, employing the RTI communication infrastructure, to enable fault-tolerance in time-warp federations. However, it should be noted that this kind of check-pointing is tightly coupled with the time synchronization protocol of the federation. Other, or complementary, solutions have to be provided for other synchronizations protocols, or cases with mixed synchronization protocols. Also, the test federation used in this work comprises no complex issues of ownership of objects in the federation. In more complex federation types, the issue of transferring ownership of objects between federates, in case of failure, must be resolved as well.

5. Future work

In estimating the cost of fault-tolerant distributed simulation, based on our approach, it is of interest to look at the size of the check-points reported to stable storage and how this potentially will degrade the simulation execution. Also, the checkpoint interval used by each federate is of importance in this perspective.

The overall time consumption and number of messages sent executing the federation, with and without fault-tolerance, will be measured and compared. This will be done to identify when the cost of having fault-tolerance will inhibit the simulation execution rather than make it more efficient, given a specific failure-rate of the federates.

It should also be noted that the effectiveness of the fault-tolerance mechanism presented here is clearly coupled with the mutual relations existing between federates. During migration, federates that are joined to the federation can still progress their execution. When the migrated federate joins, after being deployed in a new host environment, chances are that is has lagged behind the others, and thus it may start to produce messages in their past. However, this depends on the nature of the migrated federate. A federate publishing data of concern to a large audience will of course affect the effectiveness of the execution more than a federate producing data of less interest. In our test federation the mutual relation between the federates is organized in a fully connected network, and messages send to and from federates are completely randomized. In the future it will be of interest to test other federation topologies to investigate how the relations between federates influence the performance of our fault-tolerant approach.

6. References

Berchtold, C., and M. Hezel. 2001. An architecture for fault-tolerant HLA-based simulation. In Proceedings of the 15th European Simulation Multiconference, 616-620. Prague, Czech Republic.

Bononi, L., G. D'Angelo, and L. Donatiello. 2003. HLA-based adaptive distributed simulation of wireless mo-bile systems. In Proceedings of the 17th Workshop on Parallel and Distributed Simulation, 40-49. San Diego, California.

Cai, W., S. Turner, and H. Zhao. 2002. A load management system for running HLA-based distributed simulations over the grid. In Proceedings of the 6th IEEE International Workshop on Distributed Simulation and Real-Time Applications, 7-14. Fort Worth, Texas.

Damani, O. P., and V. K. Garg. 1998. Fault-tolerant distributed simulation. In Proceedings of the 12th Work-shop on Parallel and Distributed Simulation, 38-45. Alberta, Canada.

Eklöf, M., M. Sparf, and F. Moradi. 2004. Peer-to-peer-based resource management in support of HLA-based simulations. Simulation 80: 181-190.

Eklöf, M., J. Ulriksson, and F. Moradi. 2003. NetSim: An environment for network based modeling and simulation. In Proceedings of the NATO RTO Symposium on C3I and M&S Interoperability. Antalya, Turkey.

Huang, J., M. Tung, K. Wang, L. Hui, M. Lee, J. Wu, and S. Wai. 2003. Smart time management: The unified time management mechanism. In Proceedings of the 2003 European Simulation Interoperability Workshop. Stockholm, Sweden.

Jefferson, D. 1985. Virtual time. ACM Transactions on Programming Languages and Systems 7: 404-425.

Kiesling, T. 2003. Fault-tolerant distributed simulation: A position paper [online]. Available via <http://fakinf.informatik.unibw-muenchen.de/~tkiesling/documents/ftds-position-paper.pdf> [accessed March 21, 2005].

Lüthi, J., and S. Großmann. 2001. The resource sharing system: Dynamic federate mapping for HLA-based distributed simulation. In Proceedings of the 15th Workshop on Parallel and Distributed Simulation, 91-98. Lake Arrowhead, California.

Lüthi, J., and C. Berchtold. 2000. Concepts for dependable distributed discrete event simulation. In Proceedings of the 14th International European Simulation Multi-Conference, 59-66. Ghent, Belgium.

McBride, B. 2002. Jena: A semantic web toolkit. IEEE Internet Computing 6: 55-59.

McGuniess, D. L., and F. van Harmelen. 2004. OWL web ontology language overview [online]. Available via <http://www.w3.org/TR/owl-features/> [accessed March 21, 2005].

Möller, B., M. Karlsson, and B. Löfstrand. 2005. Developing fault tolerant federations using HLA evolved. In Proceedings of the 2005 Spring Simulation Interoperability Workshop. San Diego, California.

Saleem, U. 2004. Developing java web services with AXIS [online]. Available via <http://www.developer.com/java/web/article.php/3443951> [accessed March 21, 2005].

Tan, G., A. Persson, and R. Ayani. 2004. HLA federate migration. In Proceedings of the 38th Annual Simulation Symposium, 243-250. San Diego, California.

Vardanega, F., and C. Maziero. 2001. A generic rollback manager for optimistic HLA simulations. In Proceedings of the 4th IEEE International Workshop on Distributed Simulation and Real-Time Applications, 79-85. San Francisco, California.

Wang, X., S. J. Turner, M. Y. H. Low, and B. P. Gan. 2004. Optimistic synchronization in HLA based distributed simulation. In Proceedings of the 18th Work-shop on Parallel and Distributed Simulation, 123-130. Kufstein, Austria.

Yan, H., Y. Zhang, G. Sun, and L. Zhong. 2003. Research on time warp mechanism in HLA. In Proceedings of the 2nd International Conference on Machine Learning and Cybernetics, 1092-1095, Xi-an, China.

IV. Evaluation of a Fault-Tolerance Mechanism for HLA-Based Distributed Simulations

M. Eklöf, F. Moradi & R. Ayani.
Proceedings of the 20th Workshop on Parallel and Distributed Simulations (PADS), Singapore, 2006.

Abstract

Successful integration of Modeling and Simulation (M&S) in the future Network-Based Defence (NBD) depends, among other things, on providing Fault-Tolerant (FT) distributed simulations. This paper describes a framework, named Distributed Resource Management System (DRMS), for robust execution of simulations based on the High Level Architecture. More specifically, a mechanism for FT in simulations synchronized according to the time-warp protocol is presented and evaluated. The results show that utilization of the FT mechanism, in a worst-case scenario, increases the total number of generated messages by 68% if one fault occurs. When the FT mechanism is not utilized, the same scenario shows an increase in total number of generated messages by 90%. Considering the worst-case scenario a plausible requirement on an M&S infrastructure of the NBD, the overhead caused by the FT mechanism is considered acceptable.

1. Introduction

Modeling and Simulation (M&S) have an important role in realizing the concept of a Network-Based Defence (NBD). In this context, simulations provide support for efficient training, and can also function as a decision support tool for the commander. When simulation tools are used in the decision process, these must meet certain requirements. Above all, simulations must be reliable and respond in a timely fashion. The latter is of particular importance in short decision cycles. An important aspect of these requirements is support for Fault-Tolerance (FT). Mechanisms for detection of failures in a simulation, as well as measures for failure recovery are needed. If these functions are properly designed and implemented the reliability of simulation results will be increased. Also, the effectiveness of simulation executions may be increased, i.e. reruns of erroneous simulations are avoided.

Today, distributed simulation is often employed in the military domain since it efficiently decomposes a simulation system into logical units, better enabling reuse and availability of simulation models. Also, a distributed simulation system exhibits better tolerance against failures. Given that components of a simulation are distributed over several nodes, a failure will most often not affect the entire system.

The most well-known and used standard for distributed simulations within the military domain is the High Level Architecture (HLA). The HLA standard is implemented by a Run-Time Infrastructure (RTI), which can be seen as an operating system for distributed simulations. In HLA a simulation is referred to as a federation, whereas individual components of the federation are referred to as federates. Until now, FT has not been treated as a core component of the HLA standard, thus federations are typically developed disregarding this important aspect. Given that HLA is the main architecture for distributed simulations within the Swedish Defence, and simulations are crucial in the NBD, it is important to evaluate the possibilities for inclusion of FT mechanisms in HLA.

In [6] and [5] an architecture and partial implementation of an execution environment for distributed simulations, referred to as the Distributed Resource Management System (DRMS), are described. Also, in [5] an FT mechanism within the framework of DRMS is presented. This mechanism specifically addresses FT in federations synchronized according to the time-warp protocol [9]. In this paper we investigate the feasibility of the proposed FT mechanism and present some performance results. The performance is evaluated in terms of the overhead, in number of messages, caused by the FT mechanism. The FT mechanism in DRMS does not consider software errors, in terms of federates producing erroneous result, but handles situations where the host environment of a federate crashes, the federate itself crashes for some reason, or a federate's link to the RTI is lost. Moreover, we assume that federates executed within the scope of DRMS are portable, meaning that they are not bound to a specific piece of hardware and can easily be migrated between different host environments. In the present implementation of the FT mechanism, it is not possible to recover from multiple concurrent failures. However, the DRMS supports recovery from non-transient errors, i.e. permanent failures.

2. Handling fault-tolerance in distributed simulations

DRMS is a component of a network-based M&S environment, referred to as NetSim, currently under development at the Swedish Defence Research Agency (FOI). NetSim provides services for distributed storage and look-up of resources, e.g. federates, federations and computing resources. Further, the NetSim environment aims at providing services for Computer Supported Collaborative Work (CSCW), which means that users of the environment can cooperate in developing a federation, regardless of their physical location. The purpose of the DRMS in this setting is to provide transparent access to computing resources, which are utilized for execution of federations composed by a user, or a group of users. DRMS comprises services for automatic deployment of distributed simulations and fault-tolerant execution of federations. The NetSim environment is described in greater detail in [7].

DRMS is based on a service-oriented architecture, which is realized using Web Services, more specifically the Axis Web Services platform [14]. DRMS comprises two basic service types, namely a worker service and a coordinator service. A worker is responsible for execution of one or more jobs, e.g. federates, whereas a coordinator is responsible for the coordination of one or more workers when managing a batch of jobs. In addition to these basic services, the DRMS relies on a repository service. A repository is used by a worker to advertise its presence on the network and thereby its

availability for execution of jobs. Furthermore, the repository is used by a coordinator for localization of available workers. A repository also contains advertisements of other resources available on the network, federates for instance, and is therefore used as entry point when worker services fetch resource files and executable code. The implementation of DRMS is described in greater detail in [5].

2.1 Fault-tolerance in HLA

A distributed simulation, or distributed system for that matter, has a higher failure rate than a simulation or system executed on a single machine. However, a failure in a distributed system is often partial, that is, one of the components of the system fails. The failure may, or may not, affect other components of the system. In the past, several techniques for fault-tolerance in distributed systems have been developed. These techniques can be classified into two main categories; replication-based and check-pointing-based approaches [4]. In replication based approaches one or more copies of a Logical Process (LP) is maintained in addition to the main LP. In case of failure, one of these replicas will take the failed LP's place. In check-pointing based approaches, states of the individual LPs are saved on stable storage. In case of failure, an LP is restarted using the last stable state saved on stable storage.

According to [10] research in fault-tolerant distributed simulation has been quite sparse. Application of fault-tolerance techniques in the context of HLA is even scarcer. However, there is some work that aims in this direction. In [12] a structured view of fault-tolerance in parallel and distributed simulations is given and possible solutions are proposed. In [11] a Resource Sharing System (RSS) is presented that in a future extension could serve as the basis for fault-detection, check-pointing and replication of federates. In [1] a concept, named R-FED (Replica Federate), in support of fault-tolerant HLA federations is presented. As the name implies, the approach is based on replication of individual federates in a federation. Several papers address the issue of federate migration, which is an important cornerstone in designing an infrastructure for fault-tolerant distributed simulation; see for example [6], [15], [2], [3] and [11]. However, these papers usually address federate migration in the context of load-balancing and do not explicitly address fault-tolerance.

The present version of HLA is IEEE 1516-2000. Currently, work is carried out to define the next version of HLA, through the HLA Evolved [13]. An interesting aspect of HLA Evolved is that fault-tolerance has been given more focus than before. HLA Evolved is aimed at providing a common semantics for failure and mechanisms for fault-detection. At the core, two additions have been made to the Management Object Model (MOM), namely federate lost and disconnected. These interactions provide the basic mechanisms for signaling a fault from the context of a federation, through federate lost, and from the perspective of a federate, through disconnected. Upon failure, the RTI has the responsibility to do resign on behalf of the lost federate using an Automatic Resign Directive. This line of development is important for future realization of fault-tolerant distributed simulations, based on the HLA.

2.2 Fault-tolerance in DRMS

At present, a mechanism for FT in federations synchronized according to the time-warp protocol is implemented in DRMS. The recovery phase of this mechanism is

based on a rollback-recovery scheme, which is commonly used for FT in message-passing systems; see for example the survey made by Elnozahy et al [8]. In this approach, states of individual federates are periodically saved on stable storage throughout the federation execution. In case of failure, recovery of the federation is accomplished by rolling back individual federates to a consistent system state. In DRMS, federate states are check-pointed using a remote stable storage component. This means that states are distributed outside of the local scope (host environment) of an executing federate. The communication required for distribution of checkpoints, to and from the stable storage, is implemented by means of the RTI communication infrastructure, utilizing an extension to the Federation Object Model (FOM). Thus, the stable storage component is also a member of the federation. Interactions imposed by the FT mechanism are outlined in table 1.

Table 1. Interactions added to the FOM to support the fault-tolerance mechanism (P = Publish, S = Subscribe).

Interaction	Description	Federate	Stable Storage
reportCP	Reports state to Stable Storage	P	S
requestCP	Requests latest state from stable storage	P	S
latestCP	Delivers latest state to recovered federate	S	P
requestResend	Requests resend of messages with time stamp greater than GVT	P, S	-

The check-pointing protocol must assure that states, reported to stable storage, are safe i.e. a state represents a federate at a point in time that can not be invalided due to rollback. This means that federates report checkpoints for a point in time that is less than the smallest timestamp of a message that could ever be delivered to the federate. The time-stamp of states reported to the stable storage are used by the stable storage component to control the advancement of GVT. This control is required to stop advancement of GVT in case of federate failure. The check-pointing is not synchronized through-out the federation, but federates report states individually.

2.3 Federate restoration in DRMS

Next, the fault-recovery scheme of the FT mechanism is explained in greater detail based on a simple example. In case of federate failure, the DRMS automatically restores the concerned federate in a new host environment. This is accomplished by using a previously saved state from the stable storage. The state used for restoration represents the federate at the current GVT.

Consider the simple exchange of event-messages as illustrated in figure 1. Fed2 processes event-message A at time T_a, which induces scheduling of event-message B at time T_b in Fed1. Fed1 processes event-message B at time T_b, which leads to

scheduling of event-message C in Fed2 at time T_c. Next, Fed2 processes event-message C at time T_c, leading to scheduling of event-message D in Fed3 at time T_d, and so on.

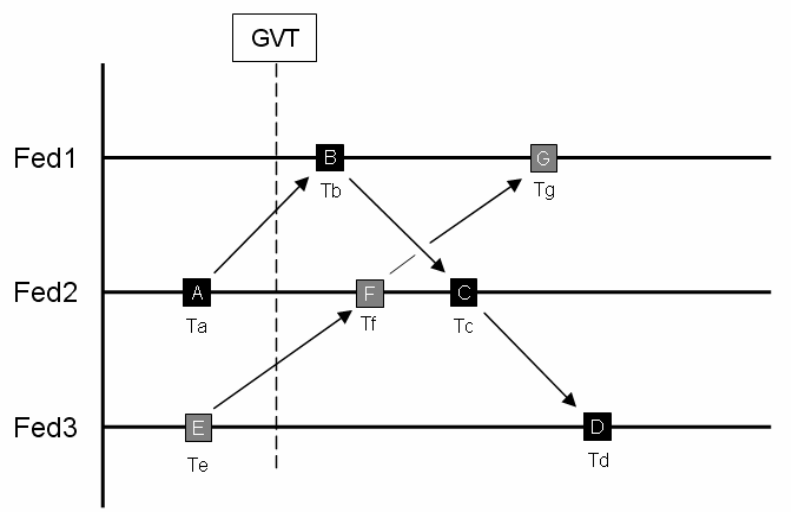

Figure 1. Exchange of event-messages in a federation.

Three types of time-stamps are associated with each event-message in figure 1, these are; T_{send}, $T_{process}$ and $T_{schedule}$. Looking at an event-message from the perspective of Fed2, for example event-message C, T_{send} represents the processing time of event-message B by Fed1, i.e. T_b. $T_{process}$ represents the processing time of event-message C by Fed2, i.e. T_c and $T_{schedule}$ represents the scheduled time of event-message D in Fed3, i.e. T_d. $T_{process}$ and $T_{schedule}$ of each event-message are inherently known to each federate. T_{send} is added to all interactions, or objects, as an additional parameter.

Upon failure of a federate its execution is resumed in a new host environment. After rejoining the federation execution, the resumed federate requests its state from stable storage using the requestCP interaction. Next, the resumed federate issues the RequestResend interaction that instructs other federates to execute the recovery procedure. The recovery procedure, carried out by all federates besides the resumed federate, can be described in terms of the following pseudo-code:

```
ID = name of failed federate

For each event-message, E, sent from ID do
      If T_send of E is greater than GVT then
            If E is not processed then
                  Delete E
            Else If E is processed then
                  Retract E

For each retracted E do
      Rollback to T_process of E

For each E destined for ID do
      If T_schedule of E is greater than GVT
            Resend E to ID
```

Given the procedure outlined above, if Fed1 fails the following sequence of events will be executed to restore the federation; Fed2 retracts event-message C, since in this case T_{send} of C is greater than GVT. The retracted event-message induces rollback of Fed2 to $T_{process}$ of event-message C, i.e. Fed2 will rollback to a state that does not record the processing of event-message C, but reflects the processing of event-message F. The rollback will in turn induce retraction of event-message D from Fed3. This causes rollback of Fed3 to $T_{process}$ of event-message D. The state used for the rollback does not record the processing of event-message D, but reflects the processing of event-message E. After completion of the rollback phase, event-messages destined for Fed1 are resent. This means that Fed2 resends event-message B, since $T_{schedule}$ of event-message A is greater than GVT. Also, Fed2 resends event-message G, since $T_{schedule}$ of event-message F is greater than GVT. After this the federation is restored and can resume normal execution.

3. Test federation

To evaluate the FT mechanism, as described above, a simple test federation was developed. This federation employs time-warp as synchronization protocol and comprises four federates. The federates form a fully connected network, i.e. each federate is capable of sending an event-message to an arbitrary neighbor federate.

The processing of an event-message in the test federation simply means updating a statistics object, describing the message exchange during a simulation run, and scheduling of the event-message in a neighbor federate. The scheduled time is randomly calculated within each federate.

The federates of the test federation process and produce events optimistically, thus when an event-message is received in a federate's past, a rollback is triggered. Similarly, when an anti-message is received that will annihilate an already processed event-message a rollback is also triggered. The rollback relies on a record of locally saved checkpoints. The advancement of GVT triggers garbage collection of this record. The test federation utilizes the following interactions:

- Event-message: this is the standard event of the federation. It is simply a message that when processed by a federate means scheduling its arrival in another federate.
- Anti-message: this message is used for annihilating sent event-messages in case of rollback. In this case the standard way of retracting messages in HLA is employed, i.e. using requestRetraction of the FederateAmbassador and retract of the RTIAmbassador.
- ReportCP: this message represents the state of a federate at a specific point in time. It is sent to the stable storage to enable restoration of a failed federate.
- RequestCP: this message requests a state of a particular federate at the current GVT from stable storage.
- LatestCP: this message is used to deliver a requested state from stable storage to the concerned federate.
- RequestResend: this message triggers resend of messages to a federate that has been migrated due to failure.

Of the message types defined above, the ReportCP, RequestCP, LatestCP and RequestResend are specifically employed to enable use of the FT mechanism. Thus, when the federation is executed without the FT mechanism, these message types are not used.

4. Experiments and results

The following section describes the experiments used to evaluate the proposed FT mechanism and the results generated through these experiments. The purpose of the experiments was to get an estimate of the communication overhead generated when utilizing the FT mechanism. Thus, the difference in number of generated messages between the case when FT is employed and the case when FT is not used was measured. Also, the difference in total number of messages generated between the FT and non FT cases, when faults are introduced during the simulation run, was measured. When faults are introduced during the simulation execution, in the non FT case, the entire simulation has to be restarted completely. During the experiments the test federation was executed using a logical time interval from 0 to 500. Table 2 outlines the characteristics of the simulation runs carried out to evaluate the proposed FT mechanism.

Table 2. Characteristics of experimental simulation runs.
The simulation was executed using a logical time interval from 0 to 500.

Simulation Run	Failures	Time of failures	Using FT
1	0	-	Yes
2	0	-	No
3	1	50	Yes
4	1	150	Yes
5	1	250	Yes
6	1	350	Yes
7	1	450	Yes
8	1	50	No
9	1	150	No
10	1	250	No
11	1	350	No
12	1	450	No
13	3	125, 250, 375	Yes
14	3	125, 250, 375	No

Figure 2 shows the total number of event-messages and anti-messages for three simulation runs that does not utilize FT. In the first case, no failures were triggered during the simulation run. In the second case, a failure was triggered at local time 250 in one of the federates. In the last case, three failures were introduced during the simulation execution. These occurred at local time 125, 250 and 375 of three different federates. When a failure occurs the federation must be restarted, since it is assumed that no checkpoints are available to allow for federate recovery. Thus, a failure at the early stages of the simulation execution will induce minimal extra communication, whereas in the extreme case, the total communication cost will increase by 100 %.

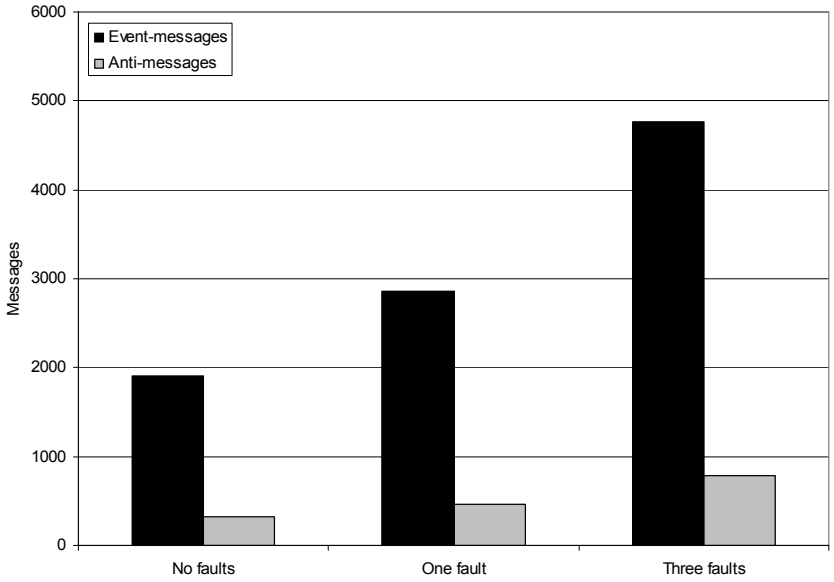

Figure 2. Total number of event-messages and anti-messages generated during simulation execution, not utilizing the FT mechanism, for three different cases; no faults, one fault and three faults.

Figure 3 shows the total number of event-messages and anti-messages for three simulation runs when the FT mechanism is used. In the first case, no failures were triggered during the simulation run. In the second case, a failure was triggered at local time 250 in one of the federates. In the last case, three failures were introduced during the simulation execution. These occurred at local time 125, 250 and 375 of three different federates. The difference between the zero faults, one fault and three faults cases is small. The total number of messages generated for each message type increases slightly as the number of faults triggered increases.

Figure 4 shows a comparison between the FT and non FT cases. The bars in this chart represent the total number of messages generated within the federation. In the one fault case the failure was triggered at local time 250 in one of the federates, whereas in the three faults case, the failures were triggered at local time 125, 250 and 375 in three different federates. As the chart illustrates, using the FT mechanism when no faults are present in the system will impose an extra cost. In this case the reportCP interactions to stable storage induce the cost. However, in the one fault case the applied approaches (FT and non FT) almost perform equally. As expected, the three faults case reflects a high number of messages for the non FT simulation execution.

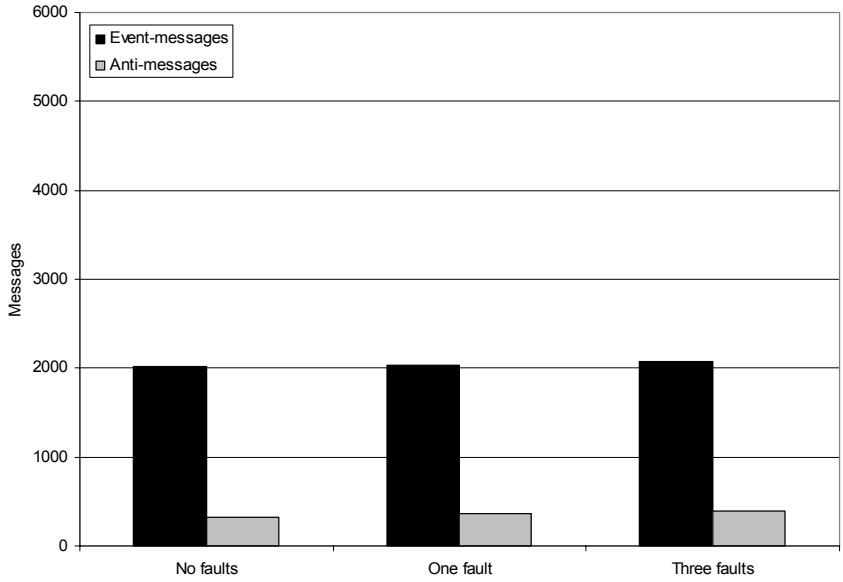

Figure 3. Total number of event-messages and anti-messages generated during simulation execution, utilizing the FT mechanism, for three different cases; no faults, one fault and three faults.

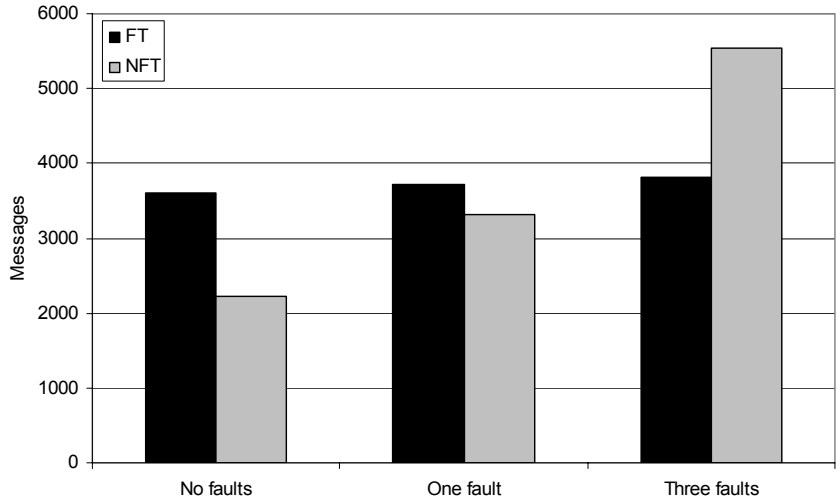

Figure 4. Total number of messages generated for the no faults, one fault and three faults cases, with and without utilization of the FT mechanism.

The failure of individual components of a simulation is probably best described in terms of a stochastic process. Making general conclusions based on mean estimates of failure times will not reflect the behavior of a real-world system. Thus, it is not entirely justified to make conclusions based on a mean value that expresses when a federate will fail during the simulation run. To provide an alternative view of the cost of having fault-tolerance, three different cases were tested. The purpose of these cases was to represent the mean failure time within five consecutive intervals of the federation execution. Thus, these cases represent a mean failure time within the first fifth, second fifth and so forth of the logical time interval used for the simulation execution. Table 3 summarizes the total number of messages generated, with and without FT, for these intervals.

Table 3. Total number of messages for the FT and non FT cases.

Failure time	FT	NFT
50	3841	2474
150	3774	2856
250	3714	3321
350	3754	3747
450	3740	4258

Figure 5 depicts the difference between the FT and non FT cases, in terms of total number of generated messages, for the intervals described above. As indicated in the chart, the overhead for the FT cases is greater than the extra communication caused by the faults in the non FT cases for the first three intervals. In the fourth interval the total communication cost is almost equal, whereas in the fifth interval the FT case shows better performance over the non FT case.

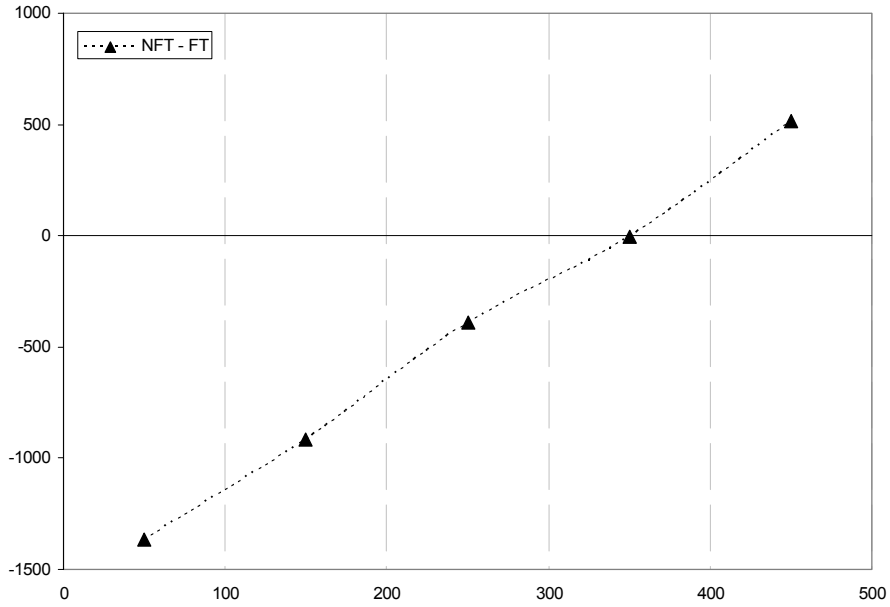

Figure 5. Difference in total number of messages, assuming one fault, between the FT and non FT cases for mean failure time of five intervals of the federation's logical time interval.

5. Discussion

When simulations are integrated in the decision process, to serve the commander in an NBD context, the requirements on the supporting infrastructure are high. An essential aspect of these requirements is enabling fault-tolerant distributed simulations, since this is the fundament for a robust execution environment that can respond to user needs in a timely fashion. The most widely adopted standard for distributed simulations within the military community, the HLA, does not in its present form treat fault-tolerance extensively. Thus, it is desirable to develop efficient and scalable methods for fault-tolerance in HLA in order to successfully deploy HLA simulations within the framework of the NBD.

We have shown that it is feasible to develop FT mechanisms within the framework of HLA. However, in our case a supporting infrastructure is needed to enable automatic re-deployment of a federate upon failure. As the HLA develops, through the HLA

Evolved track, there will be stronger support for fault-tolerance within the standard, which will ease development of robust federations.

Regardless of the approach taken for implementation of fault-tolerant federations, there will always be an extra cost associated to it. The fault-tolerance support will inherently lead to increased network traffic and/or heavier consumption of hardware resources. However, this extra cost must be evaluated in terms of the consequences that may result from not having a fault-tolerant federation. In the federation developed to evaluate the proposed FT mechanism, the cost, in terms of number of extra messages generated to handle failures, is relatively low, as indicated in figure 3. The total number of messages rises from 2013 to 2075, comparing the no faults case with the three faults case, not taking into account the reportCP interactions. The reason for this increase is the resending of event-messages and potential rollbacks caused by recovery of a failed federate. However, depending on the mutual relations existing between federates of a federation, the recovery phase may result in disparate numbers of resent event-messages and/or rollbacks. Different relations among a set of federates have not been treated in the present evaluation. Currently, the federates form a fully connected network and thus the message exchange is fairly homogeneous.

As seen in figure 4, the total number of generated messages, for the non FT cases, rises sharply as the number of faults increases, whereas in the FT cases, the increase is almost negligible. Of course, the timing of the faults has a large impact on the end result in these cases. If a simulation is considered being a critical component of an NBD framework, the worst-case must be considered a plausible scenario, i.e. failure near the end of the simulation execution. Looking at the mean failure time of the last 20% of the simulation execution in figure 5 the non FT case results in 4258 messages (given by table 3). In comparison with the total number of generated messages for the zero failures case without FT of 2221 messages, presented in figure 4, a failure in the last 20% of the simulation execution would increase the total number of messages by approximately 90% on average. Note that this estimate is for the one fault case. If additional faults are introduced in this interval the cost will grow immensely. In the one fault case, utilization of the FT approach will result in 3740 messages (given by table 3). This equals an increase in number of generated messages by approximately 68%. However, note that this figure is tightly coupled with the total number of event-messages of the federation. If a federation generates more event-messages than the test federation, the FT mechanism will perform even better (since the number of reportCP interactions will remain the same). Thus, the overhead, in number of generated messages, for having fault-tolerance is certainly justifiable, considering the worst-case scenario a likely requirement on an M&S infrastructure of the NBD.

6. Conclusions

We showed how to implement fault-tolerance in time-warp federations using a check-pointing-based scheme within the HLA communication infrastructure. The overhead cost when utilizing the FT mechanism is justifiable, especially when considering worst-case scenarios for federate failure times i.e. failures near the end of the federation execution. In our example, when the FT mechanism is employed, a failure in the last 20% of the simulation execution would result in a total of 3740 generated messages, which should be compared with the 4258 messages generated when the FT

mechanism is not employed. If no failures occur, the simulation execution will generate 2221 messages on average, given that the fault-tolerance mechanism is not used. This means that for one failure in the last 20% of the simulation execution, utilization of the fault-tolerance mechanism will increase the total number of messages by 68%. This should be compared to the case when fault-tolerance is not present where the total number of generated messages is increased by 90%. Thus, the FT mechanism will in this case decrease the total number of messages by approximately 12%. If additional faults occur the reduction will be greater.

7. Future work

As a starting point we have evaluated the overhead in number of additional messages generated by the FT mechanism. As a next step it would be valuable to estimate the degradation in the effectiveness of a simulation execution when using the FT mechanism. In this context it would be of interest to look at various sizes of the checkpoints to see how different federate types may have an impact on the effectiveness, i.e. some federates will require more memory than others for representation of their states. Also, we will consider tests of the FT mechanism where the timing of the faults are not pre-defined, but randomly generated throughout the simulation execution by a probability function at each node.

Moreover, the test federation used in this work comprises no complex issues of ownership of objects in the federation. In more complex federation types, the issue of transferring ownership of objects between federates, in case of failure, must be resolved as well. This issue is of great concern to enable use of the FT mechanism on a more general level and in more complex federation types.

8. References

[1] C. Berchtold, and M. Hezel. 2001. An architecture for fault-tolerant HLA-based simulation Proceedings of 15th European Simulation Multiconference.

[2] L. Bononi, G. D'Angelo, and L. Donatiello. 2003. HLA-based adaptive distributed simulation of wireless mobile systems Proceedings of 17th Workshop on Parallel and Distributed Simulation.

[3] W. Cai, S. Turner and H. Zhao. 2002. A load management system for running HLA-based distributed simulations over the grid Proceedings of the IEEE International Symposium on Distributed Simulation and Real Time Applications.

[4] P. Damani, and K. Garg. 1998. Fault-tolerant distributed simulation Proceedings of the 12th Workshop on Parallel and Distributed Simulation.

[5] M. Eklöf, R. Ayani, and F. Moradi. 2005. A Framework for fault-tolerance in HLA-based distributed simulations Proceedings of the 2005 Winter Simulation Conference.

[6] M. Eklöf, M. Sparf, F. Moradi, and R. Ayani. 2004. Peer-to-peer-based resource management in support of HLA-based simulations SIMULATION. Volume 80, Issue 4-5.

[7] M. Eklöf, J. Ulriksson, and F. Moradi. 2003. NetSim: an environment for network based modeling and simulation Symposium on C3I and M&S Interoperability (MSG-22). NATO Modeling and Simulation Group.

[8] E. Elnozahy, L. Alvisi, Y-M, Wang, and D. Johnson. 2002. A Survey of Rollback-Recovery Protocols in Message-Passing Systems ACM Computing Surveys. Volume 34, Number 3.

[9] D. Jefferson. 1985. Virtual time ACM Transactions on Programming Languages and Systems. Volume 7, Number 3.

[10] T. Kiesling. 2003. Fault-tolerant distributed simulation: a position paper. Available via http://fakinf.informatik.unibwmuenchen.de/~tkiesling/documents/ftds-position-paper.pdf [accessed March 21, 2005].

[11] J. Lüthi, and S. Großmann. 2001. The resource sharing system: dynamic federate mapping for HLA-based distributed simulation Proceedings of the 15th Workshop on Parallel and Distributed Simulation.

[12] J. Lüthi, and C. Berchtold. 2000. Concepts for dependable distributed discrete event simulation Proceedings of the International European Simulation Multi-Conference.

[13] B. Möller, M. Karlsson, and B. Löfstrand. 2005. Developing fault tolerant federations using HLA evolved Proceedings of the Spring Simulation Interoperability Workshop.

[14] U. Saleem. 2004. Developing java web services with AXIS. Available via http://www.developer.com/java/ web/article.php/3443951 [accessed March 21, 2005].

[15] G. Tan, A. Persson, and R. Ayani. 2004. HLA federate migration 38th Annual Simulation Symposium.