A Synchronization Algorithm for Distributed Time Stepped Simulation: An Application to Wargame

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

By:
Pooja Grover
(8043116)

Under the supervision of:

Mr. Rajesh Bhatia
Assistant Professor, CSED

Mr S. B Taneja
Scientist 'E', ISSA, DRDO

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
(DEEMED UNIVERSITY)
PATIALA – 147004

MAY 2006

Abstract

An overview of technologies concerned with distributing the execution of simulation programs across multiple processors is presented. Here, particular emphasis is placed on discrete event simulations. The main focus of this research work is on time management, a central issue concerning the synchronization of computations on different processors. Time management algorithms broadly fall into two categories, termed conservative and optimistic synchronization. A survey of both conservative and optimistic algorithms is presented focusing on fundamental principles and mechanisms. Time management in the HLA is discussed as a means to illustrate how this standard supports both approaches to synchronization. Finally a new time synchronization approach is suggested and then implemented. This approach is specifically meant for time stepped simulations.

Institute for System Studies and Analyses (ISSA) has developed software's which provide a computerized combat training environment, designed to train commanders and staff in combined arms environment for realistic battlefield conditions. The thesis work will provide a survey of the various time synchronization algorithms and will thus provide solution for synchronization of time among various distributed simulation processes.

Keywords: Wargame, Distributed Time Stepped Simulation, Synchronization, federate, HLA

Declaration

I hereby certify that the work which is being presented in the thesis entitled, ***"A Synchronization Algorithm for Distributed Time Stepped Simulation: An Application to Wargame"*** in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering at Computer Science and Engineering Department of Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision of *Mr. Rajesh Bhatia* and *Mr. S.B Taneja.*
The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other University.

(Pooja Grover)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Mr. Rajesh Bhatia)
Assistant Professor
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
PATIALA- 147004

(Mr. S.B Taneja)
Scientist 'E'
ISSA, DRDO

Countersigned by

(Dr. Seema Bawa)
Head and Professor
Computer Science and Engineering Department
Thapar Institute of Engineering and Technology
PATIALA- 147004

(Dr T.P Singh)
Dean Of Academic Affairs
Thapar Institute of Engineering and Technology
PATIALA- 147004

Acknowledgement

I wish to express my deep gratitude to Mr. S.B Taneja, Scientist 'E', ISSA, DRDO for providing his individual guidance and support throughout the Thesis work.

I am also thankful to Mr. Rajesh Bhatia, Assistant Professor, Computer Science and Engineering Department for providing assistance and invaluable support throughout the Thesis work.

I am also thankful to Mr. Chandramoulli Scientist 'E', ISSA, DRDO for providing timely assistance and encouragement, which went long way in successful completion of my thesis.

I am also thankful to Dr. Seema Bawa, Head and Professor, Computer Science and Engineering Department, for the motivation and inspiration that triggered me for my thesis work.

I would also like to thank all the staff members of ISSA who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of my thesis.

I am also thankful to the authors whose works I have consulted and quoted in this work. Last but not the least I would like to thank God for not letting me down at the time of crisis and showing me the silver lining in the dark clouds.

Pooja Grover.
(8043116).

Table of Contents

Abstract i

Declaration ii

Acknowledgement iii

Table of Contents iv

List of Figures vii

Chapter-1 Introduction 1-10

1.1 Wargame-An Overview 2

1.1.1 Military Simulation Architecture 3

1.2 Issues in Military Simulation Infrastructure 5

1.3 Need for Research 9

1.4 Objective 9

1.5 Methodology 10

1.6 Organization of the Report 10

1.7 Concluding Remarks 10

Chapter-2 Distributed Simulation 11-22

2.1 Simulation Classification 12

2.1.1 Discrete and Continuous Simulation 13

2.1.2 Entity Level and Aggregate Level 15

2.2 State of the Art in Distributed Simulation 16

2.2.1 SIMNET 16

2.2.2 DIS 16

2.2.3 ALSP 16

2.2.4 HLA 17

2.3 Distributed Simulation Architecture Options 19

2.4 Synchronization Problem 20

2.5 Advantages 21

2.6 Concluding Remarks 21

Chapter-3 Time Management 23-41

3.1 Need for Time Management 23
3.2 Problem Statement 25
3.3 Time Synchronization Algorithms 26
3.3.1 Conservative Time Management 26
3.3.1.1 Advantages of Chandy-Misra-Bryant (CMB) Algorithm 29
3.3.1.2 Disadvantages of Chandy-Misra-Bryant (CMB) Algorithm30
3.3.2 Optimistic Time Management 30
3.3.2.1 State saving Techniques 32
3.3.2.2 Advantages of Optimistic over Conservative Techniques 33
3.3.2.3 Disadvantages of Optimistic Techniques 33
3.4 Conservative Vs Optimistic for Military Simulations 33
3.5 Time Management in the HLA 34
3.5.1 Message Ordering Mechanisms 34
3.5.2 Object Management Services 37
3.5.3 Time Advance Services 38
3.5.4 Schemes for Time Management in HLA 39
3.6 Concluding Remarks 41

Chapter-4 Synchronization Approaches for Distributed Time Stepped Simulations 42-50

4.1 Piggy-Backed Time Stepped Simulation 43
4.1.1 Introduction 43
4.1.2 The Approach 43
4.1.3 Result 46
4.1.4 Limitations 47
4.2 Paced Time Stepped Synchronization 47
4.2.1 Introduction 47
4.2.2 The Approach 48
4.2.3 Results 49
4.3 Zero Lookahead Approach 49
4.3.1 Introduction 49
4.3.2 The Approach 49
4.3.3 Results 50
4.4 Concluding Remarks 50

Chapter-5 Proposed Time Synchronization Approach 51-65
5.1 The Proposed Approach 51
5.2 Methodology 54
5.3 Simulation Execution and Testing 56
5.4 Results 56
5.5 Test Plan and Output 57
5.6 Advantages 58
5.7 Limitations 58
5.8 Snapshots 59
5.9 Concluding Remarks 65

Chapter-6 Conclusions and Scope for Further Research 66-69
6.1 Conclusions 66
6.2 Summary of Contributions 67
6.3 Scope for Further Research 68

References 70-72
Appendix A – Acronyms 73
Appendix B – Glossary 74
Papers Communicated 75

List of Figures

Figure Number	Title	Page
Figure 1.1	Functional Components of Military Simulations	4
Figure 2.1	Differences between Parallel and Distributed Computers	13
Figure 2.2	Time guarantees in time-stepped simulations	13
Figure 2.3	DIS Network	17
Figure 2.4	Peer-to-Peer Approach	20
Figure 3.1	Need for Time Management	24
Figure 3.2	FIFO queues in Chandy-Misra-Bryant Algorithm	27
Figure 3.3	The Chandy-Misra-Bryant Algorithm	28
Figure 3.4	FIFO queues in Time Warp Algorithm	31
Figure 3.5	Scenario demonstrating causal and total ordering	36
Figure 4.1	Simulation Progress in Traditional Time Stepped Simulations	42
Figure 4.2	Initiation of the Simulation by the Host	44
Figure 4.3	Piggy- Backed Ready Signals from the PEs	44
Figure 4.4	Updating of FTL by Host	45
Figure 4.5	Advance Signals being sent to PEs	46
Figure 5.1	Proposed Architecture	54

Chapter 1
Introduction

The military has a rich history of using models and simulation, since simulation allows the analysis of a system's capabilities, capacities, and behaviors without requiring the construction of or experimentation with the real system. Armed forces of every country spend a huge amount of money for acquiring, designing, fielding and operating simulation systems [19]. Simulation is applied to support a variety of missions, including training, analysis, acquisition, mission rehearsal, and test and evaluation.

The military defines any training that is not real combat to be simulation. This has lead to a division of simulations into three broad categories: Live, Virtual, and Constructive. In live simulation, soldiers operate their real equipment in mock engagements. Ground forces participate in similar maneuvers, armored and infantry forces don laser gear and engage a threat force in non-lethal combat. These exercises allow combat troops to experience the rigors of living and working in the field and force them to fight against a well trained reactive opponent.

Virtual Simulation is similar to live except that the equipment is replaced with simulated mockups and the field of battle is generated by a computer. In these simulators soldiers practice many of the same tactics but at a much lower operational cost and with greater freedom in taking risks. These tanks do not burn fuel, break down, or destroy terrain, thus reducing much of the expense involved in Live Simulation. Operators are free to pull the triggers of their weapons and see the results of simulated tank rounds impacting a target. They may attempt to ford rivers, destroy building, and generally engage in behavior that is too dangerous to attempt in live simulation. Since the battlefield is sculpted by the computer it can be made to look exactly like that on which the trainee expects to operate. Live simulations, on the other hand, are limited to the terrain that can be found naturally at training sites.

Constructive Simulation, also known as wargaming, derives its name from the fact that the pieces operating on the battlefield are not individual tanks and aircraft but a construction of many different types of equipment into a single aggregated unit like

an armor company, artillery battery, etc. The first two types of simulations are used to train individuals operating equipment, this equipment is in turn controlled by leaders in command posts who see the battle in a more abstract form. Constructive simulations allow these commanders to face situations and make decisions under the stress of time and limited resources just as they will during actual combat. In a classroom situation it is easy to present students with problems in which the answers have been determined and solving them is a homework assignment. Constructive simulations immerse these commanders in a situation where the enemy is highly trained, experienced, and just as determined to win the war [18]. This enemy is unpredictable and does not always operate as the books say he should. Here soldiers discover whether the tactics they have been taught really work, here they develop confidence in their ability to operate as a team and win wars.

Since the wargame is a distributed simulation technology application, various issues linked to distributed simulation infrastructure also apply to development of wargames. These issues are discussed in Section 1.2. This thesis work will be focusing on the Time synchronization issue. Time synchronization is very important since the reliability of the results generated by a distributed algorithm depends on the accuracy of the synchronization mechanism used. Different time synchronization algorithms have been developed. This thesis work will first provide a survey of various time management approaches applicable to different distributed simulation environments and architectures mainly focusing on the time synchronization for distributed time stepped simulation. Thereby an efficient time synchronization algorithm is discussed.

1.1 Wargame: An Overview

A wargame can be defined as “a simulated military operation involving two or more opposing forces and using rules, data and procedures designed to depict an actual or hypothetical real-life situation" [20]. It is used primarily to study problems of military planning, organization, tactics and strategy. Tactical and strategic decisions are reflected in the movement of military icons on a map, testing the commander and staff’s ability to use their forces effectively. Two different simulation categories can be defined namely live simulations and analytical simulations. Live simulations are the application of real equipment in mock combat scenarios or firing ranges. These allow pilots, tank drivers, and other soldiers to practice the physical activities of war

with their real equipment. Analytical simulations are used to study problems like force composition, weapons effectiveness, and logistics issues. Analytical simulations usually differ in that they do not focus on interactive exchanges with people during a simulation run. This allows them to execute much faster or3 slower than real time without adversely impacting a human operator.

Three types of wargames are in common use today. These are as under:

i. Training Wargames:

Such games are two sided and are designed to provide the participants with decision making opportunities similar to those they may experience in combat. A control or referee group guides the actions of the team. Typically, these low level games focus attention on force levels and tactical deployments, weapon and sensor performance and inter-relationships among various warfare areas. The primary decision makers in this category are generally battle group commanders or below.

ii. Operational Wargames:

These games are generally played to validate operational plans. These may be solo or two sided games. In the former, the Commander who has made the plans himself acts as the enemy commander and tests his reactions against his own plans. In the two-sided variety the plans may be subjected to scrutiny by an outside agency not associated with planning. Many options, which may have been overlooked by the original planners, may be highlighted in such gaming.

iii. Analytical Research Games:

This is the most complex of the three types, and requires careful preparation to achieve maximum objectivity and is usually designed to study future tactical or strategic problems.

1.1.1 Military Simulation Architecture

There are six major components of a military training simulation [10]. An infrastructure ties all of these together. Figure 1.1 shows the six basic pieces of a simulation

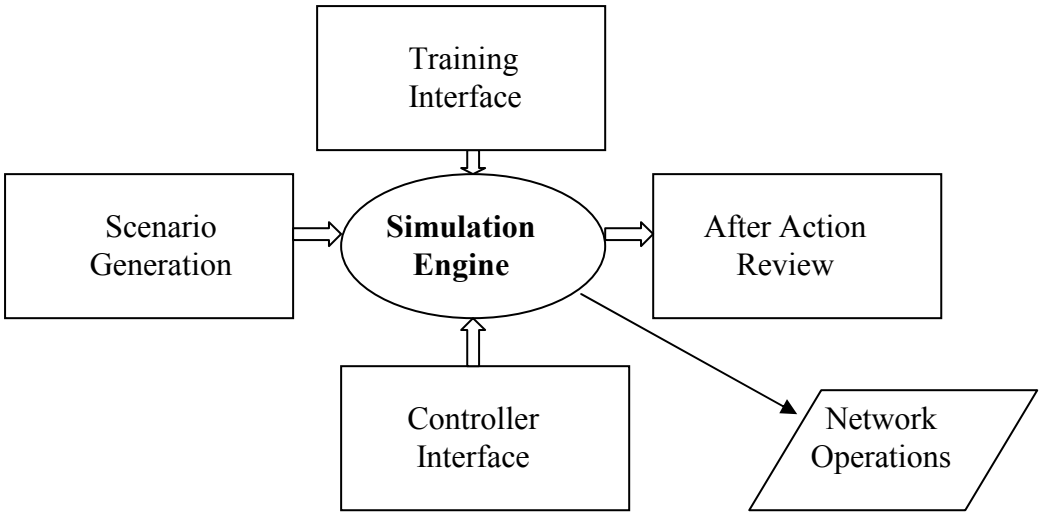


Figure 1.1 Functional Components of Military Simulations

- **Simulation Engine.** The Simulation Engine lies at the center of the whole architecture. It produces the ground combat, air strikes or intelligence collection. This contains representations of the terrain, weather, combat units, and equipment characteristics. These are operated on by orders received from the training audience and their opposing forces (OPFOR). Mathematic algorithms then execute the orders to produce the combat events and their outcomes.
- **Scenario Build**. Scenario build is responsible for creating the required data in a form that the simulation can use. The data set contains military equipment descriptions, operating characteristics, terrain, and force placement. The data may naturally exist in hundreds of different documents, libraries, and databases in varying formats. This tool began as an operator using a text editor to build a formatted data file. More advanced systems are being developed in which the data is entered graphically, reducing the amount of time required to prepare for an exercise.
- **After Action Review**. The AAR tools are responsible for collecting, analyzing, and presenting the performance of the units being trained when an exercise is finished. This allows the exercise sponsors to grasp what is

happening and adjust the exercise to maximize the lessons learned by the training audience.

- **Controllers.** The simulation is run by controllers who are responsible for starting, stopping, controlling the rate of time advancement, and maintaining the system. These typically use a graphical user interface which allows a few people to monitor and control a simulation training hundreds of soldiers. They are responsible for ensuring that the system operates efficiently during the scheduled training period.
- **Trainee.** The entire point of the simulation is to stimulate the trainee. In some cases these people will be operating a simulation specific user interface designed to allow them to enter orders and operations in a format understandable by the computer. Unfortunately, this creates a simulation artifact. It does not teach the operator to work in the environment he or she will use during actual war, though the decision process may be much the same. As the military becomes more computerized, it becomes more natural for the simulation to invisibly connect to the trainee's organic equipment and stimulate it directly.
- **Networks.** Networks join multiple simulations and perform the data transformations that are needed to allow independently developed simulations to understand one another.

The underlying architecture should be able to support the extension of the system to future problems i.e. reuse of the architecture by other simulation developers. This is one of the issues which should be considered during development of a wargame. Other major issues are described in the next section.

1.2 Issues in Military Simulation Infrastructure

There are many issues involved for successfully implementing a wargame application [19].

- **System Architectures:** The underlying architecture should be reusable by other simulation developers. The simulation should be designed in a way that certain operations could be encapsulated as libraries and used by many different customers. These libraries contained routines for generating random numbers, formatting specific reports, performing complex mathematical and

statistical operations, and managing simulation execution. Recently, the focus is on object oriented architectures since they provide greater interoperability. These architectures promise an infrastructure for simulations that may be reused by multiple projects.

- **Terrain Representation:** Military operations typically focus around physical activities on specific areas of the earth. Therefore, simulations of these activities each require a representation of this terrain. In the absence of terrain representation standards, a variety of unique and often incompatible terrain representation schemes have emerged, hindering the representation of a common terrain between multiple simulations. This incompatibility requires that multiple databases of the same terrain be built to support multiple simulations, and introduces potential (undesired) variability in the synthetic environment between interoperating simulations.

- **Behavior Representation:** Because of its complexity, behavioral modeling has traditionally been very basic. The goal has been to provide military vehicles and units with the ability to *react* to basic events in the absence of human intervention. These models allow aircraft on patrol to "decide" to return to base when getting low on fuel, rather than continuing until the aircraft falls to the ground. More recent models have attempted to increase the reasoning capabilities of simulated objects. Recent developments in Artificial Intelligence (AI) technologies, e.g. intelligent agents, potentially offer new solutions to this problem.

- **Multiresolution Modeling:** The notion of resolution in a simulation model is commonly used to describe the level of abstraction employed in the model – some measurement of the differences between the actual system under study and the model of that system. Multiresolution issues exist in single simulation models, but are perhaps made more acute by the system composition afforded by today's interoperability technologies.

- **Synthetic Natural Environments:** An emerging approach in military simulation involves the separation of models of the environment from the physical and behavioral models constituting a simulation. Independent representations of the environment make it necessary to collect and manage a large volume of data. This data may include characteristics of the terrain surface, natural and cultural features, atmosphere, sea surface, subsurface, and ocean floor. The representation of radio and acoustic energy, chemical and biological agents, natural and man-made obscurants and nuclear effects are also considered part of the environment since these create a medium within which the objects must operate. The collection and management of environmental data presents significant challenges, as does the validation of environmental models.

- **Verification, Validation and Accreditation:** There are a number of difficulties with the verification and validation of military training simulations. These include:
  - Validating models in the absence of data.
  - Cost-effective VV&A of training simulations requires a proper characterization of the training objectives – which are often not well formulated.
  - Interactive simulations raise additional difficulties by confounding the notion of repeatability that underlies many of the traditional V&V techniques.
  - Validating real-time simulations introduces a performance aspect not present in traditional simulation applications.

  In addition to these technical challenges, VV&A in the era of distributed, interoperable simulation introduces many programmatic and cultural barriers to cost-effective VV&A, such as the absence of centralized configuration management.

- **Event Management:** Simulations are dynamic representations of systems. Events execution depicts the state of the real system. Since events are such an integral part of the simulation, it is important that they be managed accurately

and efficiently. Events are usually organized into some form of list and stored through a variety of computer structures. Storage may be in the form of an ordered array, linked list, tree, or other structure. Whatever the form, each event contains information about the operation to be performed, the trigger for its execution, and the identity of the objects that it will operate upon.

- **Time Management**: Simulations contain some representation of time since they portray real world. When a functional simulation operates on a single computer system, the management of time is relatively simple. The simulation may choose to move forward in defined discrete time steps, or according to the times of events being executed. With the advent of networked and parallel computer equipment**,** simulations were developed to take advantage of this hardware. As a result, it became necessary to synchronize time across multiple software applications. This synchronization insures that events happen in an order that preserves their causal relationships. Because of delays in network message delivery it is possible for events to arrive at an object in the wrong order. Since the object can not determine whether event messages are enroute, it does not know whether the most current event in queue is the next in the execution sequence. To address this problem, techniques for both conservative and optimistic synchronization of time across processors were developed. These techniques are described in Chapter 3.

- **Information Bandwidth:** Distributed simulations cannot exist without sufficient reliable communications bandwidth for delivering events and synchronizing execution of the entire system. This bandwidth is currently one of the limiting factors on the size of a distributed simulation.

- **Interoperability:** Simulations have traditionally been independent, stand-alone systems that address specific problems and adhere to a unique architecture established by the designer. Around 1988 the military began to explore the possibility of linking multiple interactive training simulations to allow them to interoperate with one another during execution.

The Defense Modeling and Simulation Office (DMSO) developed the High Level Architecture (HLA) to replace both DIS and ALSP since those methods have proven to be very system specific and do not provide a general interoperability solution that can support future simulation systems and missions. Thus interoperable solutions are more beneficial and hence become the most preferred standards for the development of wargame simulations.

1.3 Need for Research

One of the major applications of simulation is wargaming. As these applications have grown in popularity, the developers have been searching for techniques to keep the distributed pictures of the virtual world synchronized. This is necessary to ensure that each player or user is experiencing the same version of the world, with the same cause-and-effect relationships between observed phenomena.

In the context of wargame, focus is primarily on the training of large, joint, or combined forces. Closely associated with training is mission rehearsal, in which essential coordination procedures are worked out and practiced. Some level of synchronization is required to ensure that hardware clock differences, communication delays, and host computer loads do not skew or corrupt critical exercise data. This is especially true in large-scale simulations where hundreds of dissimilar simulation nodes, with multiple software units, will be used to create the virtual battlefield. Apart from these other issues like Event Management, Time Management, Underlying Architecture Support, Behavior Modeling etc. are already discussed in section 1.2. Focusing on Time Management, it is very important to make sure that the underlying time synchronization approach is efficient and suitable for the particular wargame application, since for different type of wargames different time synchronization algorithms need to be used. .

1.4 Objective

The main objective of this research is to develop a synchronization algorithm for distributed time stepped simulation application (Wargame). This algorithm will focus on Time Synchronization issue i.e. synchronizing the clocks of all the processes involved in distributed simulation so that causal relationships are maintained and simulations are well synchronized with the System time.

1.5 Methodology

Synchronization can be accomplished by assigning one of the processes as the owner and controller of the clock. That process moves the clock forward, sending its time to all other processes. Those processes accept that time and process events accordingly. Simulations that are slaved to the master clock are expected to operate fast enough to remain apace of the clock owner. As long as the processes run at least as fast as the master this approach works fine. The suggested approach in chapter 5 follows this methodology.

1.6 Organization of the Report

The thesis is organized in six chapters. The second chapter describes the theoretical background of Distributed Simulation in detail along with the standards followed by the wargame development community. In the third chapter, the focus is on Time Management issue. The need for time management is discussed along with the Time Management algorithms i.e. Conservative and Optimistic along with their advantages and disadvantages. How HLA manages time is also discussed. The fourth chapter describes three time management approaches for Distributed Time Stepped Simulation. These approaches were useful to develop the proposed synchronization algorithm. The proposed algorithm which uses a centralized time server is described in Chapter fifth. This chapter also describes the results and discussion after implementing the proposed algorithm. The sixth chapter concludes with the scope for further research in the area.

1.7 Concluding Remarks

The future of military simulation appears to be very bright. Since this type of training is an order of magnitude less expensive that live simulation, it should be a natural growth area in a shrinking defense environment. It allows the armed forces of a country to train a force that is ready to perform their mission efficiently. There are many issues involved for successfully implementing a distributed simulation application like Time Management, Event Management, and Information Bandwidth etc.

Chapter 2
Distributed Simulation

Simulation is the process of designing a model of a real or imagined system and conducting experiments with that model. The purpose of simulation experiments is to understand the behavior of the system or evaluate strategies for the operation of the system. Since the problems of interest in the real world are very complex that a simple mathematical model cannot be constructed to represent them. In this case, the behavior of the system must be estimated with a simulation. Certain assumptions are made and a model is formed with some approximations since exact representation is seldom possible.

Simulation allows the analysis of a system's capabilities, capacities, and behaviors without requiring the construction of or experimentation with the real system. Due to these reasons its application to the military is significant. Concurrent discrete event simulation (or simply concurrent simulation) refers to the execution of a discrete event simulation program on a machine containing multiple CPUs. This includes execution on closely coupled multiprocessor machines (parallel simulation) as well as geographically distributed computers (distributed simulation).

Distributed simulation has often been used to refer to geographically distributed simulations where several different simulators are integrated into a single simulation environment [17]. Distributed Simulation encompasses several different dimensions. One dimension concerns the motivation for distributing the execution. One paradigm is often referred to as *parallel simulation*. Parallel Computer components are usually closely coupled in a computer cabin or single room. It is characteristic that the used hardware is homogeneous and connected by a specialized high speed connection, with small latency and high bandwidth. Distributed computers on the other hand, can range from being located in a single room, the same building or even spread over larger geographical distances.

The table below clearly figures out the differences between parallel and distributed technologies.

	Parallel Computers	Distributed Computers
Coupling	Tightly coupled	Loosely coupled
Processors	Homogeneous	Often heterogeneous
Physical extent	Machine room	Building, city, global
Comm. Network	Custom switch	Commercial LAN / WAN
Programming	Shared variables/ locks	Message passing
Comm. Latency (small messages)	A few to tens of microseconds	Hundreds of microseconds to seconds

Figure 2.1 Differences between Parallel and Distributed Computers

2.1 Simulation Classification

2.1.1 Discrete and Continuous Simulation

Continuous models assume continuously advancing time and a continuously changing system state. Computationally, this is represented by a time variable that advances in increments small enough to accurately simulate the continuous behavior of the model. Updates to the variables representing the state of the system are typically computed for each step in time.

Discrete-event simulations are based on models that assume an unchanging system state until an event occurs that produces an instantaneous change in the state of the system. In these simulations time advances from event to event in chronological time order. State changes in discrete simulations occur "suddenly", as a jump to next point in simulation time. This takes place in an atomic, no simulation time consuming event and can either happen in a *time-stepped* or *event driven* fashion [5].

- In *time-stepped discrete* simulations the time flow is a sequence of equal size time steps Δt as shown in figure 2.2.

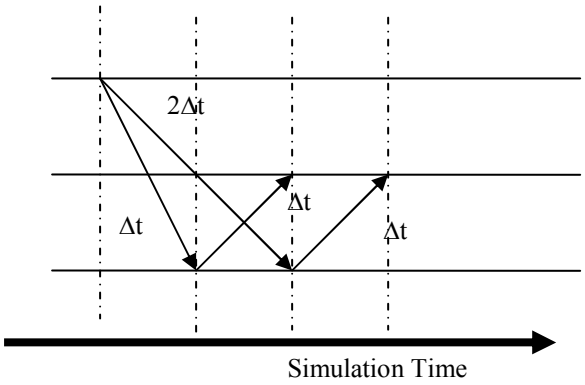


Figure 2.2 Time guarantees in time-stepped simulations

A time-stepped simulation program fills in the space-time diagram by repeatedly computing a new state for the simulation, time step by time step. Actually, not every state variable need to be modified in each time step, but the execution mechanism can only advance from one time step to the next. Actions in the simulation occurring in the same time step are usually considered to be simultaneous, and are often assumed not to have an effect on each other. This is important as it allows such actions to be executed concurrently by different computers to speed up simulation. If two actions has a causal relationship that must be accurately modeled in the simulation, the actions must be simulated at different time steps. Thus the size of the time steps is important in determining the precision of the simulation with respect to time. The size of Δt can be chosen arbitrarily, with the constraint to choose it small enough that no two event in $[t_x, t_{x+\Delta t}]$ effect another or prior events. Still Δt should be chosen to be as large as possible in order to avoid too many empty time intervals.

The following code snippet shows a real-time time-stepped simulation. Here execution is paced to wallclock time. WtoS function converts wallclock time to simulation time [18].

```
while (Simulation in progress)
    wait until ( WtoS(wallclock time) >= simulation time)
    compute state of the system at the end of this time step
    advance simulation time to the next time step
```

Advantages

i. For simulations that contain a large number of objects that are constantly changing state, it is natural to treat simulation time advance like the advance of a clock. The simulation clock ticks like a discrete watch and every object is updated to represent its state at the new time.

ii. Time-stepped algorithm has good performance for certain applications, where there are many events to be executed at each step.

iii. Most interactive simulations are structured as time-stepped simulations since no object notification is required when other objects have done something interesting to them.

Disadvantages

i. The difficulty of finding a good value for Δt marks the main disadvantages of the technique.

ii. Moreover, simulation periods without events can not be skipped and cause the processors to be idle.

- *Event driven discrete* simulations work in a different way, without state change between two events. Variables are not updated at each time step but rather when "something interesting" occurs, which is referred to as an "event." An event is an abstraction used in the simulation to model some instantaneous action in the physical system. Each event has a timestamp associated with it that indicates the point in simulation time when the event occurs. Each event usually results in some change in one or more state variables.

  The following code snippet shows an event stepped discrete simulation where simulation time is advanced depending on the timestamp of the next event.

```
While(!done)
        getNextEvent(smallest timestamped message in the queue)
        simulation time= eventTimeStamp
        <Process this event. Send it to appropriate event handler>
```

Advantages

i. In simulations where events are so few and far between that time stepping results in long periods of processing where nothing happens, event stepped is preferred approach. Simulation time would then jump from one event time-stamp to the next without representing all of the values in between. This can save a huge amount of processing and allow the simulation to finish in a much shorter period of time.

Disadvantages

i. In interactive simulations with many events occurring simultaneously, an additional mechanism must be implemented to notify objects when other objects have done something that is interesting to them. Though this mechanism works, it is rather cumbersome and can still miss notifying some events that you are interested in. That is why most interactive simulations are structured as time-stepped simulations [4].

Time-stepped simulations seem to offer the best potential for military simulations, since all computational activity for a time-step can be performed concurrently. However, it is important that the time-step be large enough to allow a significant amount of computation.

2.1.2 Entity Level and Aggregate Level

Military simulations are often differentiated based on their inherent level of abstraction. If the primary objects represented in the simulation are collections of doctrinally identifiable military assets, e.g. a tank battalion, then the simulation is referred to as an *aggregate-level simulation.* If the primary objects represented by the simulation are singular military objects, e.g. a tank, a soldier, the simulation is referred to as an *entity-level simulation.* Historically, entity-level simulations have been used to train at lower military echelons, e.g. Platoon, Company, and aggregate-level simulations have been used as the basis for training at higher military echelons, e.g. Brigade, Division. Different synchronization algorithms are applicable for different levels of war games.

2.2 State of the Art in Distributed Simulation

Distributed simulation technology has developed largely independently in at least three separate communities namely:

- The high performance computing community
- The defense community
- The Internet/gaming industry

This thesis work is mainly restricted to its application in the defense community. Many projects were undertaken which resulted in Standards which were later followed in the development of future projects.

2.2.1 SIMulator NETworking (SIMNET)

In 1988 the Defense Advanced Research Projects Agency (DARPA) initiated Simulator Networking (SIMNET) program to create multiple tank simulators that could be joined over a network such that each could detect, engage, and destroy the others. This program resulted in the establishment of important principles for simulation interaction and the creation of a network messaging protocol to exchange essential data.

2.2.2 Distributed Interactive Simulation (DIS)

DIS attempted to generalize the SIMNET technology so that it could be applied to a wider variety of combat vehicle simulators such as trucks, helicopters, fighters, ships, and soldiers.

Distributed Interactive Simulation (DIS) provides the infrastructure to build large-scale simulations by interconnecting independent simulators via a network. These simulators execute primarily in real time because of the human-in-the-loop requirements [25].

In DIS, there is no central computer. Instead, a number of computers are interconnected via a network as shown in Figure 2.2

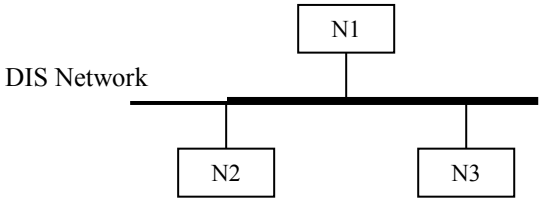


Figure 2.3 DIS Network

Each node is only responsible for the entity or entities it is simulating, and does not have to calculate what other nodes are interested in.

2.2.3 Aggregate Level Simulation Protocol (ALSP)

Aggregate Level Simulation Protocol (ALSP) was developed in 1990's. It applied the SIMNET concept of interoperability and model reuse to wargame simulations. It demonstrated interoperable training at the staff level. ALSP linked seven existing simulations from each military service by providing both the network messages and software services for insuring consistency and causality between the simulations. Time stepped being the most suitable approach for war game simulations was used in ALSP. Before stepping to the next time-step, each wargame would post an "Advance Request" to the infrastructure. These would be exchanged between all members of the virtual world and evaluated to determine the lowest time that was being requested from all of the simulations. Each instance of the infrastructure would then turn around 3and give its simulation an "Advance Grant" for the smallest time [23]. Simulations that wanted to advance to that time would do so. Others would wait for an advance grant for the time they had requested. Those that did have events for the time granted would process them and post another "Advance Request" for a time further in the future than the last one. As this continued simulation time would advance, each simulation would be able to process its events, and each simulation would have control over the rate at which the distributed time progressed. This approach to event management and synchronization became part of the Aggregate Level Simulation Protocol (ALSP) that is used to tie distributed wargames together.

2.2.4 High Level Architecture(HLA)

The Defense Modeling and Simulation Office (DMSO) developed the High Level Architecture (HLA) in the mid 1990's to replace both DIS and ALSP [29]. Those

methods have proven to be very system specific and do not provide a general interoperability solution that can support future simulation systems and missions.

It is intended to promote reuse and interoperation of simulations. The HLA effort was based on the premise that no one simulation could satisfy all uses and applications for the Defense community. The intent of the HLA is to provide a structure that supports reuse of different simulations, ultimately reducing the cost and time required to create a synthetic environment for a new purpose.

Apart from defense simulations the HLA is applicable across a broad range of simulation application areas, including education and training, analysis, engineering and even entertainment, at a variety of levels of resolution.

The HLA does not prescribe a specific implementation, nor does it mandate the use of any particular set of software or programming language. It was envisioned that as new technological advances become available, new and different implementations would be possible within the framework of the HLA.

An HLA *federation* consists of a collection of interacting simulations, termed *federates*. A *federate* may be a computer simulation, a manned simulator or an interface to a live player or instrumented facility. All object representation stays within the federates. The HLA imposes no constraints on what is represented in the federates or how it is represented, but it does require that all federates incorporate specified capabilities to allow the objects in the simulation to interact with objects in other simulations through the exchange of data.

Data exchange and a variety of other services are realized by software called the *Runtime Infrastructure (RTI)* [6]. The RTI is, in effect, a distributed operating system for the federation. The RTI provides a general set of services that support the simulations in carrying out these federate-to-federate interactions and federation management support functions. All interactions among the federates go through the RTI. The HLA runtime interface specification provides a standard way for federates to interact with the RTI, to invoke the RTI services to support runtime interactions among federates and to respond to requests from the RTI. This interface is

implementation independent and is independent of the specific object models and data exchange requirements of any federation. There are six classes of services namely [15]:

- *Federation management.* This includes services to create and delete federation executions, to allow simulations to join or resign from existing federations, and to pause, checkpoint, and resume an execution.
- *Declaration management.* These services provide the means for simulations to establish their intent to publish object attributes and interactions, and to subscribe to updates and interactions produced by other simulations.
- *Object management.* These services allow simulations to create and delete object instances, and to produce and receive individual attribute updates and interactions.
- *Ownership management.* These services enable the transfer of ownership of object attributes (and the right to modify the value of these attributes) during the federation execution.
- *Time management.* These services coordinate the advancement of logical time, and its relationship to wallclock time during the federation execution.
- *Data distribution management.* These services allow large federations to improve the efficiency of the data distribution mechanisms by reducing the amount of data that is sent between federates.

2.3 Distributed Simulation Architecture Options

Two widely used architectures for distributed simulation are the *client-server* and the *peer-to-peer* approaches.

- *Client-server approach.* It involves executing the distributed simulation on one or more server computers (which may be several computers connected by a local area network) to which clients (e.g., users) can "log in" from remote sites. The bulk of the simulation computation is executed on the server machines. This approach is typically used in distributed simulations used for multi-player gaming. Centralized management of the simulation computation greatly simplifies management of the distributed simulation system, and facilitates monitoring of the system, e.g., to detect cheating.

- *Peer-to-peer systems*. These systems have no servers, and the simulation is distributed across many machines, perhaps interconnected by a wide area network as shown in Figure 2.3. The peer-to-peer approach is often used in distributed simulations used for defense.

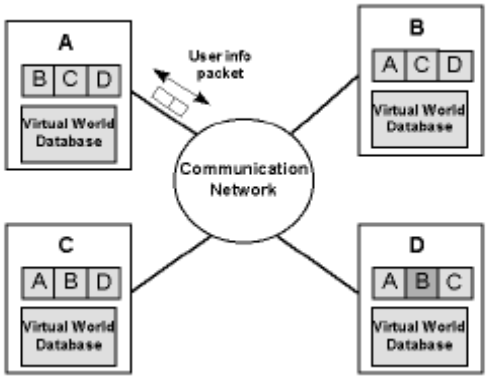


Figure 2.4 Peer-to-Peer Approach

2.4 Synchronization Problem

In distributed simulation the physical system can be viewed as a collection of physical processes that interacts in some fashion with each physical process being modeled by a *logical process (LP)*. Different LPs may execute concurrently on different processors. As each logical process executes, it processes events and generates new events. LPs communicate by exchanging timestamped events or messages. Each LP is essentially a sequential, event-driven simulator that computes the portion of the sample path pertaining to the subsystem it models. Sending an event to a logical process is equivalent to scheduling that event in the receiving processor's list of pending events. A key question, referred to as the *synchronization problem*, is ensuring that the events are processed in a correct order i.e. to make sure that an event with minimum timestamp is executed first. This can be difficult since LP can execute local generated events as well as events from remote LPs.

Two classes of synchronization algorithms have been proposed. Conservative algorithms guarantee that dependent events are never processed out of order, i.e., synchronization errors never occur. Optimistic algorithms allow synchronization errors to occur, but provide a mechanism to recover. These are described in Chapter 3.

2.5 Advantages

- Reduction of length of time to execute the simulation. By distributing the execution of a computation across N processors, one can complete the computation up to N times faster than if it were executed on a single processor.
- Larger simulations can be executed than could be executed on a single computer since memory of many computer systems can be utilized.
- Several different simulators can be integrated into a single simulation environment. For Example in military training- tank simulators, flight simulators, computer generated forces, and a variety of other models may be used to create a distributed virtual environment into which personnel are embedded to train for hypothetical scenarios and situations.
- An infrastructure simulation is an emerging field where simulators of different subsystems in a modern society are combined to explore dependencies among subsystems. For example, simulations of transportation systems may be combined with simulations of electrical power distribution systems, computer and communication infrastructures, and economic models to assess the economic impact of natural or human-caused disasters. In both these domains (military and infrastructure simulations) it is far more economical to link existing simulators to create distributed simulation environments than to create new models within the context of a single tool or piece of software. The High Level Architecture (HLA) developed by the U.S. Department of Defense defines an approach to integrate, or federate, separate, autonomous simulators into a single, distributed simulation system.
- Distributed simulations are executed over broad geographic areas. This is particularly useful when personnel and/or resources (e.g., databases or specialized facilities) are included in the distributed simulation exercise. Distributed execution eliminates the need for these personnel and resources to be physically co-located, representing an enormous cost savings.

2.7 Concluding Remarks

Simulation is an engrossing activity. It requires careful analysis of the systems, of the problems or requirements at hand. Moreover, “Models are not universally useful, but

are designed for specific purposes" [Roger Smith, 1999, Computer Game Developer's Conference]. Thus Modeling & Simulation (M&S) activity is a customized, "mostly non reusable", time consuming and hence, a costly affair. Secondly, M&S cannot prove something on its own; it is not an intelligent and automated process which can act on 'fire and forget' basis to solve any kind of problem we may imagine. Therefore, Modeling and Simulation should not be considered as a magic wand which can do wonders on its own, it must be supplemented by real exercises. It requires actual trial data and real parametric values to verify the results and to establish the validity of models used to achieve those results. It only provides us with suggestive facts, figures and visualizations based upon the inputs, the model used, and the level of details we tried to incorporate in the model.

Distributed simulations are those applications that span multiple computer devices, executables, or geographic areas. These include what are often referred to as parallel and distributed simulations (PADS) and distributed interactive simulations (DIS). These communities vary widely in their techniques for implementing a distributed simulation, but they both fall under the general category of distributed simulation.

Distributed simulation is widely applied in military training systems in which computers and executables have been joined together through techniques like the Distributed Interactive Simulation (DIS) protocol, Aggregate Level Simulation Protocol (ALSP), and the High Level Architecture (HLA). It is also used in analytical models in which networked and parallel computers divide a problem into smaller pieces that can be solved more rapidly. The entertainment community has applied distributed simulation ideas in attractions like the Battle Tech Entertainment Center and the Internet-based Virtual Worlds environment.

Chapter 3

Time Management

Time management is concerned with ensuring that the execution of the distributed simulation is properly synchronized. Time management not only ensures that events are processed in a correct order, but also helps to ensure that repeated executions of a simulation with the same inputs produce exactly the same results.

Time management algorithms usually assume the simulation consists of a collection of *logical processes (LPs)* that communicate by exchanging timestamped messages or events. The goal of the synchronization mechanism is to ensure that each LP processes events in timestamp order. This requirement is referred to as the *local causality constrain*t. For example, if events E1 and E2 occur at times T1 and T2, and where T1 < T2, and E1 writes into a state variable that is read by E2, then E1 must be executed before E2. Ignoring events containing exactly the same time stamp, it can be shown that if each LP adheres to the local causality constraint, execution of the simulation program on a parallel computer will produce exactly the same results as an execution on a sequential computer where all events are processed in time stamp order. This property also helps to ensure that the execution of the simulation is repeatable.

3.1 Need for Time Management

Time Synchronization is very important for successful implementation of a distributed Simulation application. An example below illustrates the case when time synchronization is not taken into consideration and simulators are just combined together. This can lead to problems.

Consider a real-time simulation consisting of three simulators modeling a tank, a target, and a observer. Suppose the tank simulator fires upon the target and destroys it, while the observers monitor this exchange as shown in figure 3.1. The tank simulator first generates a message indicating it has fired. This message is received by the target simulator which generates a message indicating this fact. The observer federate receives both messages, but the message containing the tank firing event has been delayed in the network, causing the target destroyed event to be received first. Thus,

the observer sees the target being destroyed before it has been fired upon! Thus the temporal relationships between the simulators in the real world are not reproduced correctly. In the simulated world, delays are encountered by a message as it travels through the network. This can lead to anomalies such as the cause appearing to happen *after* the effect.

Another problem can be different views of the real world by different simulators. This can be as a result of one simulator receiving messages late then the other simulator due to network congestion.

Another problem is that repeated executions of the simulation with the same initial state and external inputs may produce entirely different results. One reason for this is the order that messages are delivered to each federate depends on properties such as communication delays in the network that may change from one execution to the next. It is sometimes important that repeated executions of the simulation yield the same results. For example, in defense simulations, the results produced by the simulator may be used to make acquisition decisions. Simulation results may be audited by the General Accounting Office, which may require the simulations to be repeated, e.g., to verify their authenticity.

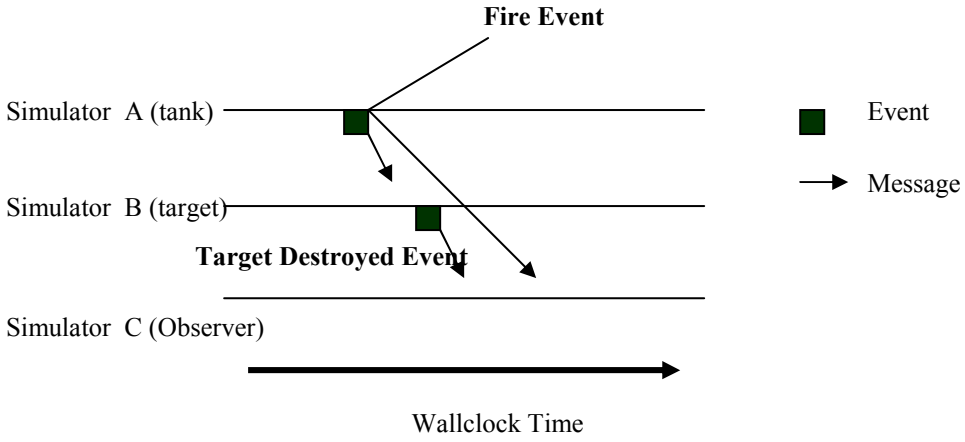


Figure 3.1. Need for Time Management:
The observer simulator receives the "target destroyed" message before the "fire" message.

3.2 Problem Statement

Synchronization is of utmost importance in the design of a distributed simulation system, since the reliability of the results generated by a distributed algorithm depends on the accuracy of the synchronization mechanism used. However unlike sequential simulation i.e. simulation that takes place on a single processor, the need for synchronization in a distributed computing environment creates increased overhead for the processors (synchronization overhead). Therefore efficiency and speed of distributed simulation is highly dependent on the type and quality of the synchronization mechanism.

The need for synchronization of multiple processes emerges naturally when several simulation processes are run on different processors. In a sequential simulation environment the common mechanism used for simulation is event driven execution. Events are created and inserted into 'future event list' which is sorted by the scheduled time of events. At each execution step the simulation engine executes the next event on the event list, removes the event from the event list and updates the simulation clock to the time of the event that is executed. Since all events are ordered in non-decreasing timestamp order in the event list, the simulation clock increases monotonically. Therefore, at any time during these executions, none of the events in the event list can have a timestamp less than simulation clock.

When multiple interacting simulation processes are executed over several processors, interactions among these processes cause external events to be inserted into the event lists of remote simulation processes. This is done via passing messages among the processes. Each message schedules a new event at the destination process. Therefore if simulation clocks of different processes are advancing independently during an execution, it is possible that process will try to schedule a remote event at another process using a message with a timestamp less than local clock of the receiving process. In this case the execution of the receiving process becomes erroneous, because the external event occurred in the past of the receiving process and was not executed at the correct time. Such events are called straggler events and this situation is known as *time ambiguity*. Two main classes of algorithms have been developed namely Conservative and Optimistic (Section 3.3). Briefly, conservative algorithms

take precautions to avoid the possibility of processing events out of time stamp order, i.e., the execution mechanism avoids synchronization errors. On the other hand, optimistic algorithms use a detection and recovery approach. Events are allowed to be processed out of time stamp order, however, a separate mechanism is provided to recover from such errors.

Thus the main objective of this research work is to provide an efficient solution for time synchronization among multiple simulation processes. This solution will solve the time ambiguity problem discussed above. The time synchronization approach is specifically meant for time stepped simulations where events are executed at each time step. The solution should be developed taking the following issues under consideration:

- Number of Simulation Processes involved.
- Number of timing signals sent.
- Need of Acknowledgement from the simulation processes participating.

3.3 Time Synchronization Algorithms

3.3.1 Conservative Time Management

The first synchronization algorithms were based on conservative approaches. The principal task of any conservative protocol is to determine when it is "safe" to process an event. An event is said to be *safe* when one can guarantee no event containing a smaller time stamp will be later received by this LP. Conservative approaches do not allow an LP to process an event until it has been guaranteed to be safe.

At the heart of most conservative synchronization algorithms is the computation for each LP of a *Lower Bound on the Time Stamp (LBTS)* of future messages that may later be received by that LP. This allows the mechanism to determine which events are safe to process. For example, if the synchronization algorithm has determined that the LBTS value for an LP is 12, then all events with time stamp less than 12 are safe, and may be processed. Conversely, all events with time stamp larger than 12 cannot be safely processed.

The algorithms developed by Chandy, Misra and Bryant in 1978 were among the first synchronization algorithms that were developed. They assume the topology indicating which LPs send messages to which others is fixed and known prior to execution. It is assumed each LP sends messages with non-decreasing timestamps, and the communication network ensures that messages are received in the same order that they were sent. This guarantees that messages on each incoming link of an LP arrive in timestamp order. This implies that the timestamp of the last message received on a link is a lower bound on the timestamp of any subsequent message that will later be received on that link. Thus, the LBTS value for an LP is simply the minimum among the LBTS values of its incoming links [27].

Their basic algorithm is given in figure 3.3. As shown in figure 3.2, each LP keeps a static FIFO channel for each LP with incoming communication. Local events scheduled within the LP can be handled by having a queue within each LP that holds messages sent by an LP to itself.

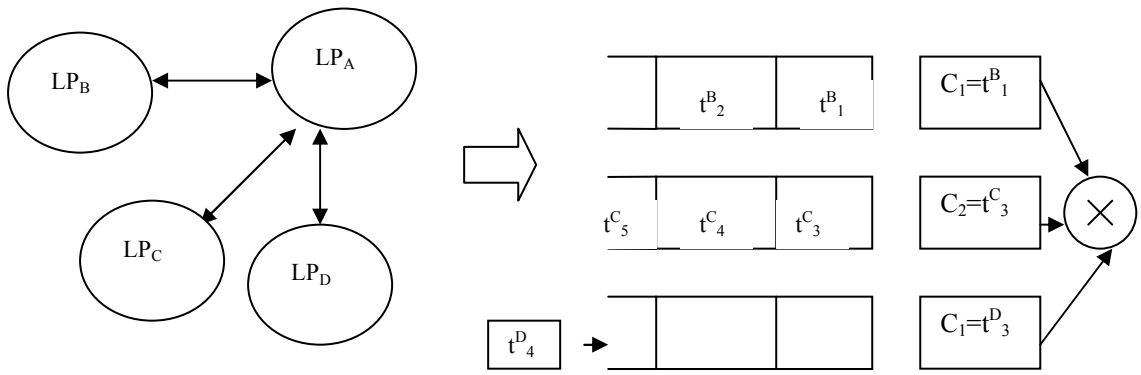


Figure 3.2 FIFO queues in Chandy-Misra-Bryant Algorithm

Each FIFO channel (input channel, IC) has a clock C_i that ticks according to the timestamp of the topmost message, if any; otherwise it keeps the timestamp of the last message

The process repeatedly selects the link with the smallest clock and, if there is a message in that link's queue, processes it. If the selected queue is empty, the process blocks. The LP never blocks on the queue containing messages it schedules for itself, however. This protocol guarantees that each process will only process events in non-decreasing timestamp order.

The Chandy-Misra-Bryant Algorithm

```
while (simulation is not over)
 {
   determine the ICi with the smallest Ci
   if (ICi empty)
      wait for a message
   else
     {
      remove topmost event from ICi
      process event
     }
 }
```

Figure 3.3 The Chandy-Misra-Bryant Algorithm

Although this approach ensures the local causality constraint is never violated, it is prone to deadlock. A cycle of empty links with small link clock values (e.g., smaller than any unprocessed message in the simulator) can occur, resulting in each process waiting for the next process in the cycle. If there are relatively few unprocessed messages compared to the number of links in the network, or if the unprocessed events become clustered in one portion of the network, deadlock may occur very frequently.

Null messages are used to avoid deadlock. A null message with timestamp Tnull sent from LP_A to LP_B is a promise by LP_A that it will not later send a message to LP_B carrying a timestamp smaller than Tnull. Null messages do not correspond to any activity in the simulated system; they are defined purely for avoiding deadlock situations. Processes send null messages on each outgoing link after processing each

event. A null message provides the receiver with additional information that may be used to determine that other events are safe to process. Null messages are processed by each LP just like ordinary non-null messages, except no activity is simulated by the processing of a null message. In particular, processing a null message advances the simulation clock of the LP to the time stamp of the null message. However, no state variables are modified and no non-null messages are sent as the result of processing a null message.

The process determines the timestamps of the null messages it sends by looking at the clock value of each incoming link. It provides a lower bound on the timestamp of the next event that will be removed from that link's buffer. When coupled with knowledge of the simulation performed by the process, this bound can be used to determine a lower bound on the timestamp of the next *outgoing* message on each output link. Whenever a process finishes processing a null or non-null message, it sends a new null message on each outgoing link. The receiver of the null message can then compute new bounds on its outgoing links, send this information on to its neighbors, and so on. This algorithm avoids deadlock.

The null message algorithm introduced a key property utilized by virtually all conservative synchronization algorithms: ***lookahead.***
Lookahead: If an LP is at simulation time T, and it can guarantee that any message it will send in the future will have a time stamp of at least T+L regardless of what messages it may later receive, the LP is said to have a lookahead of L. Lookahead is used to generate the time stamps of null messages. One constraint of the null message algorithm is it requires that no cycle among LPs exist containing zero lookahead, i.e., it is impossible for a sequence of messages to traverse the cycle, with each message scheduling a new message with the same time stamp.

3.3.1.1 Advantages of Chandy-Misra-Bryant (CMB) Algorithm

- Its base methodology is simple.
- The algorithm is straightforward to implement with simple control and data structures.
- CMB performs well as long as all static channels are equally utilized.

- Large dispersion of events in space and time is not bothersome.
- Memory consumption is conservative.
- Good performance when lookahead is large

3.3.1.2 Disadvantages of Chandy-Misra-Bryant (CMB) Algorithm

- This algorithm is pessimistic in many cases
- Large lookahead is essential for performance
- It may generate an excessive number of null messages. Consider a simulation containing two LPs. Suppose both are blocked, each has reached simulation time 100, and each has a lookahead equal to 1. Suppose the next unprocessed event in the simulation has a time stamp of 200. The null message algorithm will result in null messages exchanged between the LPs with time stamp 101, 102, 103, and so on. This will continue until the LPs advance to simulation time 200, when the event with time stamp 200 can now be processed. A hundred null messages must be sent and processed between the two LPs before the non-null message can be processed. This is clearly very inefficient. The problem becomes even more severe if there are many LPs.
- High cost of maintaining time synchronicity on a distributed system.
- It does not maximize parallelism because of single point of timing control
- Synchronization mechanism is processor *blocking,* as a consequence prone to deadlock situations. The processing can not easily handle output of the type where generated event times are not monotonically increasing (time delay generated events). This is because input event times are used by the LPs to generate the LVTH to know whether to continue processing events.

3.3.2 Optimistic Time Management

In contrast to conservative approaches that avoid violations of the local causality constraint, optimistic methods allow violations to occur, but are able to detect and recover from them.

The **Time Warp mechanism** (Jefferson 1985) is the most well known optimistic method. LPs send timestamped messages, not necessarily in non-decreasing time stamp order. There are no static communication channels between LPs, hence

dynamic creation of LPs is easy. Each LP processes events as they are received, no need to wait for safe events

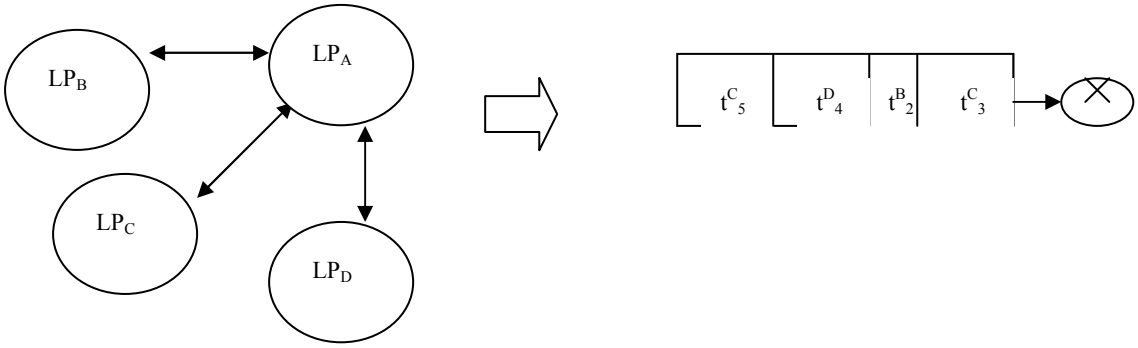


Figure 3.4 FIFO queues in Time Warp Algorithm

Local causality violations are detected and corrected at runtime. When an LP receives an event with timestamp smaller than one or more events it has already processed, it rolls back and reprocesses those events in timestamp order. Rolling back an event involves restoring the state of the LP to that which existed prior to processing the event (checkpoints are taken for this purpose), and "unsending" messages sent by the rolled back events. An elegant mechanism called *anti-messages* is provided to "unsend" messages. An anti-message is a duplicate copy of a previously sent message. Whenever an anti-message and its matching (positive) message are both stored in the same queue, the two are deleted (annihilated). To "unsend" a message, a process need only send the corresponding anti-message. If the matching positive message has already been processed, the receiver process is rolled back, possibly producing additional anti-messages. Using this recursive procedure all effects of the erroneous message will eventually be erased.

Two problems remain to be solved before the above approach can be viewed as a viable synchronization mechanism [4].

- Certain computations, e.g., I/O operations, cannot be rolled back.
- The computation will continually consume more and more memory resources because a history (e.g., checkpoints) must be retained, even if no roll-backs occur; some mechanism is required to reclaim the memory used for this history information.

Both problems are solved by *global virtual time (GVT).* GVT is a lower bound on the timestamp of any future rollback. GVT is computed by observing that rollbacks are caused by messages arriving "in the past." Therefore, the smallest time-stamp among unprocessed and partially processed messages gives a value for GVT. Once GVT has been computed, I/O operations occurring at simulated times older than GVT can be committed, and storage older than GVT (except one state vector for each LP) can be reclaimed. GVT computations are essentially the same as LBTS computations used in conservative algorithms. This is because rollbacks result from receiving a message or anti-message in the LP's past. Thus, GVT amounts to computing a lower bound on the time stamp of future messages (or anti-messages) that may later be received.

3.3.2.1 State Saving Techniques

There are three main software state-saving techniques:

- ***Copy state saving***. State of a process is saved before the execution of each event. This technique has short state recovery because any state to be restored due to rollback is immediately available. However it suffers high state saving cost and large memory size.

- ***Sparse state saving***. State is saved periodically. The period may be a certain number of event executions of a time interval. The advantage of this approach is the cost of state saving is reduced compared with copy sate saving. However during the state recovery, it causes a higher cost because of the re-execution of intermediate events starting from the last state saved. Some checkpoint intervals can improve the performance. However if the interval is poorly selected, the performance may be worse than copy state saving. Statically deciding the optimal interval to maximize the performance is very difficult. Several methods were proposed to adaptively select the interval

- ***Incremental state saving***. Only the portion of the state that has changed is saved. Incremental state saving requires users to write code to assist saving and restoring the states because it is hard to automatically detect the state changes. The advantages that these incremental state saving methods provide are smaller memory requirements and memory copy cost. The limitation is that the method is only applicable to the situation where there is large state size and each event

execution only changes a small portion of state. It may cause higher state saving and recovery cost because of extra computation required to log the state changes.

3.3.2.2 Advantages of Optimistic over Conservative Techniques.

- They can exploit greater degrees of parallelism. If two events *might* affect each other, but the computations are such that they actually don't, optimistic mechanisms can process the events concurrently, while conservative methods must sequentialize execution.
- Conservative mechanism generally rely on application specific information (e.g., distance between objects) in order to determine which events are safe to process. While optimistic mechanisms can execute more efficiently if they exploit such information, they are less reliant on such information for correct execution. This allows the synchronization mechanism to be more transparent to the application program than conservative approaches, simplifying software development.
- Interactive simulations can be enabled.
- Can be hopefully general-purpose.

3.3.2.3 Disadvantages of Optimistic Techniques.

- Optimistic methods may require more overhead computations (state saving, GVT, rollbacks) than conservative approaches, leading to certain performance degradations.
- Potentially large amount of memory is required.
- Optimistic simulation executives are more complex.

3.4 Conservative Vs Optimistic for Military Simulations

It is necessary for certain logical processes to be able to predict their future message-passing behavior far enough into the future to allow some other logical process to advance its clock (deadlock avoidance protocols that rely on prediction demonstrate only the sufficiency of behavior prediction). The ability to predict future behavior is very limited in battlefield simulations, implying that the synchronization constraints and overhead of avoiding deadlock are likely to adversely affect performance. The Time Warp mechanism of rolling back clocks avoids the behavior prediction problem, but does so at the cost of extensive memory requirements, and the potential threat of

having rollback “thrashing”. While Time Warp is an aesthetically pleasing idea, its utility on large real-world problems has yet to be empirically demonstrated.

Synchronization is a well-studied area of research in the distributed simulation field. There is no clear consensus concerning whether optimistic or conservative synchronization performs better; indeed, the optimal approach usually depends on the application. In general, if the application has good lookahead characteristics and programming the application to exploit this lookahead is not overly burdensome, conservative approaches are the method of choice.

3.5 Time Management in the HLA

The HLA provides a set of services to support time management. A principal consideration in defining these services was the observation that different federates may use different local time management mechanisms and have different requirements for message ordering and delay. Two major categories emerged. One class of simulations were designed to created virtual environments for training and test and evaluation (e.g., hardware-in-the-loop) applications. The execution of these simulations is paced by wallclock time, and synchronization algorithms to guarantee time stamp ordering of events are typically not used. Achieving low, predictable delays to transmit messages are important. A second class of simulations are those that require synchronization algorithms, in part to ensure proper ordering of events, and in part as a means to ensure that executions are repeatable, i.e., multiple executions of the same simulation with the same inputs yield exactly the same results. These simulations may use event stepped or time stepped execution mechanisms locally. It was envision that some federates may be executing on a parallel processor, and may be using conservative or optimistic synchronization mechanisms within their federate. The HLA time management services were designed to accommodate this wide variety of applications.

3.5.1 Message Ordering Mechanisms

Message ordering characteristics specify the order and time at which messages may be delivered to federates and are central to the HLA time management services. The following five message ordering mechanisms were specified in the first version of the HLA Time Management design document [30]:

- *Receive Order.* Messages are passed to the federate in the order that they were received. Logically, incoming messages are placed at the end of a first-in-first-out (FIFO) queue, and are passed to the federate by removing them from the front of this queue. This is the most straightforward, lowest latency ordering mechanism.

- *Priority Order.* Incoming messages are placed in a priority queue, with the message time stamp used to specify its priority. Messages are passed to the federate lowest time stamp first. This service does not prevent a message from being delivered in a federate's "past" (time stamp less than the federate's current time), but it is less costly in terms of latency and synchronization overhead than the time stamp ordered delivery mechanism. Priority order with best effort delivery may be used for federates where sequences of messages require ordering, but the increased latency associated with either reliable delivery or guaranteed order cannot be tolerated. For example, speech packets may utilize this service.

- *Causal order.* This service guarantees that if an event E "causally precedes" another event F, then any federate receiving messages for both events will have the message for E delivered to it before the message for F. For example, E and F might indicate firing a weapon, and the target being destroyed, respectively; if causal ordering is used, a federate observing both events will be notified of the fire event before it is notified of the destroyed event.

- *Causal and totally ordered.* In the causally ordered service defined above, messages corresponding to events that are *not* causally related (referred to as concurrent events) may be delivered to federates in any order. The causal and totally ordered service extends causal ordering to guarantee that for any pair of concurrent events, messages for these events will be delivered to all federates receiving both messages *in the same orde*r, thereby defining a total ordering of events. This service is commonly referred to as CATOCS (causally and totally ordered communications support)

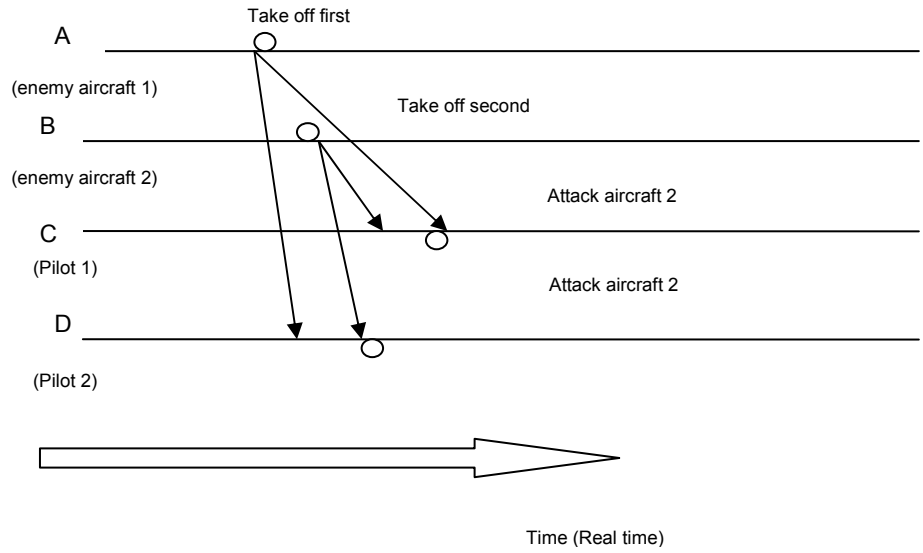


Figure 3.5. Scenario demonstrating causal and total ordering.

Figure 3.5 illustrates where CATOCS may be useful. Two federates (A and B) modeling enemy aircraft are taking off from an air field. Two pilots are assigned to intercept, with pilot 1 (federate C) given orders to attack the first enemy aircraft to take off, and pilot 2 (federate D) assigned to attack the second. Assume the "take-off" events are concurrent, e.g., the aircraft are taking off from different runways. Without total ordering, messages for the take-off events may arrive at the two federates in different orders. Figure 4.1 shows a scenario where pilot 1 incorrectly believes aircraft 2 took off first, while pilot 2 correctly believes aircraft 1 took off first. The end result is both pilots attack aircraft 2! CATOCS would circumvent this anomaly by ensuring that both pilots see the same ordering of events. Both pilots may perceive an incorrect order (e.g., both might believe aircraft 2 took off first if the messages from Federate A are delayed), but the result of this error is likely to be less severe than having both pilots attack the same aircraft.

- *Time stamp order (TSO).* Messages utilizing this service will be delivered to federates in time stamp order. Further, the RTI also ensures that no message is delivered to a federate "in its past," i.e., no TSO message is delivered that contains a time stamp less than the federate's current time. Time stamp order is normally used for analysis applications which are often not paced by wallclock time where correct ordering of events and repeatable execution are

important. A conservative synchronization protocol is used to implement this service. All federates receiving messages for a common set of events will receive those messages in the same order, i.e., a total ordering of events is provided. The RTI provides a consistent tie breaking mechanism so messages containing identical time stamps will be delivered to different federates in the same order. Further, the tie-breaking mechanism is deterministic, meaning repeated executions of the federation will yield the same relative ordering of these events if the same initial conditions and inputs are used, and all messages are transmitted using time stamp ordering.

3.5.2 Object Management Services

The object management services include primitives to schedule and retract events, and primitives to receive messages. For example, **Update Attribute Values** and **Send Interaction** schedule events. The sender assigns a time stamp to the event to indicate when it is to occur, and specifies the category of transportation service (reliability and message ordering) that is to be used. The RTI delivers messages to the federate by invoking services that must be provided by the federate. Specifically, the RTI invokes the federate's **Reflect Attribute Values** and **Receive Interaction** services to deliver messages denoting state changes and interactions. *Event retraction* refers to the ability of a federate to retract or unscheduled a previously scheduled event. This is a common discrete-event simulation primitive often used to model interrupts and other preemptive behaviors. Event retraction is also utilized by optimistic federates to implement anti-messages.

The **Update Attribute Values** and **Send Interaction** services return an event handle that is used to specify the event that is to be retracted. If the RTI at the destination federate receives a retraction request for an event that is not buffered in the RTI (e.g., because the corresponding message has already been forwarded to the federate), the retraction request is forwarded to the federate.

3.5.3 Time Advance Services

The time advance primitives serve several purposes. First, it provides a protocol for the federate and RTI to jointly control the advancement of logical time. The RTI can

only advance the federate's logical time to T when it can guarantee that all TSO messages with time stamp less than or equal to T have been delivered to the federate. At the same time, the federate must delay processing any local event until logical time has advanced to the time of that event, or else it is possible it will receive a TSO message in its past.

The time management primitives also control the delivery of messages to the federate. TSO messages will not be delivered until the receiving federate has requested a time advance up to at least the time stamp of the message. In addition, the time management primitives provide information to the RTI that is used to synchronize the execution, as discussed below.

Two services for advancing logical time are defined: **Time Advance Request** and **Next Event Reques**t. Time Advance Request is intended to be used by time-stepped federates, and Next Event Request by event-driven federates. Each invocation of either primitive eventually results in the RTI calling **Time Advance Grant** to indicate that logical time has been advanced, and all TSO messages with time stamp less than or equal to the grant have been delivered [29].

Time Advance Request with parameter t requests an advance of the federate's logical time to t. When used in a time-stepped simulation, t will usually indicate the time of the next time step. Invocation of this service implies that the following messages are eligible for delivery to the federate: (i) all incoming receive ordered messages, and (ii) all messages using other ordering services with time stamp less than or equal to t. The federate may simply note the occurrence of these events for later processing, or immediately simulate actions resulting from the occurrence of the events. When the RTI can guarantee that it has passed all TSO messages to the federate with time stamp less than or equal to t, logical time is advanced to t, and the RTI calls the federate's **Time Advance Grant** primitive. At this point, a time-stepped federate may proceed to simulate the next time step.

Next Event Request with time parameter t requests an advance of logical time to t, *or* the time stamp of the next TSO message from the RTI, whichever is smaller. When used in an event driven federate, t will usually indicate the time stamp of the next local event within the federate. After the primitive is invoked, the federate will either

(i) deliver the next TSO message (and all other TSO messages containing exactly the same time stamp) if that message has a time stamp of t or less, and advance logical time to the time of that message, or (ii) not deliver any TSO messages and advance logical time to t. In either case, a **Time Advance Grant** is issued to indicate completion of the request. Other non-TSO messages may also be delivered as a result of invoking this primitive. If no TSO messages are delivered as a result of this request, this indicates to the federate that it may process its local event with time stamp t because no externally generated TSO messages with time stamp less than or equal to t are forthcoming.

3.5.4 Schemes for Time Management in HLA

HLA does not support the concept of wall-clock time for a federation as a whole. Each federate has a logical time and the purpose of time management is to coordinate the advance of the logical times of all federates.

The HLA allows flexibility in the way a particular federate handles time. It supports a number of time-management schemes, including the following [29]:

- **No time management:** Each federate advances time at its own pace

- **Event-driven:** The federate processes both local events and those generated by other federates in time-stamp order, and federate time typically advances to the time stamp of each event as it is processed.

- **Time-stepped:** Each time advance made by the federate is of a fixed duration of simulation time called a **time step**. The simulator does not advance to the next time step until all simulation activities associated with the current time step have been completed. *Activity scanning*, another time-flow mechanism sometimes used in discrete-event simulation, is essentially a variation of the time-stepped mechanism. Federates proceed through periods during which they exchange messages at the same time until they agree to advance their logical times together.

- **Parallel discrete-event simulation:** Federates executing on multiprocessor systems may be synchronized internally. A synchronization protocol used for that purpose can be either *conservative* or *optimistic*. In a conservative protocol, each *logical process* within the federate must process its events in time-stamp order. Federates advance time only when it can be guaranteed they will receive no past events. Optimistic synchronization protocols allow logical processes to process events out of time-stamp order but provide a means to recover from such errors, typically through the use of a roll-back mechanism.

- **Wallclock-time driven:** Wall-clock-time driven federates do not require that events be processed in time-stamp order. These simulations usually have hard and/or soft real-time constraints and so interactions with humans and/or physical devices occur in timely fashion.

Each federate determines its own degree of involvement in the time-management process. It can choose one of the following:

- *Not to participate* in time management. This is the default state of a federate when it joins a federation.
- To be *time-regulating*, in which case it is capable of generating time-stamp ordered (TSO) events.
- To be *time-constrained*, in which case it is capable of receiving TSO events.
- To be *both time-regulating and time-constrained*, in which case it can both generate and receive TSO events.

The important point about TSO events is that it is possible to determine the order in which they were sent, and they are delivered to the receiving federates in this order. Receive ordered (RO) events, on the other hand, are delivered in the order in which they are received and, lacking time stamps, it is impossible to determine the order in which they were sent. Since a federate that is not time-constrained cannot receive TSO events, an event that is sent with a time stamp (sent as a TSO event) will be received as a RO event, without the time stamp, if the receiving federate is not time-constrained. RO events bear no relationship to logical time.

3.6 Concluding Remarks

Time is especially important in distributed simulations used for analysis because of causality and repeatability. In a simulation running on a single process time is easy to deal with. Regardless of how time is dealt with, a single process can ensure that events occur in a logical sequence corresponding to real events being simulated. For a simulation running in multiple processes time is not a problem if the processes are running synchronously. That is if a mechanism exists to ensure that each process advances time in sync with every other process. However there can be a problem if multiple simulation processes advance time asynchronously. Under such circumstances an event simulated by one process can occur in the past of part of the simulation running in another process. Further complications can arise when multiple simulation processes are running on separate processors that are connected by a network. In this case there is the further problem of network latency to deal with. Time synchronization algorithms fall into Conservative and Optimistic. A synchronization algorithm should be chosen depending on the need of the application and the way in which time advances in the application. The HLA allows flexibility in the way a particular federate handles time. It supports a number of time-management schemes like No time management, time stepped or event driven. Every federate can select its own way of advancing time and can still synchronize with other federates. This is because HLA time management services were designed to accommodate a wide variety of applications.

Chapter 4

Synchronization Approaches for Distributed Time Stepped Simulations

Computer simulations are widely used by the scientific community and industry today for studying, analyzing and predicting the behavior of real-world systems. In this Chapter the focus will be on the time-stepped approaches that have been commonly used to synchronize events for war game simulation on a parallel or distributed platform.

In the conventional time-stepped approach, the time interval to be advanced after synchronization is performed is usually predetermined and is equal to the step-size. All participating entities in the simulation are at the same time-step at any point in wallclock time. Typically the entire span of simulation is divided into equal-sized time steps and the simulation advances from one time step to the next.

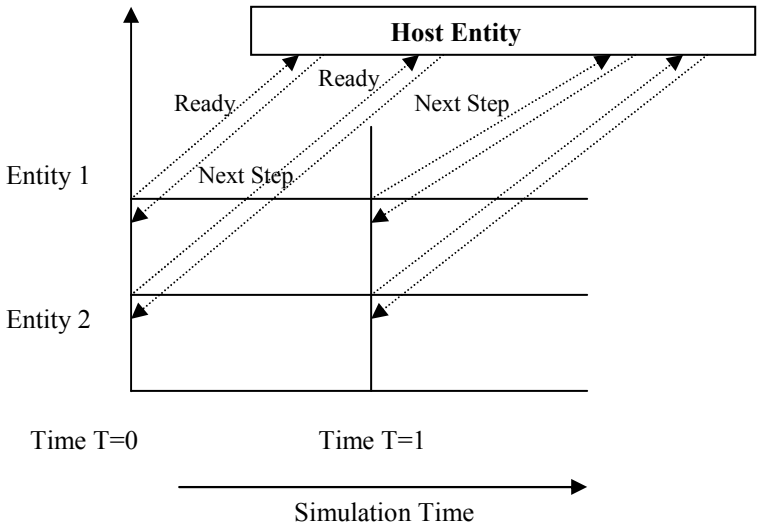


Figure 4.1 Simulation Progress in Traditional Time Stepped Simulations

In the real world the density of events can be highly varying depending on the period of time being simulated. Defense simulations have marked differences in event densities during time of war (high event density combined with need for fine

granularity and real-time response) as compared to peace time (relatively low density and response time is not mission critical). This thesis work is aimed at optimizing the synchronization for such simulations without affecting causality constraints and without violating any real-time constraints.

In this Chapter the focus will be on three synchronization approaches especially for time stepped simulations namely:

- Piggy-Backed Time Stepped Simulation
- Paced Time Stepped Synchronization
- Zero Lookahead Approach

Based on the pros and cons of these approaches a Synchronization algorithm based on Zero Lookahead approach is developed which is discussed in Chapter 4.

4.1 Piggy-Backed Time Stepped Simulation

4.1.1 Introduction

The optimization proposed in this technique is an attempt to apply some concepts from event-driven simulations to the traditional time-stepped mechanism. This approach is based on the fact that a lot of processing time is invested by the Processing Elements (PEs) at each time step even though some time steps may not contain any events. This technique informs the PE about a time-step only when there are events to be processed at that step. To achieve this, each PE informs the host (controller) about the future events spawned by it. Though this technique reduces the number of synchronization events compared to the conventional technique, it also introduces an overhead in terms of processing time at the host and at the participating PEs.

4.1.2 The Approach

To implement the piggy-backing technique, two lists are maintained namely Pending Event List (PEL) as in traditional time stepped approach and a new Future Time List (FTL) at the Host[24]. A simulation proceeding using the piggybacking technique involving two PEs and a central Host is illustrated below.

Let E1, E3 and E4 be the local entities residing at PE1 and E2 and E5 be the ones residing at PE2. To initiate the simulation the Host adds a *Start* event destined to all the participating PEs in its FTL and then dispatches it to all the PEs (Figure 4.2)

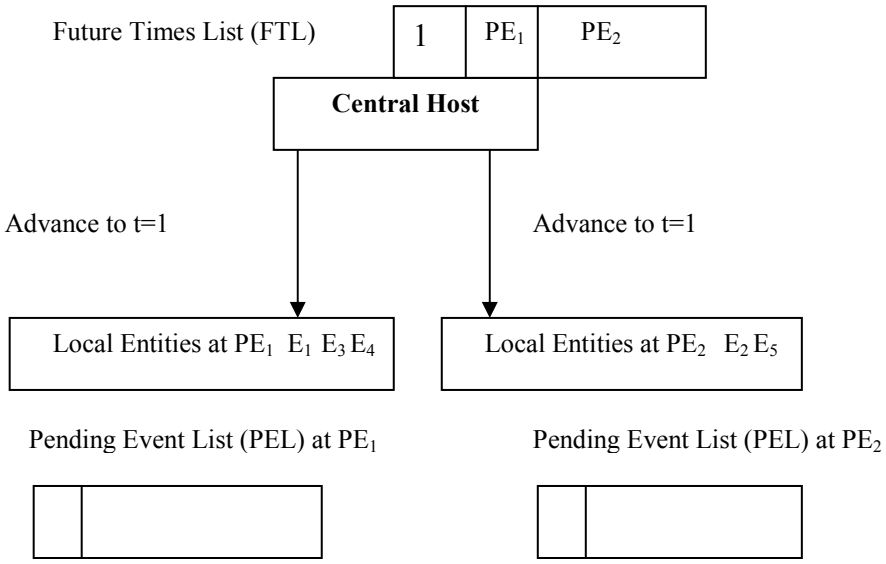


Figure 4.2 Initiation of the Simulation by the Host

When PEs receive the Start message from the host they scan through the PEL and execute the residing local entities. This might result in new events being generated which are meant to be processed during future times. The generated new events are added to the PEL.

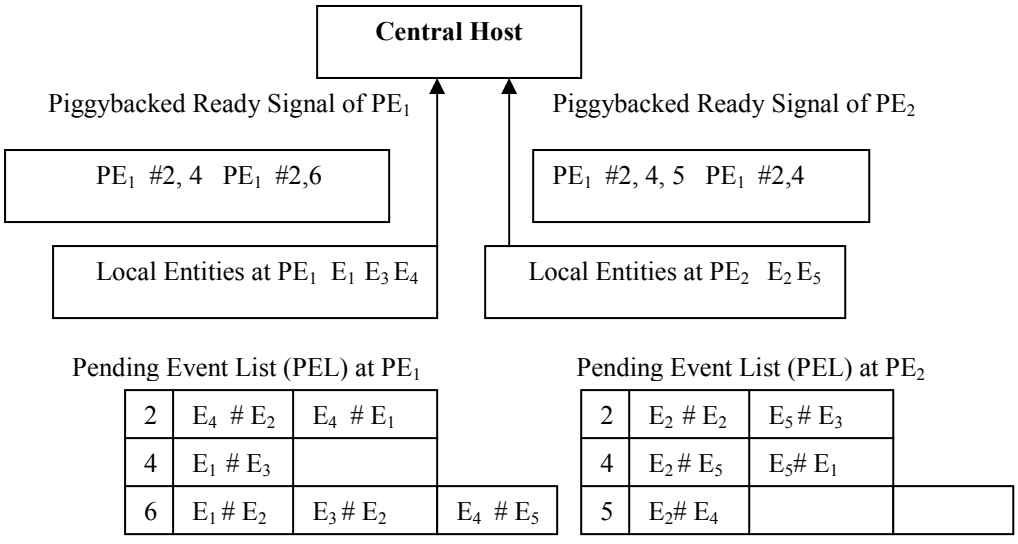


Figure 4.3 Piggy- Backed Ready Signals from the PEs

Once all the local entities have been executed the PE sends the *Ready* message to the host piggy-backed with information about the new events that have been spawned as shown in Figure 4.3

This information is the time-stamps that destination PEs need to be informed about in order to receive *Time Advance* signals from the Host corresponding to those time-stamps. The Host extracts this piggy-backed information on receiving the *Ready* message and updates its FTL, as shown in Figure 4.4.

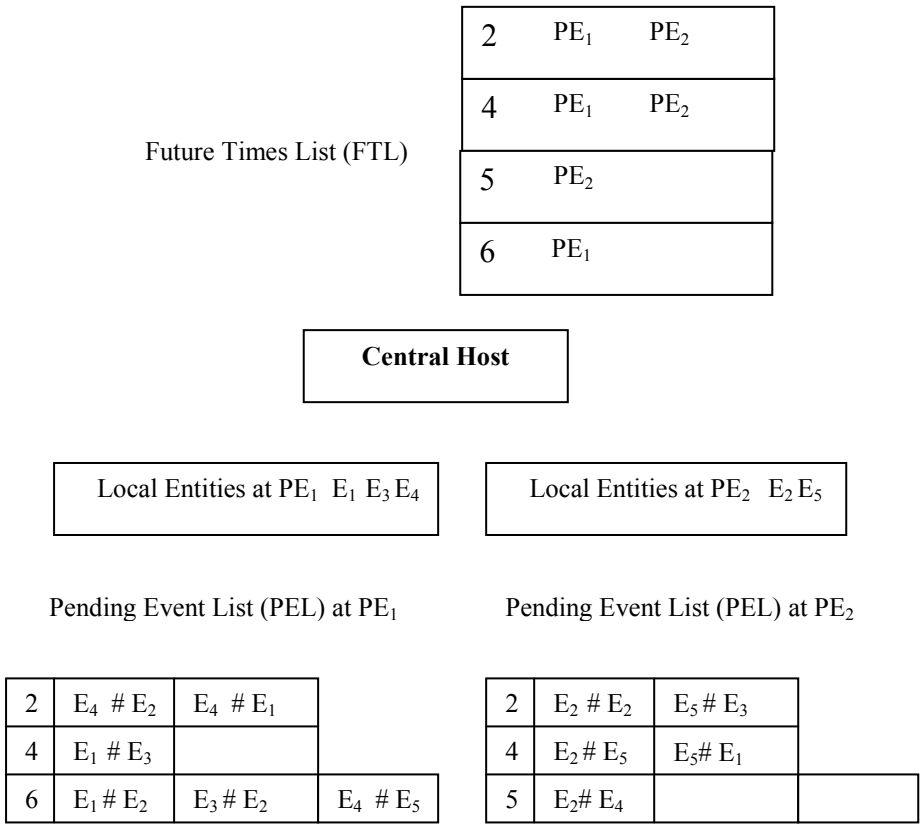


Figure 4.4 Updating of FTL by Host

The Host then sends *Advance to t* where *t* is the minimum time-stamp in its FTL, to those PEs corresponding to time *t* in the FTL. In this case, the Host sends an *Advance to t=2* message to PE1 and PE2, as shown in Figure 4.5. Once the PEs have been informed about the advance, the list corresponding to *t=2* in the FTL is emptied.

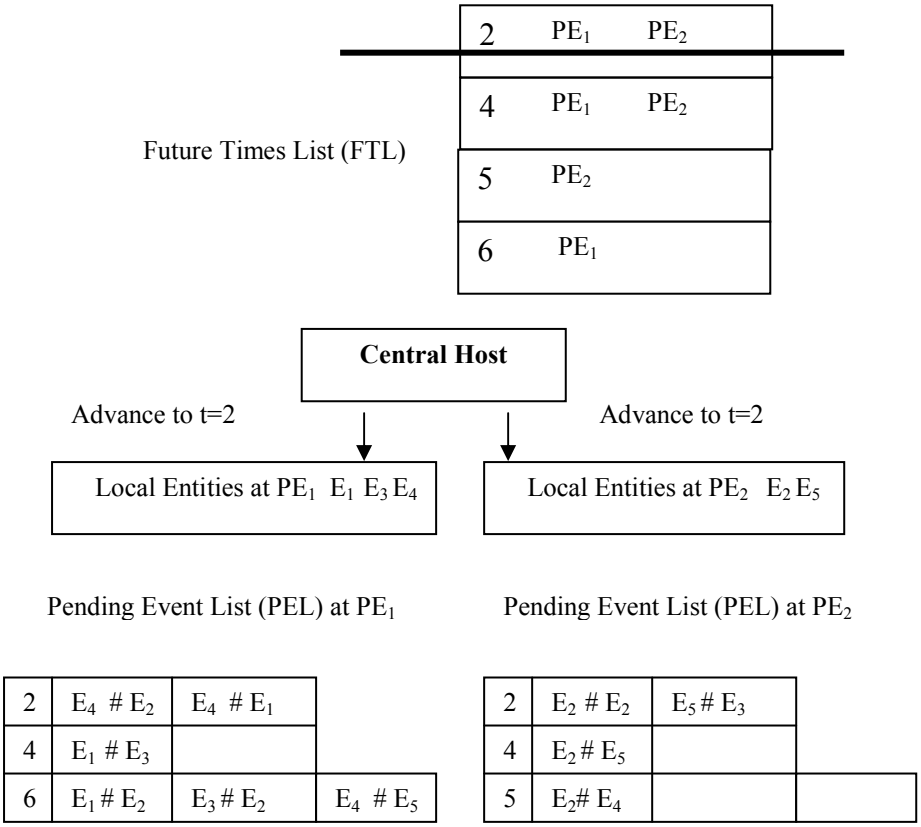


Figure 4.5 Advance Signals being sent to PEs

After receiving the *Advance* message from the Host, the PEs scan through the PEL, execute the residing local entities and then send the piggy-backed *Ready* signal to the Host again. Host only waits for *Ready* messages from those PEs that had been sent *Advance* messages during the previous time-step. When the Host receives information that it already possesses in its FTL it conveniently ignores the information.

4.1.3 Result

This technique introduces an optimization to traditional timestepped simulations, particularly in the domain of wargame simulations.

However when the number of events spawned in a time step (Event coverage) equals 1, the performance of this technique equals that of the conventional technique. Moreover, the introduction of the FTL results in some overhead at the host. Since we are reducing the number of overhead messages at the cost of introducing an additional data structure at the central host it is important to measure the physical time spent by the host in maintaining the data structure. This time is affected by two parameters –

the number of time stamps already in the FTL and the number of time stamps newly introduced.

4.1.4 Limitations

Though the optimization suggested in this technique addresses simulations with sparse event coverage it does not provide for utilization of any available lookaheads between participating entities.

4.2 Paced Time Stepped Synchronization

4.2.1 Introduction

This technique uses the idea of using a time stepped synchronous execution mechanism to solve the problem of synchronization. It is a conservative approach that paces the execution of simulation using real time clocks.

Since the root cause of time ambiguities is asynchronous execution of LPs i.e. simulation clocks of LPs advance independently from each other. For instance if the Local Virtual Time (LVT) of receiving process advance faster than that of the source LP then it is highly possible that a message coming from source LP will cause a time ambiguity.

4.2.2 The Approach

The solution for time ambiguity problem is to execute the source and receiving LPs synchronously. Hence a mechanism that will advance LVTs in a coordinated fashion will solve this problem.

In a time stepped execution, at each time step all simulation clocks advance by a small constant value and at the end of the step all events/message occurred during that step is processed. This involves a lot of communication overhead sine at each step all LPs would send and receive at least one message for coordination purposes.

To reduce the communication overhead a reference real time clock can be used to pace the advance of time steps[2]. If all the simulation clocks can be advanced at a common rate synchronized with the rate of advance of a reference wall clock then it is possible to advance the clocks of federates at a fairly synchronous way. Thus

communication overhead will be reduced since the LPs communicate to the reference clock when their local hardware clocks drift far enough.

Assuming that execution of all LPs start at the same time. At each time step each LP will spend exactly the same amount of wallclock time and therefore all LVTs will advance at the same rate.

$dLVT_i/6dt = 1/k$ …..(i) where $dLVT_i/dt$ is the rate of advance of the simulation clock of LP_i with respect to the wallclock.

k=1 means that the simulation is running at wallclock speed i.e. real time simulation. As the value of k decreases simulation speed increases.

and $k > \Delta c_{max}/\Delta T_{min}$ …..(ii) where Δc_{max} denotes the maximum communication delay and ΔT_{min} denotes the minimum simulated event time among all messages.
Equation (ii) defines the lower bound on the time scaling factor for proper synchronization. This lower bound corresponds to the fastest execution speed while maintaining synchronization throughout the entire simulation execution. Thus in order to provide better performance it is possible to adjust the time scaling factor dynamically during an execution using ΔT values of imminent messages. This method is called "*adaptive pacing*" of the distributed simulation execution. To adjust the time scaling factor a control system is used where each LP is coupled with a "*local synchronization agent*" or LSA. Each LSA collects data from the simulation process in order to predict imminent messages in that LP. Based on these messages each LSA notifies a "*global synchronization module*". GSA receives message specific data from the LSAs such as when a message will be fired and its ΔT value and by examining this data it broadcasts adjusted value of time scaling factor to LSAs. Each LSA listens to the GSA for a change of the value of the time scaling factor and then adjusts the speed of its coupled simulation process. For LSA to predict imminent messages it is formed of a simple finite state automaton representation of the coupled simulation process [3].

4.2.3 Results

Adaptive time-stepped pacing mechanism is a viable alternative to the existing synchronization techniques. However there are many issues to be addressed such as tolerating to jitter in the communication network, abnormalities caused by drifting hardware clocks and accuracy of timing. Such synchronization techniques are already deployed for satellite and cellular communication networks.

4.3 Zero Lookahead Approach

4.3.1 Introduction

Conservative parallel and distributed synchronization mechanisms require a quantity known as lookahead to enable deadlock avoidance. However lookahead can be difficult to calculate in certain applications. Hence this approach supports the so called Zero lookahead systems. Activity scanning framework is the organizing principle behind this approach. Activity scanning based simulation model is organized around conditions and actions that should be taken when the associated action is satisfied. Together, a condition and its associated actions form an event.

4.3.2 The Approach

This technique extends the basic Activity scanning world view to a distributed fixed time simulation[7]. Activities are viewed as logical processes.

With each LP there is an associated process scheduler.

The following steps are followed by simulation executive for LP:

- LP calculates its next state for time t and sends self generated output.
- LP marks the end of this output and requests to advance to time t.
- LP then enters its main scanning phase within which LP either receives a grant, which indicates no LP in the federation sent output during the previous scan phase or the LP receives input which it may respond.
- When the LP receives grant, LP updates its local clock and start from step 1.

The following steps are followed by the simulation executive for Process scheduler:

- It receives and forwards as necessary LP output, marker and requests to advance.

- It also receives input for its LP and phase markers sent from other schedulers in the federation.
- If any LP sent output the scheduler enters its scanning loop. Within the loop, if any input is waiting for its LP, the scheduler sends the input to the LP and marks its completion.
- Then it receives LP generated output and completion marker and again receives input from the other schedulers within the federation.
- If the LP has no input during a scan, the scheduler forwards a 'no output' marker to the rest of federation and waits the results of the scan.

In this approach, time does not advance as long as federates are generating output and each federate is guaranteed to receive, process and respond to any externally generated output during each scan.

4.3.3 Results

This approach inherently supports zero lookahead and is described in terms of HLA services. However application of this approach within the context of mixed or dynamic time steps is being developed.

4.4 Concluding Remarks

In the conventional time-stepped approach, the time interval to be advanced after synchronization is performed is usually predetermined and is equal to the step-size. Thus at each time step the Server waits for the processes to issue a Ready signal. As a result of which the server sends next time advance signal. To minimize these synchronization messages at each time step and also to make sure that the synchronization is maintained in the entire execution run, variations of the conventional time stepped approach can be followed. Three such approaches were described in this chapter. All these approaches ensure that synchronization is maintained at minimal overhead.

Chapter 5

Proposed Time Synchronization Approach

In a wargame application it is essential that distributed pieces of the system be strictly synchronized to allow them to vary the rate of progression of time, achieve identical event ordering and stop or restart the simulations in a coordinated manner. Wargames should be able to progress forward at different rates, such as twice the speed of real-time or half real-time.

Conservative synchronization algorithms have been widely used in non-military simulation. We have chosen a conservative time management approach (time stepped) for wargame simulation for two reasons: better control over synchronization overhead and its performance implications, and decreased complexity. Moreover the concept of rolling back makes no sense in the scenario of a war. In this chapter an efficient time synchronization approach is suggested.

5.1 The Proposed Approach

The proposed approach makes use of a centralized time server which sends time after a fixed update frequency to all the processes (clients) involved in a simulation run. The approach is thereby implemented and different cases are run to study that time synchronization is maintained in all the cases.

For successfully executing a wargame, it is very important that the processes participating in the simulation run, are synchronized. With each process two 'times' are associated namely System Time and Game Time.

Game Time: This is the logical time associated with the wargame. It is entered at the start of a wargame.

System Time (Wallclock time): Time during execution of simulation (hardware clock).

With respect to Time Server we call it as ***Time Server Game Time (TGT) and Time Server System Time (TST).*** With respect to clients (processes) we call it as ***Client Game Time (CGT) and Client System Time (CST)***.

The approach is tied onto the client server architecture and includes the following main entities as shown in figure 5:

- ***Process Configuration Application***
- ***Time Configuration Application***
- ***Hicon Application***
- ***Time Server***
- ***Clients (Processes)***

Process Configuration Application

There can be many processes involved in a wargame. The number of processes, the machine on which process is running, machine IP Address can vary each time we execute. Thus the following details about the processes involved in the simulation should be entered and stored in a file.

- Name of the Process involved
- Machine Number on which that process will run
- IP Address of the Machine on which that process will run.

This information is stored in the file so that details of time configuration for each process can be set later according to the requirements.

Time Configuration Application

In a simulation execution there can be a case where a process does not require time from the server. Such a process executes without considering current Simulation Time or System Time. Thus timing details about each process is separately stored through time configuration application in a separate file. The following timing information is associated with each process:

- Whether the Process requires time or not.
- If Time is required, whether the process requires handshake facility i.e. the process will acknowledge back to the server once it receives time or not.

Along with the timing information, frequency at which time update (*frequency of update*) will be sent by the server is specified through this application and stored in the file.

Hicon Application

Game Time is entered through the Hicon Application (Hicon App), which is responsible for storing the Game Time in the database. Along with the Game Time, scaling factor also needs to be specified. Scaling f2actor determines the speed of the simulation such as twice the speed of real time or half real time. By default scaling factor will be 1, i.e. wargame will be advanced with the rate of real time.

Time Server

Time Server is the owner and controller of the master clock. It is responsible for sending time to all the processes involved in the simulation run. It accesses the Game Time as stored by the Hicon App in the database. As the wargame is run this time is updated and this time sent after an interval of 'update frequency'. As this time is updated it is saved as Current Game Time in the database. Time Server also stores its System Time in the database when the wargame is started.

The following database tables are maintained:

- ***Game Time***: It holds the Game Time as entered by the HICON App at the start of the wargame.
- ***Current Game Time:*** It holds the current Game Time value as the wargame runs. Time Server stores the current Game Time value (TGT) in this table before sending the Game Time to the clients (processes).
- ***System Time:*** Time Server stores its System Time (TST) when the wargame is started in this table.
- ***Current System Time:*** Time Server stores the current System time (TST) value as the wargame is executed in this table.

The process and time configuration data is saved in the following files:

- ***Process Configuration file:*** Process Configuration Application stores the details of the processes participating in the simulation run in this file. The details includes:
  - Name of the Process involved
  - Machine Number on which that process will run
  - IP Address of the Machine on which that process will run.

- ***Time Configuration file:*** The timing details of the processes participating in the simulation run are entered into this file by the Time Configuration Application. The details include processes requiring time and handshake facility.

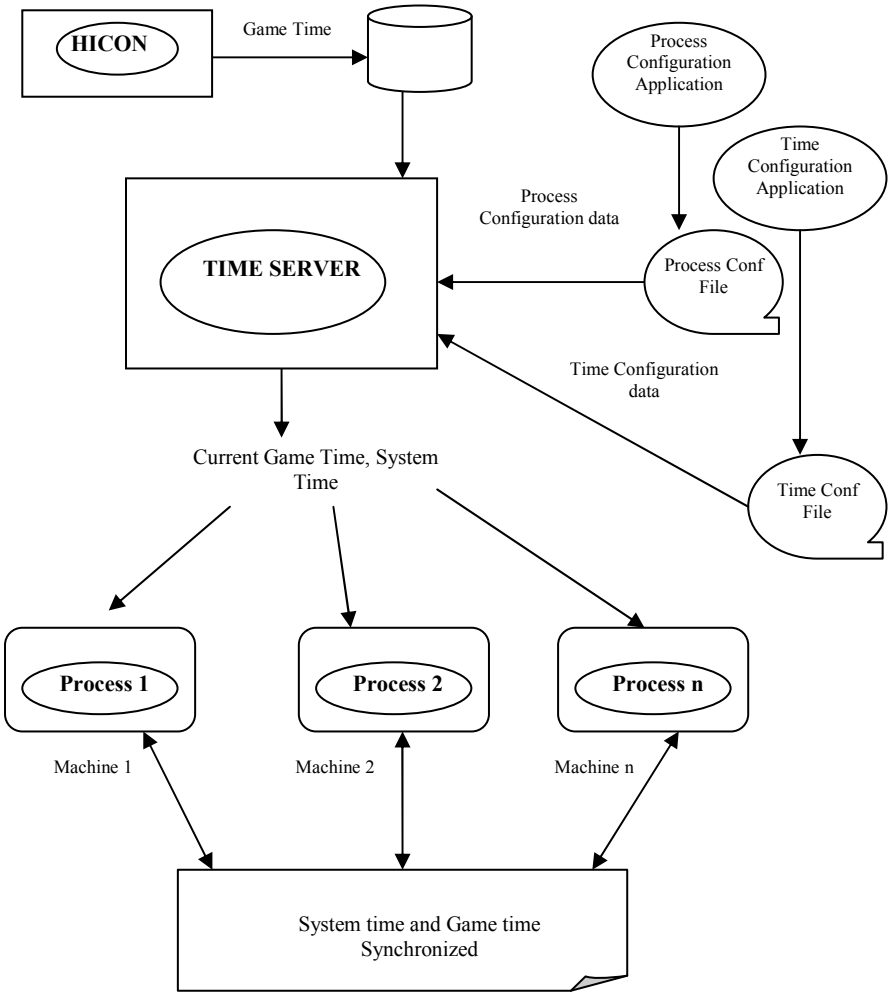


Figure 5.1 Proposed Architecture

5.2 Methodology

In the time-stepped approach, the time interval to be advanced after synchronization is usually predetermined and is equal to the step-size. Typically the entire span of simulation is divided into equal-sized time steps and the simulation advances from one time step to the next. This time interval size depends upon the type and purpose of the wargame. For example any tactical wargame for training purpose would have a time interval set for 1 minute while on the other hand an Airfare wargame would require a time interval less than a minute.

A wargame simulation would typically have front end and back ground processes.
The front end client (s) provides graphically interfaces for the player to input the orders for maneuvering the units. The background processes actually execute these orders. Most often these clients distributed across the node in a network.
Hence it is very important to make sure that the wallclock time and Game time are same for all the clients (processes). In order to maintain synchronization in the wall clock times, there are two main available options:

- To manually set same 'system time' on all the processes before starting the simulation execution.
- To develop a methodology which is transparent to the 'system time' of each process.

The Approach developed in this work follows the second option i.e. Time Synchronization will be maintained irrespective of the 'system time' on each process.
Synchronization can be accomplished by assigning one of the distributed simulation process (server) as the owner and controller of the clock. That process moves the clock forward, sending its time to all other processes. Those processes accept that time and process events accordingly.

5.3 Simulation Execution and Testing

Time Server sends time to all the interested processes i.e. processes which need time. This data is stored in the Time Configuration file. Time Server has two available options:

I. Time Server sends only Game Time to the processes (Clients)
II. Time Server sends Game Time and its System Time to the processes (Clients)

The simulation execution was tested with these two options. During the simulation run it was checked whether time remain synchronized or not.
Within each of the above cases two more cases can arise:

i. System Time of all processes is same.
ii. System Time of all processes is different.

Case I. If Time Server sends only Game Time

Option i. If System Time of all processes is same then System Time of all clients are always synchronized. As a result time synchronization is maintained.

Option ii. If System Time of all processes is different then different processes will see different System Time's. Game Time seen on all clients will also be different. This can result in incorrect ordering of messages and events. As a result time synchronization will not be there. This proves that apart from 'Game Time' from Time Server some other synchronization mechanism is needed.

Case II. If Time Server sends Game Time and its System time

Option i. If System Time of all processes is same then System Time on all Clients is always synchronized. As a result Game Time is also synchronized. Thus the simulation run is always safe.

Option ii. Even if System Time of all processes is different, since in this case System Time is also sent along with Game Time, synchronization is maintained.

Every client calculates the *Elapsed Time* as difference between its own System Time (CST) and Time Server's System Time (TST). As client undergoes processing and as its system clock advances, this Elapsed Time is added to the Game Time as sent regularly (sent after *frequency of update*) by the Time Server (TGT). As a result Current Game Time (CGT) at all the clients will be same. Elapsed time is also added to the System Time (CST) to get current System time. Hence same System Time is seen by all processes irrespective of their System Time's (as shown by hardware clock). Thus time synchronization will be maintained. Thus this approach is transparent to the system time of all processes.

5.4 Results

Platform Used

Hardware: Pentium IV Processor

Operating System: Windows XP Professional

Development Software: NetBeans, Jdk 1.4

Deployment Details:

Machine	IP	Application Deployed
1	140.9.215.4	HICON (Client 2)
2	140.9.215.6	Client 1
3	140.9.215.16	Time Server

5.5 Test Plan and Output

Test Cases were planned to evaluate Game Time Synchronization with Client's System clocks synchronized and not synchronized. They are tabulated as follows:

Case I: Time Server sends only Game Time

Option 1: Client's System clocks are synchronized

Test Data and Output

Time Server	Client 1	Client 2
TST: 3:59 TGT: 4:30	CST: 3:59 CGT: 4:30	CST: 3:59 CGT: 4:30
TST: 4:04 TGT: 4:35	CST: 4:04 CGT: 4:35	CST: 4:04 CGT: 4:35

Option 2: Client's System Clocks are not synchronized

Test Data and Output

Time Server	Client 1	Client 2
TST: 03:59 TGT: 4:30	CST: 03: 59 CGT: 4:30	CST: 04:04 CGT: 4:30
TST: 04: 04 TGT: 4:35	CST: 04:04 CGT: 4:35	CST: 04:09 CGT: 4:40

Case II: Time Server sends System Time and Game Time

Option 1: Client's System Clocks are synchronized

Test Data and Output

Time Server	Client 1	Client 2
TST: 3:59 TGT: 4:30	CST: 3:59 CGT: 4:30	CST: 3:59 CGT: 4:30
TST: 4:04 TGT: 4:35	CST: 4:04 CGT: 4:35	CST: 4:04 CGT: 4:35

Option 2: Client's System Clocks are not synchronized

Test Data and Output

Time Server	**Client 1**	**Client 2**
TST: 3:59 TGT: 4:30	CST: 3:59 CGT: 4:30	CST: 3:59 CGT: 4:30
TST: 4:04 TGT: 4:35	CST: 4:04 CGT: 4:35	CST: 4:04 CGT: 4:35

(Since in this case Time Server sends its own System Time (TST) along, each Client sets its System time to this time)

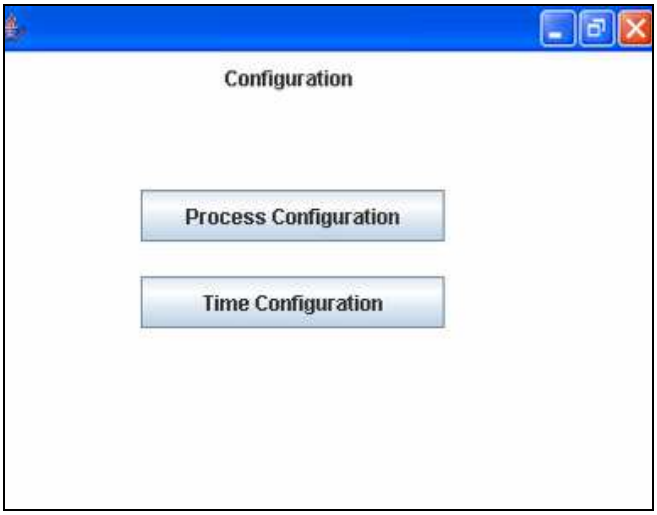
5.6 Advantages

- Processes can be distributed across the network.
- Since sending Game Time and System Time by the time server will maintain time synchronization in both the cases (System time of clients same/ not same), there is no need of manually setting system time of all the clients before simulation execution. This could be a tedious task if there are a large number of processes participating in the simulation.
- Since it is a conservative approach, no extra overheads are involved in rolling back or saving states of each process.
- It is a simple approach and does not require extra resources like memory, processor etc.

5.7 Limitations

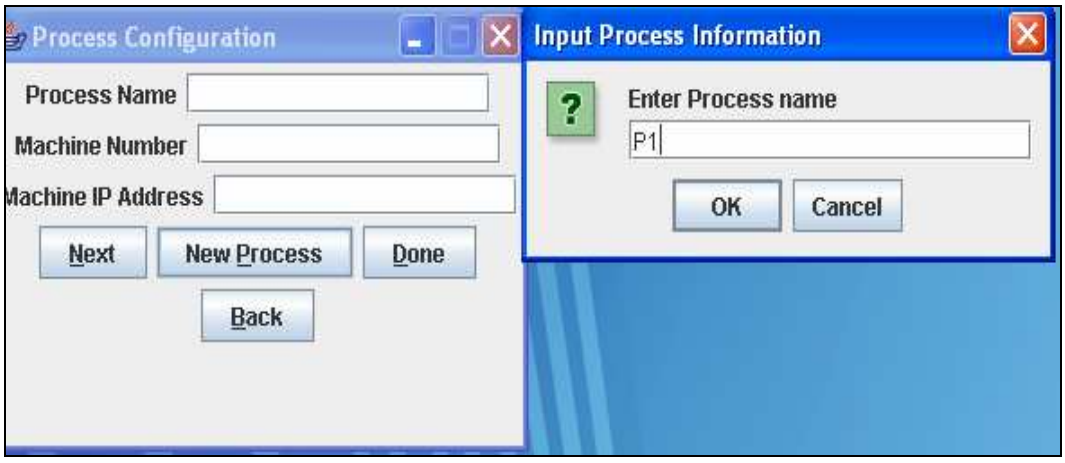
- It is assumed that frequency of the clocks running on different processes (clients) is same i.e. all the clocks pace at same speed. This could not be the case always because of manufacturing or depreciation errors.
- The case where one of the process stops execution due to hardware/ software failure is not taken into consideration.
- The case where acknowledgement or time packet is lost/ corrupted on the way is not taken into consideration.
- The case where time packet is lost/ corrupted on the way from time server to process is not taken into consideration.

5.8 Snapshots



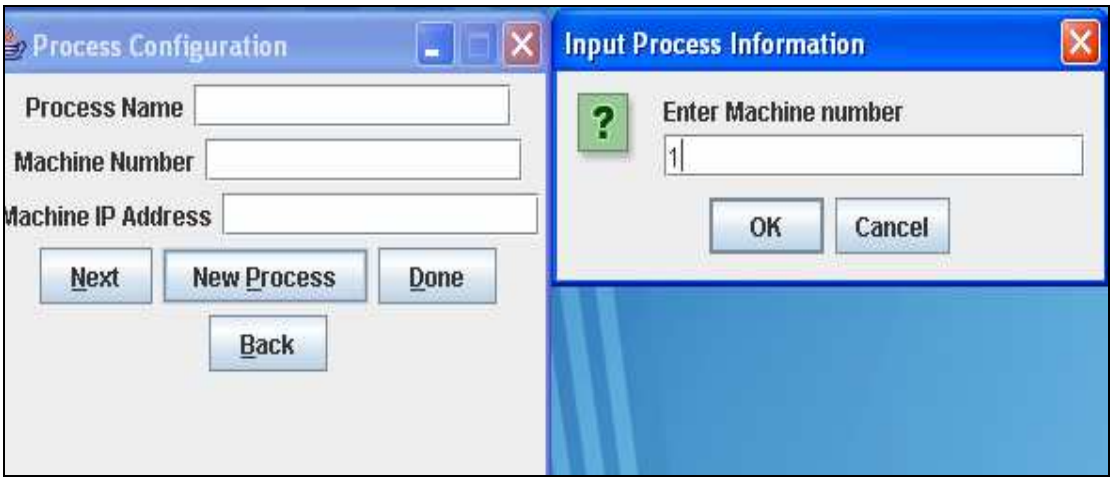
Screen #1

The first Interface through which one can set configurations for the processes involved in simulation execution.



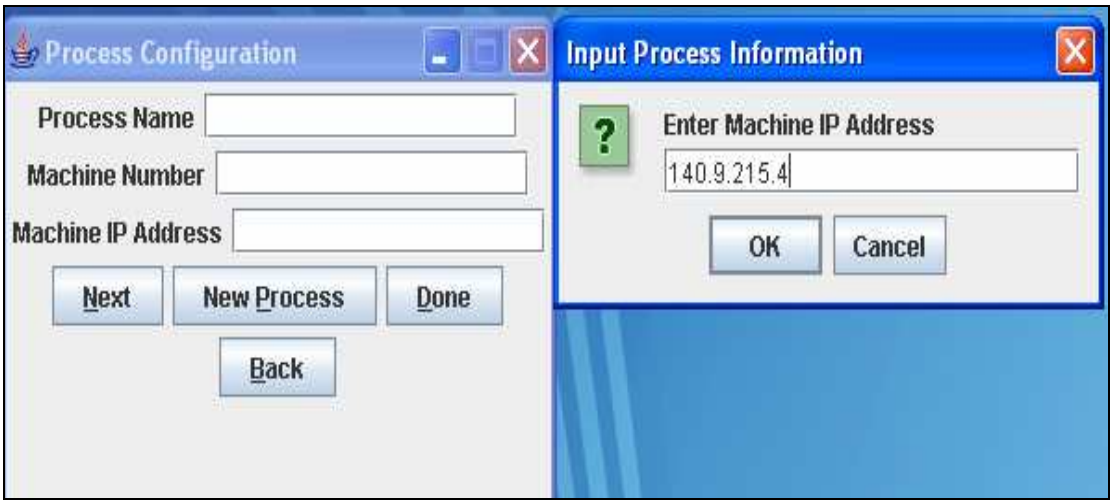
Screen #2

Process name for the first client (process) is entered through this Interface



Screen #3

Machine number of the Machine on which this process is to be run is entered through this Interface

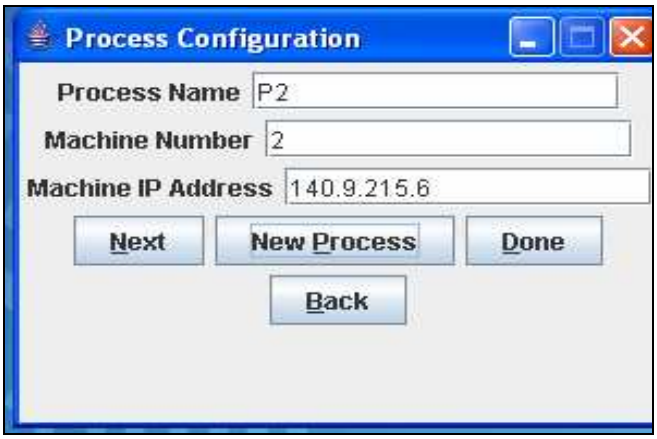


Screen #4

Machine IP Address of the Machine on which this process is to be run is entered through this Interface

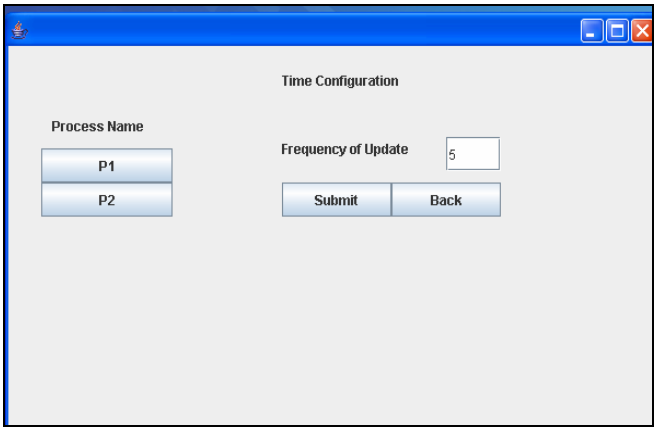
Screen #5

This Screen displays the information for process 1 as stored in the Process Conf file.



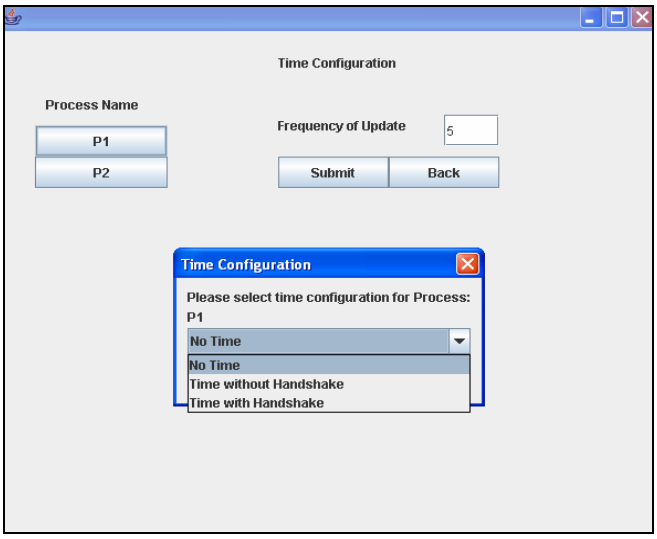
Screen #6

This Screen displays the information for process 2 as stored in the Process Conf file.



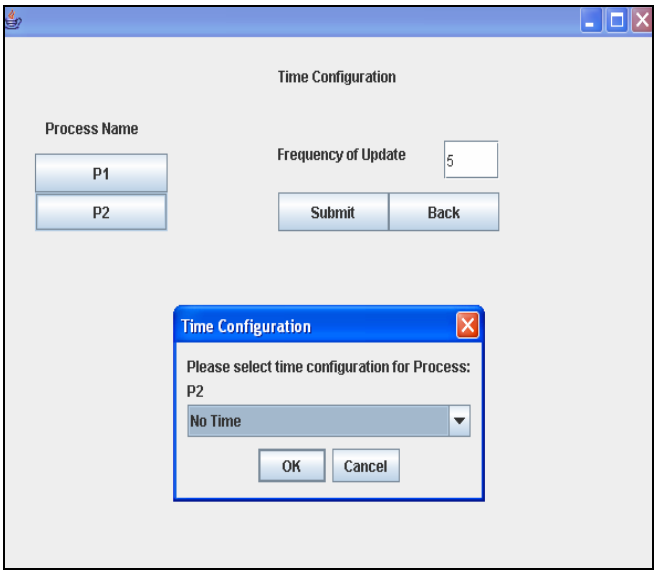
Screen #7

This is the Interface through which we can set the Time Configuration Information for the Clients (processes) we selected (as stored in the Process Conf file). Also *frequency of update* is set through this interface.



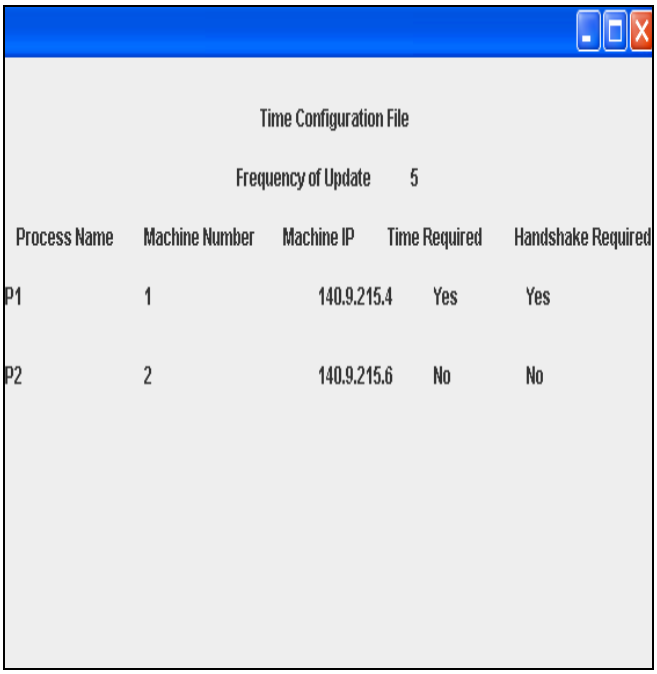
Screen #8

This Screen shows the various options for Time Configuration available for Process P1.



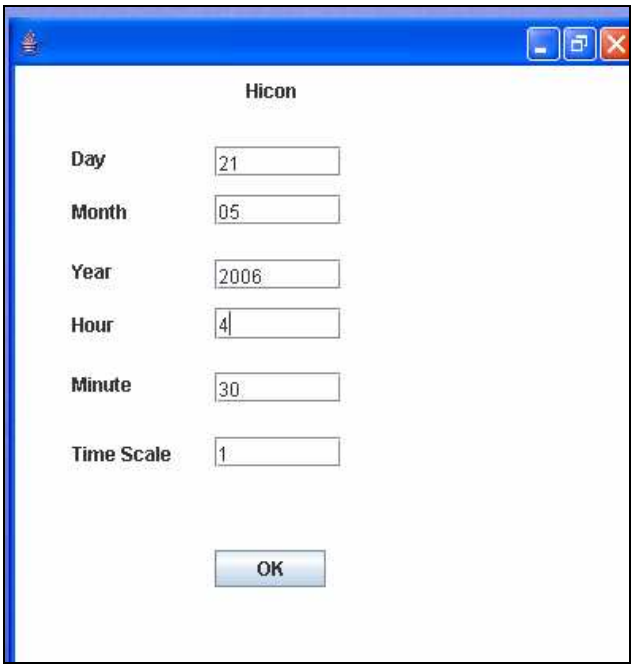
Screen #9

This Screen shows the setting of the "No Time" option for the Process P2



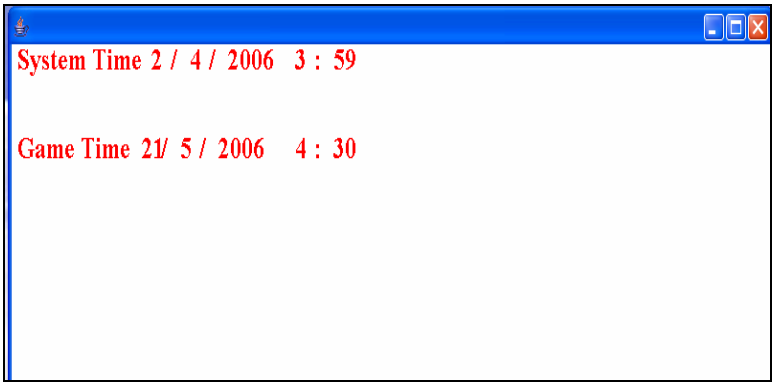
Screen #10

This Screen shows the Time configuration Information for Process P1 and P2 as stored in the Time Conf file.



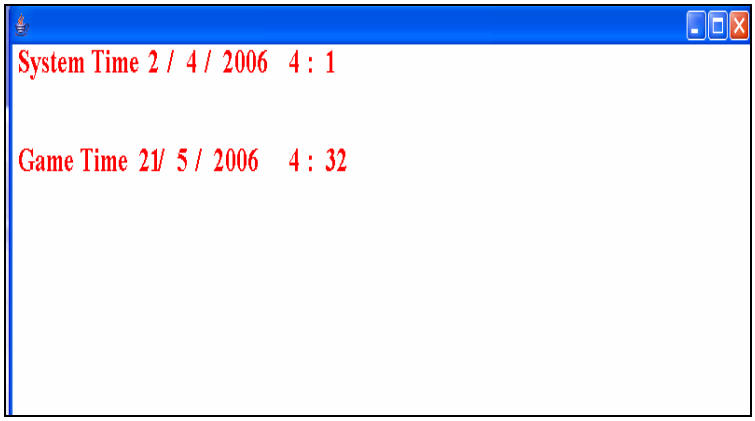
Screen #11

This is the HICON Interface through which Game Time and Scaling factor is entered and then stored in the database.



Screen #12

This Screen shows the System Time and Game Time as displayed on P1 and P2 immediately after Time Server sent Game Time and System Time



Screen #13

This Screen shows the System Time and Game Time as displayed on P1 and P2 two minutes after Time Server sent Game Time and System Time

5.9 Concluding Remarks

An efficient time synchronization approach was discussed in this chapter. This approach is simple yet effective in maintaining time synchronization. This approach will solve the earlier problems of the wargame community like setting the system time of all the processes before starting a wargame. Two different options were executed and tested i.e. sending only game time and sending both game time and system time of time server. Testing results clearly demonstrates that the latter option is more effective.

Chapter 6

Conclusions and Scope for Further Research

6.1 Conclusions

Beginning with research and development efforts in the 1970's, research in distributed simulation systems has matured over the years. Much of the early research in this area was motivated purely by performance considerations. As processor speeds have continued to increase at an exponential pace, performance alone has become less of a motivating factor in recent years.

There are many issues involved for successful implementation of wargame. Time management is an important issue that must be addressed in distributed simulations. It is a central issue concerning the synchronization of computations on different processors. The time management system includes mechanisms to ensure timestamp ordered delivery of messages, we well as mechanisms for LPs to advance logical time so that the LPs does not receive messages with time stamp less than its current logical time. Time management algorithms broadly fall into two categories, termed conservative and optimistic synchronization. Conservative algorithms take precautions to avoid the possibility of processing events out of time stamp order, i.e., the execution mechanism avoids synchronization errors. On the other hand, optimistic algorithms use a detection and recovery approach. Events are allowed to be processed out of time stamp order, however, a separate mechanism is provided to recover from such errors.

In the conventional time-stepped approach, the time interval to be advanced after synchronization is performed is usually predetermined and is equal to the step-size. All participating entities in the simulation are at the same time-step at any point in wallclock time. Typically the entire span of simulation is divided into equal-sized time steps and the simulation advances from one time step to the next.

Three synchronization approaches especially for time stepped simulations namely: Piggy-Backed Time Stepped Simulation, Paced Time Stepped Synchronization and Zero Lookahead Approach were discussed. Based on the pros and cons of these

approaches a new Time Synchronization approach based on Zero Lookahead approach is developed.

The main objective of the research was achieved through the design of the framework for synchronization of time between different LPs participating in the simulation. Synchronization makes sure that time ambiguities never occur and time advances smoothly time step by time step. Thus, times perceived by all LPs will be synchronized.

In this approach, a centralized time server acted as an owner and controller of the master clock. It sends timing signals to all participating LPs after a fixed period of time. These timing signals ensure that all time in all LPs remain synchronized irrespective of the system time on the LPs.

6.2 Summary of Contributions

Institute for System Studies and Analyses (ISSA) has developed software's which provide a computerized combat training environment, designed to train commanders and staff in combined arms environment for realistic battlefield conditions.

This work will also help the people new in wargame development community to have an insight on the various issues concerned with the development of wargames.

This work will also help the amateurs. Since the theoretical background discussed in this thesis work is sufficient to have an idea of how wargames are developed and how synchronization is maintained between different entities participating in the wargame.

It also provides a survey of the various time synchronization algorithms and their pros and cons. This will be useful since different type of wargames needs different time management solutions. A solution for the type of wargame to be developed can be found out.

Also the standards followed across the world by the wargame development communities are discussed. This will help to be in place with the latest technologies in the development of wargames.

Thereby an efficient solution for synchronization of time among various distributed simulation processes is developed. This solution can be helpful in the development of future wargames or will provide time synchronization if applied to the existing wargames.

The implementation, integration and testing of the algorithm requires lots of resources and time, which is difficult under the preview of the project. Moreover the application of the algorithm in the real world scenario will only prove its efficiency. Thus this work will provide an algorithm for synchronization of time among LPs in a distributed simulation application (wargame). The present study is very timely and important, as it will provide the distributed simulation community an insight into the synchronization of time in the development of wargames.

6.3 Scope for Further Research

The time synchronization solution developed in Chapter 5 gives an efficient solution for time ambiguities. Given below are the further research options with the respect to the approach developed.

- Varying the time scaling factor and executing the wargame with different speeds such as twice the speed of real time will further explore whether time synchronization is maintained.

- Developing provisions for successful implementation of the wargame in the following scenarios:

  Case 1: One of the LP shuts down.

  Case 2: Processing speed of the LPs vary, some being too slow.

  Case 3: Time Server waiting for acknowledgement from an LP. Acknowledgement packet is lost or corrupted on the way.

  Case 4: LP does not receive time packet from time server due to packet being lost or corrupted.

Due to time constraints the above mentioned cases were not taken into consideration. But these can be explored as part of the future research work.

In general these are the Research Directions for Distributed Simulation Technology:

Synchronization

Past work in synchronization has focused on execution on multiprocessor platforms. As a practical matter, platforms using standard networking hardware will continue to dominate the marketplace. While prior work in parallel and distributed simulation traditionally treats the network as a black box, significant advantage can be realized by exposing network characteristics to the simulation executive. Realization in geographically distributed computing environments such as the global Internet where large communication latencies cannot be avoided also presents new technical challenges.

Distributed Virtual Environments (DVEs)

Parallel and distributed simulation research has traditionally focused on analytic simulation applications. Distributed virtual environments (DVEs) for training and entertainment have emerged as an important domain where this technology may also be applied. Parallel and distributed simulation techniques have much to offer in addressing problems such as repeatability and correct ordering of causally related events that arise in DVEs. Training and entertainment have different requirements than analytic simulations. For example, maintaining precise time stamp order processing of events is often not essential because humans may not be able to perceive “incorrect” orderings of events. On the other hand, real-time execution of the simulation is essential. Relatively little attention to date has been paid in the parallel simulation community to the problem of ensuring timely delivery of results.

Heterogeneous Distributed Simulations

Most of the work thus far in the parallel simulation community has focused on homogeneous simulations where the entire simulation is built from scratch using a parallel simulation language or library.

References

[1]. Bernard P.Zeigler, Herbert Praehofer, Tag Gon Kim, "Theory of Modeling and Simulation", Harcourt India Private Ltd/ Academic Press, Second Edition

[2]. Bertan Aluntas, Richard A. Wysk, "Paced Time Stepped Synchronization of Parallel and Distributed Simulations"

[3]. Bertan Aluntas, Richard A. Wysk, "A Framework for Adaptive Synchronization of Distributed Simulations" at Winter Simulation Conference, 2004, pp 371-373

[4]. Bruno R. Preiss, Wayne M. Loucks, "Memory Management Techniques for Time Warp on a Distributed Memory Machine"

[5]. Daniel Teckentrup "Parallel Discrete Event Simulation on Multicomputer applied to Network Performance Evaluation", submitted at Technical University of Berlin, 2003, pp 3-9

[6]. Department of Defense, Defense Modeling and Simulation Office "*RTI Programmer Guide*", Version 3.2, 6.1-6.3

[7]. Ernest H. Page, "Zero Lookahead in Distributed Time Stepped Simulations", *Simulation Digest*, 26(2), September 1997, pp. 4-13

[8]. Fabian Gomes, Brian Unger, John Cleary "Language Based state saving extensions for optimistic Parallel Simulation" 2001

[9]. Fujimoto, R.M. 1999. Exploiting Temporal Uncertainty in Parallel and Distributed Simulations. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, ed. R. Fujimoto and S. Turner, 46-53. Piscataway. New Jersey: Institute of Electrical and Electronics Engineers.

[10]. Kalyan Perumalla, Parallel and Distributed Simulation Systems and the High Level Architecture, Nov 28, 2005, pp 5-31

[11]. Kevin Jones, Sameer R. Das, “Combining Optimism Limiting Schemes in Time Warp based Parallel Simulations” at Winter Simulation Conference, 1998

[12]. Michael Richter, “ Java Based Conservative Distributed Simulation”, Proceedings of Winter Simulation Conference, 1997, pp 381-383

[13]. Morris Stoman, Jeff Kramer, “Distributed System and Computer Networks”

[14]. Nancy Roberts, David Anderson, “Introduction to Computer Simulation” Addison-Wesley Longman Publishing Co., Inc

[15]. R.M Fujimoto, “Time Management in the High Level Architecture” Simulation 71, 6 (December 1998), 388-400

[16]. Richard M. Fujimoto “Parallel and Distributed discrete Event Simulation: Algorithms and Applications” at Winter Simulation Conference, 1993

[17]. Richard M. Fujimoto, “Distributed Simulation Systems” at Winter Simulation Conference, 2003, pp 1-8

[18]. Roger D. Smith “Vertical Interface Methodology for Connecting Constructive and Virtual Level Simulations” submitted at Century University, 1994, pp 1-9

[19]. Roger Smith, ”Essential Techniques for Military Modeling and Simulation” at Winter Simulation Conference, 1999, 16-17

[20]. Roger Smith, Ernest H. Page, “Introduction to Military Training Simulation: A guide for Discrete Event Simulationists” at Winter Simulation Conference, 1998

[21]. Roger Smith, Strategic Directions in Distributed Simulation, Volume 2, Simulation 2000 Series, pp 7-9

[22]. Roger Smith, Synchronizing Distributed Virtual Worlds, Volume 3, Simulation 2000 Series, pp 5-22

[23]. Roger Smith, The Engine Behind the Virtual World, Volume 1, Simulation 2000 Series, pp 9-14

[24]. S.C Tay, G.S.H Tan, K.Shenoy, ”Piggy Backed Time Stepped Simulation with Super Stepping” at Winter Simulation conference, 2003, pp 1077-1080

[25]. Sandra Cheung, Margaret Loper, “Synchronizing Simulations in Distributed Interactive Simulations” at Winter Simulation Conference, 1994

[26]. Stephen J. Turner, Wentong Cai, Ji Chen, “A Middleware Approach to Causal Order Delivery in Distributed Simulations”

[27]. Wentong Cai, Stephen J. Turner, “An Algorithm For Reducing Null-Messages of CMB Approach in Parallel Discrete Event Simulation “(1995)

[28]. Wentong Cai, Stephen J. Turner, Bu-Sung Lee, “An Alternate Time Management Mechanism for Distributed Simulations”, ACM Transactions on Modeling and Simulation, April 2005, pp 110-112

[29]. “Design and Implementation of HLA Time Management in the RTI Version F.0” at Winter Simulation Conference, 1997, pp 372-376

[30]. “HLA Time Management Design Document”, Version 1, August 15, 1996, pp 1-11

Appendix A

Acronyms

SIMNET	SIMulator NETworking
DIS	Distributed Interactive Simulation
ALSP	Aggregate Level Simulation Protocol
HLA	High Level Architecture
DMSO	Defense Modeling and Simulation Office
RTI	Runtime Infrastructure
LP	Logical Process
PADS	Parallel and Distributed Simulations
LBTS	Lower Bound on the Time Stamp
GVT	Global Virtual Time
RO	Receive Order
CO	Causal Order
CATOCS	Causally and Totally Ordered Communications Support
TSO	Time Stamp Order
PE	Processing Elements
PEL	Pending Event List
FTL	Future Time List
LVT	Local Virtual Time
LSA	Local Synchronization Agent
GSA	Global Synchronization Agent
DVE	Distributed Virtual Environments

Appendix B

Glossary

Model	Physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process.
Simulation	Method for implementing a model over time.
M & S	Use of models, including emulators, prototypes, simulators, stimulators, either statically or over time, to develop data as a basis for making managerial or technical decisions
Simulator	A device, computer program, or system that performs simulation;
War Game	Simulation game in which participants seek to achieve a specified military objective given pre-established resources and constraints.
Virtual simulation	Simulation involving real people operating simulated systems.
Live simulation	Simulation involving real people operating real systems.
Constructive simulation	Simulation that involves simulated people operating in simulated systems.

Papers Communicated

[1]. Pooja Grover "Time Synchronization Approach for Distributed Time Stepped Simulation: An Application to Wargame" at BMAS 2006, IEEE International Behavioral Modeling and Simulation Conference sponsored by Cadence, Mentor Graphics and Synopsys to be held in San Jose, California, USA from September 14-15, 2006 **[Communicated].**