PATIM: PROXIMITY AWARE TIME MANAGEMENT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

AYDIN OKUTANOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF COMPUTER ENGINEERING

SEPTEMBER 2008

Approval of the thesis

"PATIM: PROXIMITY AWARE TIME MANAGEMENT"

submitted by **Aydın Okutanoğlu** in partial fullfillment of the requirements for the degree of **Doctor of Philosophy in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen ________________
Dean, **Graduate School of Natural and Applied Science**

Prof. Dr. Volkan Atalay ________________
Head of Department, **Computer Engineering**

Prof. Dr. Müslim Bozyiğit ________________
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Ali Saatçi ________________
Computer Engineering Dept., Hacettepe University

Prof. Dr. Müslim Bozyiğit ________________
Computer Engineering Dept., METU

Assoc. Prof. Dr. Halit Oğuztüzün ________________
Computer Engineering Dept., METU

Assoc. Prof. Dr. Veysi İşler ________________
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ali Doğru ________________
Computer Engineering Dept., METU

Date: ________________

ABSTRACT

PATIM: PROXIMITY AWARE TIME MANAGEMENT

Okutanoğlu, Aydın
Ph.D., Department of Computer Engineering
Supervisor: Prof. Dr. Müslim Bozyiğit

September 2008, 148 pages

Logical time management is used to synchronize the executions of distributed simulation elements. In existing time management systems, such as High Level Architecture (HLA), logical times of the simulation elements are synchronized. However, in some cases synchronization can unnecessarily decrease the performance of the system. In the proposed HLA based time management mechanism, federates are clustered into logically related groups. The relevance of federates is taken to be a function of proximity which is defined as the distance between them in the virtual space. Thus, each federate cluster is composed of relatively close federates according to calculated distances.

When federate clusters are sufficiently far from each other, there is no need to synchronize them, as they do not relate each other. So in PATiM mechanism, inter-cluster logical times are not synchronized when clusters are sufficiently distant. However, if the distant federate clusters get close to each other, they will need to resynchronize their logical times. This temporal partitioning is aimed at reducing network traffic and time management calculations and also increasing the concurrency between federates.

The results obtained based on case applications have verified that clustering improves local performance as soon as federates become unrelated.

Keywords: Clustering, Data Distribution Management, HLA, Interest Management, Logical Time, RTI, Time Management

ÖZ

PATİM: YAKINSALLIK TABANLI ZAMAN YÖNETİMİ

Okutanoğlu, Aydın
Doktora, Bilgisayar Mühendisliği Bölümü Bölümü
Tez Yöneticisi: Prof. Dr. Müslim Bozyiğit

Eylül 2008, 148 sayfa

Mantıksal zaman yönetimi dağıtık bir şekilde çalışan simülasyonların senkronizasyonlarında kullanılmaktadır. Bugüne kadar geliştirilmiş olan zaman yönetimi sistemlerinde, örneğin Yüksek Seviye Mimari, simülasyon elemanlarının mantıksal zamanları simülasyon içerisindeki zamanı düzenleyen tüm diğer elemanlara bağlıdır. Ancak bazı simülasyon uygulamalarında bu koşul gereksiz olarak sistemin performansını düşürmektedir. Önerilen zaman yönetimi mekanizmasında, dağıtık simülasyon içerisindeki federeler mantıksal olarak birbirleriyle ilişkisiz kümelere ayrılmaktadırlar. Federelerin birbirleriyle ilişkileri onların simülasyon uzayı içerisindeki yakınsallıklarıyla ölçülmektedir. Bu sayede federe kümeleri göreceli olarak birbirlerine daha yakın federelerden oluşmaktadır.

Federe kümeleri arasındaki mesafe belirli bir miktardan daha fazla olduğunda, bu federeler arasıdaki mantıksal zaman sekronizasyonunu devam ettirmeye gerek yoktur. PATiM mekanizması içerisinde federe kümeleri birbirleriyle uzak mesafelerde olduklarında federe kümelerinin mantıksal zamanları senkronize edilmemektedir. Ancak bu federe kümeleri birbirlerine tekrar yaklaşmaya başladıklarında, mekanizmanın onların mantıksal zamanlarını tekrar senkronize etmesi gerekecektir. Simülasyon sırasındaki bu geçici bölünmelere izin verilmesi sayesinde mesajlaşma trafiğini ve zaman yönetimi hesaplama zamanlarını azaltmayı ve federeler arasıdaki paralleliği arttırmak amaçlanmaktadır.

Örnek uygulamalardan elde edilen sonuçlar federe kümelenmesi yöneteminin federeler birbirlerinden uzaklaştıkları andan itibaren yerel performansı arttırdığını göstermiştir.

Anahtar Kelimeler: HLA, Ilgi yönetimi, Kümelenme, Mantıksal Zaman, RTI, Veri Dağıtım Yönetimi, Zaman Yönetimi

ACKNOWLEDGMENTS

I would like to express thanks to my supervisor Prof. Dr. Müslim Bozyiğit for his continuous guidance from the very beginning of this thesis work.

I would also like to acknowledge the support given by Associate Prof. Dr. Halit Oğuztüzün, as well as Prof. Dr. Ali Saatçi as members of Thesis Follow-up Committee from the early stage of this study.

I am grateful to my friend Hatice, who has helped me in editing the thesis text and encourage me through out the thesis work.

Finally, I am in depth to my wife Canan, without whose encouragement it would have been impossible to complete this thesis.

To my wife Canan

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AS	Approaching Speed
CD	Critical Distance
DDM	Data Distribution Management
HLA	High Level Architecture
LBTS	Lower Bound on Time Stamp
PATiM	Proximity Aware Time Management
RS	Routing Space
RTI	Runtime Infrastructure

CHAPTER 1

INTRODUCTION

Parallel and distributed simulation is an emerging technology for collaborative simulation. It enables models to be run at geographically dispersed sites and it deals with ways of using multiple processors in a single simulation. Each participant constructs his/her own model, and agrees upon the information interchange mechanisms that are going to be used for the interoperation of the models. These types of simulation systems find relevance in many applications, including civilian applications such as telecommunication networks, physical system simulations, traffic simulations, transportation simulations, distributed multi-player gaming, and non-civilian applications like battlefield simulations.

Ideally, the results produced by a parallel simulation run must match those that are produced by an equivalent sequential run. To achieve this match, parallel execution must be properly synchronized in order to preserve the right orderings and dependencies during computation of the simulation states across the processors. One of the challenges of this synchronization is encountered in minimizing the runtime execution overheads incurred during parallel execution. Thus, it is important to keep the overhead within acceptable limits to produce better overall execution times in comparison with sequential runs.

A large simulation system is usually built up by a combination of various simulation components which are developed at different times by different teams. To promote the interoperability and reusability of such simulations, the USA Defense Modeling and Simulation Office (DMSO) has been working on the formation of a common technical framework for simulations since 1995. The efforts of the

DMSO led to the advent of the High Level Architecture (HLA).

The HLA generalizes and builds upon the results of Distributed Interactive Simulation (DIS) and other related efforts such as Aggregate Level Simulation Protocol (ALSP). The primary mission of HLA is to create a synthetic, virtual simulation environment by systematically connecting individually developed simulations that are executed at geographically dispersed locations. It defines an architecture where simulation components, referred as federates in the HLA, complying with a set of HLA rules can interact with each other via the services defined in the HLA Runtime Infrastructure (RTI) to create a combined large scale simulation.

Ideally, simulations exactly reproduce the temporal relationships among the events that occur in the real world being modeled. However, the heterogeneous delays associated with the modeling computations and message transmissions over the network may lead to a violation of such relationships. For example, an observer may see a tank destroyed before it was hit by an aircraft, where the tank and aircraft are simulated at two separate computers. These temporal anomalies or distortions in the event ordering cause simulations to deviate from the real world.

The objective of time management services of HLA is to reduce the occurrence and effects of these temporal anomalies in order to meet the requirements of the simulation. To facilitate the reuse and interoperation of the separate simulations, a number of design rationales have been proposed and incorporated into the definitions of current time management services. Some important design principles are time management transparency, flexibility of accommodating various internal time management mechanisms, minimal communication latency, and minimal computational and communication overhead.

A fundamental issue to be addressed by the time management services of the HLA is the ordering of the messages that are processed by a federate. Each federate in the federation generates a stream of events modeling its behavior and exchanges messages representing those events to other federates. Since each of these events takes place at an exact moment in time, the messages representing the respective events must be handled in the correct order at all federates.

Although there are a group of ordering mechanisms in the distributed synchronization area, such as receive order, priority order, causal order, causal and totally delivered order and timestamp order, in distributed synchronization area, timestamp ordering is the main focus in which every event is given a timestamp indicating the time the event occurred.

Each federate has a logical time value indicating its point on the timeline of the simulation environment. When a federate wants to send an event, its current logical time is given as a timestamp to this event. On the receiving side, events are processed with ascending timestamps.

The main problem of conservative time management mechanisms like the time stamp ordering mechanism, is the high overhead of interchanging and processing the synchronization information. In distributed implementations of time management mechanisms, individual elements need to know about time related states of other elements in order to advance their internal logical time. However, distributing this information between simulation elements decreases the performance because of both time and resource required to transfer and process this information.

1.1 Motivation

In existing time management systems, such as in HLA, the logical times of the federates are regulated by all the federates that take part in the simulation. A federate can advance its logical time only after states of all other federates allow. This approach, in fact, should decrease the level of concurrency during the overall simulation.

During simulations, a group of federates could come logically close but fall apart from the rest of the federates. For example, in a space-based simulation application, there could be a number of federates simulating spaceships and ground stations. Initially when they are all on Earth, they could regularly interact. It is normal that logical times of all federates are synchronized. However, after a while, a group of spaceships may leave Earth to go to other planets, making the communication with Earth impossible. In this case, there will not be any interaction between spaceships and ground stations for a period of time. Thus, the federates on different planets, including Earth, will constitute different groups. In this study, these logically related group of federates are referred as Federate Clusters.

In fact, there will be no need to synchronize the logical times of federates that are not related. If the logical time dependency between two federate clusters is broken, their concurrency level is expected to increase as their logical times will be constrained by less number of federates. Additionally, there will be performance improvements in the time management algorithm with reduced problem space, in terms of the number of federates [50, 51].

The only standard HLA supplied mechanism that could be used to break logical time dependencies between two or more groups of federates is disabling or enabling time regulating properties of the federates. By disabling time management, a federate could not send or receive TSO messages, as its logical time synchronization with other federates is broken. However, in most cases, disabling time management is not a desired feature. First of all, the aim is to construct two different time consistency groups; not disabling time any group. Secondly, having disabled time management, federates in that group start processing events in receive order, which is not a desired situation.

1.2 Summary of Contributions

In this study, a new time management mechanism is proposed, in which federates are divided into logically related groups referred as Federate Clusters. The dynamic clustering of this mechanism is based on proximity relations between federates, in which federates are grouped into federate clusters by comparing the distances between them in the virtual space. Distances of the federates are calculated using the declared update and subscribe regions of the federates by the Data Distribution Management services of HLA. Federate cluster structure could dynamically change during the simulation execution due to the movements of the federate within the virtual space. On the other hand, federates of the clusters are always related with each other, maintained by dynamic property of proposed clustering algorithm, named as Proximity Aware Time Management (PATiM).

In PATiM mechanism, logical times of federate clusters can be independent of each other. When federates are distant to each other, there is no need to synchronize their logical times as there will not be any interaction between these federates as long as they are distant. In this study, this logical time independency is utilized in order to increase the performance of the underlying time management algorithm.

The aim of this algorithm is to reduce overhead of the time management mechanism by increasing concurrency, thus reducing unnecessary blocking of the federates. Performance improvements are made possible as the whole federation is divided into small synchronization groups. The required synchronization related messages and LBTS calculation time are dramatically reduced due to this dynamic division.

The notion of distance is very important in PATiM mechanism. Distance changes between the federates are measured in order to break and reconstruct the logical time consistency. Thus, the formation of federate clusters is completely dependent on the distances between the federates. In order to measure distances that are meaningful, the simulations should use time-stepped algorithms; in that case, position of the federate will not change dramatically and immediate changes or jumps will not occur. Even if the term of distance in PATiM does not have any relation with the real distance, it is easier to understand it with physical simulations of real systems.

To test the proposed mechanisms, they are implemented within the scopes of example simulations. Currently there are not any fully implemented and open source HLAs; therefore a middleware architecture for integrating the PATiM mechanism to RTI-NG 1.3, which is a free but not open source HLA implementation, is developed. This implementation is used to make a group of validations and performance analyses in order to show the effectiveness of the PATiM mechanism.

1.3 Organization of the Thesis

This thesis is organized as follows: Chapter 2 gives some background information, which includes distributed simulations, parallel and distributed simulations, distributed interactive simulations, High Level Architecture and federation communities. Chapter 3 describes the theory behind the proposed dynamic clustering mechanism and proximity aware time management mechanism. Chapter 4 gives information about the design and implementation of the proposed mechanisms. Additionally, in this chapter there is a validation section of the PATiM mechanism. In Chapter 5, an analytical analysis of the proposed method is given. Finally, Chapter 6 states the conclusion of this study and suggests future work.

CHAPTER 2

BACKGROUND ON DISTRIBUTED SIMULATIONS

2.1 Distributed Simulation

Distributed simulation has attracted considerable amount of interest in recent years. This interest is mainly a result of the fact that large simulations in many applications consume enormous amount of time and processing power on sequential computers. The need for a distributed system that combines simulation environments that are distributed over many computers and locations is another reason for this interest. The main issues in the distributed simulation area are synchronization of time and data, fault tolerance, locality management, parallelization of simulation processes, scalability, interaction and coordination between simulation processes and event distribution [66, 31, 14, 73, 37, 7].

The classification of distributed simulations is based on the involvement of real world entities to the simulation system [28]. The simulation systems that fall in the first category are fully composed of logical components. These types of simulations, called Parallel and Distributed Simulations (PADS), are generally used for statistical purposes or for performance evaluations. Distributed synchronization, discrete events and time management [36, 40] are the issues in these simulations. For the simulation systems of the second category, entities from real world are directly included in the interactions within the simulation systems. These types of real-time simulation systems are generally referred as Distributed Interactive Simulations (DIS). DIS tools form a simulation infrastructure intended to support interoperability between separately developed simulators, systems and human participants.

2.2 Parallel and Distributed (Discrete Event) Simulations

In Parallel and distributed Simulation (PADS), a set of real world entities are modeled as logical processes. In this process oriented approach, there are continuous interactions between logical processes, which simulate the interactions between physical objects in the real world. The logical processes contain a set of state information corresponding to the physical process and logical simulation clock. The interactions between physical processes are modeled by time-stamped messages that are exchanged between the logical processes. In order to lead a correct simulation, logical processes should process events in an ascending timestamp order.

PADS systems can be divided into two main categories based on their synchronization techniques as conservative or optimistic [26, 52]. Conservative algorithms strictly avoid the possibility of incorrect sequencing of events by including strategies to determine which events are safe to process at each point in the simulated time. As a result, this creates the potential for a deadlock. Optimistic algorithms, on the other hand, use a detection and recovery approach, which is based on detecting incorrect event execution sequences, and determining a rollback mechanism for recovery. Both techniques are discussed further in the following subsections.

2.2.1 Conservative PADS Techniques

Conservative PADS techniques are based on the idea of determining when it is safe to process an event. By processing only safe event, simulation systems guarantee that no incorrect executions can be generated. This safety information can be generated using the timestamps of the events that are sent within the event messages.

The first conservative algorithm is the CMB (Chandy, Misra and Bryant) algorithm [18, 17]. In this algorithm, every logical process send the outgoing event messages in the timestamp order. This rule guarantees that the last message from an incoming channel has the highest timestamp value within the messages received from that channel, which is the clock of that channel. Incoming messages are kept in a queue in a descending timestamp order. The process repeatedly selects the channel with the smallest channel clock and gets the smallest message to include in the process. The important point here is that the logical process blocks if not all the channel queues have messages. The reason for this is to ensure that all processes have sent a message, so that no other messages with smaller timestamp values than the selected message, has been undelivered. Of course, this method can lead to deadlock conditions because of infinite blockings of the logical processes. These blocking operations are the sources of unnecessary waiting, because generally new messages from other logical processes have higher timestamp values than the smallest message in the local queues. To overcome this problem, null messages are used. Every logical process regularly sends null messages, which carry correct timestamp values indicating the current logical time. By using this method, blocking times are bounded to the interval where the null messages are sent and deadlocks caused by recursive blockings are avoided. However, this method brings the overhead of sending too many null messages. An improvement is the demand based null messages, in which the logical process that has an empty message queue will request a null message from the corresponding process.

Another method to ensure safety of processing of an event is using the time windows. In this method, only unprocessed events with timestamps larger than the lower edge of the window can be processed. Dividing the physical system into subparts, for which time windows are defined, increases the concurrency. Additionally, there are some mechanisms making assumptions about the underlying network topologies in order to improve the performance [60]. Besides CMB, there are other time management mechanisms reported in the literature [33, 57, 25, 34].

2.2.2 Improvements to CMB Algorithm

The basic problem of the CMB algorithm is its high number of message requirement for synchronization of distributed simulation elements. In this algorithm, logical processes send several null messages for each processed message. A number of methods have been developed in order to reduce the number of these synchronization messages [49, 69, 10, 10, 55].

The Null Message Cancellation [56] algorithm is based on the fact that null messages do not contain any other information other than the synchronization data. Therefore, total number of null messages can be reduced by removing unnecessary ones. There are two types of unnecessary null messages. The first one includes null messages that appear before any real message. These messages are not necessary because immediately after sending this message, CMB algorithm will send a real data message containing the necessary synchronization information. The other type of unnecessary null messages includes null messages that appear after the last real message, except the last one. This last null message represents the greatest timestamp of the queue. Information carried by any null message between the last real message and the last null message will be overwritten by the null message next to it. So, these messages can also be cancelled. Null message cancellations can be implemented by using a modified message queue structure so that messages holding the greatest timestamp of all null messages can be cancelled individually. This value could be used in the deciding whether to send or cancel the current null message.

Another optimization on CMB algorithm is applied by optimizing the simulation loop algorithm [54]. In standard CMB algorithm, all created messages related to events are released to output channels after every event processing. If the simulation loop is changed, logical processes first process all available events coming from incoming channels and buffer outgoing messages. After event processing is finished, all generated messages are released. If the simulation loop does not generate an outgoing message for a channel, a null message that carries information about current logical time of the logical process is generated and released. Thus, only necessary null messages are sent over the communication network.

2.2.3 Optimistic PADS techniques

In optimistic approaches [70, 16, 5], the system lets the logical processes to continue their executions without blocking them. On an error condition, optimistic techniques will detect the error and recover from it. The most known optimistic mechanism is the Time Warp. In Time Warp, a causality error condition is specified as the time a logical process receives an event that has a smaller timestamp value than the local time clock.

In this algorithm, if a process receives an event from another process that has a timestamp earlier than the local clock, the process rolls back and repeats its simulation to take the new messages into account. In addition, the effects of the rolled back processes should be undone, which can lead to successive rollbacks of other logical processes. The state of each process since the last correct time, called Global Virtual Time [63] (GVT), must be stored periodically in order to be able to roll back more than one logical process. GVT is the minimum of the logical time vectors for all logical processes and the timestamps of all messages that are sent but not acknowledged, i.e. message on the network. Additionally, irrecoverable operations, such as disk I/O, should not be committed until GVT is larger than the logical time of the operation.

2.2.4 Hybrid Techniques

Conservative and optimistic approaches to PADS are likely to encounter some limitations when the size and complexity of the simulation system increases. The blocking problem and the sensitivity to look ahead in the conservative protocols, cascading rollbacks and state saving problems in the optimistic protocols are the key limitations for the respective approaches. A hybrid algorithm can be dynamically switched between optimistic and conservative mechanisms whenever it encounters too many numbers of rollbacks. Clustering of closely related logical processes is another hybrid technique. In this mechanism, intra-cluster events are handled optimistically whereas inter-cluster events are handled conservatively.

2.2.5 Casual Ordering

Conservative and optimistic approaches provide message ordering schemes that use a "all" or "nothing" type of ordering. On the other hand, casual ordering provides a partial message ordering scheme for simulations. This new approach captures the "cause and effect" relationship among the events and enforces an ordered processing of two events only when such a "cause and effect" relationship exists between them [41, 13].

The notion of causality was first introduced by Lamport [40] in a general context of a distributed system. A distributed system consists of a group of asynchronous processes that cooperate to achieve a common goal by exchanging messages. Those processes do not have a shared memory and a common clock does not exist for them. The action of each process in a distributed system can be modeled as three types of events: internal event, send message event, and receive message event. Causality captures the "cause and effect" relationship among events and establishes a partial order called "happened before".

If an event e_2 is fired after another event e_1 is received there is a cause-and-effect relation between these events and e_1 is "happened before" e_2. Thus, these events are delivered to another simulation node in that order. However, if there is not any cause-and-effect relation between two events, they are assumed to occur concurrently and causal ordering does not guarantee the delivery of them. So, this ordering could relax the total ordering of the timestamp ordering, but it is undesirable in some cases. For example, let us assume that two planes, A and B, fire to a tank without having an existing cause-and-effect relation. Assume that the event of plane A is delivered to the tank node in the first place and has fired it. There is no guarantee about the ordering of these two fire events and a third observer could receive the fire event of B first and may think that plane B fired the tank rather than A.

To maintain causality, any message M should not be delivered to a process P_i until all other messages sent to P_i, which are called the causal predecessors of M and the SEND events of which occur before SEND(M), have been processed. By attaching message M the information about all its causal predecessors, destination processes of M are able to process the message only after all of its causal predecessors have been delivered. This traditional approach requires control information packaged within each message with N^2 size.

Underlying network topologies or communication patterns of processes are used to decrease the size of the required control information. Thus, a more promising approach is based on the direct dependency tracking technique, where only the immediate causal predecessors of message M, whose delivery state may not be known, are tracked and propagated with M. The foremost advantage of the direct dependency tracking technique is that it achieves optimality without relying on assumptions over the underlying communication network and thus is feasible for a wider class of applications. Although the upper bound of the space complexity of the control information appended to messages is still $O(N^2)$, the size of the control information is likely to be much smaller on the average.

2.2.6 Fault Tolerance

In a distributed simulation, a crash of any logical process causes the entire computation to halt. In such a case, if there is no any fault tolerance system present, the only solution would be to restart the entire system. However, this would result in losing all the simulation work data of hours or even days, which is clearly unacceptable. Thus, some form of fault-tolerance is required to minimize the amount of lost work data.

Most of the studies on fault tolerant distributed simulations are focused on optimistic simulations because of their natural implementation of rollback and checkpoint mechanisms [22, 72]. Therefore, in this section fault tolerance issue for optimistic mechanisms is mentioned.

On a crash of a process, it is assumed that it loses all of its volatile memory. To reduce the amount of wasted computation, it periodically writes its checkpoints to a stable storage. After a failure, it is restarted from its last stable checkpoint. Here, the time warp protocol provides a useful mechanism for the fault tolerance. Crash of a logical process is similar to a late event in a time warp mechanism. However, the original mechanism provided by the time warp cannot be applied to a fault situation such as a process crash. In a native time warp algorithm, when a roll back occurs because of a late message, a process should send anti-messages corresponding to messages it sent after the last checkpoint. However, in a process crash this process cannot be succeeded.

There is a fault tolerance protocol especially designed for distributed simulations [22]. In fact, this algorithm is very similar to the standard fault tolerance mechanism used for a group of distributed communicating processes. In this algorithm each logical process saves its states to stable storages periodically. After a crash, it restores the latest checkpoint and starts its execution from there. At that point, the process broadcasts a message indicating the rollback event with a timestamp checkpoint. Upon receiving this message, other processes check for any dependencies. Dependency here means that the processes receive and process an event message from the rolled back process which was sent between the checkpoint time and the rollback time. Of course, this message is an orphan message that should not be sent at all in a rolled back state. If other processes find such invalid dependencies, they are also rolled back to the previous situation.

2.3 Distributed Interactive Simulations

Distributed Interactive Simulations (DIS) [1] can be viewed as a collection of autonomous simulators (e.g. tank simulators), each generating its own virtual environment representation (e.g. battlefield). Each simulator sends messages, called Protocol Data Units (PDUs), whenever its state changes in a way that might affect another simulator. Typical PDUs include movement to a new location, such as firing, etc. For the interoperability among separately developed simulators, a set of standards are defined. These standards define not only the format and content of the PDUs but also the information of when to send PDUs.

DIS systems are designed especially for training systems that include human interactions. Additionally, testing real world systems is another purpose of the DIS systems. In order to properly model these requirements, DIS systems have some design principles [26]:

- Autonomy of simulation nodes: Each simulation node in the distributed environment is an autonomous agent, which can independently enter or exit the simulation. Nodes have their own local real-time clock. Simulators are not required to determine which other simulators must receive PDUs. They either broadcast all the messages to others and receivers decide to process the PDU, or a runtime system decides which nodes should receive the PDU.

- Transmission of state change information only: In order to minimize the amount of messages sent over the network, nodes only send state changes. For example, while traveling in a straight line with constant velocity, it is not necessary to send location update messages. Only a "keep alive" message is sent regularly, so that newcomers can get the state information of other nodes.

- Dead Reckoning [11] algorithms: All simulators use common algorithms to extrapolate the current state of other entities between state updates. Dead-reckoning model (DRM) is an approximation of the true simulator node. In practice, DRM is a simplified model of the true simulator and it can lead to errors. For this reason, the true simulator node also facilitates a copy of its own DRM and compares the real value and the DRM generated value. Whenever the difference between them exceeds a threshold value, it sends an update message.

- Simulation time constraints: Because humans cannot distinguish time differences smaller than 100 milliseconds, a communication latency of up to this is required. Lower latencies can be necessary for other simulators such as non-training simulators, e.g. testing of weapon systems.

Even with dead-reckoning, DIS protocols require enormous amount of messages to be sent over the network. The main decrease is achieved by using relevance filtering, in which messages are not broadcasted but sent only to the relevant nodes. The relevance here is data dependent and the nodes have to decide which data are related to which nodes. Other methods are compression of messages, bundling two or more messages in one, and fidelity management [61], in which different degrees of details are sent to different nodes.

2.3.1 Synchronization and Time Management

The synchronization and time management mechanisms are responsible to ensure temporal correlation between simulation nodes. While PADS simulation protocols guarantee that all logical processes observe the same timestamp ordered sequence of events, DIS does not guarantee this. Correlation problems can occur because of lost messages and lack of a mechanism that ensures that events are processed in a timestamp order. Partial solution for this problem is that the receiver compensates late arrivals by determining the communication delay in transmitting the message. This communication delay can be calculated using past message transmission delays, or it can be assumed that there is a synchronized real-time clock between the sender and receiver to determine the latency by simply computing the difference between the send and receive times.

In general, time synchronization problem is solved by synchronizing local clocks to a standard clock called Coordinated Universal Time (UTC). By using some methods, the UTC time is broadcasted to all simulation nodes. Radio broadcasting, dial-up time service, global positioning system (GPS) or Network Time Protocol (NTP) is used for this purpose [44].

2.4 High Level Architecture (HLA)

The High Level Architecture (HLA) [21, 6] provides a specification of a common technical architecture to use with a wide range of distributed simulation environments. Its main purpose is to support interoperability among different simulation systems, which is actually a similar requirement for distributed interactive simulation systems.

HLA architecture is composed of a number of functional components which are shown in Figure 2.1. The first key components are the simulations themselves, or more generally the federates. A federate can be a computer simulation, a supporting utility (e.g. a viewer or data collector), or an interface to a live player.

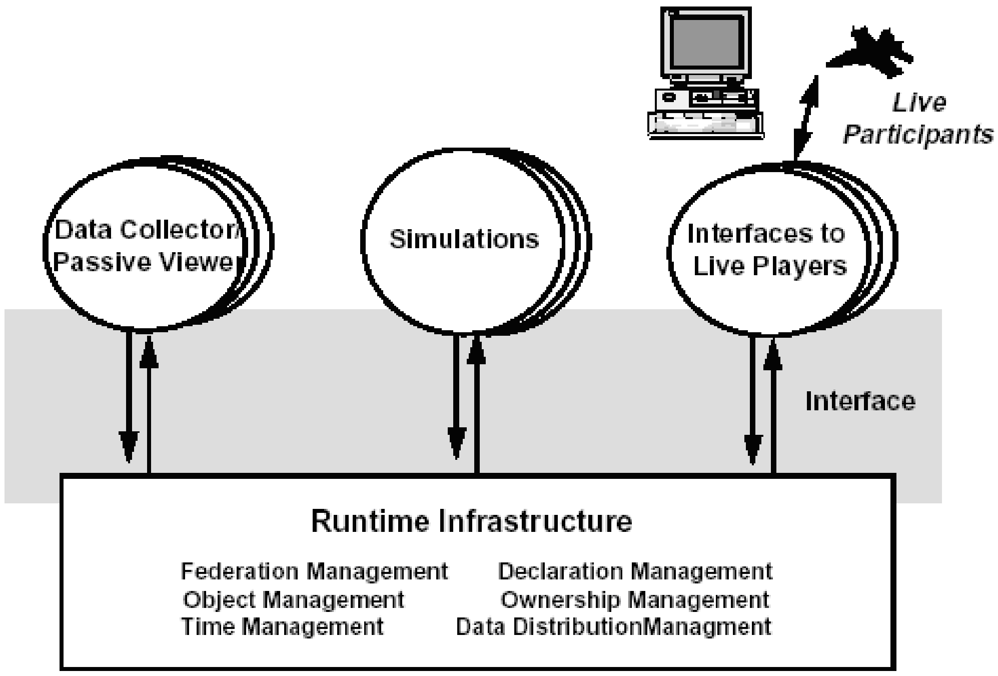


Figure 2.1: HLA Architecture

The second functional component is the Runtime Infrastructure (RTI). In fact, RTI is a kind of distributed operation system for the federation. It provides a set of services that support the interaction. All interactions between the federates flow through the RTI. Between the federates and the RTI there is the runtime interface which provides a standard way for the federates to interact with the RTI.

The HLA is a blueprint to be used to develop the necessary infrastructure in order to promote interoperability and reusability within the modeling and simulation community. A key component of the HLA is the Interface Specification that defines the standard services that simulations utilize for coordination and collaboration during an exercise. There are two sides of the Interface Specification; the services implemented by the individual simulations themselves and the services implemented by the common Runtime Infrastructure (RTI).

As the RTI is an interface specification, it is envisioned that multiple implementations, potentially providing domain specific benefits, could be developed. The RTI provides services that fall into six categories:

- Federation Management, which deals with the creation of new federates,
- Declaration Management, dealing with the declaration of objects and their attributes and operations such as subscribe and update declarations
- Object Management
- Ownership Management, which deals with which federate currently owns objects,
- Time Management, which deals with time and message ordering, and
- Data Distribution Management dealing with distribution of update information on object attributes to other federates.

The specifications of these services have evolved through prototyping and working group activities. Modules that compose the RTI architecture are shown in Figure 2.2.

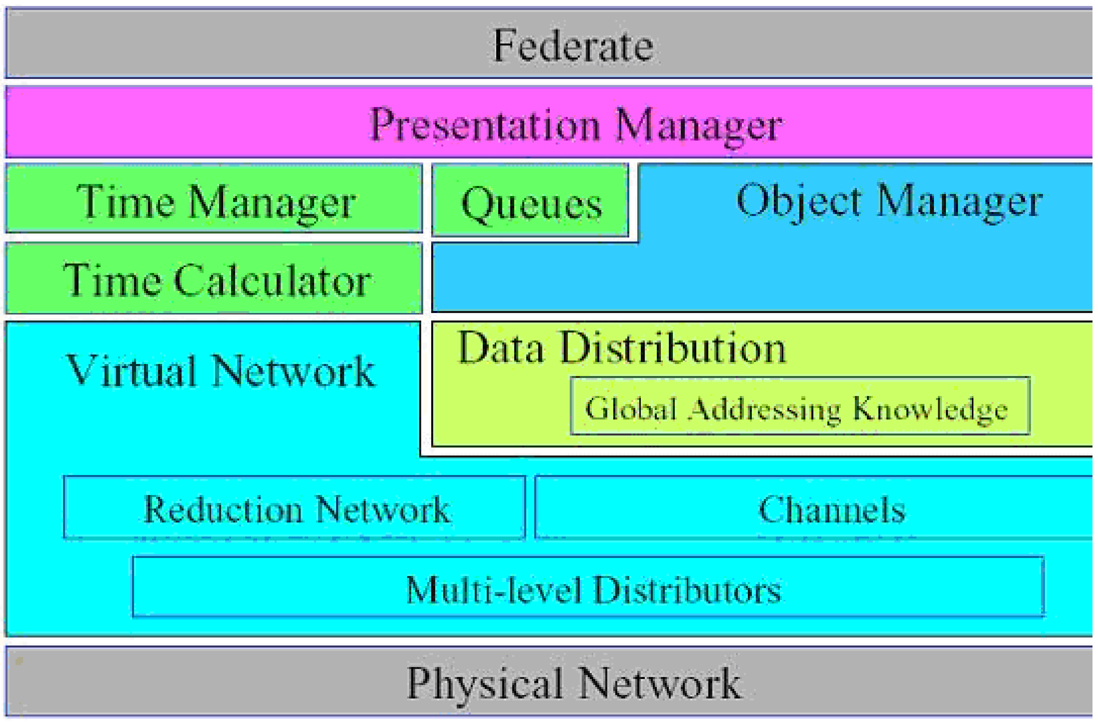


Figure 2.2: RTI Modules

As it can be seen in the figure, the federate accesses the RTI via the Presentation Manager, which presents the language-specific API to the user. Internally, the Presentation Manager converts the supported APIs into a common format before passing service requests and data to other RTI components. There are a group of services to support the Presentation Manager, including the Time Manager, Queues and the Object Manager. The Time Manager maintains the ordering information for the local federate and determines which data items within the Queues may be released to the federate without violating the data ordering requirements of the federate.

The Time Calculator in the RTI architecture is a support module for the Time Manager. The Time Manager is responsible to maintain an up-to-date value for the system-wide Lower Bounds Time Stamp (LBTS) and use LBTS in determining data that can be released to the federate.

The Object Manager is responsible to maintain the current list of objects produced and consumed by the local federate. It uses the Data Distribution component to efficiently transport data from producers to consumers.

Internal to the Data Distribution component, the Global Addressing Knowledge component maintains the information needed to segment data flow in order to minimize undesired package receipt.

The Virtual Network component abstracts communications with the other RTI components and isolates the RTI from variations in networking technology. A generic Channel is used to provide both point to point and point to multipoint communications. Internal to the Virtual Network, a Reduction Network component provides support to RTI components that require efficient access to system-wide information. The Multi-Level Distributor is an optional component, which supports large-scale federations through hierarchical message transmission.

2.4.1 Data Distribution Management

The aim of Data Distribution Management (DDM) is to limit and control the volume of the data exchanged during the simulation, and to reduce the processing requirements of simulation hosts by relying events and state information only to those applications which require them, while exploiting the features of the computer network architectures [8, 64, 46].

The fundamental Data Distribution Management construct is a routing space. A routing space is a multidimensional coordinate system through which the federates either express an interest in receiving data (subscribe) or declare their intention to send data (publish). These intentions are expressed through:

- Subscription Region: Bounding routing space coordinates that narrow the scope of interest of the subscribing federate.
- Update Region: Bounding routing space coordinates which are guaranteed to enclose an object's location in the routing space.

Both subscription and update regions can change in size and location over time as the interests of the federate can change or the location of an object in the routing space can change. An object is discovered by a federate when at least one of the attribute of the object gets into the scope of the federate, i.e. if an only if:

- the federate has subscribed to the attribute
- the update region of the object overlaps the federate's association to another region.

DDM enables federates to specify by object class and attribute name the types of data they will send or receive, while narrowing down the specific instances of data. Federates decide which of the federation routing spaces are more useful for them and define the portions of those routing spaces that specify regions, or logical areas of interest particular to the federate, by putting bounds (extents) on the dimensions of the selected routing space. Specifying a subscription region, the federate tells the Run Time Infrastructure (RTI) the data in which it is interested in and which fall within the extents of the region specified by that federate. Specifying an update region and associating this update region with a particular object instance is a contract from the federate to the RTI. Due to this contract, the federate will ensure that the characteristics of the object instance, which map to the dimensions of the routing space, fall within the extents of the associated region at the time the attribute update is issued. This implies that the federate is monitoring the added characteristics for each of the attribute it owns. As the state of the objects changes, the federate may need to either adjust to the extents of the associated regions or alter its association to another region.

Figure 2.3 shows one update region (U1) and two subscription regions (S1, S2) within a two dimensional routing space. In this example, U1 and S1 overlap so attribute updates from the object associated with U1 will be routed to the federate that created S1. In contrast, U1 and S2 do not overlap so attributes will not be routed from the federate that created U1 to the federate that created S2.

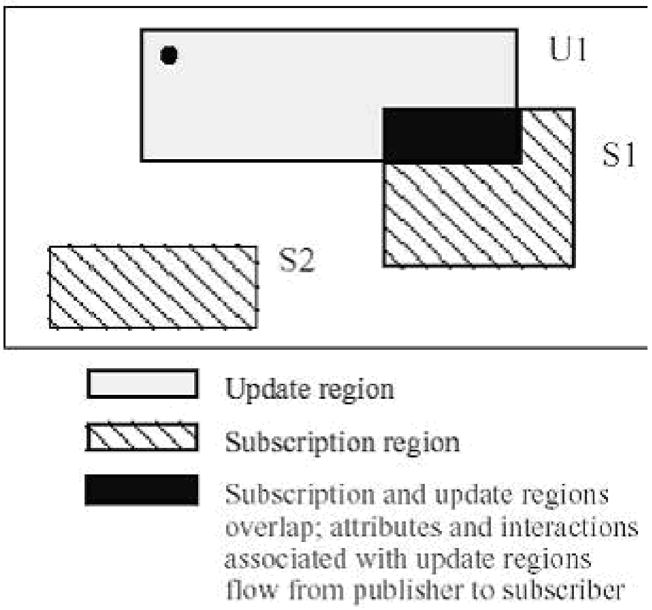

U1
S1
S2
Update region
Subscription region
Subscription and update regions overlap; attributes and interactions associated with update regions flow from publisher to subscriber

Figure 2.3: Two-dimensional Routing Space Example

When an update region and subscription region of different federates overlap, the RTI establishes a communication between the publishing and subscribing federates. The subscribing federates receive only the object class attributes to which they subscribed, although they may receive individual updates outside their subscription region, depending on the precision of the routing space implementation. In Figure 2.3, S1's federate will receive attribute updates from the object associated with U1 as their regions overlap even though the object itself is not within S1.

Each federate can create multiple update and subscription regions. Update regions are associated with individual objects that have been registered to the RTI. A federate might have a subscription region for each sensor system being simulated.

The fundamental observation leading to this strategy is that practically all of the real world objects being simulated will be interested in only a fraction of the objects contained in the entire simulation. For example, a plane is primarily interested only in ground vehicles, such as tanks, that are close to it. It is probably not concerned with the position of any other tanks, some of which might be thousands of miles away. This real-world phenomenon of limited interest would not be taken into account if a DDM is not omni-present. The host that simulates the plane cannot choose to receive only the information about certain objects, i.e., the ones that are within the plane's radar range. Instead, it gets an update message every time when a tank, vehicle or aircraft changes its position. Consequently, this information is always transmitted to all hosts, just in case one should need it.

The cost of this irrelevant data transmission can be very high. If a host receives a large amount of transmissions containing data that are irrelevant, it will waste a significant amount of time and processing resources while receiving and perhaps reading the data. Even if unnecessary data are somehow dismissed by the host without requiring any computation, they will have contributed to increase the traffic on the network. Sending unwanted data to a host puts an unnecessary burden on the sending host (and especially on the receiving host), as well as on the network, which may be inundated by such useless transmissions that will most probably be ignored when they reach their destination.

In RTI implementations multicasting is used to effectively distribute the data to the distributed federates. It is a good way of data distribution; however, it has its own challenges: multicast hardware currently supports a limited number of multicast groups, on the order of a couple thousand. A second drawback is the time required to reconfigure multicast routers as this time can be as big as the total allowable latency for message delivery. As a result, most implementations using multicast have used static assignments of multicast groups, usually fixed to geographic regions. In this type of assignment, when a federate enters to a geographic location, local RTI component joins the multicast groups associated with that region so that the messages for that region can be delivered to that federate.

There are a couple of methods that utilize the multicasting protocol for data distribution management. These methods are mainly divided into two groups as fixed grid-based algorithms and sender based algorithms.

Fixed Grid-Based

Fixed grid-based allocation of multicast groups scheme was used in the initial implementations of the HLA-RTI. It associates multicast groups with cells defined by a grid system overlaid on the terrain [65].

To illustrate how this scheme works, a simple scenario can be illustrated. In Figure 2.4, a grid overlay on the terrain to define cells, and a two-dimensional grid system are shown. Second part of the figure shows how the grid cells are populated with subscribing and publishing regions. The plane is subscribing to the terrain within its radar range, which has been mapped to thirteen cells that are distinguished in the figure by gray shading. Each tank is publishing to a terrain area mapped to one or two cells, which are illustrated in the figure by placing a tank icon in them. The four cells representing both the subscribing region of the plane and the publishing region of a tank have both gray shading and a tank icon.

After publication and subscription regions are mapped to cells, each cell has been associated with a multicast group, i.e., all units within a specific grid cell are assigned to a membership of that multicast group.

In the Fixed Grid-Based (FGB) approach, cell size, and the one-to-one mapping of cells to multicast groups is fixed at the initialization step of the system. The advantage of this approach is that it minimizes inter-federates communication needed to perform DDM because a federate does not need any information from other federates in order to set up multicast group assignments for its units. Us-

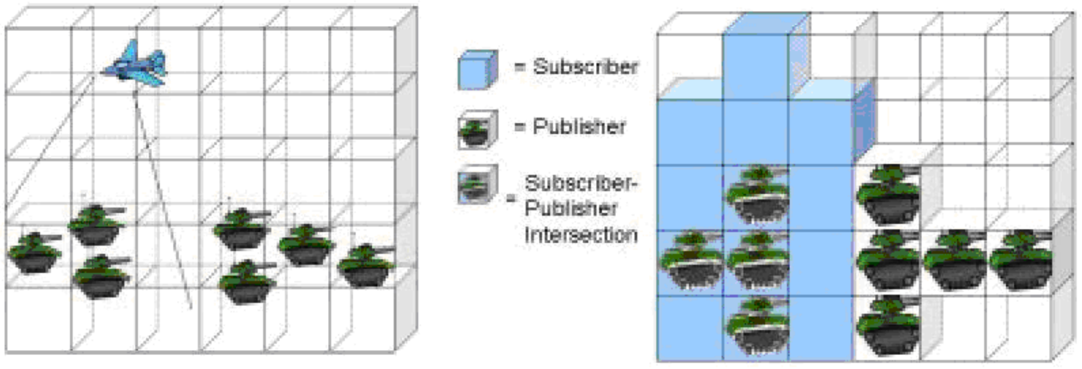


Figure 2.4: Region Grids

ing the fully distributed grid algorithm implemented for the RTIs routing spaces means that determining the destination address was fully deterministic. In this context, routing spaces refer to grids. Therefore, at run-time a federate can assign its units to multicast groups simply by checking the cell-to-group mapping already established during the initialization step [45].

A disadvantage of this strategy is that it requires large number of multicast groups to cover a wide terrain. Experimental studies of this algorithm have found that the optimal cells size is about 2 to 2.5 km. Thus, for a large battlefield, the cell size might easily require thousands of multicast groups. Optimizing the cell size is the main concern in this approach [4]. Associating multiple cells to the same multicast group would reduce the number of groups needed but it decrease the accuracy of DDM filtering, and it would require additional filtering at the source. This tradeoff might not be acceptable in terms of increased cost and complexity.

Multi-level Grids

To combat the problem of using large numbers of multicast groups, the use of multi-level grids has been proposed. This approach is based on the observation that the limit on the number of multicast groups and their associated maintenance overhead is most critical at the wide area network level. A hierarchy of grid-based filters is used, and the cell resolution increases from the wide area network to local area network layers so as to decrease the number of multicast groups needed at the wide area network level while still obtaining the benefits of finer filtering at the local area network level. Another multi-tiered filtering algorithm uses a grid for the first tier of filtering and the Sender-Based strategy for the second tier. However, these approaches are significantly more complex than the fixed grid-based approach.

Sender-Based

Instead of performing a cell-to-group mapping at system initialization, the RTI 1.3 allocates multicast groups dynamically. Like its predecessor RTI-s, the RTI 1.3 uses a grid to determine intersections of subscription and publication regions. However, multicast groups are allocated based on which federates need to send data they are publishing, not on which cells are part of the intersection. For each publisher, a subscriber set is derived specifying which federates are subscribing to cells to which that publisher is publishing.

The approach used by the RTI 1.3 is similar to the distribution list algorithm where each mover maintains a list of sensors that want information about it. A distributed grid-based approach is used, as in the RTI 1.3, to determine the intersection between the areas in which a mover is operating in a sensor's coverage. Here, movers can be linked to publishers, sensors to subscribers, and distribution lists to subscriber sets.

The assignment of multicast groups based on subscriber sets, or distribution lists, can be done by dynamically assigning a multicast group to each publisher created in the simulation, and then have each subscriber in the subscriber set join that group. Subscribers to the group will change as the publisher and subscribers regions of interest change. A possible optimization designed to reduce the number of multicast groups used involves assigning multicast groups only to those publishers that have a non-empty subscriber set.

Clustering

Assigning multicast groups dynamically, based on the number of publishers that have non-empty subscriber sets, may have the disadvantage of requiring large number of multicast groups if there are many publishers. Clustering is a technique that has been used to reduce the number of multicast groups needed. In this technique, multiple publication regions are grouped into a cluster, which is then treated as a single unit and is assigned to one multicast group.

Clustering publication regions, whose cells are in close proximity in the routing space, has the advantage of reducing the number of multicast groups that a subscriber may need to join. This is true, since the subscriber might join only the groups assigned to the cluster instead of all the groups associated with multiple publishers. However, clustering publication regions for publishers that have disjoin subscriber sets may cause several subscribers to receive data in which they have no interest, which partially defeats the purpose of DDM. Nonetheless, the subscriber can of course filter out the unneeded data.

Dynamic Grid-Based

Dynamic grid-based algorithm [67] is a hybrid approach of the static grid based and the sender based approaches. It combines the simplicity and ease of implementation that is associated with grid based method, and the reduction of multicast groups used, which is connected with the sender based strategy.

As in the grid-based method, in this strategy cells are defined by overlapping the terrain with a grid. However, multicast groups are not statistically assigned to all of the cells in the grid. As in the sender-based approach, multicast groups are dynamically allocated based on the current publication and subscription regions in the system. A multicast group is allocated to each cell that has been determined to be part of the intersection of a publication region and a subscription region.

In that algorithm, a distributed grid is created by distributing the cells among the nodes that participate in the simulation. The node to which a cell is distributed is said to be the Owner of that cell. Cells can be distributed over the nodes for example modulo operation. The publishing and subscribing information is hold by the nodes that own the cells. Federates that want to publish and subscribe to such cells should send a message to the owners of that cells (possibly more than one owner)

Publishing and subscribing information of the federates is hold in a bit array, which contains an entry for each federate. When an owner node receives a subscription, it checks if there is a publisher/subscriber match for that cell. If there is one, the requesting federate simply joins that cell, and its information is recorded. If there is no previous match, owner checks whether this new publishing/subscribing creates a match for that cell. If yes, a new multicast group for that cell is created and the requesting federate as well as all previous subscriber and publisher federates join that cell. If this new request does not generate a publisher/subscriber match, the information related with the operation is recorded. The owner informs federates on which should join or leave the multicast groups according to subscribe/unsubscribe and publish/unpublish operations.

Published events should be distributed to the subscribed federates only if corresponding regions overlap. Brute force algorithms [53] check all pairs of regions to find an overlap. Grid-based or hybrid approaches try to reduce the space of the check operation by dividing the total space. However the final decision for overlapping regions should be made by checking the extents of these regions. A group of improvements are proposed for this overlap finding operation [38, 59, 58].

State Update Frequency Optimizations

HLA DDM mechanism is constructed on relevance filtering, in which interests of the federates are defined by the update and subscribe regions. Another group of message traffic reduction techniques is based on arranging the state update frequency. This frequency can be determined by the distance between objects, speed of these objects, and the number of objects around them, etc.

In a space-based quantization scheme [42], the distance between two agents can be more than one critical distance corresponding to spatial relations, thus allowing communication in a more tunable fashion. A quantum is assigned to each distance range created by critical distances. A quantum refers to a rule for transferring or discarding messages from sender agent to receiver agent, and this rule is called a "filtering policy". The quantum size determines how big a change must occur before a message is sent across the network. This approach allows multiple quantum sizes, and thereby allowing communication frequency to be controlled as a smoother function of distance (Figure 2.5).

Another frequency determination method is the flexible state update mech-

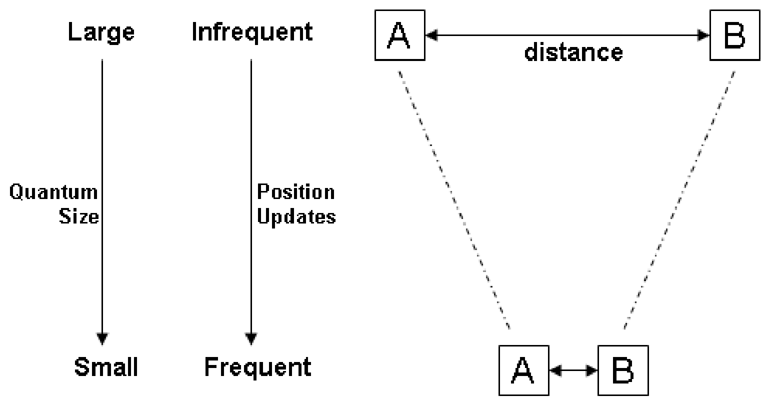


Figure 2.5: Space-based quantization scheme

anism [75] designed for war game simulations. An entity's update frequency is determined by its potential impact on the simulation, i.e. by its moving speed, its distance to other entities and the number of entities around it. Relative importance of the simulation entities is determined and formalized based on the following observations:

An entity has an area of influence on other entities:

- If the entity has influence on a large number of entities, then the state update of entity e may have a great impact on the whole simulation.
- If the entity is close to some other entities, then the state update of the entity may have great impact on other entities (e.g. for collision detection)
- If the speed of the entity is high, it should be updated frequently in order to eliminate anomalies.

2.4.2 Time Management

Time management in HLA [23, 24] is concerned with the mechanism to control the advancement of each federate in the simulation time. A federate may specify whether it wants to use a coordinated or an uncoordinated time. The time management mainly deals with the coordinated time advancement of the federates. The most critical information in time management is the Lower Bound on Time Stamp (LBTS) value and its calculation. LBTS is the minimum of logical time plus the lookahead values of all federates in the federation. It bounds the logical times of individual federates, so that the logical time of any federate could be greater than this value.

Time management services mainly deal with two aspects of federation executions. These are transportation services and time advancement services.

Transportation Services

Both transportation services and time advancement services together manage how the messages are delivered to the federates with different reliability, message ordering and cost. Reliable message delivery means that RTI utilizes some mechanisms, like retransmission, so that a sent message will eventually be delivered to the intended destination. These mechanisms will bring some extra latency during transmissions. On the other hand, the best effort message delivery service can attempt to minimize this latency, but with cost of lower probability of delivery.

Message ordering characteristics specify the order and time at which the messages may be delivered to the federates. Five different ordering mechanisms are specified in the early specifications of HLA: receive, priority, causal, causal and totally ordered, and timestamp ordered. These five mechanisms, in turn, give increased functionality but at increased cost. In current HLA specification (HLA 1516 [2]) only two of these ordering schemes are supported, which are Receive order and Timestamp order. The reason for this reduction is that the complexity of the implementation and execution cost of these ordering mechanisms is relatively high in comparison to their usage frequency.

- Receive order: Messages are delivered to the federate in the order they are received. A FIFO queue is used for this purpose.
- Priority order: Incoming messages are placed in a priority queue, with the timestamp used to specify their priority. Messages are passed to the federate in the order of having the lowest timestamp. This method does not prevent to deliver a message that is in the past of the federate; that is, a message delivered to a federate can have a smaller timestamp than the federate

- 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 of F [74].

- Causal and totally ordered: The causal and totally ordered service extends causal ordering to guarantee that for any pair of concurrent events, messages will be delivered to all federates in the same order, thereby defining a total ordering of events. This type of ordering is referred as CATOCS (casually and totally ordered communications support).

- Timestamp order (TSO): In this service, messages are delivered to the federates in a timestamp order. Further, RTI guarantees that no messages are delivered to federates in their past. All federates receiving the same set of events will receive those messages in the same order, so that total ordering of events is provided.

Receive order is the unmanaged ordering mechanism working as FIFO. This mechanism is generally used in human-in-the-loop training simulations in which each update event overrides previous information. Timestamp ordering is a total ordering mechanism and it is perfectly suited for analysis simulations. Causal ordering is somewhere between receive ordering and timestamp ordering [41, 13]. In this ordering, events are distributed according to cause-and-effect relations.

Lookahead

The timestamp order services require specification of lookahead. Lookahead [35] is defined as the minimum distance to the future where a TSO event will be scheduled. A lookahead value is associated with each federate. If the lookahead value of the federate is L, all TSO events from that federate must have at least the timestamp value of the federate plus L. This lookahead value means that the federate is able to look ahead into the future for the given unit of time. In other words, the federate predicts events at least L time units ahead of its current logical time. Lookahead value is very important for simulations requiring guaranteed message ordering services to achieve acceptable performance.

A single lookahead value is designated by each federate. This value may change during execution, but reductions in the lookahead do not take effect immediately. If a federate reduces its lookahead by K units, this change takes effect after federate advances this amount of time in order to prevent incorrect operations.

In order to explain the reasons for a lookahead specification, consider a federation consisting of many federates. Suppose a federate D is at logical time 10. This federate could generate a TSO message with timestamp 10 to each other federates. This implies that none of the other federates can advance beyond logical time 10, which is a very undesirable situation. With a lookahead value, for example 5, all other federates can advance their times up to 15, which increases the concurrency between the federates.

Determining lookahead is completely dependent on the simulation domain information, and cannot be determined by RTI automatically. Some lookahead determination clues could be:

- Physical limitations concerning how quickly a federate can react to an external event
- Physical limitations about how quickly one federate can effect another federate
- Possible tolerance to temporal inaccuracies
- In time-stepped simulations, lookahead value will probably be the time step amount
- Non-preemptive behavior in the simulation model

Time advance services

In HLA federates may or may not want to use coordinated time in their executions. In such an environment, time management services should know which federates' information should be counted in time computations and which federates require the results of these computations. When the federate declares that it will be time regulating, it declares that it could generate timestamp ordered events and so it should be in the LBTS computation. When a federate declares that it is time constrained, it declares that it can receive timestamp ordered messages and so it requires the results of LBTS computations. Time management has some services that can be used by federates in order to declare these time related information flags.

At any instant during the execution, different coordinated time federates may be at different logical times. Federates should explicitly advance their times, and this operation is not completed until RTI grants the request. The RTI will only grant an advance to logical time T when it can guarantee that all timestamp ordered messages with timestamp less than T have been delivered to the federate. There are two primitives which can be used by federates to advance their logical times: *Time Advance Request*, which is used by time-stepped federates, and *Next Event Request*, which is used by event-driven federates.

Invocation of *Time Advance Request* service implies that all messages with timestamps less than or equal to the given value are eligible to delivery to the federate that invokes the request. After a federate invokes *Next Event Request* either all timestamp ordered messages, with less than or equal to given timestamp value, will be delivered to this federate and the federate will advance its logical time to that given time or if there is not an event that satisfies these conditions, it only advances its logical time to the given value.

Implementation Example

There are not any standard methods for implementing time management functionality. In general, there are two main divisions for the implementation of these services: Central approach and distributed approach. Central approach is straightforward in implementation. Every federate in a distributed simulation environment sends its time related requests (Time Advance Request, Next Event Request, etc.) to the central RTI component. Central RTI calculates the global LBTS (or GALT in HLA 1516) and according to this limiting value, responses to the requesting federate.

Distributed time management algorithms are more challenging. Without having a central component, global time management related state information should be hold in the distributed environment. State information contains the individual logical time and lookahead values of each federate. Also LBTS calculation should be done on each federate in order to advance the logical time of the local federate. The main constraint of distributed time management techniques is the reduction on state update messages. On the other hand, these techniques are standard advantages of distributed systems over central techniques, including non-existence of single point of failure and distribution of system load over a number of processing nodes, etc.

General discussion about distributed time management algorithms are given in section 2.2.1 Conservative PADS techniques. In this section, a fully distributed time management architecture for time advance services implemented in RTI version F.0 [15] is mentioned.

The time advance services serve several purposes. First, they provide a protocol for the federate and the RTI to jointly control the advancement of logical time. The RTI can only advance the logical time of the time constrained federate to T when it can guarantee that all TSO events with timestamps less than or equal to T have been delivered to the federate. At the same time, conservative federates must delay processing any local event until their logical time has advanced to the time of that event, while optimistic federates will aggressively process events and rollback when they receive a TSO event that occurred in their past.

To insure that the RTI properly delivers TSO events, a conservative synchronization protocol implements the TSO event delivery service and it is used to advance logical time. The principal task of the protocol is to determine a value called Lower Bound on Time Stamp (LBTS) for each federate, which is defined as a lower bound on the timestamp of future TSO events that the federate will receive from other federates.

Among time constrained federates, there are three subclasses of federates: (i) conservative event-driven, (ii) conservative time-stepped, and (iii) optimistic. For each of these federate subclasses, the following three time advance services have been devised.

Time advance request with parameter t requests an advance of the logical time of the federate to t. This service is intended to be used by conservative time-stepped federates where t denotes the time of the next time step. Invocation of this service by a time constrained federate implies that the following events are eligible for the delivery to the federate: (i) all receive order events, and (ii) all TSO events with the same timestamp that are less than or equal to t. When the RTI can guarantee that it has passed to the federate all such events, the RTI invokes Time Advance Grant service.

Next Event Request with parameter t requests an advance to logical time t, or to the timestamp of the next TSO event from the RTI, whichever is smaller. By invoking this request, the federate guarantees that it will not produce any new TSO messages in the future with a timestamp less than t plus its lookahead. After invocation of this service, the RTI will deliver all receive order events and either (i) deliver the next TSO event (and all other TSO events with exactly the same timestamp) if the timestamp of that event is less than or equal to t and advance logical time to the time of that TSO event, or (ii) not deliver any TSO events and advance logical time to t.

Flush Queue Request with parameter t requests an advance to logical time t, or to the timestamp of the next TSO event, or LBTS, whichever is smaller, and deliver all receive order and TSO events currently residing within the RTI. This service is intended to be used by optimistic federates, where t denotes the time of local event within the federate that is to be processed.

RTI F.0 time management implementation assumes that all federates are fully connected by point-to-point, reliable, FIFO communication links. Additionally, this RTI is a single-threaded library that links directly to the program of the federate. By having a single threaded RTI, the possibility of race conditions and deadlocks is reduced. Moreover, time consuming IPCs and context switches are eliminated. However, the consequence of this decision is that it complicates the implementations of the federate. In particular, the federate must explicitly yield control to the RTI so that the RTI can process service requests and deliver data to the federate.

As shown in Figure 2.6, RTI is composed of three main parts: (i) RTI ambassador, which serves as an interface for marshaling service requests to appropriate RTI internal managers, (ii) Federate ambassador, which serves as an interface for marshaling service request responses, and (iii) Internal RTI Managers

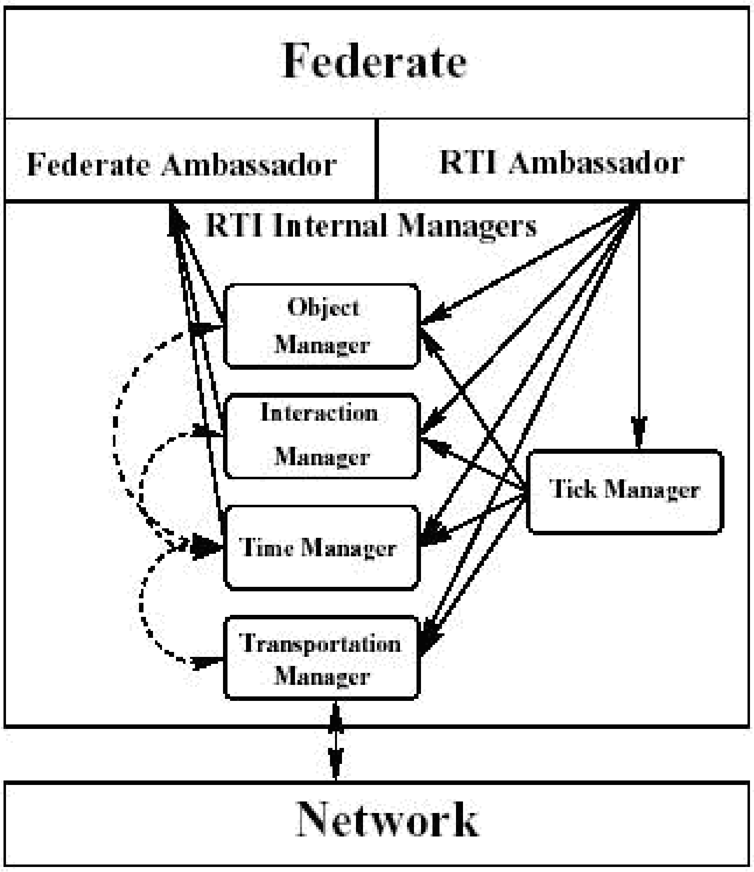


Figure 2.6: RTI F.0 Object Hierarchy

The Internal RTI Managers support five categories of runtime services. The Object Manager service implements the object creation, destruction, ownership, publication and subscription services. The Interaction Manager implements the services that create and destroy interactions as well as interaction publication and subscription services. The Time Manager implements the services for advancing logical time. The Transportation Manager is used to send and receive data and supports the services provided by other managers. The federation management services are implemented in the RTI Ambassador. Tick Manager is used to give control to the RTI, which is invoked by a tick service implemented in RTI Ambassador.

The process is as follows: when a federate requests the tick service, that request is sent to the Tick Manager. The Tick Manager then invokes the tick service provided by each of the RTI Internal Managers to perform their necessary functions. For example, suppose a federate issues a Time Advance Request for some time, t. For the RTI, to process this request, the tick service must be invoked. Upon doing so, the Tick Manager would invoke the Transportation Manager's tick service. Having the single thread of control, the Transportation Manager would deliver pending events from the network up to the Time Manager, where they are enqueued to the appropriate queue (either TSO or receive order). When the tick service of the Transportation Manager is completed, control is returned back to the Tick Manager which immediately gives control to the Time Manager by invoking its tick service. Then, the Time Manager, seeing there is a pending Time Advance Request, examines the receive order and TSO queues, and delivers the appropriate events to either the Object Manager or the Interaction Manager, which directly forward the event, if necessary, to the Federate Ambassador. Note that the Tick Manager was bypassed during the delivery of events from the Time Manager to the Federate Ambassador. This was done to expedite the event delivery process. After the tick service of the Time Manager is completed, the Tick Manager gains control and invokes the tick services on the Object Manager and the Interaction Manager. Once, the Tick Manager has "ticked" all the managers, control is returned to the federate. In practice, the federate will need to invoke the tick service more than once (i.e. "poll" the RTI) before the Time Advance Grant will be issued.

Time Advance Request

The algorithm that is executed when a federate issues time advance requests is shown in Figure 2.7.

First, the Time Manager checks whether the requested time is bigger than the current logical time of the federate. Because the federate can shrink its lookahead during federation execution, the time manager must rearrange the current lookahead. This shrinking requires gradual decrease in the lookahead. This operation is done using currentLookahead = max(specifiedLookahead, LT + currentLookahead − t) operation, where LT is current logical time and t is the requested time. In this operation, the Lookahead value is the lookahead value specified by the federate. This expression guarantees that the federate will advance k units of time before shrinking its lookahead k units.

```
Time Advance Request( FederationTime t)
    case state of
        IDLE
            if t < LT then
                throw FederationTimeAlreadyPassed()
            end if
            currentLookahead := max( specifiedLookahead,
                LT + currentLookahead − t)
            requestTime := t
            LT := t
            if timeRegulating then
                send newLT((LT + currentLookahead),
                    requestTime)
            end if
            state := TIME_PENDING
            more := TRUE
            fifoRemainingToDequeue := size of FIFO queue

        EVENT_PENDING
        TIME_PENDING
        FLUSH_PENDING
            error TimeAdvanceAlreadyInProgress
    end case
```

Figure 2.7: Time Advance Request Service

Next, a newLT message that contains LT + currentLookahead and the request time is broadcasted to all other federates. This message can be viewed as a null message from the CMB null message algorithm. When another time constrained federate receives the newLT message, the logical time plus the lookahead value is stored and it is used to compute LBTS. LBTS for a federate is defined as the minimum of all other federates' logical clock plus lookahead values and it is computed every time a new LT value arrives.

Afterwards, having sent the newLT message, the state of the Time Manager is set to Time Pending. Now, to tell the Tick Manager that the Time Manager has more work to complete, *more* is set to TRUE.

At this point, the size of the receive order event queue is stored before returning control to the federate. When the federate yields control to the RTI via the tick service, the Tick Manager will invoke the Time Manager's tick service. In this service, the delivery of all receive-order events occurs before any TSO events are considered for delivery. Delivering all receive order events will take several invocations of this service since only one event is delivered per invocation.

```
Do Time Pending()
    case findCases() of
        DELIVER_DATA:
            dequeue one TSO message
            more := TRUE

        GRANT:
            grant LT
            state := IDLE
            more := FALSE

        PENDING:
            fifoRemainingToDequeue := size of FIFO queue
            if fifoRemainingToDequeue = 0 then
                more := FALSE
            else
                more := TRUE
            end if
    end case
```

Figure 2.8: Do Time Pending Services

Next, after the receive order events have been delivered, TSO events may be delivered if the federate is time constrained. As the state of the Time Manager is Time Pending, the Do Time Pending decision service (see Figure 2.8) is invoked, which determines which, if any, TSO events are to be delivered to the federate so that the Time Advance Grant is issued. By comparing the request time, the timestamp of the head event of the TSO event queue and the LBTS, the proper case grouping can be determined.

In this determination, the RTI must consider the relationships between (i) the timestamp of the event at the head of the TSO queue, (ii) the requested time to which the federate wishes to advance and (iii) the LBTS. Shown in Figure 2.8, there are six relationship cases between these three factors determining how a decision is reached on whether the RTI should (a) grant, (b) deliver a TSO event, or (c) do nothing. Head denotes the timestamp of the event at the head of the TSO queue. LBTS denotes the lower bound timestamp on any event that could be sent to this federate in the future. t denotes the time to which the federated has requested to advance.

The case grouping names shown in Figure 2.9 denote the action taken by the Time Manager generally in a particular case. Consider the Grant Cases (cases 1, 3, and 5): it is observed that the request time is always smaller than either the head of the TSO queue or the LBTS. Because of this, it is safe for the Time Manager to issue a Time Advance Grant to the request time, t. Likewise, in the Deliver Data Case (case 2), since the head of the TSO queue is the smallest of the three factors, the event that is at the head of the TSO queue can be delivered. Last, in the Pending Cases (cases 4 and 6), the RTI can neither grant nor deliver data since both the request time and the head of the TSO queue are greater than or equal to the LBTS. Note that, there are some exceptions to the actions taken by the Time Manager, depending on the time advance service.

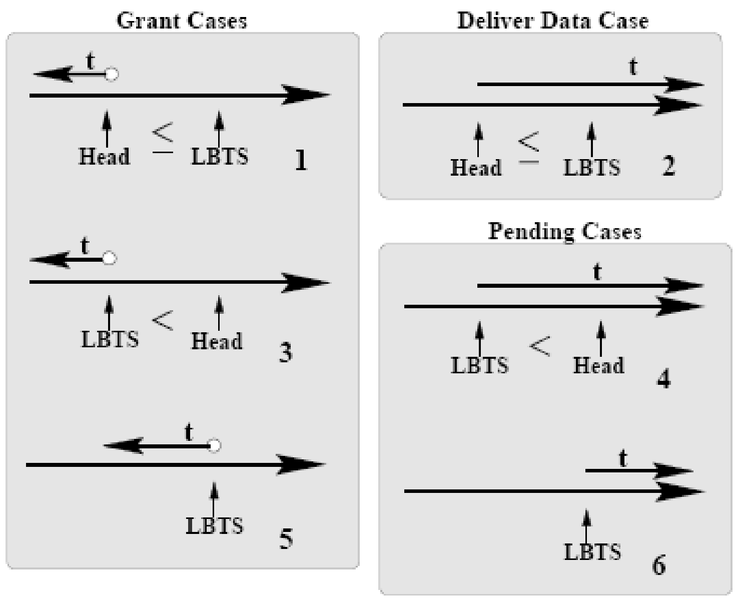


Figure 2.9: Timestamp Order Event Queue Case Groupings

If the case is Deliver Data, this service will deliver a single TSO event and set more equal to TRUE. If the case is Grant, then a Time Advance Grant is issued for the logical time of the federate, LT, and the state is set to Idle and more is set to FALSE, indicating this service request is completed and the Time Manager has no pending work to perform. If the case is Pending, no actions can be taken and the number of receive order events is recalculated to be used in delivering of receive order events in future invocations of the Time Manager's tick service. If there are receive order events to deliver, more is set to TRUE. Otherwise it is set to FALSE.

2.5 Federation Communities

The federation communities are a group of communicating federations and RTIs. The purpose for federation communities is to connect separated federates. This federation division generally exists because of different network locations. As the existing RTIs do not have the feature that connects federates on, for example, different WANs, a gateway is generally used to get them together [47, 30]. These gateways become the communication channels between sub-federations (see Figure 2.10).

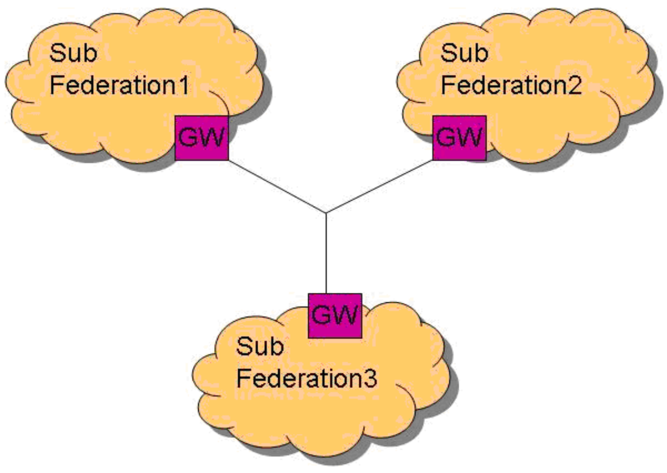


Figure 2.10: Federation Community

Hierarchical federation [48] is a type of the federation communities in which every lower level federation is a federate of a higher level federation. In this type of federation community, gateway federate is connected to both lower level user federation and higher level gateway federation, and reflects the updates of local federation to other user federation through the gateway federation(s). A hierarchical federation is shown in Figure 2.11 as an example.

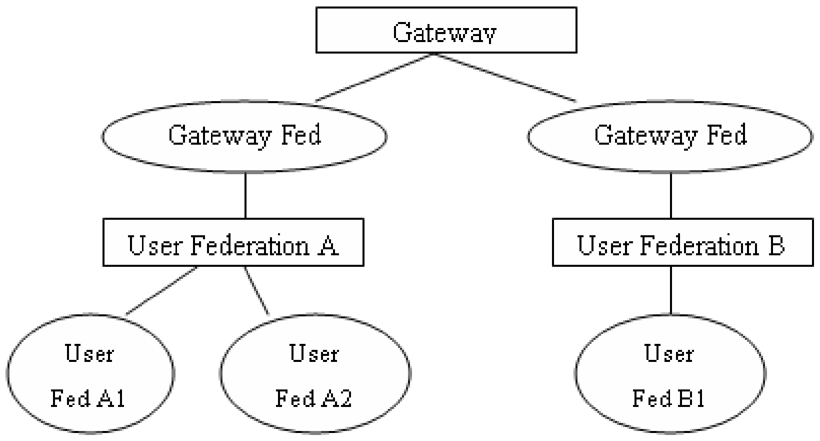


Figure 2.11: Hierarchical Federation

Hierarchical federations aim also to hide the critical information between federate groups. Information that is produced within federates are distributed to other federates with specifications defined in the object models in FOM and SOM. In some applications, information is needed to be hidden from some group of other federates. For example, in a battlefield simulation, information of movement of a group of war elements should be only shared within ally federates. This kind of information hiding can be realized using subscriptions to only related parts of simulation objects. However, this kind of policy cannot be forced.

Hierarchical federation architecture can be used to force information hiding between federations. Additionally, the gateways or proxies that are used to connect federations in hierarchical federations cannot be trusted by any of federations because these gateways join to all federations. In a sample architecture [12], there is a corresponding gateway federate for each federation. These gateways are added only to owner federations. Connections between federations are supplied by the RTI Gateway. Gateway federates of each federation connect to RTI Gateway and the exchange of information is completed there. Thus, information hiding is guaranteed as each gateway federate is under the responsibility of the federations.

2.5.1 Time Management in Hierarchical Federations

All user federates in hierarchical federations act as they are in the same local federations. This brings the issue that logical time of all time constrained federates are determined by the all time regulating federates in the global federation community. As not all user federates in the hierarchical federations are connected to the same RTI, low level RTIs cannot handle time synchronization operations. The gateway federates should get the time requirements of user federates and synchronize their logical time to each other.

Hierarchical federations are oriented so that federations are hierarchically grouped in a tree structure manner. In a study [20], federate proxy components that connect different federations are distributed over the network. Distributed Federate Proxy Components (DFPC) architecture is shown in Figure 2.12. In this architecture, DFPC components are joined to each federation. They are also connected to upper level gateways called SimNode. This hierarchical architecture of distributed federate proxy results in finer information sharing in between federations.

This distributed and hierarchical structure of DFPC is used to manage logical time between federations. In this study, all DFPC are defined as time regulating and time constrained, so that their logical time depend on both lower level user federates and higher level gateway federates. In addition, their logical times affect both user federates and gateway federates. However, if there is not user federates present that are time regulating, there is no need to make the gateway federate as time regulating, because this federation will not generate time-stamped events. Similarly, if there is not user federates present that are time constrained, there is no need to make gateway federate as time constrained.

Logical time of each DFPC is constrained by its lower level federation and upper level SimNode [20]. Similarly, each DFPC regulates logical time of these. Similar conditions are applied for SimNode elements. In this architecture, the top

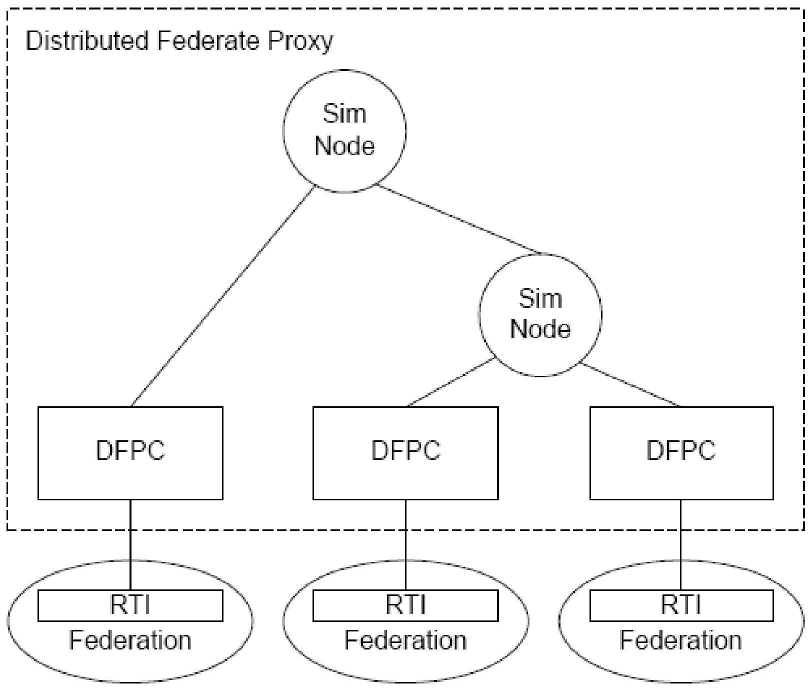


Figure 2.12: Distributed Federate Proxy Architecture

level component of the hierarchical federation is to determine the element used as a base for the logical times of the rest of the federation community. Thus, LBTS of root element is at most the earliest possible timestamp assigned to a message generated by any federate in the entire hierarchical federation community. This implies that all time advance decisions are made by this top level element.

2.5.2 Data Distribution Management in Hierarchical Federations

In hierarchical federations, individual federates are connected to each other with a connection node such as a gateway federate or a proxy. In this architecture, any event that has an effect on a special region in one federation should be reflected to all other federations with the same effect.

In the hierarchical filtering mechanism [43], each subscription region of a gateway for an event (an interaction or an attribute update) is determined by the union of subscription regions of the lower level federations of the gateway. In order to construct this mechanism, whenever a federate in a lower level federation subscribes to an event with a region, the gateway federate should subscribe the same event with the same region. Thus, in the federation, events are reflected to the next federate only when update and subscription regions overlap.

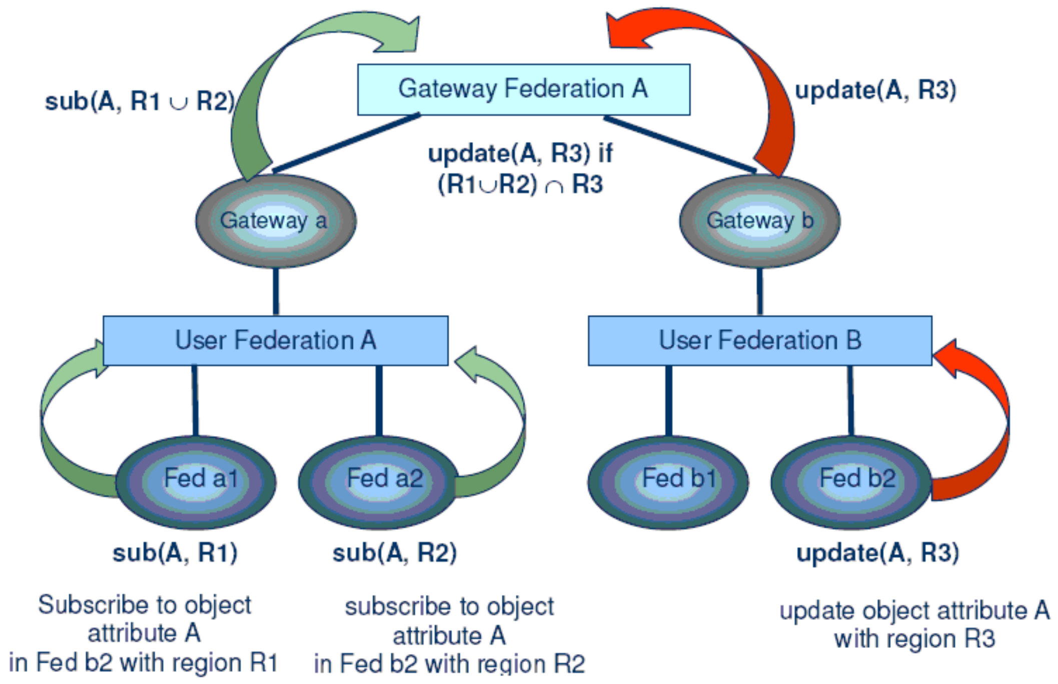

sub(A, R1 ∪ R2)
Gateway Federation A
update(A, R3)
update(A, R3) if (R1∪R2) ∩ R3
Gateway a
Gateway b
User Federation A
User Federation B
Fed a1
Fed a2
Fed b1
Fed b2
sub(A, R1)
sub(A, R2)
update(A, R3)
Subscribe to object attribute A in Fed b2 with region R1
subscribe to object attribute A in Fed b2 with region R2
update object attribute A with region R3

Figure 2.13: DDM implementation in Hierarchical Federations

In Figure 2.13, an example is shown for hierarchical filtering. Both Fed *a1* and Fed *a2* in User Federation A subscribe to the same object attribute A within region R1 and R2 respectively. Their subscriptions are propagated via the gateway a to Gateway Federation A. Thus, in Gateway Federation A, the subscription region for object attribute A is the union of the region R1 and R2. Similarly, in Fed *b2* of User Federate B, object attribute A is associated with region R3. Its associated region is also propagated via gateway *b* to the Gateway Federation A. So if Region R3 has no overlap with the union of Region R1 and R2, the attribute updates are filtered at Gateway Federation A.

CHAPTER 3

PROXIMITY AWARE TIME MANAGEMENT

3.1 Problem Statement

In HLA, simulation elements that are distributed around computing nodes use logical times in order to synchronize their events. Logical times ensure that events in simulations are processed by federates in synchrony. In a distributed simulation environment, there are time regulating federates and time constrained federates constrained by these regulating federates. In order to simplify the case, assume that all federates in the federation are both time regulating and time constrained. This means that all federates could generate timestamp ordered events and could process timestamp ordered events. In existing time management algorithms, a federate could advance its logical time only if the global state of the federation allows this operation. The condition to allow a federate to advance its logical time is that after the federate advances its logical time, for example, to LT1, distributed simulation architecture should not deliver any timestamp ordered messages which have smaller timestamps than LT1. The time management architecture should check all logical time states of all federates before allowing a time advance operation.

Each federate can advance its logical time only after states of all other federates allow this operation. This fact actually decreases the overall concurrency level of the simulation. The objective of this study is to improve the concurrency whenever possible.

In some of the simulations, a group of federates could be separated from others and these federates could interact only within themselves. For example, in a space simulation, there could be a number of federates simulating spaceships and ground stations. Initially, when they are all on Earth, they could interact with each other. It is normal that logical times of all federates are synchronized with each other at this level. After a while, one or more spaceships may leave Earth in order to go to Mars or to some other planet on which communication with Earth is not possible. In this case, there will not be any interaction between spaceships and ground stations for a period of time. When spaceships are on Mars, federates simulating spaceships and ground stations constitute two different groups, and federates in each group interact only with federates of the same group. In this study, these logically related group of federates are called Federate Clusters.

In fact, there is no need to synchronize the logical times of federates that are currently not related with each other. If the logical time dependency between these two federate clusters are broken, their concurrency level is possibly increased as their logical times will be constrained on less number of federates. Additionally, because of this partitioning, there will be some performance improvements in the time management algorithms.

The only standard HLA supplied mechanism that could be used to break the logical time dependency between two or more groups of federates is disabling and enabling time constrained and time regulating properties of the federate. Federates using these properties are removed from the dependency list of time management. By disabling time management, the federate could not send or receive TSO messages, and its logical time synchronization with other federates will be broken. Thus, in most cases, disabling time management is not a desired feature. First of all, the aim is to construct two different time consistency groups, not disabling time management in one group. Because, after disabling time management, federates in that group start processing events in the receive order, which is not a desired situation.

3.2 Proximity Awareness in Time Management

In a distributed simulation environment, the federates move around the virtual space defined by the simulation. In some cases, the federates become so distant that they could not communicate with each other for a period of time. If these non-interacting periods can be determined, these separated federates can be allowed to advance their logical times independent of each other.

In this study, a new time management mechanism is proposed in which federates are divided into logically related groups. These groups are called federate clusters. In this context, a Proximity Aware Time Management (PATiM) mechanism to manage federate clusters' logical times is proposed. The aim of this mechanism is to increase performance of time management mechanism through increasing concurrency by reducing unnecessary blocking of federates.

The proposed **proximity-aware** mechanism tries to detect the non-interacting time periods between federate clusters by checking distances between federate groups in the virtual space. In this proximity relation, a volume in routing space is calculated for every federate cluster and distances, in turn, are calculated according to these volumes. When a distance between two federate clusters exceeds a predefined threshold value, it is decided that these two federate clusters are distant to each other and they will not interact as long as they continue to be far from each other.

Separate federate clusters do not need to be time consistent because of the non-interacting period between them. Breaking the logical time dependency between federate clusters will result in inconsistencies between their logical times. Of course, broken logical time synchronizations between clusters should be reconstructed if they want to interact again.

When the logical time dependency between two different clusters is broken, a number of performance improvement opportunities are expected. The first one is the dramatic decrease in the number of synchronization related messages, because the big federation is divided into smaller synchronization groups. Another performance improvement occurs in the form of reduction in the LBTS calculation. Further, because of the broken logical time constraint between groups of federates, the concurrency of the federation will be increased. The detailed analyses of these performance improvements are given in Chapter 5.

The data distribution management described in HLA specification allows detection of intervals in which federates are not interested in each other's events. When regions of two federates do not overlap, or when there is a distance between them, it can be said that these federates will not interact. The amount of distance between these regions determines how long this non-interaction period will continue.

Logical divisions between federates in a federation could dynamically change. For instance, in a battle field simulation example, sub-battles could take place in distant parts of battle field. So, federates in the same sub-battle would constitute a federate cluster. However, at some point in time, a group of airplanes in a sub-battle at Diyarbakir could move to Istanbul to join the battle there as a result of a tactical decision. In this case, logical relations between these airplanes and other war elements at Diyarbakir are broken and they become logically related with battle elements in Istanbul. It is also possible that they remain unrelated for the rest of the battle.

In the proposed dynamic cluster formation mechanism, a federate cluster is formed for each group of federates that are close and related to each other. During simulation execution time, federates could move away from their federate cluster and become closer to another cluster because of their internal logic. In this case, moving federates will also leave the previous federate cluster and join the new cluster to which it gets closer. Because of this dynamic property of federate cluster formation, the membership in a cluster will always change.

The scope of the time management is based on proximity degree between the federates. Federates are said to be time consistent when they are in the same cluster and they are said to be time inconsistent when they belong to different clusters.

In the proposed synchronization method, a volume in routing spaces for each federate cluster is calculated. By measuring the distances between these calculated volumes, a decision is reached on whether some federate clusters are distant to each other, based on a distance threshold value. When clusters are distant from each other, none of the subscribe and update regions will overlap with each other, so no interactions could take place between them in terms of the Data Distribution Management states.

During the design of Proximity Aware Time Management (PATiM) mechanism, some assumptions are made. In this subsection, these assumptions are analyzed.

Firstly, it is assumed that federates work with time-stepped manner. This means that each federate iterates in order to advance its logical time. Logical time must have a one-to-one relationship with physical time. This assumption is required because our local synchronization and resynchronization mechanisms require that federates should not jump instantaneously from one location in routing space to another location. If this occurs, critical distance calculation of PATiM will fail. Simulations obeying this assumption are generally physical world simulations such as particle simulations.

The second assumption is that the federates should strictly use the DDM services in order to publish and receive events. PATiM mechanism uses subscribe and update regions to determine the non-interaction periods between federate clusters. If a federate sends events without using the DDM services, non-interacting period determination operation will fail. Additionally, these declared regions are used to calculate the distances between the federates. If a federate fires an interaction without a region, this interaction will be delivered to the entire federation. If federates are distant to each other at that time and if their logical times are not synchronized, the receiving federate could process an event in its past. This interaction event will break our assumption that these federates will not interact when they are distant.

As an important assumption for performance the simulation is expected to have non-interacting periods between some groups of federates, to be able to divide them into federate clusters. In a traffic simulation example, federates simulating vehicles possibly declare that they are interesting in for example 2 km radius area. These federates will not be interested in events occurring outside this boundary. Using this boundary information, PATiM mechanism could divide the group of federates that are distant to construct federate clusters. However, if federates in the simulation could process events from all over the virtual environment, then there is no chance of clustering the federation and there will not be any advantage of the proposed mechanism.

In this work we mainly focus on the simulations of real world spatial environments. In such simulations, routing space is composed of dimensions of spatial coordinate system which have continuous property. Thus approaching speed concept is understandable in the usual sense.

3.3 Dynamic Cluster Formation

In the proposed dynamic cluster formation mechanism, a federate cluster is formed for each group of federates that are close and related. During simulation execution, federates could move away from their federate cluster and become closer to another cluster because of their internal logic. In this case, the moving federate should leave the previous federate cluster and join the cluster to which it gets closer. Because of this dynamic property of federate cluster formation, federates in a cluster will always be logically related with each other during simulation execution.

The relevance of federates is taken to be a function of a degree of proximity which is defined as the distance between them in the virtual space of the simulation. In PATiM, a federate is represented in virtual space by the union of the update and subscribe regions that are defined by the federate. These regions actually determine the positions of any federate in the virtual space. The distances between federates are calculated by measuring the distances between their unioned volumes.

In order to be able to calculate meaningful distances, dimensions of routing spaces should have continuous property allowing the distance between two federate clusters to shrink or expand. An example routing space obeying this condition is the world's coordinate system. Federates in such a space could only move continuously, without jumping. However, in discrete routing spaces, federates could jump from point to point, as in the gears of a ground vehicle. A third kind of routing space has discrete representation with continuous property. For example, a routing space representing floors of an apartment would be declared as discrete. However, a change of the value of the federate from 1 to 3 means that this federate will have to pass through floor 2 as well. So, these kind of routing spaces are also treated as continuous.

During the course of the simulation, cluster formation algorithm continuously repeats two actions:

- Checks distances among federates within federate clusters to validate dynamic formation.
- Checks distances between pairs of federate clusters to determine whether they should be changed.

3.3.1 New Cluster Formation

This step checks whether an existing federate cluster should be decomposed into two different federate clusters. An increase in distance between groups of federates is an indication of a breakaway in logical relation between these groups. In this case, the unrelated groups of federates in this cluster are divided into two different clusters as shown in Figure 3.1.

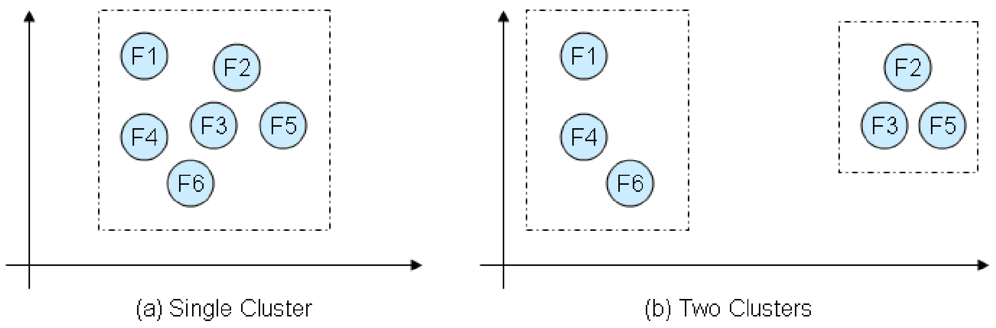


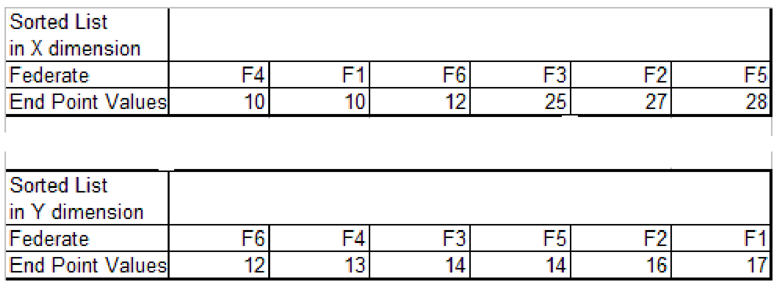
Figure 3.1: Federate Cluster Division

In distance calculations, subscribe and update regions defined by the DDM services are used in a modified version of Sweep-and-Prune algorithm [19]. This algorithm constructs a list for each dimension of routing space that includes the values of end points of update and subscribe regions of the federates. Sorted distance lists give information on relative positions of the federates in the virtual space.

In calculation of the sorted list, temporal coherence property [19] is exploited, meaning that positions of these federates in virtual space do not change dramatically between successive time steps. In a traffic simulation, for example, the federates simulating traffic vehicles advance positions of vehicles according to the simulated speed of the vehicle and the amount of the time step. Most probably, this position update moves the vehicle by an amount in meters (not kilometers) which is relatively very small compared to the size of the virtual terrain. The locations of the federates will not change drastically during these updates. Thus, the distance lists sorted will have close values of distances, and re-sorting them using insertion sort is expected to have linear complexity in the number of federates in the cluster.

A federate cluster is divided into two parts according to the proximity criteria if there is a sufficiently large gap in at least one of the sorted lists. When the difference between two successive values in any of the sorted lists is greater than a predefined threshold value, the algorithm divides the federate cluster into two different clusters.

Figure 3.2 gives sorted lists for two dimensions of the clusters, which were shown in Figure 3.1 (b). For simplicity, only one end point value is shown for each federate. Assuming the threshold is 10, there is a gap between values of F6 and F3 in the sorted list in X dimension. As a result, the original federate cluster is divided into two clusters.



Sorted List in X dimension						
Federate	F4	F1	F6	F3	F2	F5
End Point Values	10	10	12	25	27	28

Sorted List in Y dimension						
Federate	F6	F4	F3	F5	F2	F1
End Point Values	12	13	14	14	16	17

Figure 3.2: Sorted lists in two dimensions

In this cluster formation algorithm, the amount of the gap is the approximate distance between federate groups. It is an approximate value because an exact calculation for distance between two points is not done. The algorithm searches for a gap in sorted lists and takes the value of this gap as the distance. In fact, this value is smaller than the actual distance between these two federates.

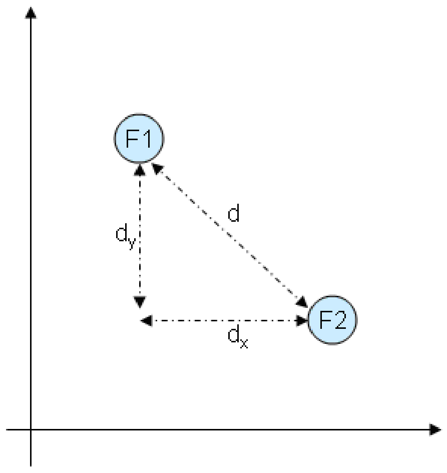


Figure 3.3: Dimensional distances

The distance between two federates is approximated by the maximum of dimensional distances because the algorithm tries to find a gab in at least one of the dimensionally sorted lists. Gaps in these lists represent the dimensional distances between the federates. In Figure 3.3, distance components for two federates are shown. In the new cluster formation algorithm, dimensional distances, dx and dy, are known, however the actual distance d is not calculated. Approximate distances are enough for the proposed algorithm because the important issue is just determining the proximity of the federates. In fact, this algorithm just tries to determine whether the distance between two federates exceeds a predefined threshold value. If one of the dimensional distances exceeds this threshold, it is obvious that the actual distance will also be exceeded because of the triangle rule.

The only effect of using approximate distances is the possible late division of federate clusters. The algorithm checks for dimensional distances to find a gap that exceed the threshold. However, the actual distance is greater than any dimensional distance. Thus, the cluster could possibly be divided earlier if the algorithm would have checked whether the actual distance exceeds the threshold. On the other hand, the possible late division of the proposed algorithm does not generate any problems.

3.3.2 Merging two clusters

In this step, the proposed algorithm tries to determine if two different federate clusters should be joined because of the decrease in the distance between them. For this purpose, the algorithm calculates the positions of federate clusters and then checks if the distance between the federates is smaller than a threshold value. If this condition holds, the two clusters are joined to form a new cluster.

In order to determine the location of the federate clusters, union of all update and subscribe regions of the federates in the cluster is calculated. This multi-dimensional volume represents a cluster in a given routing space. Unioned volumes are constructed by getting the maximum of the end point values of all update and subscribe regions. In Figure 3.4, the unioned volume of an example federate cluster including five federates is shown.

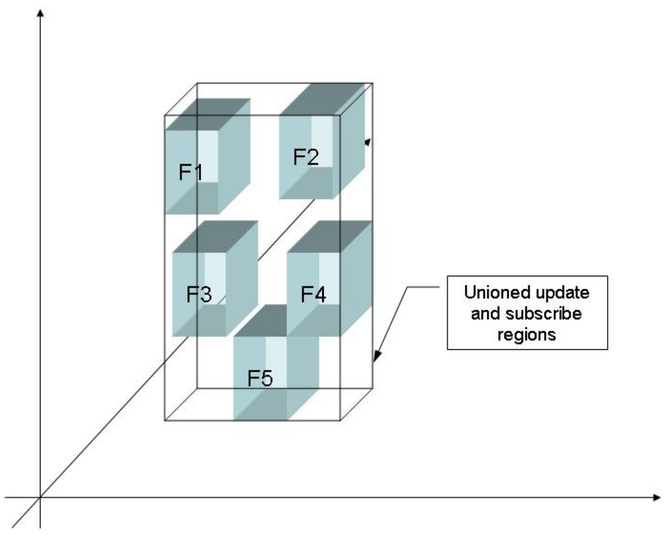


Figure 3.4: Union mechanism for a 3-dimensional routing space

The union of update and subscribe regions of federate cluster (FC) in the routing space RS_k is computed as:

$$U_{FC_i}^{RS_k} = \cup \left\{ SR^{RS_k}\left(F_j\right) \cup UR^{RS_k}\left(F_j\right) \right\} \; \forall F_j \in FC_i \tag{3.1}$$

In this formula, $SR^{RS_k}\left(F_j\right)$ is the subscribe region of j^{th} federate (F_j) in k^{th} routing space (RS_k) and $UR^{RS_k}\left(F_j\right)$ is the update region of F_j in RS_k.

Distance between FC_a and FC_b according to routing space RS_i is computed as

$$d^{RS_i}\left(FC_a, FC_b\right) = Distance\left(U_{FC_a}^{RS_i}, U_{FC_b}^{RS_i}\right) \tag{3.2}$$

Distance() relation calculates the distance between two regions in terms of the distance between two rectangular volumes in a n-dimensional space.

When distance between two federate clusters is smaller than a predefined threshold value for a given routing space, the algorithm combines these two clusters. The condition for this decision is formulated as following:

$$d^{RS_i}\left(FC_a, FC_b\right) \leq \omega^{RS_i} \; \forall RS_i \tag{3.3}$$

The threshold given in the formula above is called the *Cluster Join Threshold.* These threshold values for routing spaces should be given to the algorithm as explicit parameters.

The proposed method assumes that as time passes, unified update and subscribe regions of federates or clusters either get closer or get away from each other. This means that, the combined regions move in the corresponding routing space. As a result of this movement, the distance between these federate clusters changes. This change determines the corresponding approaching speeds of the clusters. Actually the movement of combined regions could have two different meanings. One of them is that individual regions, to which the federates registered for update or subscribe, move in the routing space; meaning that the federate owning the region, updates its parameters so that the position of the region in the routing space is changed. The other possibility is that the federate unregisters from one of the regions and registers to another one that has a different location. In both cases, the resulting combined region of the federate cluster will be moved from one location to another.

3.3.3 Clusterability of distributed simulation

In the proposed method, the federates are separated into interest groups called federate clusters because of their mobility around the virtual terrain. PATiM mechanism utilizes the clustering of federations in order to increment the time management performance. Proposed mechanism, on the other hand, is based on the observation that distant federates are not interested in the events of each other. By using Data Distribution Management services, the federates guarantee that they are not interested in events that happened out of their subscribe area.

The traffic simulation example perfectly fits to these observations. Vehicles in a traffic simulation move around the virtual city and they can only see events that occur around them. However, the situation seems to be more complex for some type of simulation objects like radar systems. It can be very though that they can see everything in the virtual terrain. But generally, they have also a range in which they can see events around them. Of course their ranges are much bigger than a car in a traffic simulation.

How much a federation is divided into clusters is mainly dependent on the simulation domain. If simulation objects move in a small region, then there will not be many clusters. On the other hand, if federates are spread around on a wide virtual space, there will be a number of clusters. In a traffic simulation example, if vehicles turn around one or two blocks, there will be possibly only one cluster. However, when they move around the whole city, there will be more clusters.

Advantages of the PATiM mechanism appear when there are two or more clusters. Thus, higher average cluster count in a federation is better. Average cluster count is related to the duration of divisions as well as the total number of clusters. For example, if two groups of traffic vehicles are getting closer repeatedly and become distant only in small periods, there are two clusters but the average cluster count is smaller than two. In another case, if these two groups of vehicles become distant for longer periods, average cluster count and gain of PATiM will be higher.

Another factor affecting average cluster count is the selection of clustering and joining thresholds. The threshold values determine when to divide and join clusters during the simulation execution. Small threshold values result in early division and late join of clusters. Of course, these operations result in higher cluster counts. However, having small threshold values may lead to the ping-pong effect. This effect occurs when clusters rejoin after a small amount of time they have divided. This effect downgrades the performance of PATiM mechanism because of the overhead of dividing and joining the clusters. Therefore, threshold values should be selected so that after the distances between the federates exceed these values, there should not be any logical relation between the federates.

In general, performance gain of PATiM mechanism is highly dependent on the increased clustering within the federation. If there is a simulation element which is interested in the whole simulation space, there could not be a clustering in the simulation. Of course, in this situation, there will not be any performance improvements supported by the PATiM mechanism. During the simulation execution, average cluster count should be high in order to maximize the performance improvement, and this condition is mainly dependent on the simulation domain and on the behaviors of the simulation objects that are owned by the federates.

3.4 Time Management

PATiM federates are said to be time consistent when they are part of the same cluster, otherwise they are said to be time inconsistent. As an example, consider a space simulation, in which a group of spaceships travels to a distant planet and then returns. It is assumed that at a large distance to Earth, the communication between the spaceships and the Earth station will be broken. In a distributed simulation environment, a routing space representing a 3-dimensional space could be created for this application. Using update and subscribe regions, broken communication situation could be simulated. The group of spaceship federations constitutes a federate cluster because there is a tight logical relation between them. Let's assume that the computing nodes that execute space ship federations have higher capacity and could run faster than computing nodes of Earth station federates. When the spaceship cluster is getting far from the Earth, a non-interactive period begins between the spaceship group and the earth station group. During this period, the faster cluster, in this case the spaceship cluster, would be unnecessarily blocked from advancing its logical time. According to the proposed algorithm, when the distance between the spaceship group and the Earth exceeds a threshold value, the faster federate cluster could advance its logical time without waiting for the slower Earth federate cluster. The threshold value in this case could be selected according to the distance after which the communication equipments could not contact each other.

The time management is based on a sequence of consistent and inconsistent execution periods. It initiates with a single federate cluster containing all the federates with synchronized logical times. When the distance between federates gets bigger, the cluster will be divided into two new clusters and the logical time dependency between them will be broken. This inconsistent period is called as the local synchronization period.

On the other hand, inconsistent distant clusters may approach to each other during future time periods. In this case, they will start to interact when they are joined because of the proximity criteria. These possible interactions must occur in a time synchronized environment. Thus, a resynchronization operation is executed to synchronize the logical times of the approaching clusters.

3.4.1 Local Synchronization Period

When two federate groups are in different federate clusters, they enter into the local synchronization period. In the local synchronization period, the logical times of distant parties are not dependent on each other. This means that logical time dependencies of these federates are broken. Logical times of time-constrained federates are only dependent on time-regulating federates of the federate clusters they belong to. This will be achieved by modifying the LBTS calculation algorithm. The LBTS value of each federate will be the minimum value of logical time plus the lookahead values of time-regulating federates in the current federate cluster.

As federates always use DDM based message filtering, when they are on different federate clusters, their updates are not destined to each other. Thus, the individual cluster-wide logical times will not cause any logical problems. Logical problems could occur if a federate receives an external event from a federate that is in another federate cluster; as in this case, it may have to process the event in its past. However, this is not possible as all of the federates declare their interest areas using the update and subscribe regions and the PATiM mechanism breaks logical time dependencies with the help of the proximity degree that is calculated using these regions.

3.4.2 Resynchronization Period

During simulation executions, different federate clusters may get close because of the internal logic of the federates. Before these clusters are joined, their logical time should be synchronized as federates will initiate interaction when they get in the same cluster. The purpose of this resynchronization period is to prepare federate clusters for the joining operation. In this period, some operations are done in order to synchronize logical times of federate clusters. There are two different methods for this resynchronization; the first one is the optimistic approach and the second one is the conservative approach.

Optimistic Approach

Optimistic approach is the naive way of resynchronization of two federate clusters that get closer to each other. In this approach, logical time of the slower federate is advanced to the logical time of the faster one.

This method delivers the messages in a logically consistent order. However, this method may not work for simulations that contain internal timed events, because these internal events in slower federate clusters could be missed when the logical time is jumped to a future value. For example, if the federate cluster 1 has an internal event scheduled to occur at logical time 5, this internal event will be missed when the logical time is advanced to logical time 6. Especially time stepped simulations will have internal events at each time step. For example, in a traffic simulation, the federates simulating the vehicles will have time steps of 2 unit of logical time. In all of these time increments in steps, vehicles forward their positions with a calculated amount. These position change operations are of course an internal event in each federate.

One solution to this problem is processing these internal events at the moment of the jump operation. However, it cannot be guaranteed that these early processing of internal events have no negative side effect on the system.

Conservative Approach

Conservative approach is designed to overcome the internal event problem of the optimistic approach. In this method, logical times of the federate clusters are aimed to be synchronized slowly. Because of the local synchronization period, logical time of one of the clusters is smaller than the other one before initiating the resynchronization period. Conservative approach provides a chance to the slower federate to catch up the faster one.

In this approach, the faster cluster is blocked from advancing its logical time so that the slower cluster could catch it up before the proximity condition is broken. In this resynchronization period, the slower federate cluster continues its execution normally and advances its logical time. At some point in time, the logical times of both clusters become synchronized. At that point, resynchronization period will finish and the normal synchronized execution continues.

In Figure 3.5, there are two federate clusters each consisting of three federates. Messages are shown with timestamps within parentheses. For example, $m_5(3)$ indicates a message sent from P2 with timestamp 3. Subscript value 5 indicates the identification number of this message. In this example, four successive periods are shown. The first one is a normal single cluster period. In this period, all six federates are in the same cluster. All messages are forwarded to the entire federate cluster and logical times of all the federates are synchronized. At real time t_1, two federate groups become distant and their logical time synchronization is broken at that point. The original cluster is divided into two new clusters. Federates in these different clusters advance their logical times independent of each other. During this period there is a single event in cluster 1 and there are 4 events in cluster 2. At the end of the local synchronization period, cluster 1 has logical time 3 and cluster 2 has logical time 6. At time t_2, the clusters start getting closer to each other and resynchronization period begins. In this period, the faster cluster, cluster 2, is blocked from advancing its logical time. During this blocked period, cluster 1 continues to advance its logical time. Resynchronization ends when logical times of both clusters are synchronized. At the end of the resynchronization period, the federate clusters join into a single cluster and all the federates continue their normal synchronized execution.

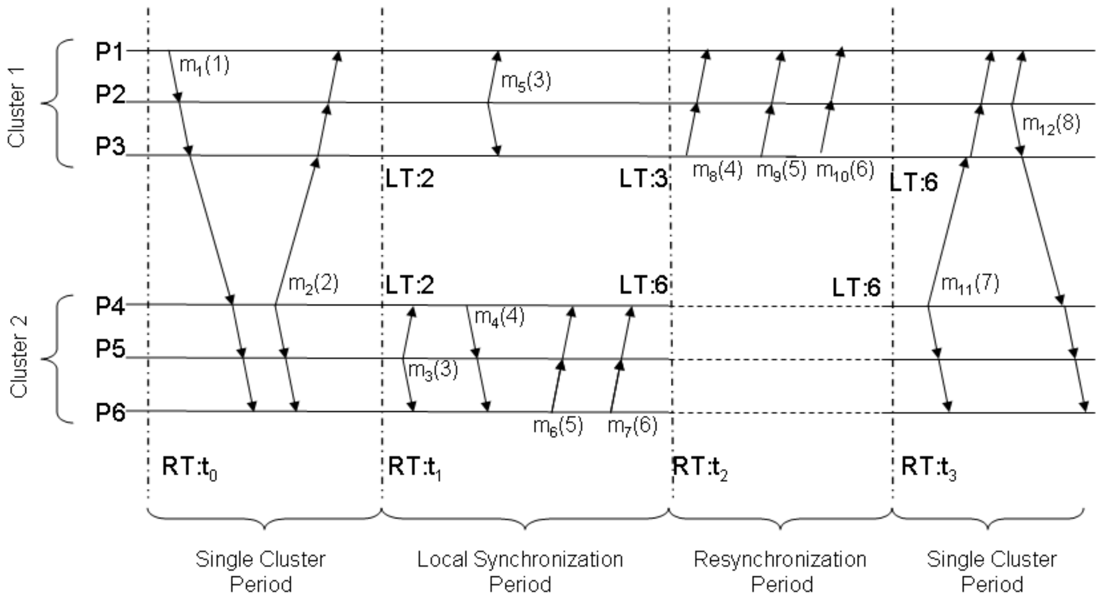


Figure 3.5: Resynchronization Period

The most critical point of resynchronization operation is to find the starting time of the resynchronization period. The idea is that the resynchronization period should be long enough to provide the opportunity to the slower federate cluster to catch up with the faster one. If there is not enough time for the slower cluster to catch up the faster one, they cannot become logical time synchronized when they become interested in the interactions of each other.

In order to detect the resynchronization start time, a relation between the distances of clusters and their logical times is constructed. In this relation, movements of the federate clusters and their logical time differences are checked in order to initiate the resynchronization period. The conservative resynchronization method emphasizes the required opportunity provided to the slower federate to catch up with the faster cluster by using the distance between them. The slower federate will take some distance during the resynchronization period. Conservative approach tries to find a minimum distance which is traveled by the slower federate between the resynchronization start time and the logical time synchronization point. This minimum distance is called as the critical distance. The resynchronization period initiates when the distance between two federate clusters becomes equal to the critical distance.

Main parameter to detect the minimum distance to initiate the resynchronization period is the speed of approaching federate clusters. Approaching Speed (AS) is the amount of distance change between these clusters in one unit time. In other words, approaching speed is the distance traveled by two federate clusters in one unit time.

For example, in a distributed war simulation, there could be two federates simulating tanks that are getting closer to each other. In Figure 3.6, there are two snapshots of this simulation example. In part (a) logical time of the first tank F1 is 10 and the logical time of the second tank F2 is 9. The distance between them is 60 at this point in the simulation. After a while, the simulation progresses to another state, shown in part (b). Tanks move toward each other and the distance between them is down to 50. Their logical times are also advanced to 13 and 11, for F1 and F2 respectively. These two federates get closer to each other by 10 units of distance after advancing their logical times 5 units in time. The approaching speed of these two federates is 2 during this period.

During the simulation runtime, the approaching speeds of cluster pairs are computed in regular intervals for all routing spaces. The AS between two federate clusters (FC_a and FC_b) for a given routing space RS_i is formulated as

$$AS^{RS_i}\left(FC_a, FC_b\right) = -\frac{\nabla d^{RS_i}\left(FC_a, FC_b\right)}{\nabla LT_{FC_a} + \nabla LT_{FC_b}} \tag{3.4}$$

Where ∇LT_{FC_a} is the logical time difference of FC_a, ∇LT_{FC_b} is the logical time difference of FC_b and $\nabla d^{RS_i}\left(FC_a, FC_b\right)$ is the change in distance between these clusters. Dividing this distance change by the total logical time change value, the approaching speed value is calculated, which means how much these federate clusters are getting closer per unit logical time. The division result is

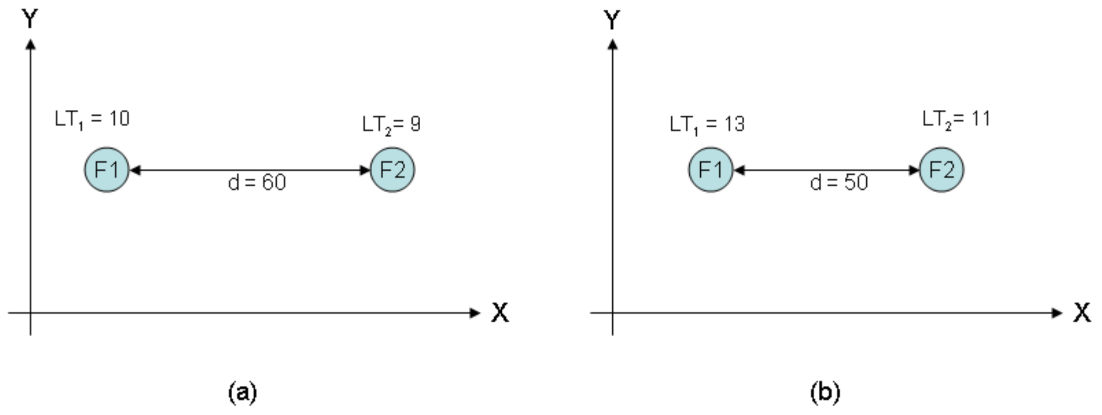


Figure 3.6: Approaching speed example

negated so that the approaching speed is positive as clusters approach each other and is negative otherwise.

When the distance between clusters becomes smaller or equal to the critical distance, it means that the minimum distance required for the slower cluster to catch up with the faster one is being reached, and the algorithm initiates the resynchronization period.

Critical distance of two federate clusters, $CD^{RS_i}(FC_a, FC_b)$, is calculated as:

$$CD^{RS_i}(FC_a, FC_b) = AS^{RS_i}(FC_a, FC_b) \,.\, (Abs\,(LT_{FC_a} - LT_{FC_b})) + \partial .\, (Abs\,(LT_{FC_a} - LT_{FC_b})) \tag{3.5}$$

The first part of this formula is the core approaching distance of the two federate clusters, and it represents the distance that could be taken by the slower non-blocked cluster. This is a function representing the current logical time difference. This component guarantees that when these two federate clusters approach each other with the current Approaching Speed, they could diminish and end the logical time difference between them. However, this part does not guarantee the correct joining operation if the clusters dramatically change their approaching speeds during the approaching phase.

The second part of this formula is the slack distance amount which is added to critical distance value in order to consider a possible speed change of the clusters. ∂ is a simulation domain dependent value determined for each simulation. It is the maximum speed for all simulation objects in the simulation and it should be externally supplied to the algorithm. For example, in a traffic simulation, the simulation designer could easily say that none of the simulation objects, i.e. vehicles in the traffic environment, will exceed the speed of 400 km/h. This value should be given to the PATiM mechanism by considering the logical time step value. For example, one unit of logical time represents one second of simulation time and one unit of routing space value represents one meter; so, the ∂ value should be 1440 by changing km value to meter and hour value to second.

In calculation of critical distance, ∂ value is multiplied by the logical time difference of the clusters in order to find the required distance to diminish the current time difference, even if the clusters move with maximum speed. Adding this value to the distance based on the current approaching speed gives the critical distance value, which takes into account any speed increase or any new simulation object with a higher speed during the resynchronization phase.

Handling simulation maximum speed (∂) PATiM algorithm guarantees that if two federate clusters enter the local synchronization period, they will be safely resynchronized when they get close to each other.

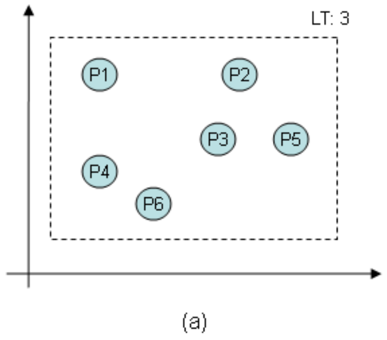
When the distance for one of the routing spaces is smaller or equal to the critical distance of that routing space, the resynchronization period will be initiated. This condition is depicted as:

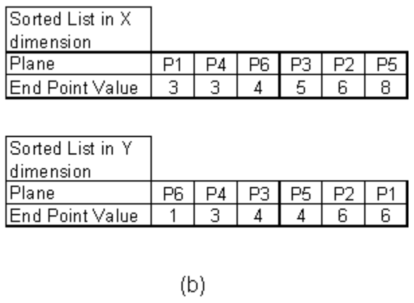
$$d^{RS_i}(FC_a, FC_b) \leq CD^{RS_i}(FC_a, FC_b) \qquad \exists RS_i \tag{3.6}$$

The resynchronization period is completed when the slower federate cluster catches up with the faster one. After the resynchronization period, the clusters enter their normal period.

3.4.3 An example case for time management

In order to understand time management mechanisms better, an example case is analyzed in this section. In a distributed war simulation example, there is a federation containing six federates which are simulating different planes in a battle environment. The federation defines a two-dimensional routing space for this terrain. The virtual space of this simulation is composed of a large two dimensional terrain. In part (a) of Figure 3.7, initial placement of the six planes on the virtual space is shown. At this state of the simulation, there is only one federate cluster and its logical time is 3. One unit of logical time represents one minute in the simulation. The federates are also shown at their locations in this two dimensional routing space. In part (b) of this figure, dimensional sorted lists are shown for this initial state. End point values in these tables are in kilometers. As it can be seen, there are not any gaps in these sorted lists.





Sorted List in X dimension						
Plane	P1	P4	P6	P3	P2	P5
End Point Value	3	3	4	5	6	8

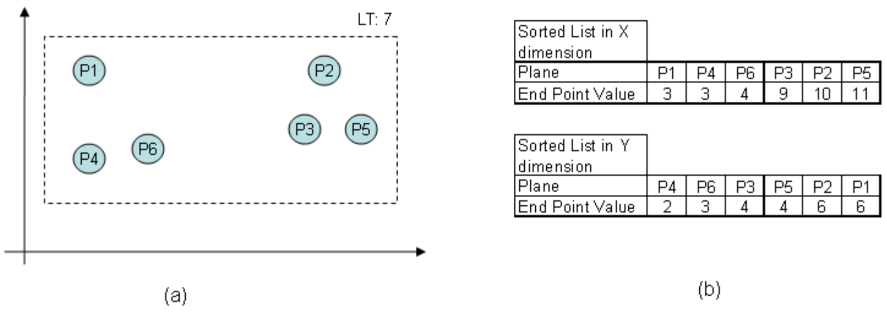
Sorted List in Y dimension						
Plane	P6	P4	P3	P5	P2	P1
End Point Value	1	3	4	4	6	6

(b)

Figure 3.7: Initial normal period

During the simulation execution, three of the federates, P2, P3 and P5, are getting away from the other federates. In the situation shown in Figure 3.8, , these three federates are relatively more distant to the other federates. However, currently there is not any gap that exceeds the threshold value of 8 kilometer as it can be seen from part (b) of the figure.

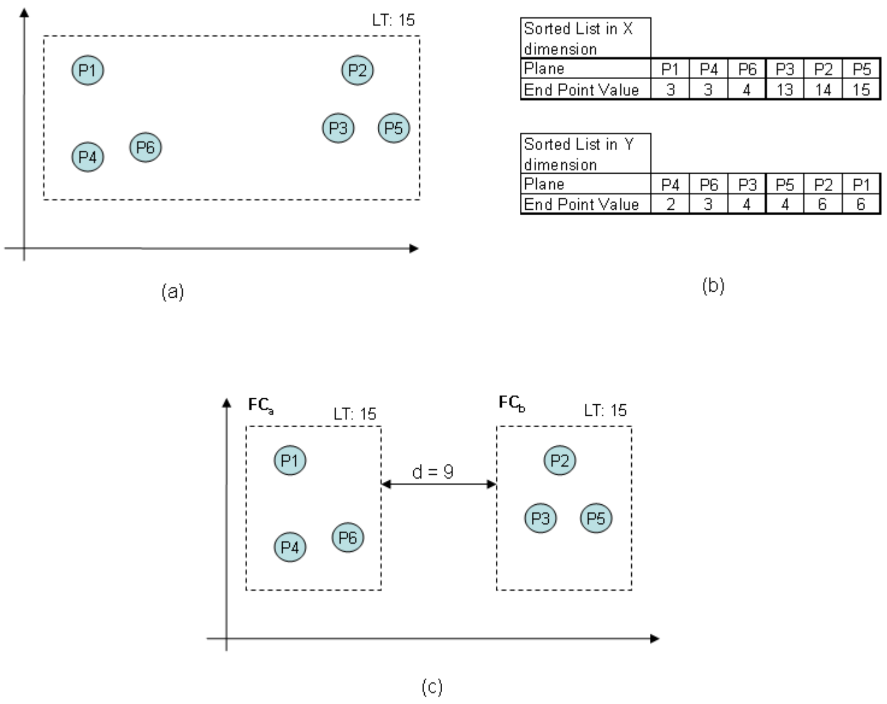
After a while, the gap value in sorted list of the X dimension becomes 9, which is greater than the threshold value. In parts (a) and (b) of Figure 3.9, the positions of the federates and dimensional sorted lists are shown respectively, which depict the case before the division operation. In part (c) of this figure, the state after the division operation is shown. In this case, the original cluster is divided into two clusters from the gap point of the X dimensional sorted list. At that point, logical time dependency between these federate groups is broken. The current distance between these two clusters is 9 km.



Sorted List in X dimension						
Plane	P1	P4	P6	P3	P2	P5
End Point Value	3	3	4	9	10	11

Sorted List in Y dimension						
Plane	P4	P6	P3	P5	P2	P1
End Point Value	2	3	4	4	6	6

Figure 3.8: Federate groups getting distant



Sorted List in X dimension						
Plane	P1	P4	P6	P3	P2	P5
End Point Value	3	3	4	13	14	15

Sorted List in Y dimension						
Plane	P4	P6	P3	P5	P2	P1
End Point Value	2	3	4	4	6	6

Figure 3.9: Division case

After a while, the distance between these two clusters increases even more and their logical times become inconsistent as it can be seen from Figure 3.10. In this case, the approaching speed (AS) of these two federates is calculated as 2, which means that the distance between these two clusters is changed by 2 km in one minute. In critical distance (CD) calculation for this simulation, maximum speed value of the simulation is defined as 3. This means that none of the planes could move faster than 3 kilometers in a minute. The calculated critical distance is 15 at this state. As the distance between these federates is smaller than the critical distance, they are now in the local synchronization period.

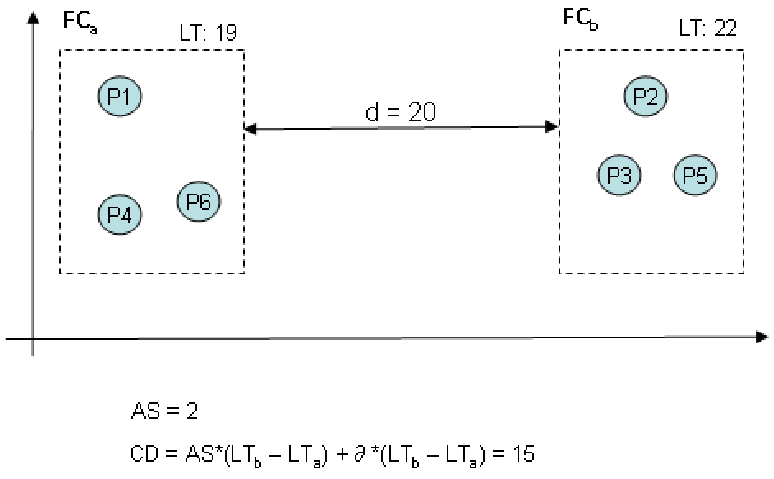


Figure 3.10: Distant case

After some time passes, the clusters get closer to each other and the distance between them decreases and becomes smaller than the calculated critical distance, as shown in Figure 3.11. In that case, they enter the resynchronization period. After the resynchronization period, their logical times become consistent, and they enter their normal synchronization period.

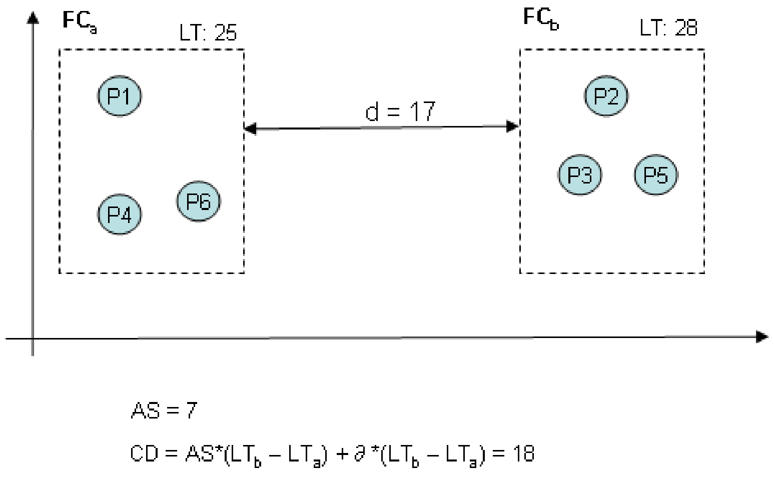


Figure 3.11: Resynchronization case

In order to show the relationship between distance, approaching speed and critical distance graphically, a run scenario will be analyzed, in which there are two federate clusters, FC_a and FC_b. Graphical representations of the example values are shown in Figure 3.12. In the scenario, the two clusters mentioned are first in a normal period. After that, they begin to get away from each other and then they get closer again. In the upper part of the figure, the relation between the logical times of the two clusters is shown. In the lower part, calculated distances, critical distances and approaching speeds are shown. The threshold value (τ) is assumed to be 20 in these calculations.

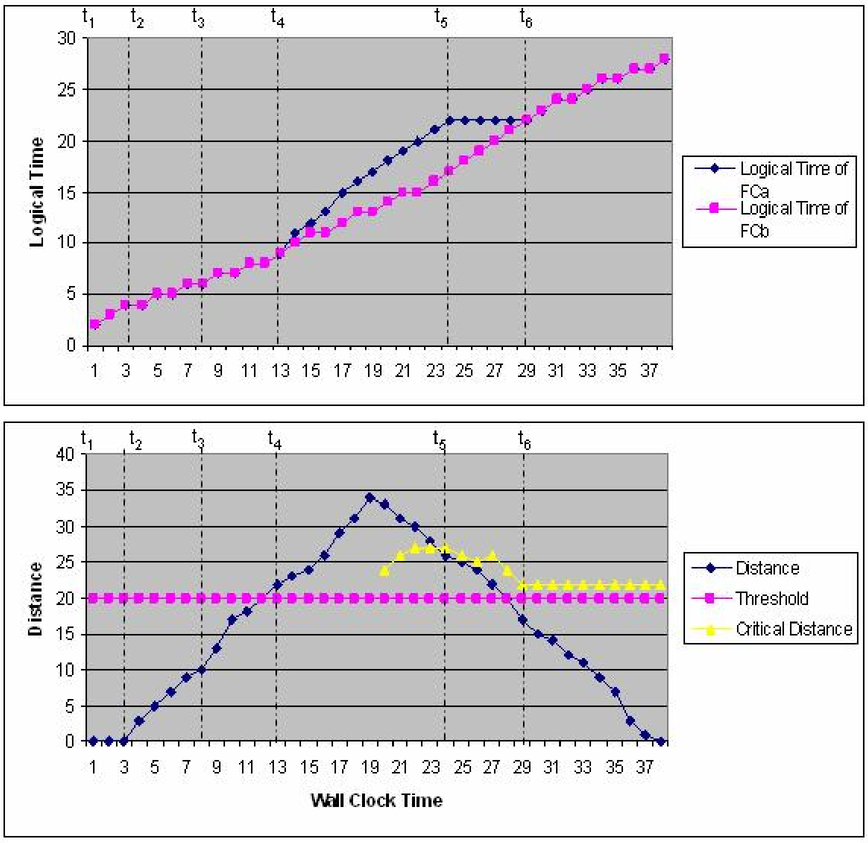


Figure 3.12: Graphs of example

Between wall-clock times t_2 and t_4, the distance between the two clusters increases continuously. At time t_4 the distance between them exceeds the threshold value and they enter the local synchronization period. This period continues until t_5. At time t_5, the distance between them becomes smaller than the calculated critical distance and the resynchronization period begins. As the faster cluster is FC_a, the algorithm blocks its logical time but allows FC_b continue its execution. After their logical times become synchronized at time t_6, both continue their executions in a normal period.

3.5 Multiple Cluster Handling

In this section, the proposed mechanism will be analyzed for cases containing more than three federate clusters. In that case, all clusters could independently become distant or get closer to each other. The proposed algorithm will never lead to a deadlock, in which every federate cluster is blocked because of another cluster. The reason for this is that, in the proposed algorithm, a cluster is only blocked because of another cluster when its logical time is greater than the other. On the other hand, there exists at least one federate cluster in the federation which has the minimum logical time. This leads to the assumption that there should be always at least one federate cluster that is not blocked. This non-blocked cluster will unblock the cluster that waits for it after the resynchronization period, and finally, all clusters in the blocked clusters chain will eventually become unblocked. In the following figures an example for the multiple federate cluster case is analyzed.

In Figure 3.13 there are 3 federate clusters. For simplicity, it is assumed that there is only one routing space which contains two continuous dimensions. Clusters are continuously getting closer to each other. Figure shows the real time t1, in which all federate clusters are distant to each other. The left side of the figure shows the timelines of the federation and the right side depicts the combined regions of clusters on the routing space of the federation. In this case, logical times of all federate clusters are independent.

In Figure 3.14, a later point (real time t_2) is shown. Between t_1 and t_2, FC2 gets closer to the FC1. Change of the logical times of the federate clusters is shown in part (a) of this figure. The distance between them becomes smaller or equal to their critical distance at time t_2. At that time, FC1 is blocked and FC1 and FC2 enter a resynchronization period. Blocked federate cluster FC1 is shown with dashed red lines in part (b) of the figure. During this period, FC3 is getting closer to FC2 and FC2 is getting closer to FC1. However, the distance between them is still greater than the critical distance.

In Figure 3.15, real time t_3 is shown. Between t_2 and t_3, FC3 gets closer to FC2. At time t_3, FC2 is blocked for FC3. At this time, FC1 is waiting for FC2

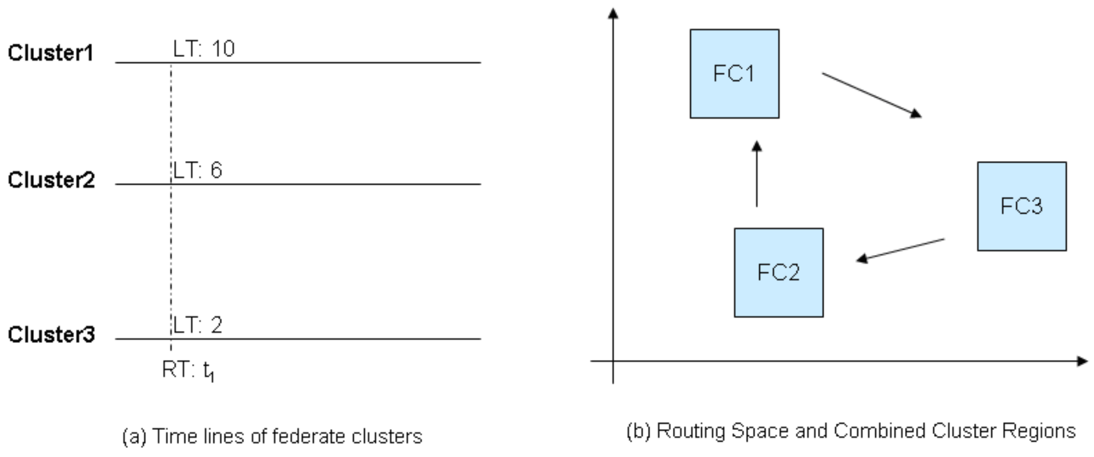

Cluster1
LT: 10
Cluster2
LT: 6
Cluster3
LT: 2
RT: t_1
(a) Time lines of federate clusters
FC1
FC3
FC2
(b) Routing Space and Combined Cluster Regions

Figure 3.13: Example case, separate clusters

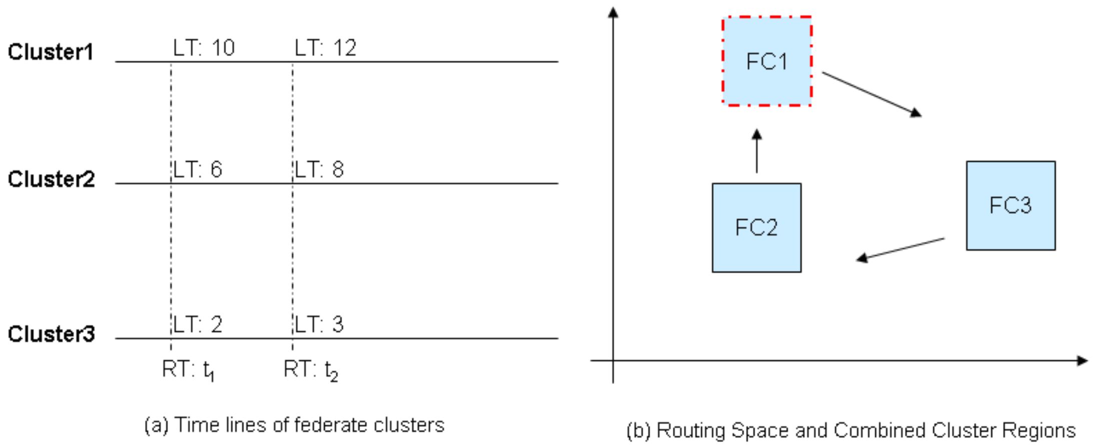


Figure 3.14: Real time t_2

while FC2 is waiting for FC3; and only FC3 is running. The dashed line for FC1 between t_2 and t_3 in the timelines of clusters shows that this cluster is blocked during this period.

Between t_3 and t_4, shown in Figure 3.16, FC3 gets closer to FC2 and at time t_4, their logical time difference between them becomes zero. In that case, FC2 and FC3 are combined to form FC4. In the timeline part, it is shown that timelines of FC2 and FC3 have ended. Before this merge operation, FC1 was blocked for FC2, however, when FC2 was removed, FC1 becomes blocked for the newly created federate cluster FC4, which has replaced FC2 and FC3.

In general, when a federate cluster is removed because of a joining operation, the other clusters that were waiting this cluster, begin to wait the newly created

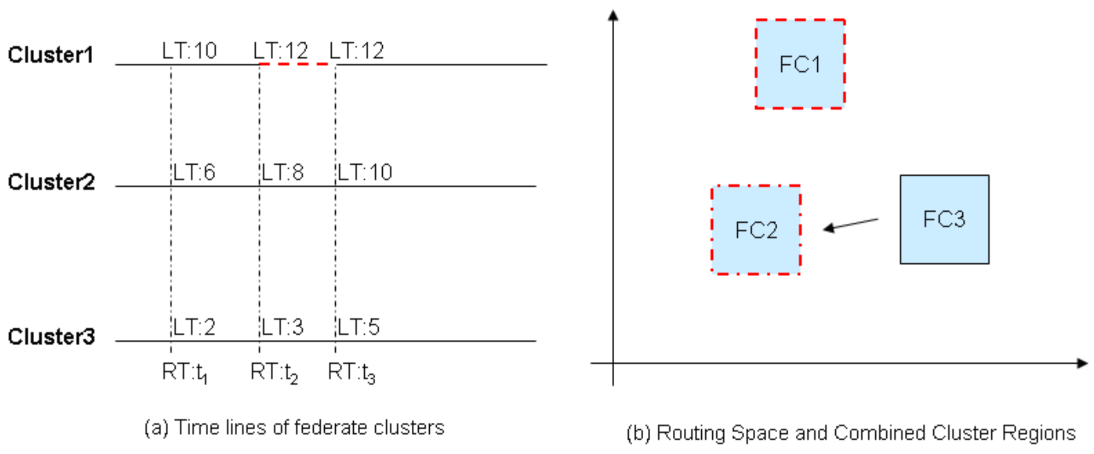


Figure 3.15: Real time t_3

cluster. After some time, the logical time of the new cluster becomes equal to the blocked clusters and the resynchronization period will end.

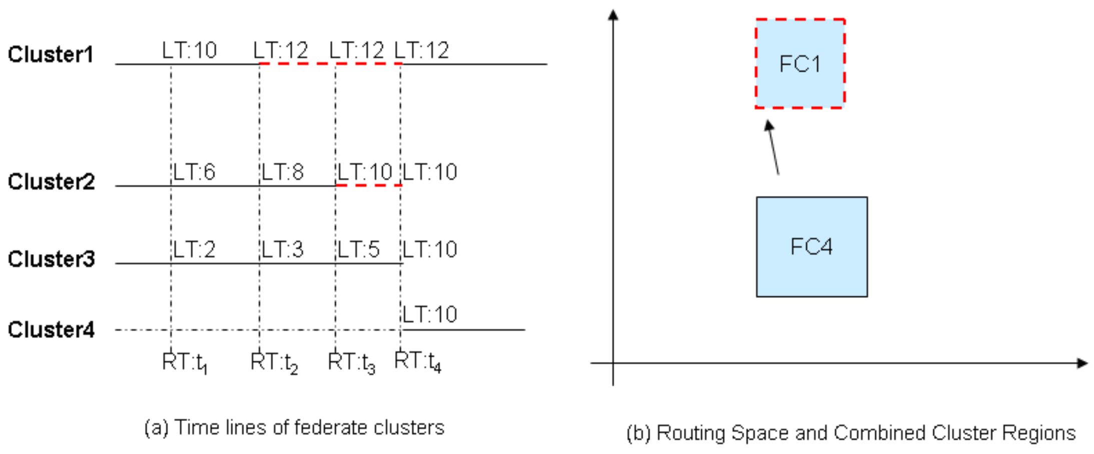


Figure 3.16: Real time t_4

Between t_4 and t_5, shown in Figure 3.17, FC4 gets closer to FC1 and their logical times become equal at time t5. At that point, the algorithm merges these two clusters to make FC5. At the end of this period, there is only one big federate cluster in the federation.

The important point in the multiple federate cluster case of PATiM is that, the "waits for graph" of blocking federate clusters has never a cycle in it. This originates to the fact that an edge of this graph is created only when the logical time of a federate cluster is greater than the logical time of other clusters, and this type of graph could not have a cycle at all.

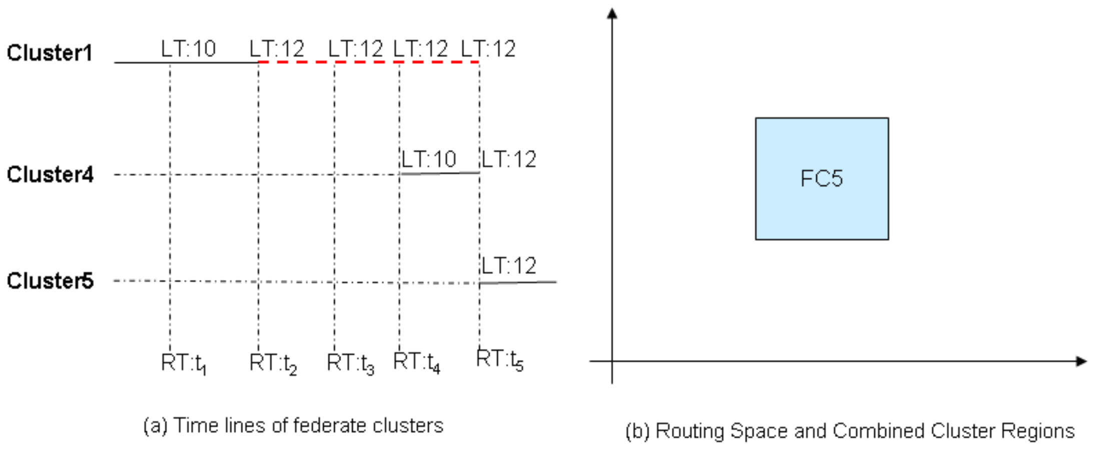


Figure 3.17: Real time t_5

CHAPTER 4

DESIGN AND IMPLEMENTATION

4.1 Design

The proposed algorithm could be implemented in both centralized and distributed manners. In a distributed implementation case, there is not any central RTI component in order to implement time management or dynamic clustering related operations. Customized time management operations are done in each LRC (Local Runtime Component) of the federates. Necessary time queues and LBTS calculations are the most critical ones of these operations. Additionally, in a distributed time management algorithm, synchronization related messages (null-messages) are necessary in order to calculate LBTS in each LRC. The distributed mechanism has disadvantages such as increased number of synchronization messages, replicated data structures like time queues and replicated LBTS calculation operations. However, distributing the time management algorithm removes the single point of failure and shares the load of calculating LBTS for the entire federation.

Dynamic cluster management in distributed PATiM implementation is replicated on the federate LRCs. In each federate cluster, a coordinator LRC is selected to execute the cluster management algorithm. The LRC that owns the minimum identification number is selected as the coordinator for each cluster. This selection algorithm quickly finds a coordinator.

These coordinators continuously check for possible divisions of current clusters while checking the critical distances to other clusters to use in resynchronization and joining operations as well. These operations first need the information about subscribe and update regions of all the federates in the federations. This information comes from the DDM implementation of RTIs. More detailed information about how RTIs transfer this region information to all of the federates is given in section 4.5. Other information required for the critical distance checks is the LBTS values for all other clusters. Each coordinator transfers the LBTS information of the cluster to all other coordinators. The received LBTS values are used to calculate critical distances.

Another possible implementation is centralized time management. In this implementation, there is a central RTI component and all federates send their time related requests, like time advance requests, to this central component. This component calculates corresponding LBTS for each of the federates and sends back the grant messages declaring that it is appropriate. In central implementation, dynamic cluster management is done by a central component where it calculates the federate clusters and advances the logical times of these federates.

4.2 Middleware Approach

There are two main methods to implement the proposed mechanism. The first one is to modify an existing HLA implementation in order to extend its time management services for the PATiM mechanism. This approach is not feasible, because currently there are not any fully implemented and open source RTI implementations. Further, understanding and modifying an existing infrastructure sometimes becomes much more complex and time consuming than initiating a completely new implementation.

Another approach to implement the proposed mechanism is to use one of the existing RTIs and extend it using a middleware approach. In this approach, it is required to capture some HLA requests and responses in order to implement the proposed mechanism.

PATiM proposes a time management mechanism in which there are a number of federate groups that have time synchronization within the group but there are no time synchronizations across separate federate groups. The existing HLA standard does not allow this kind of grouping; therefore, the time management mechanism should be re-implemented. The middleware approach used in this study re-implements the time management mechanism.

In a HLA based federation, every federate interacts with the RTI through the interface composed of the *RTIambassador* and the *FederateAmbassador*. Before the federate can invoke any RTI services, it must first create an instance of the *RTIambassador*. It is natural to put the middleware between the federate and the *RTIambassador* so that messages from the federate can be intercepted and analyzed.

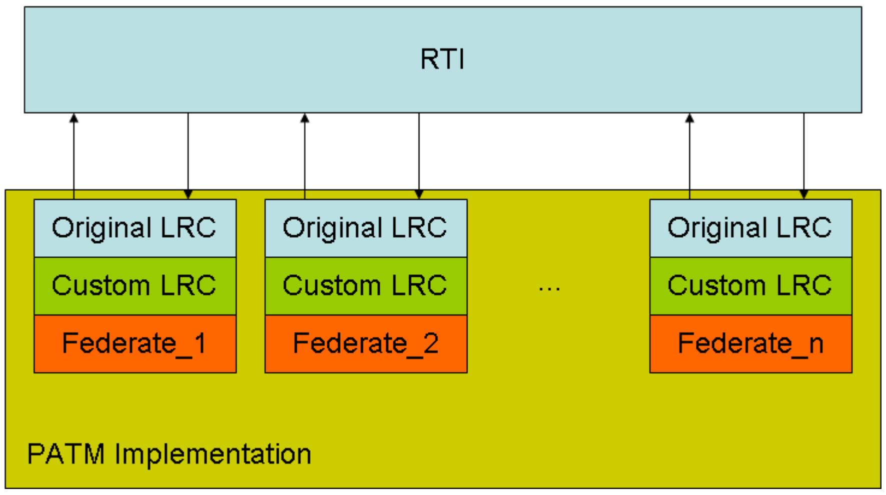


Figure 4.1: General PATiM design

In this PATiM middleware design, a number of federates are implemented, interacting with each other via the RTI services. Each federate component contains an original LRC (Local Runtime Component) which comes with the RTI implementation and a custom LRC which implements the PATiM time management with the federate code itself, as it can be seen in Figure 4.1.

In Figure 4.2 there is a detailed diagram of a single federate component. In order to implement the custom time management mechanism, the original RTI ambassador and the federate ambassador are enveloped with the custom

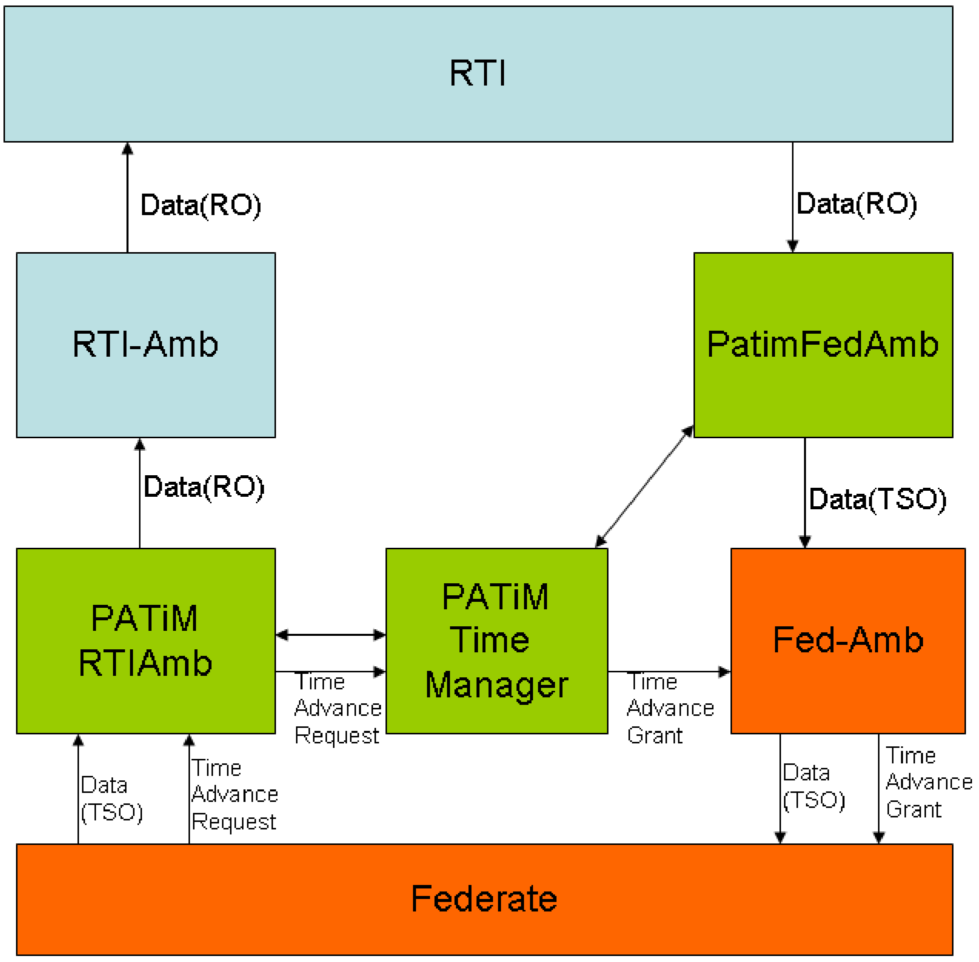


Figure 4.2: Middleware approach

ambassadors. In this figure, blue components are original RTI implementations. Greens ones are customized RTI components. Red components are implemented by the federate designer to use the RTI services.

The user federate at the bottom of the figure will use these custom ambassadors in order to utilize HLA services. The proposed middleware layer forwards all requests for HLA services to the original ambassador, except the ones related to time management. Time management functionalities are to be performed completely by the PATiM layer.

As a result of the insertion of these middleware classes, the only change to the federate code is that the instance declaration of the *RTIambassador* is replaced by an instance declaration of *PatimRTIambassador*. The method calls made by the federate remain exactly the same. When the federate joins the federation, the

PATiMRTIambassador initializes the instance of the *PatimFederateAmbassador* by providing a reference as a parameter to the *FederateAmbassador* of the user federate. The *MiddleFederateAmbassador* is then passed to the RTI, instead of the *FederateAmbassador* of the user federate, as the callback reference to the federation.

4.3 Time Management

In PATiM layer, time information that comes from the federates is encoded into tag fields of HLA services and on the receiving side, these tag fields are decoded so that receive-order messages are converted into timestamp ordered messages. In PATiM, following time management services are implemented:

- Enabling/disabling time regulation property
- Enabling/disabling time constrained property
- Time advance request
- *updateAttributeValues* and *reflectAttributedValues* with TSO messages
- *sendInteraction* and *receiveInteraction* with TSO messages

In Figure 4.3, a summarized initialization process of a federate is shown. The first action of a federate in the initialization process is to join the federation execution. This request is directly forwarded to the original RTI component. When a user federate enables time management by sending a request, *PatimRtiAmb* will utilize this information within itself and will not forward this message to original *RTIAmb*. Apparently, the RTI will recognize the federate as not time managed at first. Then, the *PatimRtiAmb* informs the PatimTimeManager that the federate tries to enable time regulation and time constraint properties. This information, in turn, enables customized time management for that federate. Later, the federate completes the desired publish and subscribe operations in the initialization phase. These publish and subscribe operations are not related with the customized time management and they are directly forwarded to the original RTIAmb and then to the RTI.

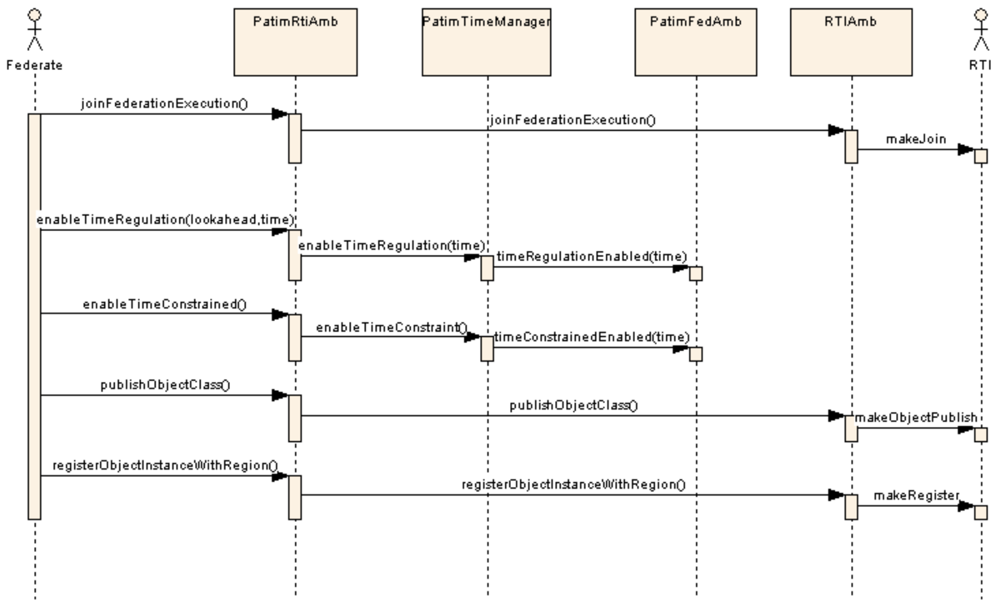

Federate
PatimRtiAmb
PatimTimeManager
PatimFedAmb
RTIAmb
RTI
joinFederationExecution()
joinFederationExecution()
makeJoin
enableTimeRegulation(lookahead,time)
enableTimeRegulation(time)
timeRegulationEnabled(time)
enableTimeConstrained()
enableTimeConstraint()
timeConstrainedEnabled(time)
publishObjectClass()
publishObjectClass()
makeObjectPublish
registerObjectInstanceWithRegion()
registerObjectInstanceWithRegion()
makeRegister

Figure 4.3: Federate initialization

As shown in Figure 4.4, when a federate tries to advance its logical time, it makes a call to the *PatimRtiAmb* as a *timeAdvanceRequest* sending the desired time as an argument as well. As the *PatimTimeManager* gets the request, it first evaluates the LBTS (Lower Bound on Time Stamp) for this federate and then decides whether it should allow this time advance. *PatimTimeManager* calculates the LBTS value by considering logical time plus lookahead values of the federates which are in the same cluster. To do this, it calls the *calculateClusterLBTS* method of the *PatimFederateCluster* object. This method finds the minimum of logical time plus the lookahead values of all the federates, except the calling federate.

If a time advance operation is appropriate according to the calculated LBTS value, the time manager of the federate calls back the *timeAdvanceGrant* callback of the federate ambassador. Otherwise, owning federate enters into the *TimePending* phase in which it wait for the LBTS value to become appropriate to advance to the requested logical time.

As mentioned before, distributed time management algorithms require synchronization related messages in order to distribute the logical time state of individual federates. *PatimTimeManager* uses *broadcastNullMessage* method for

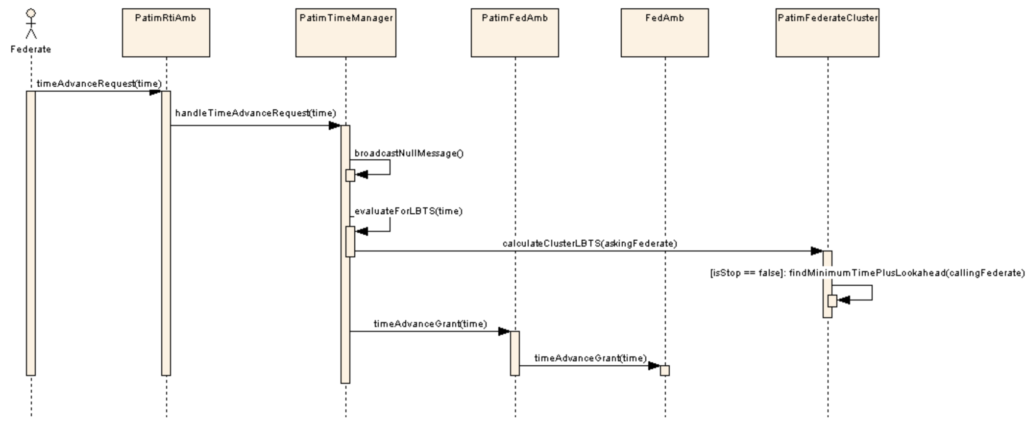


Figure 4.4: Advance time

this purpose. In this method, a null-message containing time-advance request information is broadcasted to all federates that are in the same federate cluster with the federate that made the request.

The broadcasted null-messages are used to evaluate LBTS values inside other federates. In Figure 4.5, the operations that are completed when a null-message is received are shown. Null messages are transported as an interaction within the existing RTI infrastructure. In fact, these messages should be transported as internal messages but as an existing RTI implementation is used, the internal structure of it cannot be changed. When this special null-message interaction is received by the *PatimFedAmb*, it is sent to the *PatimTimeManager*. Null messages contain information of the logical time plus the lookahead values of the sending federates. In each *PatimTimeManager*, there is a list of logical time plus lookahead values for all other federates. This list is used to evaluate the LBTS value by finding the minimum among the values on the list.

When the *PatimTimeManager* receives a null message, it updates the corresponding the logical time plus the lookahead value of the sending federate on its list. After that update, *PatimTimeManager* finds the minimum value of this list, which will be the new LBTS value for the owning federate. At that point, the time manager looks whether the owning federate is in a time pending phase. If so, *PatimTimeManager* grants time advance request of the federate if the requested time is appropriate compared to the new LBTS value.

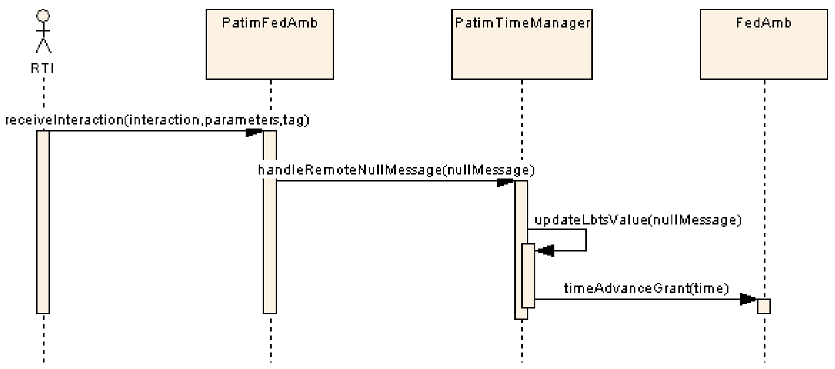

RTI
PatimFedAmb
PatimTimeManager
FedAmb
receiveInteraction(interaction,parameters,tag)
handleRemoteNullMessage(nullMessage)
updateLbtsValue(nullMessage)
timeAdvanceGrant(time)

Figure 4.5: Null message receive

When a federate tries to send object updates, it calls the updateAttributeValues function of the PatimRtiAmb, as shown in Figure 4.6. As the federate uses the time management services provided, it supplies also the time information of update with this function call. *PatimRtiAmb* sends this logical time of update to the time manager so that the manager checks the validity of the sent time with respect to the current logical time and the lookahead value of the federate. After this validity check, *PatimRtiAmb* encodes the logical time information to a string format as a tag value and calls the *updateAttributeValues* of original RTIAmb, which is not time managed. This is done because PATiM uses custom time management services and the RTI sees this federate as non time managed; thus the *updateAttributeValues* of original RTIAmb is called without any time information. The RTI, then, delivers this update information with the receive order constraint to the destination.

In the simulation iteration, the federate needs to call the tick method of the RTI ambassador after updating object attributes and sending interactions in a single threaded process model. The working mechanism of the customized tick method is shown in Figure 4.7.

When the federate calls the tick method of the *PatimRtiAmb*, the customized ambassador calls the tick method of the original RTI ambassador. After this call, if there are some events (updates or interactions) waiting in the original RTI, *reflectAttributeValues* or *receiveInteraction* methods of the *PatimFedAmb* are called by the RTI. As a custom time management is used in this mechanism,

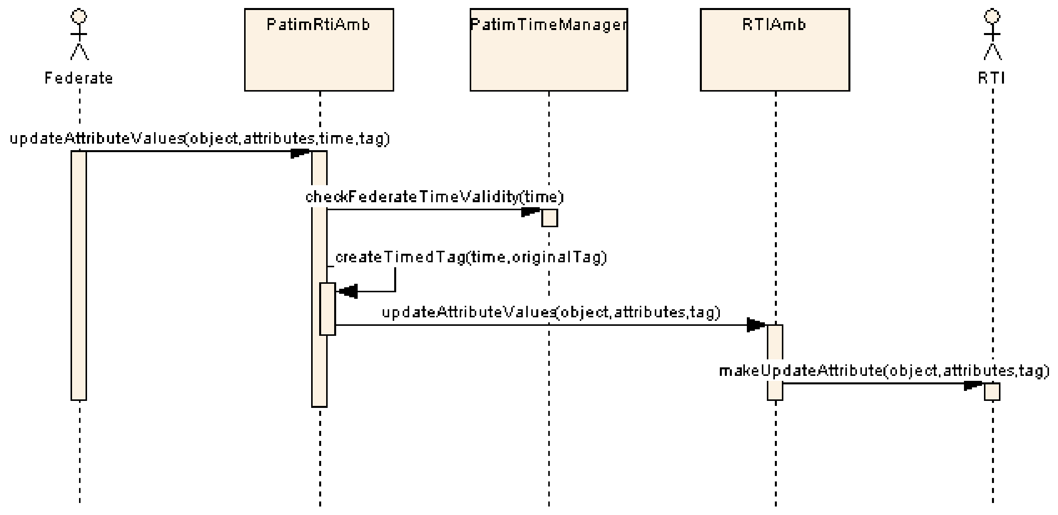

Federate
PatimRtiAmb
PatimTimeManager
RTIAmb
RTI
updateAttributeValues(object,attributes,time,tag)
checkFederateTimeValidity(time)
createTimedTag(time,originalTag)
updateAttributeValues(object,attributes,tag)
makeUpdateAttribute(object,attributes,tag)

Figure 4.6: Send data

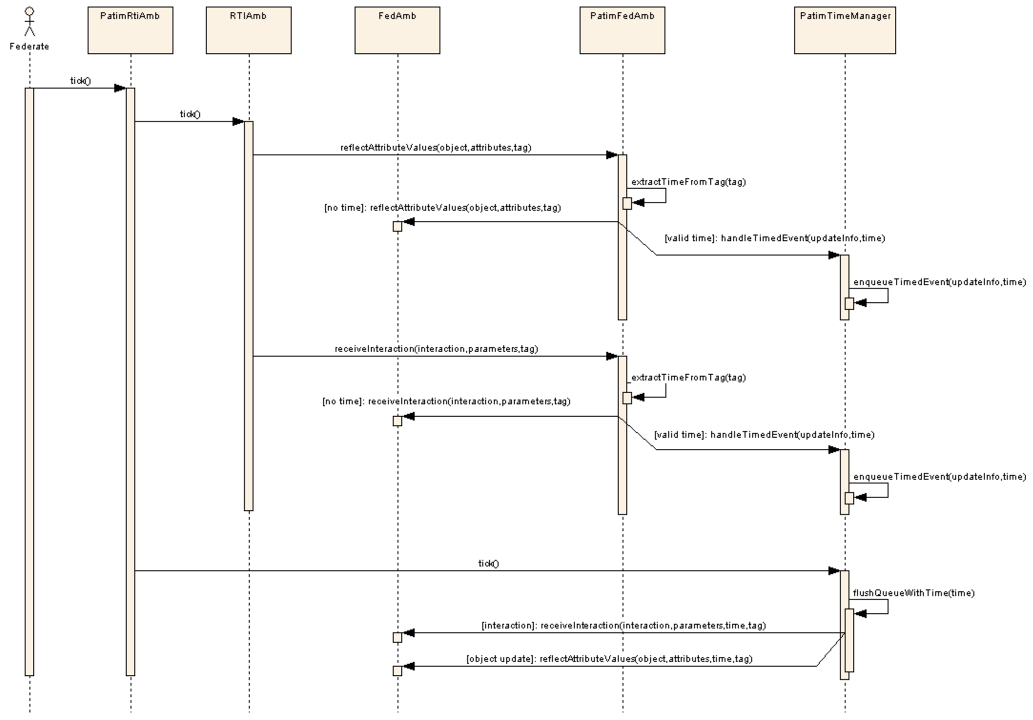

Federate
PatimRtiAmb
RTIAmb
FedAmb
PatimFedAmb
PatimTimeManager
tick()
tick()
reflectAttributeValues(object,attributes,tag)
extractTimeFromTag(tag)
[no time]: reflectAttributeValues(object,attributes,tag)
[valid time]: handleTimedEvent(updateInfo,time)
enqueueTimedEvent(updateInfo,time)
receiveInteraction(interaction,parameters,tag)
extractTimeFromTag(tag)
[no time]: receiveInteraction(interaction,parameters,tag)
[valid time]: handleTimedEvent(updateInfo,time)
enqueueTimedEvent(updateInfo,time)
tick()
flushQueueWithTime(time)
[interaction]: receiveInteraction(interaction,parameters,time,tag)
[object update]: reflectAttributeValues(object,attributes,time,tag)

Figure 4.7: Federate update (tick)

RTI distributes events in receive order. If the federate sends events as timestamp order, encoded time information is added to the tag field of the event callback. Custom time management mechanism decodes this logical time information from the tag. These receive order events are converted to timestamp order events and these events are enqueued in the *PatimTimeManager*. There will not be any logical time information in the tag field because events are actually in receive order. In this case these receive order events are distributed directly to the original federate ambassador.

After the original RTI ambassador finishes its job in the current tick call, *PatimRtiAmb* calls the tick method of the *PatimTimeManager* in order to deliver available timestamp ordered events to the federate ambassador. The time manager holds a queue of timed events, which are sorted according to their logical times. The customized time manager delivers events the logical time of which are smaller than the LBTS value of the owning federate.

4.4 Cluster Management

Dynamic federate clustering algorithm is executed in all iterations during the simulation execution. Cluster management is implemented by continuously checking the clusters for possible join and divide operations, while controlling distances and approaching speeds. The joining operation of federate clusters is used to join clusters at the end of resynchronization period.

In distributed implementation of the PATiM mechanism, in each cluster, a federate is selected as a coordinator. This coordinator federate has to manage the federate clusters using the *PatimManager* class. Necessary information required for the management of clusters includes regions data of all the federates in the federation and logical time values of all the clusters. Region information is already transferred to all LRC components for the data distribution services. Logical time values are also interchanged between coordinator federates. In the central implementation, the same *PatimManager* class is used by the central RTI component; thus all necessary information is already available.

Main method in the *PatimManager* class is the *update()* method, see Figure 4.8. First action of this method is to find the pair wise distances, approaching speeds and critical distances of all federate clusters. In order to calculate the distances between clusters, the invoked method needs to find the location of each cluster in the virtual space. For this purpose, all update and subscribe regions of all the federates that are joined are calculated. These joined regions represent the clusters and the distance between them is calculated. After calculating the distances, they are compared with the critical distance values. If the calculated distance is smaller than the critical distance value, then the faster cluster will be stopped. This will block all the federates inside that cluster and their time advance requests will not be granted.

In order to find the clusters that should be joined, logical times of cluster pairs, which previously entered a resynchronization period, are compared. If their logical times are synchronized, a new cluster will be created that combines both clusters.

The last operation is to find the divided clusters. This operation is done based on the dimensional sorted lists that are maintained in each cluster.

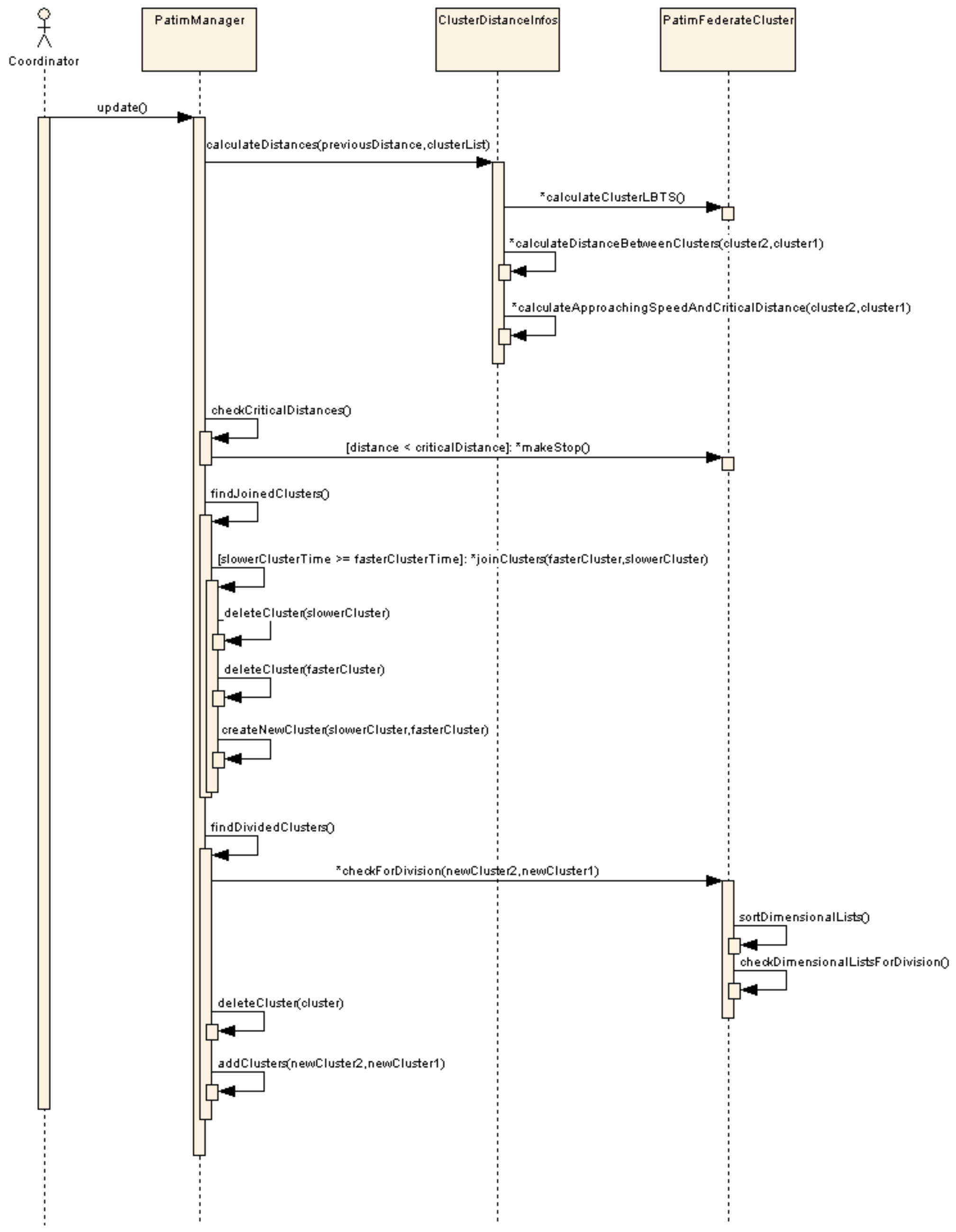

Coordinator
PatimManager
ClusterDistanceInfos
PatimFederateCluster
update()
calculateDistances(previousDistance,clusterList)
*calculateClusterLBTS()
*calculateDistanceBetweenClusters(cluster2,cluster1)
*calculateApproachingSpeedAndCriticalDistance(cluster2,cluster1)
checkCriticalDistances()
[distance < criticalDistance]: *makeStop()
findJoinedClusters()
[slowerClusterTime >= fasterClusterTime]: *joinClusters(fasterCluster,slowerCluster)
deleteCluster(slowerCluster)
deleteCluster(fasterCluster)
createNewCluster(slowerCluster,fasterCluster)
findDividedClusters()
*checkForDivision(newCluster2,newCluster1)
sortDimensionalLists()
checkDimensionalListsForDivision()
deleteCluster(cluster)
addClusters(newCluster2,newCluster1)

Figure 4.8: PATiM update

4.5 Integration with Existing RTIs

In order to implement the distributed PATiM algorithm using existing RTI and LRC implementations, some information is required for each customized LRC. In the proposed algorithm, each federate calculates the cluster formations and a federate is chosen as the master federate when the cluster is formed. This master federate calculates also the critical distances for all other clusters. This critical distance calculation requires the following information:

- Information on all publish and subscribe regions of local and remote federates: Each federate in the federation calculates the distances between federates and then calculates clustering. Every federate knows the federation wide cluster formation information.
- Information on the logical times of the federate itself as well as the entire federation: Each cluster is required to know logical times of its own and all other remote federate clusters. This is required to calculate critical distances, and periods of local synchronization and resynchronization.

4.5.1 Background on DDM implementations

In this section some background information on existing DDM implementations is given. This information is focused on how existing DDM implementations handle the regions of federates to filter data.

RTI NG

Conceptual DDM model in RTI NG [32] is given in Figure 4.9. This figure represents an abstraction of the purposes of the processing and the data flows needed to support the DDM.

In this model, one subscriber and one publisher is presented. Object attributes and interaction parameters are transferred through a communications infrastructure. DDM permits federations to abstractly specify and communicate their data requirements to the RTI. The RTI establishes connectivity so that the

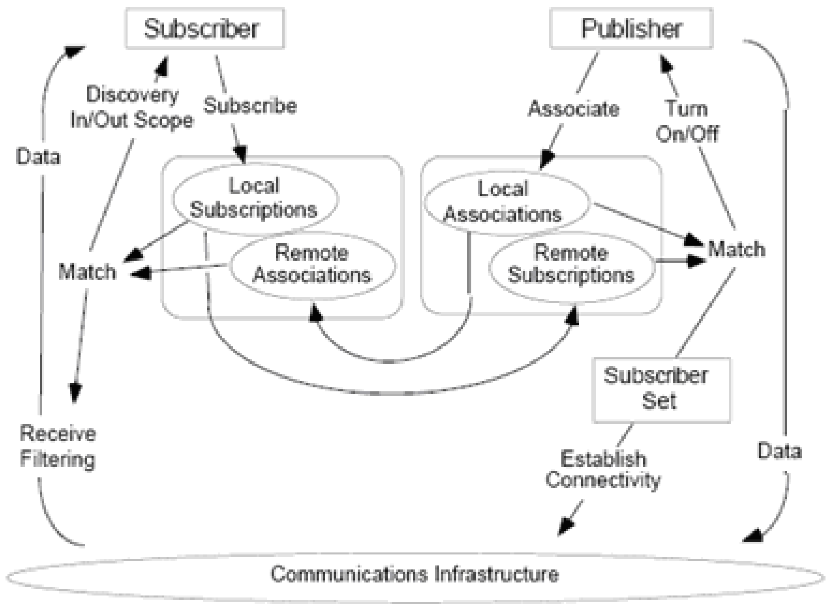


Figure 4.9: Conceptual Model of DDM in RTI 1.3

publisher's data is delivered to appropriate subscribers and the system resources are used conservatively to meet the performance goals of the federation.

In order to establish the essential connectivity, publishers and subscribers state their data requirements using associations and subscriptions. Associations are used by publishers. These associations represent either the relations between a region, an object, and the object attributes (depicted as <region, object, attributes>), or the relations between a region and an interaction class (depicted as <region, class>).

Subscriptions are used by subscribers and they are also represent relations, similar to associations. As depicted in Figure 4.9, databases of associations and subscriptions are maintained in the LRC. Local subscriptions and associations are maintained in a Local Associations database and a Local Subscriptions database. The contents of these local databases are reflected remotely. As a result of this reflection, the contents of the Local Associations database of the publisher are available to subscribers as a Remote Associations database. Similarly, local subscriptions are reflected remotely to publishers to construct Remote Subscriptions databases.

The matching operation consists of detecting intersections between regions and classes/attributes of associations and subscriptions. The result of matching is a subscriber set, which identifies the owner of each subscription that each association matches to. The subscriber set is used to determine destinations to send a given data. Also an empty subscriber set means that the given data is turned off and cannot be transmitted.

MAK RTI

MAK RTI implements the DDM using the distributed region approach [71]. In this approach, the LRC of each federate exchanges its region information with remote LRCs (or with other RTI components); so the region information is distributed among all LRCs. The individual LRCs perform matching between local and remote regions. The LRC uses these region matches to establish communication channels between publishers and subscribers.

The advantage of distributed region matching is that the LRC can establish communication channels that carry relevant data only to receiving federates. Other unsubscribed attributes may also be carried through the channel, but this multiple association problem exists practically in any approach, including the fixed grid approach. Also, the transmissions of attributes and the interactions are always sent to a single channel. The disadvantage is the considerable overhead of distributing the region information and performing matching between regions.

In the implementation of this algorithm, each LRC distributes its region set and region information to all other LRCs. When a federate changes region extends or changes the subscription information of the regions, the LRC immediately distributes this information to other LRCs. Each LRC, then, matches its local region sets against any remote sets. Region set matching includes attribute matches and the determination of region overlaps.

4.5.2 Integration

In order to implement the PATiM mechanism, it is needed to have all the region information of the federation, and the subscriptions and associations to these regions performed by the federate. To implement the proposed mechanism in a fully distributed manner, this information should be available to the distributed nodes, which could be the LRCs. Additionally, the availability of this information is dependent on the implementation of the DDM mechanisms. As the HLA does not specify the implementation details, any specific RTI implementation may be implemented in a way that this information is available in LRCs. Besides, even if this information is available in LRCs, neither the HLA 1.3 nor the HLA 1516 specifications have any mechanisms to get them from the LRCs.

On the other hand, two of the main RTI implementations analyzed above, MAK RTI and RTI 1.3, are implemented so that region related information, which belong to all federates in the federation, is available to implement DDM mechanisms. These LRC components could be extended so that this information could be reached by using some interfaces.

In our PATiM implementation, region related information is distributed exclusively by using extra messages. Currently, there are not any open source RTI implementations that implement full DDM and Time Management services. So, full integration to LRC components is not completed yet. Of course, distributing the region related information exclusively brings extra cost to the algorithm. However, this performance penalty is not counted for the proposed mechanism as this information is actually available in the LRC implementation.

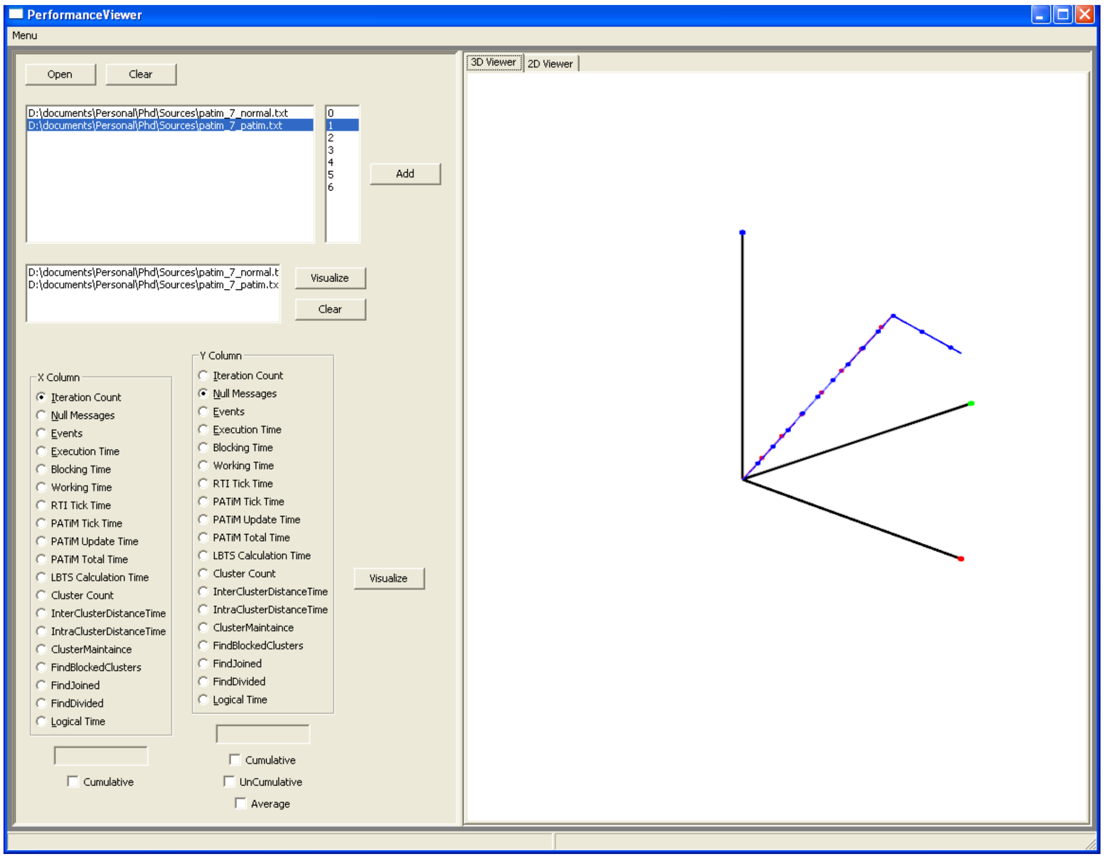
4.6 Performance Visualization

During execution of simulation scenarios, performance measures of each iteration are logged to a file. Performance items that are measured separately are listed below;

- Iteration Count: Current iteration number of simulation execution
- Null message count: Cumulative total null messages sent from all federates
- Total event count: Number of events (updates and interactions) generated by all federates

- War update time: Total update time of one simulation iteration
- Blocking iteration count: Cumulative total blocking iteration count of all federates
- Working iteration count: Cumulative total working iteration count of all federates
- RTI tick time: Tick execution time of RTI NG 1.3.
- PATiM tick time: Update execution time of PATiM mechanism. This value is the total time management related time.
- PATiM update time: Update time of PATiM. This value is the cost of dynamic cluster management algorithm.
- PATiM total time: Tick plus update time of PATiM.
- LBTS calculation time: Separate LBTS calculation time.
- Cluster count: Current cluster count of execution
- Inter-cluster distance calculation time: Distance calculation cost between clusters.
- Intra-cluster distance calculation time: Distance calculation cost for checking cluster division.
- Cluster maintenance time: Execution time for maintaining the clusters.
- Find blocking cluster time.
- Find joined cluster time.
- Find divided cluster time.
- XY positions of each federate.
- Logical time plus lookahead values of each federate.

These logged values are used to analyze the performance of the proposed mechanism. The first group of measurements is used to plot the time-location graph for individual federates. A time-location graph contains three dimensions for a simulation having a two dimensional virtual terrain. The two dimensions represent the X and Y positions of the federate and the third dimension represents the logical time of the federate. This graph represents the movement of the federate as its logical time increases. An example for the time-location graph is shown in Figure 4.10. In this figure, time-location graph of the federate F1 is shown, for both the normal and the PATiM mechanisms.



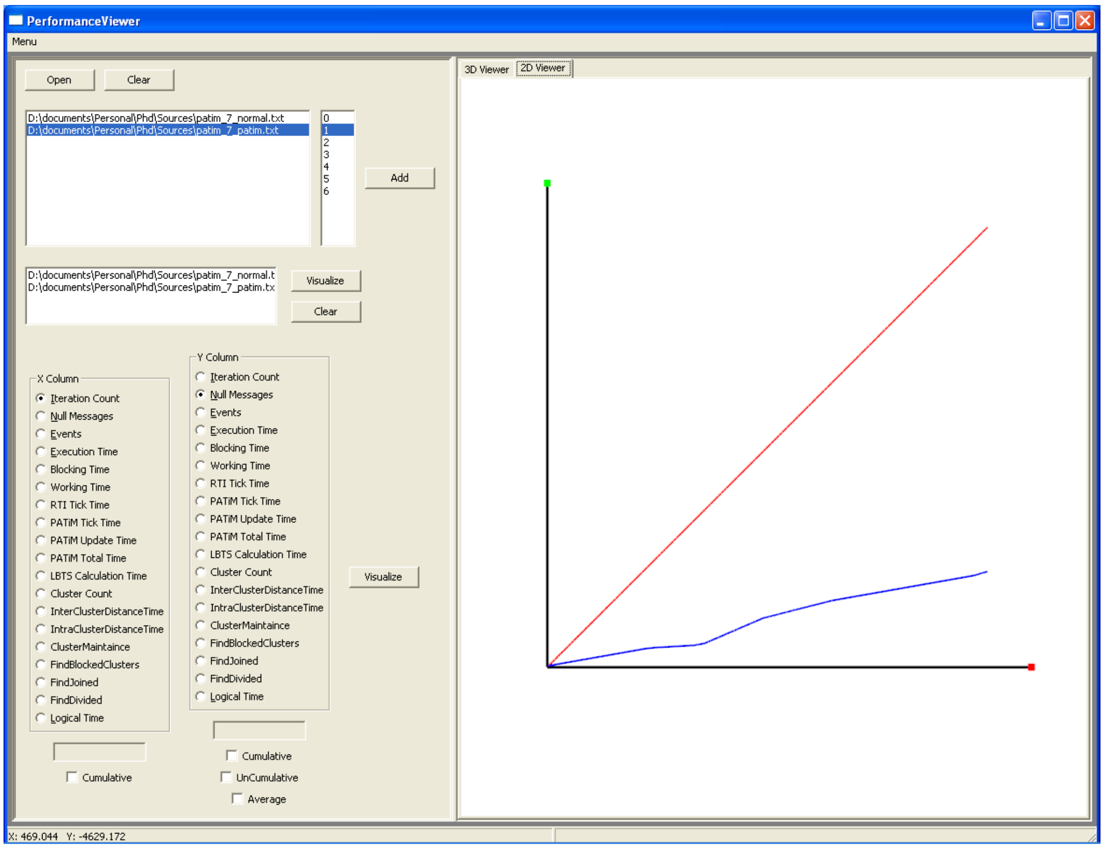
D:\documents\Personal\Phd\Sources\patim_7_normal.txt
D:\documents\Personal\Phd\Sources\patim_7_patim.txt

D:\documents\Personal\Phd\Sources\patim_7_normal.t
D:\documents\Personal\Phd\Sources\patim_7_patim.tx

X Column: [✓ Iteration Count]
Y Column: [✓ Null Messages]

Figure 4.10: Time-location graph property

Performance visualization is done using two dimensional graphs. The values that are plotted for each dimension can be selected using radio boxes placed on the left side of the graphical user interface. By using this flexible comparison mechanism, different perspectives of the implemented mechanism can be viewed easily. An example performance graph comparing iteration count versus null message count is shown in Figure 4.11. The graph mechanism includes some utilities in order to simplify the visualization process. The values of the Y dimension can be reorganized so that the graph plots the values as a cumulative for each iteration. This property is used when the user wants to see cumulative results on instantaneous values, such as cluster count. It is further possible to plot instantaneous values for a cumulative performance element, such as a null message count. Small and noisy values can be averaged in order to determine the tendency of the plotted values, such as the LBTS calculation time.



```
D:\documents\Personal\Phd\Sources\patim_7_normal.txt
D:\documents\Personal\Phd\Sources\patim_7_patim.txt
```

0 [✓ 1] 2 3 4 5 6

```
D:\documents\Personal\Phd\Sources\patim_7_normal.t
D:\documents\Personal\Phd\Sources\patim_7_patim.tx
```

X Column: [✓ Iteration Count] Null Messages Events Execution Time Blocking Time Working Time RTI Tick Time PATiM Tick Time PATiM Update Time PATiM Total Time LBTS Calculation Time Cluster Count InterClusterDistanceTime IntraClusterDistanceTime ClusterMaintaince FindBlockedClusters FindJoined FindDivided Logical Time

Cumulative

Y Column: Iteration Count [✓ Null Messages] Events Execution Time Blocking Time Working Time RTI Tick Time PATiM Tick Time PATiM Update Time PATiM Total Time LBTS Calculation Time Cluster Count InterClusterDistanceTime IntraClusterDistanceTime ClusterMaintaince FindBlockedClusters FindJoined FindDivided Logical Time

Cumulative

UnCumulative

Average

X: 469.044 Y: -4629.172

Figure 4.11: Performance measure graph property

4.7 Validation of Proposed Algorithm

PATiM mechanism provides a better performance by reducing network messages and calculation times of the LBTS value that is essential to realize time management requirements. Beside this performance improvement, it is very critical to get same simulation results with the ones that are produced by the normal time management mechanisms. Getting the same results in same simulation executions means that every object in the federation should be in the same state in a given logical time, and every event in distributed simulation should be at the same logical time.

In order to show this equivalence of simulation executions, a distributed simulation example of a war scenario is analyzed. A distributed war simulation contains a number of plane and tank federates. Each of the actors has its own path to follow on the virtual terrain. Planes can fire to enemy planes and tanks, tanks can fire to enemy tanks. There is no randomness in firing and path following in order to get the same results. Each fire operation will be successfully destroying the target, and a damaged target will not be able to make a further operation.

The same war scenario is executed first with the normal time management algorithm and then with the PATiM algorithm. During these executions, middleware recorded the current logical time, federate position, and some other performance related data to a file. Another state analyzer program read files of different executions of the same simulation scenario and drew a three dimensional representation of the simulation states. Afterwards, the three dimensional graphs were taken from this tool and the situations for these two time management algorithms were depicted.

In the simulation state graphs, there are three axes which represent logical time, x and y coordinates. These graphs represent the position of war elements with respect to the logical simulation time. There are two data lines in Figure 4.12 representing war element Tank1's simulation state for the PATiM mechanism and for the normal time management mechanism. As it can be seen from the graph, Tank1 is on exactly the same point at each logical time for both the PATiM and

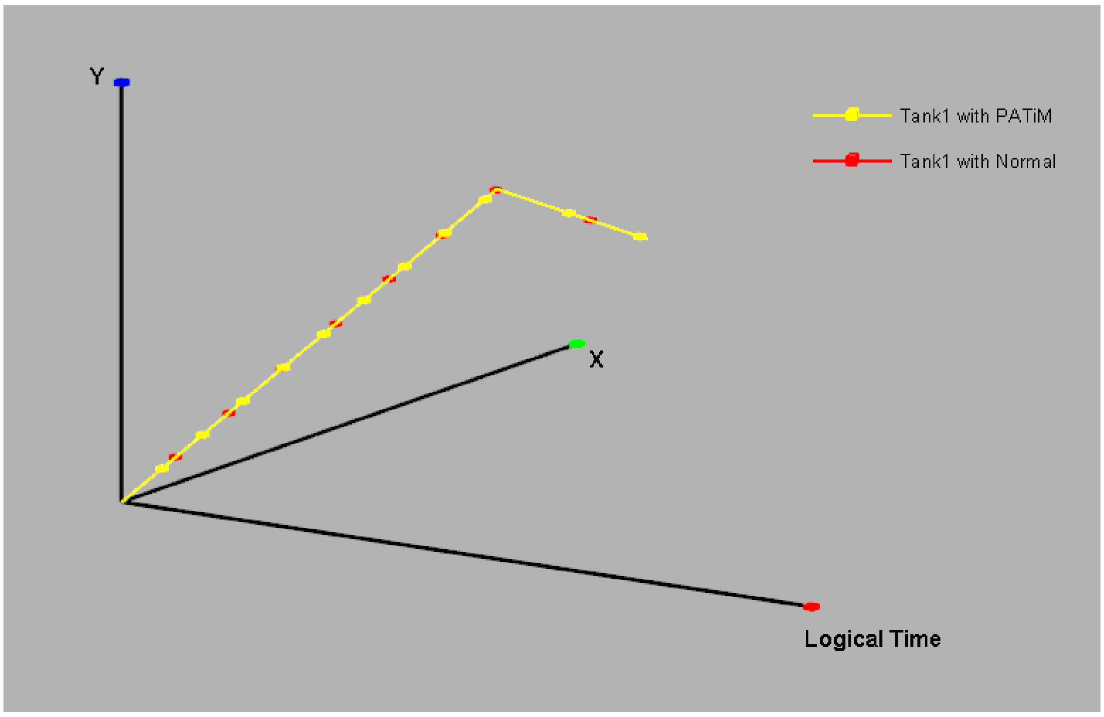


Figure 4.12: Tank1 Simulation State Graph

the normal time management mechanisms. Simulation state graph for another war element Plane1 is shown in Figure 4.13. Again, both PATiM and normal time management mechanisms generate same states for the same logical time values.

In Table 4.1, events generated within the simulation scenario and logical time values that these events exist at, for both PATiM and the normal time management mechanisms, are listed. As it can be seen from the table, all events happen at the same logical time for both time management mechanisms.

Table 4.1: Event times

Event Description	PATiM	Normal time management
Tank3 is fired by Plane1	456	456
Tank2 is fired by Plane2	890	890

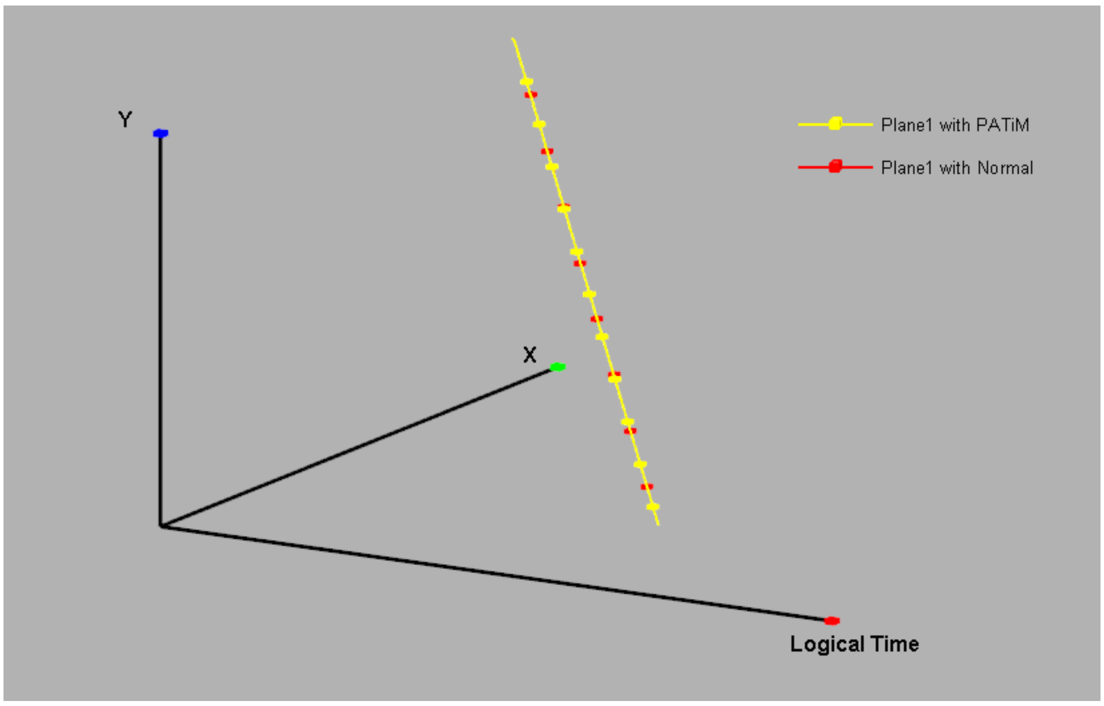


Figure 4.13: Plane1 simulation state graph

CHAPTER 5

ANALYTICAL RESULTS

In this chapter, analyses of the proposed PATiM mechanism from different perspectives are given. In order to analyze the advantages and overheads of the proposed mechanism, a group of distributed simulation scenarios are executed and different components of these executions are analyzed. The metrics chosen are reduction of the number of synchronization related messages, logical time, concurrency, and the LBTS computation overhead. Two different simulation applications are modeled; the first one is a simple war game simulation and the second one is a transportation simulation. Two common properties of these simulations are the involvement of real world physical entities and the time-stepped organization of the simulation applications.

5.1 War Game Simulation

As mentioned before, the proposed mechanism is applicable to time-stepped simulations. Especially distributed simulations, in which there is a simulation of physical entities, are perfectly suited. In order to evaluate the performance of the proposed PATiM mechanism, a group of distributed simulations are created. The first scenario is a distributed war simulation, in which there are two teams composed of planes and tanks. There are different numbers of federates (i.e. war elements) in each simulation scenario. Each war element moves on the two-dimensional space following a predefined path. In each iteration of the simulation loop, war elements try to increment their logical time values by using the *timeAdvance()* method of the time management services. Each war element possibly has a different time increment value which is used to increment its logical time. These different time increment values are used to simulate computational environments of different speeds.

In the real world, each federate most probably runs on different computational power and their logical time increments, i.e. their advance speed in the simulation, is also different. They further move on the virtual two dimensional terrain as their logical time values are incremented. The movement amount is determined by the time change value and the speed of the vehicle. A war element has a fire range and when opposite side war elements enter its range, it fires at them. Every fire operation has to be successful so that the simulation execution is repeatable.

All of the federates in the simulation environment are executed on the same processor in order to simplify the setup process of this experimental environment for different scenarios. However, these federates communicated with each other only using the HLA services, similar to the normal federates. The results of these executions are logged in order to analyze the performance. The results of these executions generated by the PATiM mechanism are compared with the normal time management mechanism. The normal time management mechanism implemented is based on the time management algorithm given in section 2.4.2.4.

5.1.1 Message Count Comparison

Distributed time management algorithms use null-messages in order to assure consistency. Of course, these null messages are just extra messages which are used to inform other federates about their internal states. Reducing the number of null messages is very important for a synchronization algorithm.

In Figure 5.1, comparison of null message counts for both normal distributed time management mechanism and PATiM mechanism is given. In this graph, the change of the total null message count is depicted as simulation steps forward. Iteration count is the number of steps that the simulation executes. As it can be seen from the figure, null message generation of the PATiM mechanism is dramatically smaller than the normal mechanism. This reduction comes from the fact that the PATiM mechanism partitions the federation by utilizing the clustering mechanism, and null messages are only used for inter-cluster synchronization.

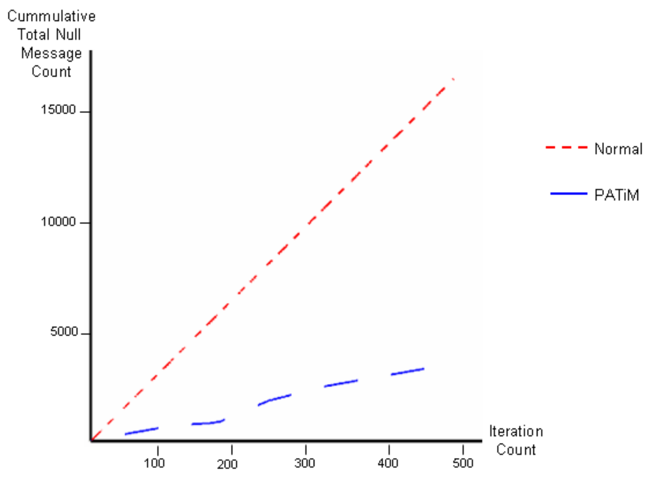


Figure 5.1: Null message count

As mentioned before, in a distributed time management mechanism, null messages should be distributed to all other time constrained federates with reliable multi-unicast links. This means that every time an advance operation takes place, a group of messages will be distributed to other federates in the federation. If all of the federates try to advance their logical times there will be $n(n-1)$ messages, because all federates are dependent to all other federates. Here comes the first performance improvement opportunity. When federation is divided into federate clusters and federate's logical times depend only on the federates within the same cluster, this decreases the total number of synchronization messages dramatically. When we divide the whole federation into synchronization groups by clustering the federates, there will be smaller number of federates within each federate cluster than in the original federation.

In the inequality 5.1 below, the left side represents the message count of the normal distributed time management mechanism. The right hand side represents the message count of PATiM mechanism, where there are two federate clusters in the federation.

$$n(n-1) > k(k-1) + t(t-1) \qquad \forall n = k + t \tag{5.1}$$

When k is replaced with (n-t), the inequality is converted to following one.

$$n(n-1) > (n-t)(n-t-1) + t(t-1) \qquad \forall n = k + t \tag{5.2}$$

This inequality is reduced to $n > t$ $\forall n = k+t$ which is always true for positive non-zero integer values. This inequality depicts that when a federation containing n federates is divided into two federate clusters with k and t number of federates, PATiM mechanism always results in smaller number of synchronization related messages. This equation could be generalized to any number of federate clusters as follows;

$$n(n-1) > \sum_{i=1}^{m}(k_i(k_i-1)) \qquad where\ n = \sum_{i=1}^{m} k_i,\ m > 1\ and\ \forall k_i > 0 \tag{5.3}$$

This inequality can be reduced as following;

$$n^2 - n > \sum_{i=1}^{m}(k_i^2 - k_i) \tag{5.4}$$

$$n^2 - n > \sum_{i=1}^{m}(k_i^2) - \sum_{i=1}^{m}(k_i) \tag{5.5}$$

$$n^2 - n > \sum_{i=1}^{m}(k_i^2) - n \tag{5.6}$$

$$n^2 > \sum_{i=1}^{m}(k_i^2) \tag{5.7}$$

This last inequality 5.7 can be proved by manipulating assumption condition;

$$n > \sum_{i=1}^{m}(k_i) \tag{5.8}$$

$$(n)^2 > \left(\sum_{i=1}^{m}(k_i)\right)^2 \tag{5.9}$$

$$n^2 > \left(k_1 + \sum_{i=2}^{m}(k_i)\right)^2 \tag{5.10}$$

$$n^2 > k_1^2 + 2\sum_{i=2}^{m} k_i k_1 + \left(\sum_{i=2}^{m}(k_i)\right)^2 \tag{5.11}$$

$$n^2 > k_1^2 + 2\sum_{i=2}^{m} k_i k_1 + \Big(k_2 + \sum_{i=3}^{m}(k_i)\Big)^2 \tag{5.12}$$

$$n^2 > k_1^2 + 2\sum_{i=2}^{m} k_i k_1 + k_2^2 + 2\sum_{i=3}^{m} k_i k_2 + \Big(\sum_{i=3}^{m}(k_i)\Big)^2 \tag{5.13}$$

When this operation continues until m number of steps we get the following equation;

$$n^2 > \sum_{i=2}^{m} k_i^2 + \sum_{i=1}^{m-1}\Big(2\sum_{j=i+1}^{m} k_i k_j\Big) \tag{5.14}$$

$$\sum_{i=1}^{m-1}\Big(2\sum_{j=i+1}^{m} k_i k_j\Big) > 0 \qquad where\ m > 1\ and\ \forall k_i > 0 \tag{5.15}$$

As it is known that the above equations 5.14 and 5.15 are true, inequality 5.7 is always true for all positive non-zero integer n and k_i values where n is equal to summation of all k_i. As it is known that the inequality 5.7 is true, the original inequality 5.3 is always true. This theoretically proves that the PATiM mechanism always generates less number of null messages because clusters will always contain non-zero positive number of federates and the summation of federate counts in clusters is equal to the total number of federates in the federation.

Mainly, the simulation model determines the degree of clustering. This depends on the ratio of total average distances between the federates and on the clustering threshold values. If the federates in a distributed simulation widely spread around the virtual space, then there will be higher number of clusters. Widely spread means that federates are positioned at large distances. In this case, the average distance between the federates is relatively higher than the cluster thresholds. As mentioned before, threshold values are determined by the nature of the simulation domain.

In fact, a threshold value is a distance in virtual space at which the communication between the federates is broken down. In an example simulation of the traffic environment, vehicles can process events happened within approximately 1 km distance. In this case, the simulation designer could assign the threshold value as 2 or 3. If the terrain is about 100 km^2 and the vehicle federates are more likely to travel distant points on the terrain, there may be higher number of clusters during the execution. However, in another run, if the vehicles have special radars on them and they can process events, i.e. could see other vehicles, within 60 km distance, probably the simulation designer will define the threshold value as 70 km. By having this threshold value and 100 km^2 virtual terrain, clustering algorithm would allow at most 2 clusters at the same time.

Figure 5.2 plots null message counts against the iteration counts comparing highly clustered and lightly clustered federations.

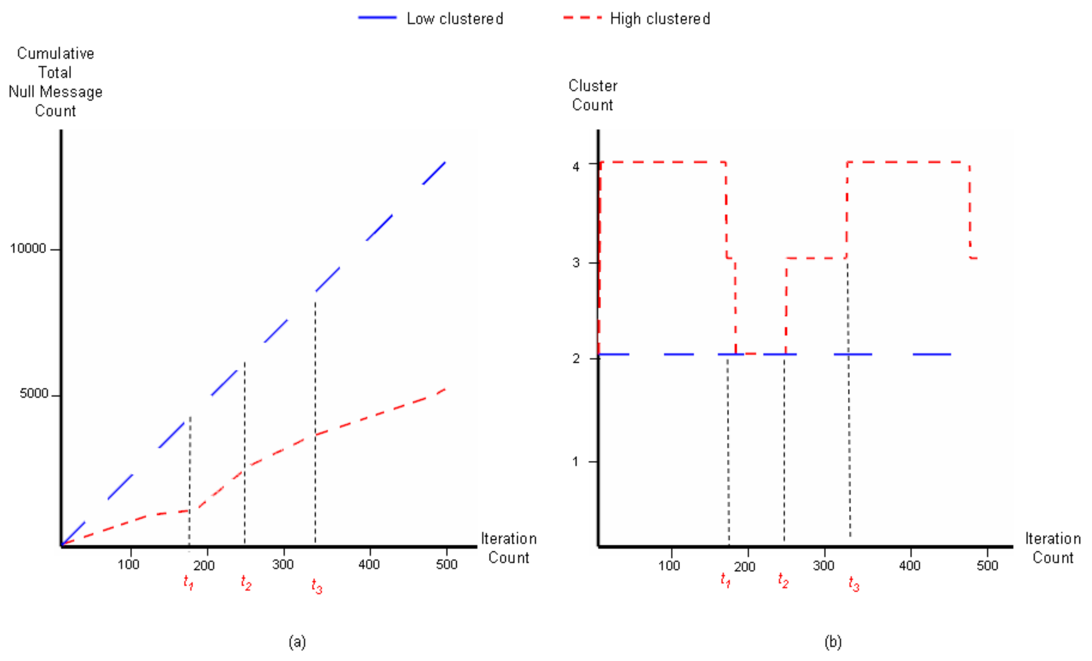


Figure 5.2: Clustering effect on null message counts

In order to analyze the effect of clustering, two different scenarios are executed. In one of the scenarios, there are 4 different federate groups. In this scenario, the federates travel more distances in distant groups on the virtual terrain. In Figure 5.2 the number of clusters is shown on the right part and the cumulative total null message counts as a function of iteration count is plotted on the left.

As it can be seen from left graph of Figure 5.2, in the highly clustered federation less number of null messages is produced. Before the iteration t_1 there are 4 clusters in the high clustered scenario and the slope of the null message count curve is much smaller than the low clustered scenario. However between iterations t_1 and t_2 there is only one cluster and the slope of both curves is nearly the same. Between iterations t_2 and t_3 there are 3 clusters in the high clustered scenario. This number of cluster results in a smaller null message count than in the low clustered scenario, but in this case the slope of the message count curve is higher than the one before the iteration t_1.

Another factor that affects cluster count during simulation executions is the values of clustering and joining thresholds. If these values are relatively smaller than virtual space dimensions, there will be higher number of clusters at average. Figure 5.3 shows the null message count as a function of iteration count for the same scenario, but different thresholds.

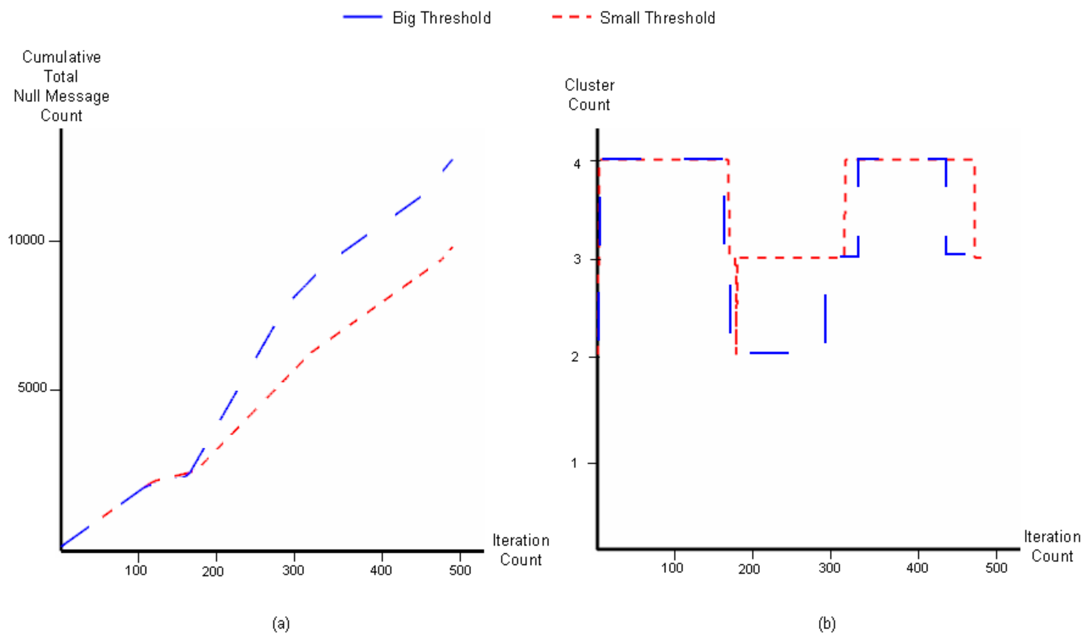


Figure 5.3: Effect of threshold values on null message counts and cluster counts

The difference between executions in this figure is the amount of thresholds. In one of the executions, threshold values are 120 and 90 for clustering and the joining thresholds respectively; for the other execution, they are 80 and 65, respectively. In both executions, initially they have the same number of federate clusters and null message counts. After a while, different threshold values start to affect the number of clusters and the null message counts. As it can be seen from the left graph, null message counts for executions that have smaller threshold values are less than the other one. By having smaller threshold values, the PATiM mechanism divides existing clusters earlier and joins clusters later. This results in higher average cluster counts and decreased null message counts.

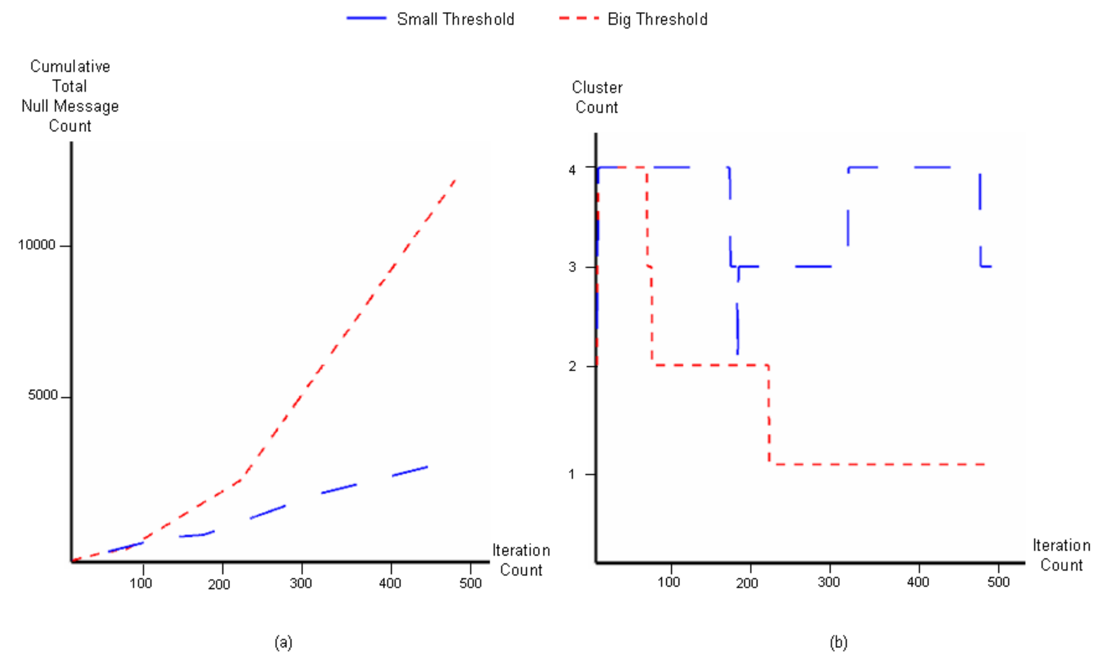


Figure 5.4: Effect of different threshold values

A more dramatic difference between null message counts is obtained when the difference between threshold values is increased. In Figure 5.4 there are null message count and cluster count comparisons of different executions of the same simulation scenario that have different threshold values. In the first execution, values for clustering and joining thresholds are 300 and 270, respectively. In other execution, these values are 80 and 65. As it can be seen from the graphs, small thresholds generate higher average cluster counts and smaller null message counts.

5.1.2 Event Count Comparison

In the distributed war simulation there are a number of events involving attribute updates and interactions to take place. In Figure 5.5, a comparison of total number of events happened during the simulation executions are presented.

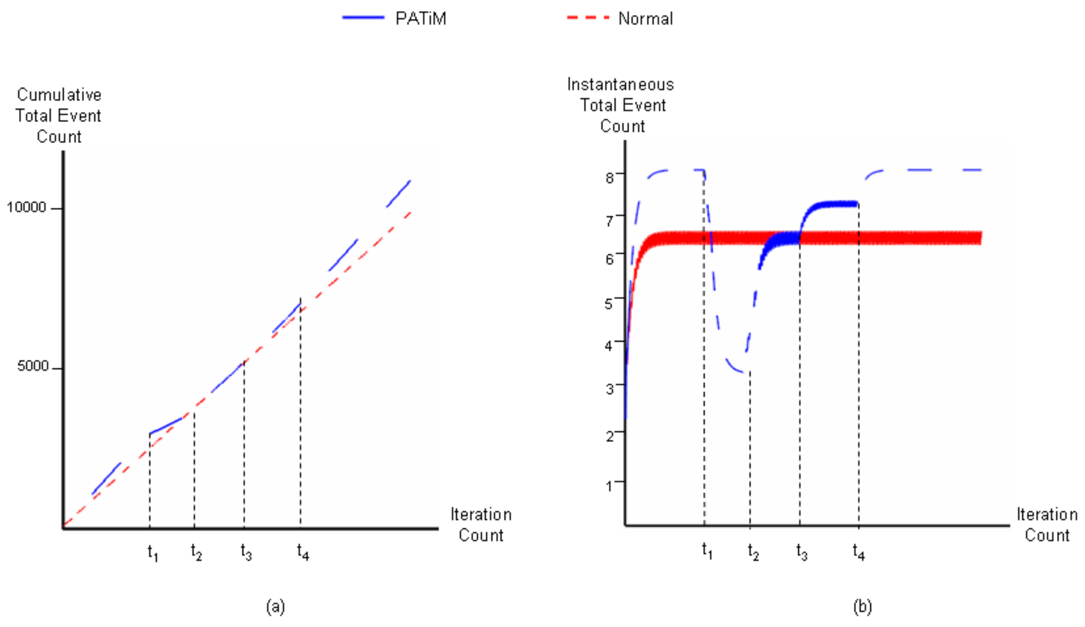


Figure 5.5: Event count comparison

In these graphs, event throughputs of both mechanisms are shown. The graph on the left shows cumulative total event counts, where event count is the sum of all previous event counts plus the current ones. The graph on the right shows instant total events in each iteration. Before time t_1, some of the federates are in different federate clusters. Thus, event generation of the PATiM mechanism is higher than the normal time management mechanism. During this period, in normal time management mechanism, event generation speed of each federate is regulated by the slowest federate in the federation. This fact generates a relatively constant total event generation speed during the simulation execution. However, in the PATiM, federates are divided into three clusters and execution speeds of these clusters are independent of each other. In this case, the fastest cluster could generate events more frequently than the slowest cluster, and therefore, there is a higher total event generation in the PATiM mechanism.

Some of event generation capabilities could not be used in the normal time management mechanisms because slower federates block the faster ones. This unused capabilities are allowed to be used only if federates are distant to each other because of the logical time independency. However, when they are getting closer to each other, their logical times have to be resynchronized. This resynchronization operation is completed by blocking the faster clusters. This blocking results in sharp decreases in the amount of event generation because blocked federates cannot generate events. At time t_1 in Figure 5.5, federate clusters start to get closer to each other and they enter the resynchronization period, which starts at t_1 and ends at t_2. In this period, faster federates are blocked and they could not generate events. This blocking of some federates obviously decreases the total simulation event generation speeds.

At time t_2 resynchronization period ends and clusters are joined into one greater cluster. Between times t_2 and t_3, there is only one federate cluster and event generation speed of the PATiM and normal time management mechanism is nearly the same. Between times t_3 and t_4, some of the federates again move away from other federates and thus they are separated from the large cluster. In this interval, there are two federate clusters in the PATiM execution. After t_3 faster federates again generate events more frequently than slower federates and total event count amounts have greater values than in the normal time management mechanism. At time t_4, one of the clusters is divided and the total cluster count of federation becomes three. In this case, event generation of federation in the PATiM mechanism increases to even higher values.

In Figure 5.6 event count comparison for another execution containing 50 federates is shown. In part (a) and (b) of the figure, cumulative total and total event count graphs are shown. In part (c) cluster count changes are shown. In this execution federates are highly distant to each other and the average cluster count during this simulation is higher. This situation causes higher differences between normal and PATiM executions in their event counts. In this simulation, there are four clusters before iteration t_1. Between t_1 and t_2, the cluster count has increased to five and event count generation has become higher. At iteration t_2 the resynchronization period begins and it continues until t_3. During this period, event generation of the PATiM mechanism is dropped because a fast cluster is blocked and could not generate any events. At iteration t_3, the resynchronization period ends and the event generation speed of the PATiM mechanism increases to its original level.

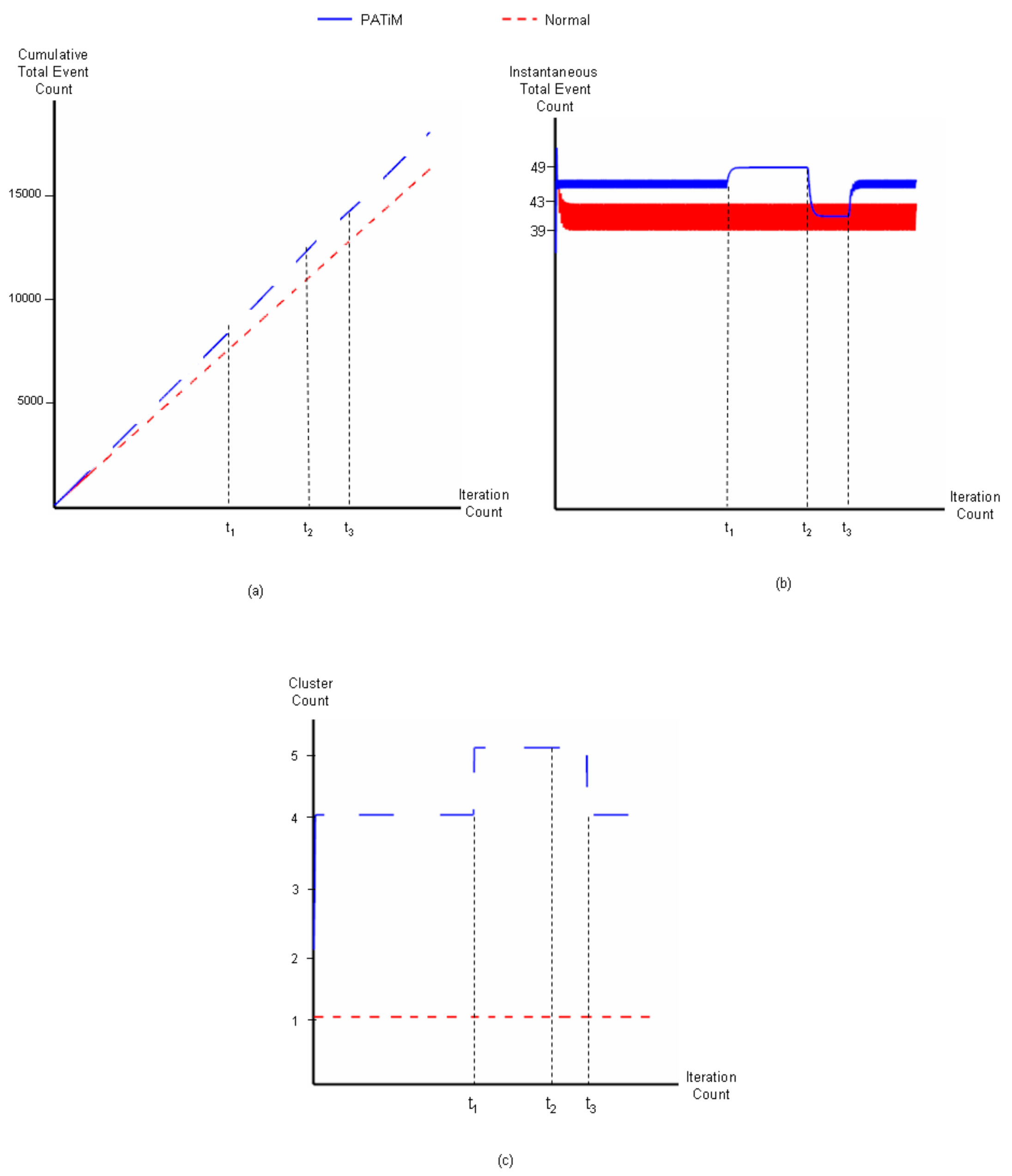


Figure 5.6: Event count comparison for 50 federate

5.1.3 Logical Time Comparison

In distributed simulations, each federate simulating an actor tries to advance its logical time in each iteration. Of course, in some of the iterations, the time management mechanism does not allow some of these time advance operations because of the slower federates. In Figure 5.7, graphs of logical time changes of the federate F4 are shown. On the left graph, logical time changes with respect to simulation iteration count are shown. Similar to previous graphs, before time t1, federates are distant to each other and there is more than one federate cluster. Being in a faster cluster, logical time of F4 increases with a higher speed in the PATiM mechanism when compared to the normal time management mechanism, because slower federates block F4 in some iterations.

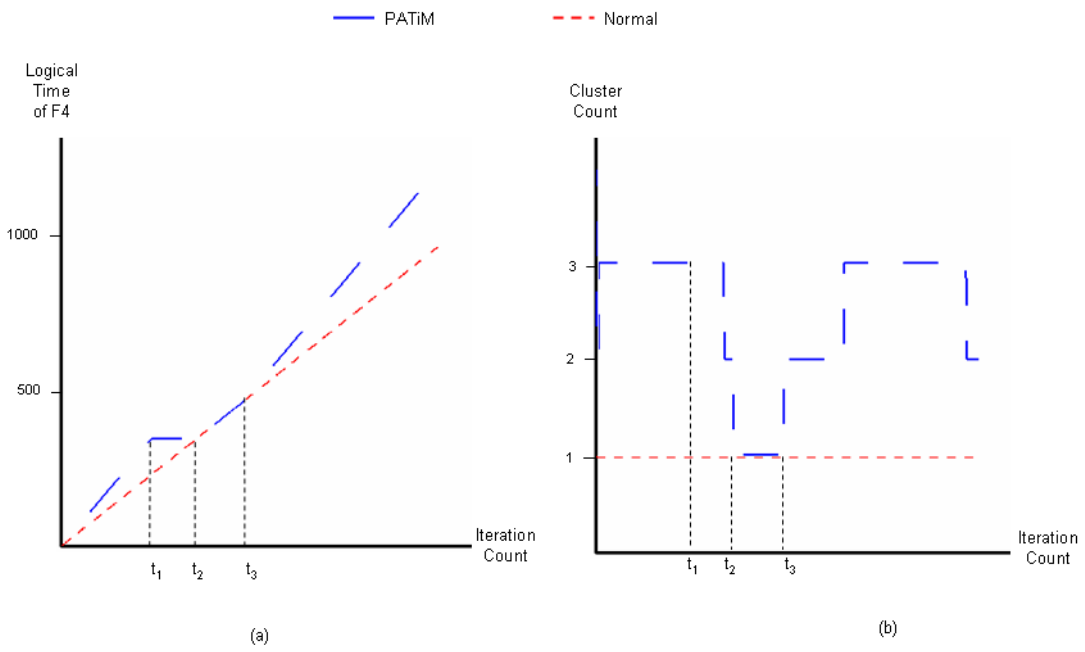


Figure 5.7: Logical time comparison of federate F4

In fact, before time t_1, there are 3 clusters in the federation, FC_1, FC_2 and FC_3. Federate F4 is in FC_1. During t_1 and t_2, two of them, FC_1 and FC_2, are in resynchronization period because they are getting closer. During this resynchronization period, F4 is blocked by the PATiM mechanism because the cluster it belongs to is faster than FC_2. Thus its logical time is constant during this period while the logical times of slower federates increase and finally reach the logical time of F4 at time t_2. At t_2, the resynchronization period between these two clusters ends and they construct a new cluster called FC4.

Between t_2 and t_3, there are two federate clusters in the federation, FC_3 and FC_4. Federate F4 is in cluster FC_4 with a slower federate. Thus, during this period, logical time increase of F4 in both the normal and the PATiM mechanism is the same. In the normal time management mechanism, logical times of all federates are regulated by the slowest federate in the simulation. Thus, in normal time management mechanisms, constant logical time increases are obtained.

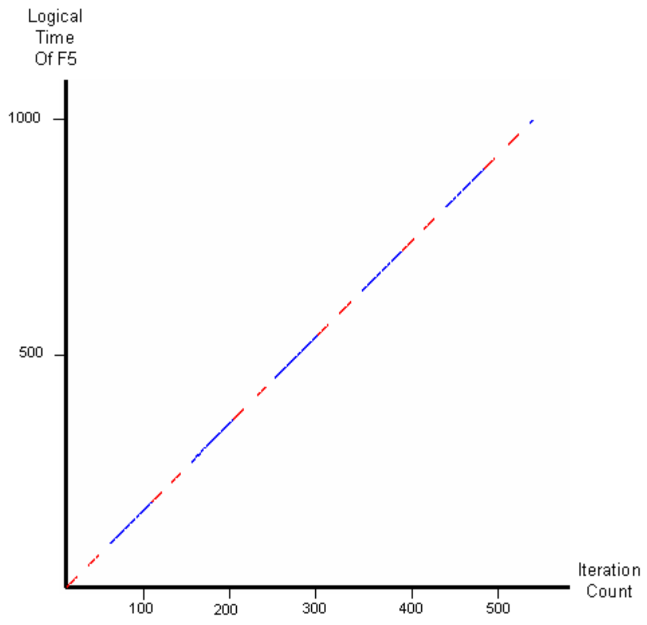


Figure 5.8: Logical time comparison of federate F5

In Figure 5.8, logical time changes of federate F5 are shown. F5 is, in fact, the slowest federate in the federation. Thus, in normal time management mechanism its logical time value constrains all other logical times of the other federates. On the graph, the logical time and iteration count comparison of the federate F5 is shown. As it can be seen from the graph, in both PATiM and normal time management mechanisms, logical time values of F5 is the same. This result comes from the fact that F5 is the slowest federation in the system. In both mechanisms, it has not been blocked by any of the other federates, but it blocks the others.

5.1.4 Concurrency Comparison

The PATiM mechanism divides the federation into highly dependent subparts based on the distance between the federate groups. Resulting clusters contain federates that are dependent on each other but not related to the federates outside the group. For a period of time, the logical time dependency between federates of these clusters is broken and there will not be any interaction between them. Thus, the concurrency within the federation is increased during this non-interacting period. As mentioned in section 3.4.1, these periods are called local synchronization periods.

In order to measure the concurrency within the federation, the total working and blocking iteration counts for all the federates in the federation are measured. Working iterations are iterations in which the federate advances its logical time and forwards its position in the virtual terrain. In blocking iterations, the federate cannot advance because of at least one slower federate. In a time stepped federation execution, the concurrency of that execution can be said to be high if working iteration count is high and blocking iteration count is low.

The same simulation scenario is executed with 7 federates using the PATiM and normal time management mechanisms. Graphs in Figure 5.9 show the total working and blocking iteration counts of all the federates as the simulation iteration count increases. In parts (a) and (b) of this figure, cumulative and normal blocking iteration counts in the PATiM execution are given respectively. In parts (c) and (d), cumulative and normal working iteration counts of the normal time management execution are given. As it can be seen from the graphs, in the PATiM mechanism, blocking iterations are less and working iterations are higher than in the normal time management mechanism. Before iteration t_1, there are three federate clusters in the PATiM and all federates in these clusters have equal execution speeds. Thus, there is no blocking iteration during this period. Between iterations t_1 and t_2, these two federate clusters enter the resynchronization period and the federates of the faster cluster are blocked. During this period, this cluster blocking operation results in a high increase in the blocking counts. Between iterations t_2 and t_3, values are the same for both mechanisms. This comes from the fact that between these iterations, there is only one federate cluster in the PATiM mechanism and it generates blocking iteration counts similar to the normal time management mechanism. At iteration t_3, the cluster count becomes two and at iteration t_4, the cluster count becomes three.

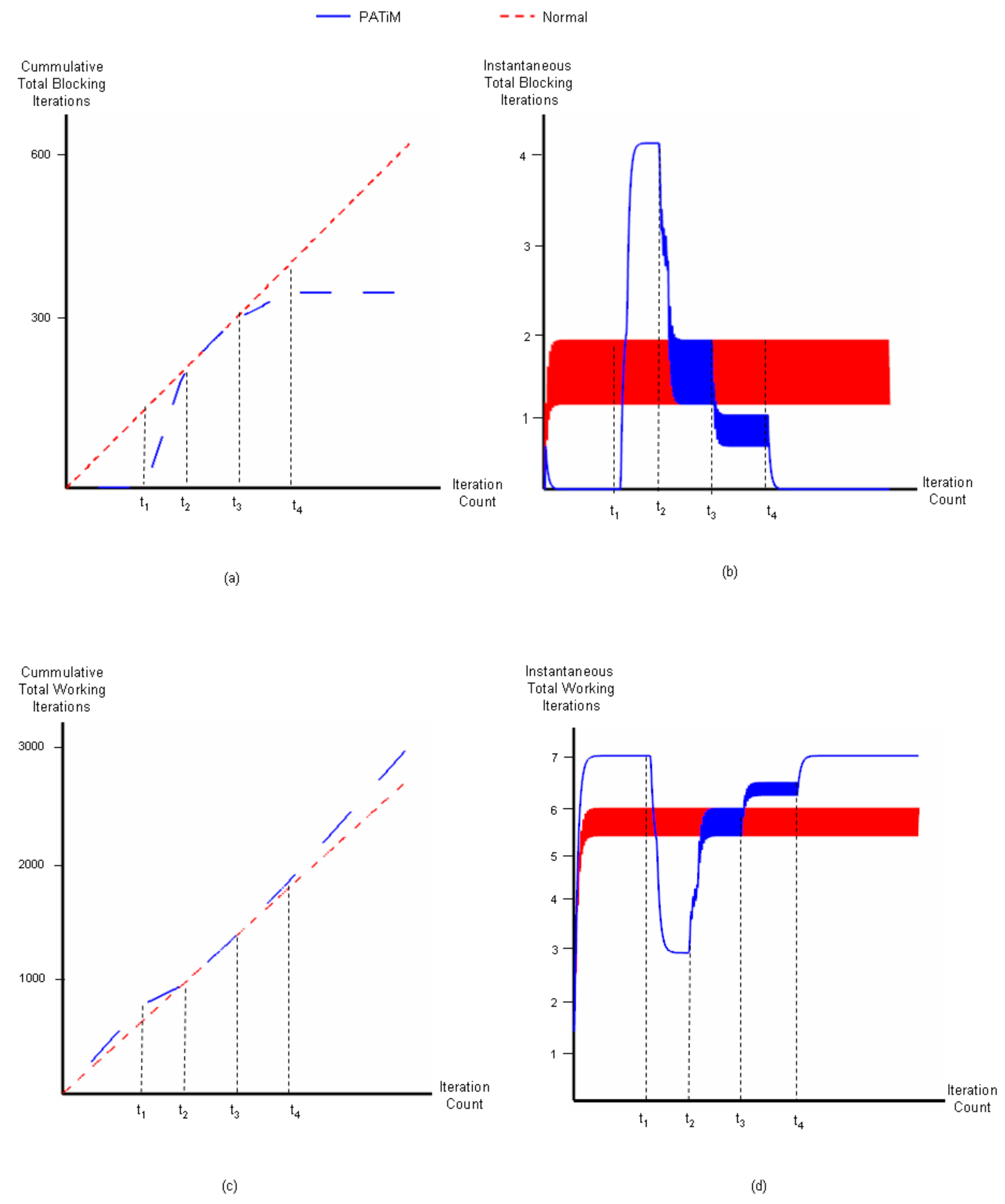


Figure 5.9: Concurrency results

Blocking and working iteration counts in the PATiM mechanism is affected mainly by two factors, one of which is the average cluster count. If the number of clusters in the simulation execution is high, then the concurrency will increase. The second factor is the formation of federate clusters. If there are federates having similar execution speeds in each federate cluster, than the concurrency of the execution will increase, because there are less blocking operations that would result due to a slower federate.

In Figure 5.10, blocking and working iteration count graphs are given for a different simulation execution. In this execution, there are again 7 federates but federate clusters are distant in most of the iterations and the average number of clusters is higher than the previous execution. As it can be seen from the graphs, the number of blocking iterations is less and the number of working iterations is higher in this execution. Before iteration t_1, there are two federate clusters in the PATiM. These clusters contain federates that have similar execution speeds and during this period, the number of blocking iterations is zero. At iteration t_1, the resynchronization operation between these two clusters is initiated and 4 of the federates, which are in the faster cluster, are blocked.

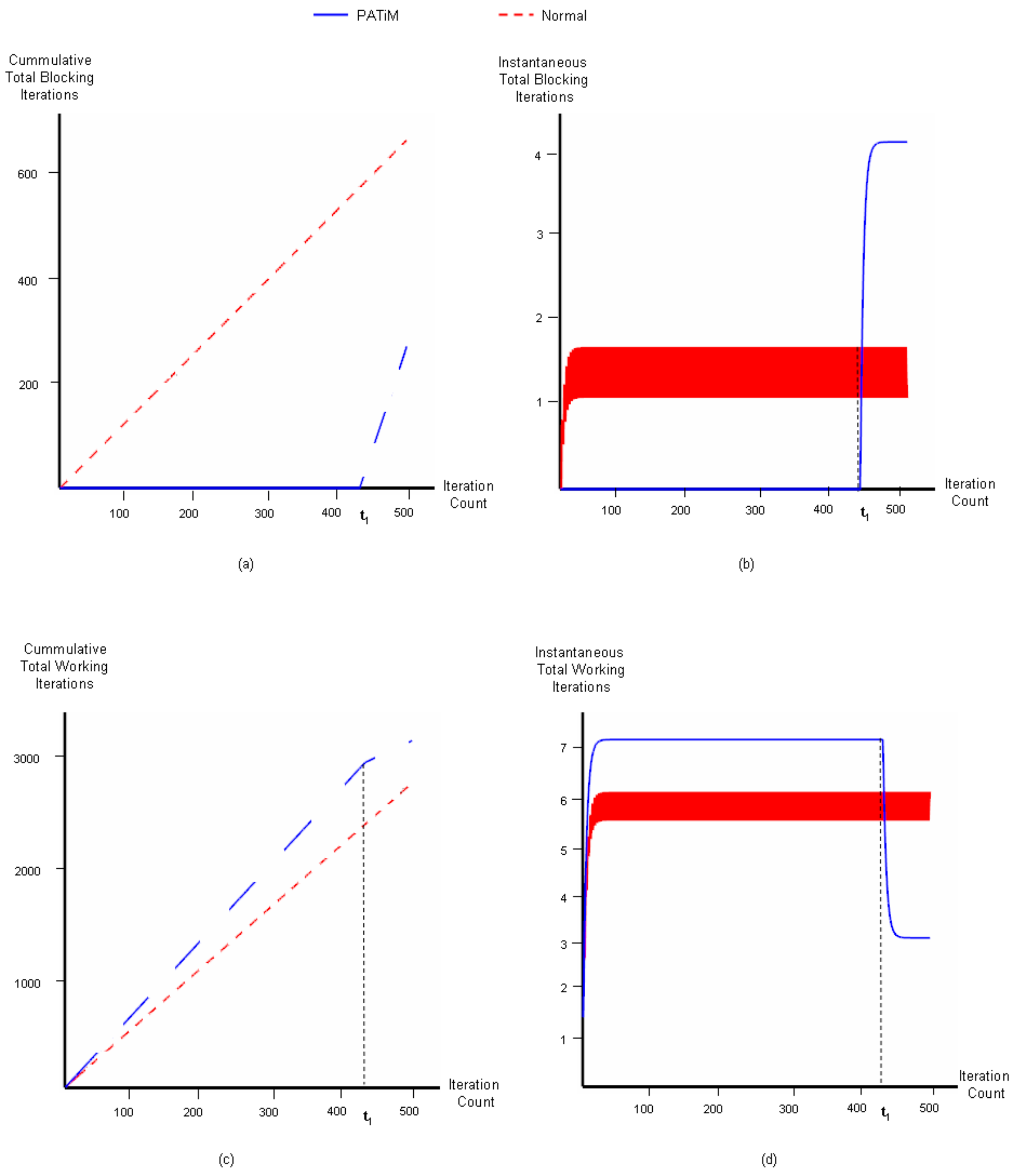


Figure 5.10: Concurreny results for a different execution

5.1.5 Execution Time Analysis

In order to evaluate the performance of the PATiM mechanism with respect to the increase in the federate count, three different distributed simulation scenarios, containing 7, 20 and 50 federates, have been run. In fact, the following execution time analyses are only related with the time management executions. The RTI data distribution and federation management related to execution times are not important in this study because these algorithms are not optimized by the proposed mechanism. In order to measure the performance of the PATiM mechanism, different components of execution time values are separately analyzed. These components are the LBTS computation time and the PATiM update time.

LBTS calculation time analysis

The LBTS computation is the hearth of the time management system. As mentioned before, this value determines the minimum logical time value for any federate in the distributed federation. Effectively computing this value dramatically increases the time management performance.

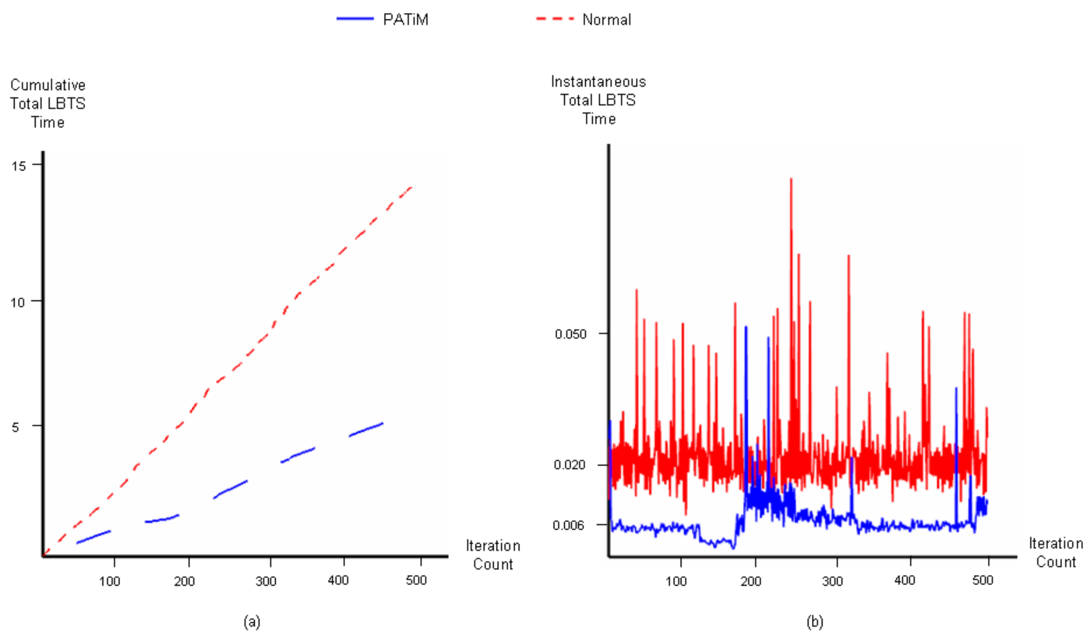


Figure 5.11: LBTS Calculation Times for 7 Federates

In Figure 5.11, two LBTS calculation time graphs for a simulation execution that contains 7 federates, are shown. In part (a) of the figure, cumulative time values are shown in milliseconds. In part (b) individual calculation time is given for each iteration. As it can be seen from these graphs, the PATiM LBTS calculation times are relatively less than the normal time management mechanism.

In order to calculate the LBTS value for a federate, the minimum of all the logical time plus the lookahead values should be calculated. Priority queues that are implemented using heaps are used in order to find the minimum value. Complexity of finding the minimum is $O(1)$. In each time advance operation, the priority queue should be updated, so that the minimum value is on the top of the heap. For this operation, the node containing the logical time plus the lookahead value should be found and its value should be updated. After that, the heap property of this list should be reconstructed. The worst case complexity of this operation is $O(\log n)$. Total complexity of the LBTS calculation for n federates is $O(n \log n)$. This complexity value is valid only in the normal time management mechanism. In the normal method, the minimum value is needed to be calculated for all federates in the federation. On the other hand, in PATiM mechanism, this complexity value is reduced because the federation is divided into smaller parts. Federates in the clusters need to find the minimum value only across federates that are in the same cluster. When the whole federation is divided into synchronization groups by the clustering federates, there will be smaller number of federates within each federate cluster considering the entire federation. The following inequality shows this condition;

$$n \log n > \sum_{i=1}^{m} (k_i \log k_i) \qquad where\; n = \sum_{i=1}^{m} k_i,\; m > 1 \; and \, \forall k_i > 0 \tag{5.16}$$

In equation 5.16, the left side represents the total count of comparison operation to find the LBTS in a normal distributed time management mechanism. The right side represents the comparison count of the PATiM mechanism where there are m number of federate clusters in the federation. This equation depicts that, when a federation containing n federates is divided into m numbers of federate clusters, the PATiM mechanism always requires less comparison operation to find the LBTS value.

In order to proof this inequality, the right side can be extracted to the following;

$$A = \sum_{i=1}^{m}(k_i \log k_i) = k_1 \log k_1 + k_2 \log k_2 + ... + k_m \log k_m \qquad (5.17)$$

New value $k^{'}$ is introduced in order to change the equality 5.17 to an inequality. If all k_i values are replaced with a value which is greater or equal to it, the following inequality is produced.

$$A \leq \left(\sum_{i=1}^{m} k_i\right) \log k_i^{'} \qquad where\ k^{'} = \max(k_i) \qquad (5.18)$$

$$A \leq n \log k^{'} \qquad (5.19)$$

$$n > k^{'} \ \Rightarrow\ \log n > \log k^{'} \ \Rightarrow\ n \log n > n \log k^{'} \qquad (5.20)$$

From inequalities 5.19 and 5.20 the following inequality could be produced

$$n \log n > n \log k^{'} \geq A \qquad (5.21)$$

This resulting inequality 5.21 proves the original inequality of 5.16, which states that the PATiM mechanism has always smaller LBTS calculation times than the normal time management mechanism, when there is clustering during simulation execution.

In Figure 5.12, tthere are LBTS calculation time graphs for two federation executions, containing 20 and 50 federates. In part (a) and (b) of the figure, there are cumulative and individual LBTS calculation time value graphs for an execution that contains 20 federates. In parts (c) and (d), LBTS calculation time graphs are given for an execution containing 50 federates.

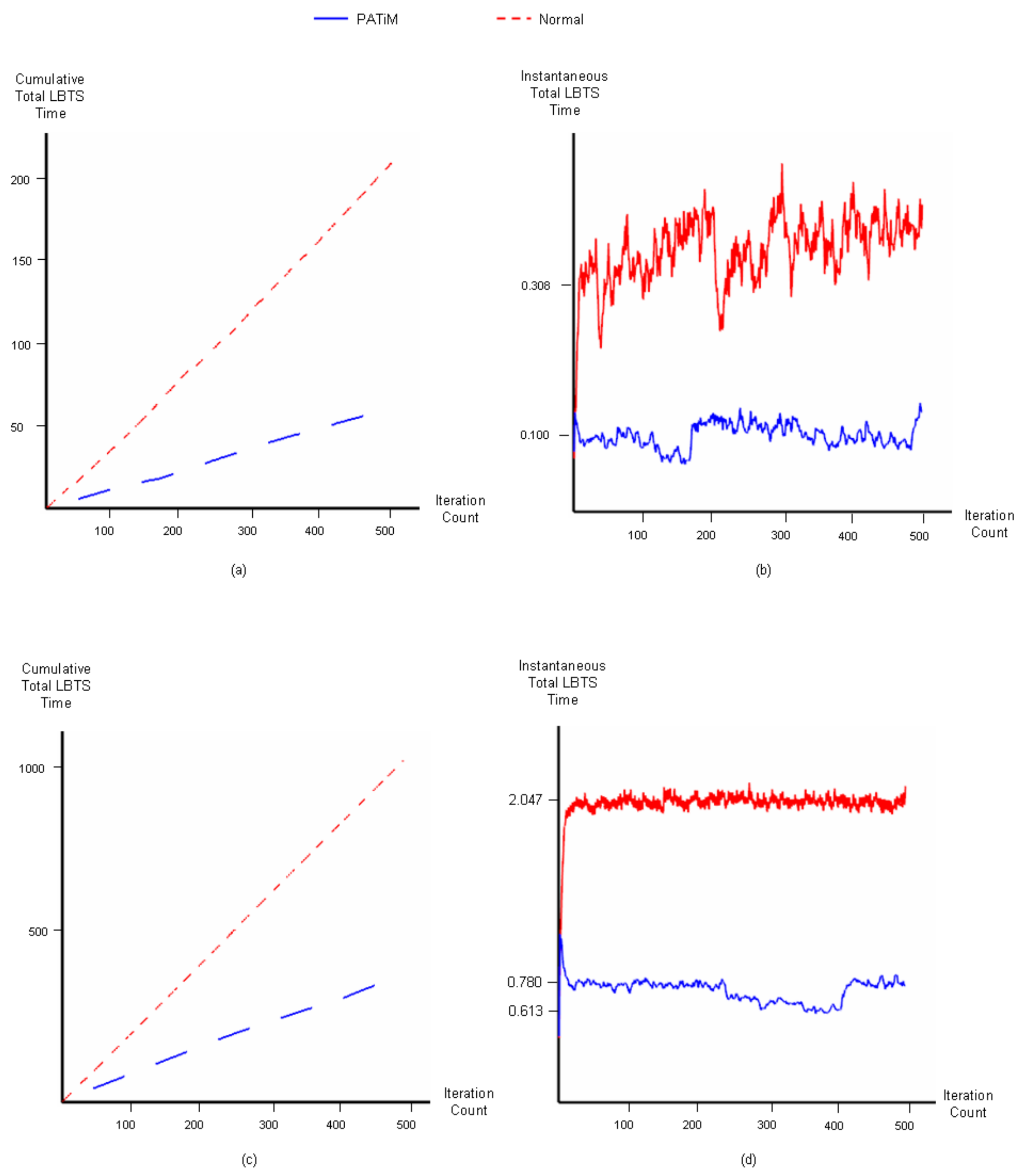


Figure 5.12: LBTS Calculation Times for 20 and 50 federates

PATiM Update Time Analysis

In the update method of PATiM mechanism, dynamic cluster formation related operations are executed. This method is the actual overhead of the proposed mechanism because in the normal time management mechanism, there is no dynamic clustering. In this section, execution time values of the update method for different simulation executions are given. Further, complexity analysis of this method for different operation groups is given.

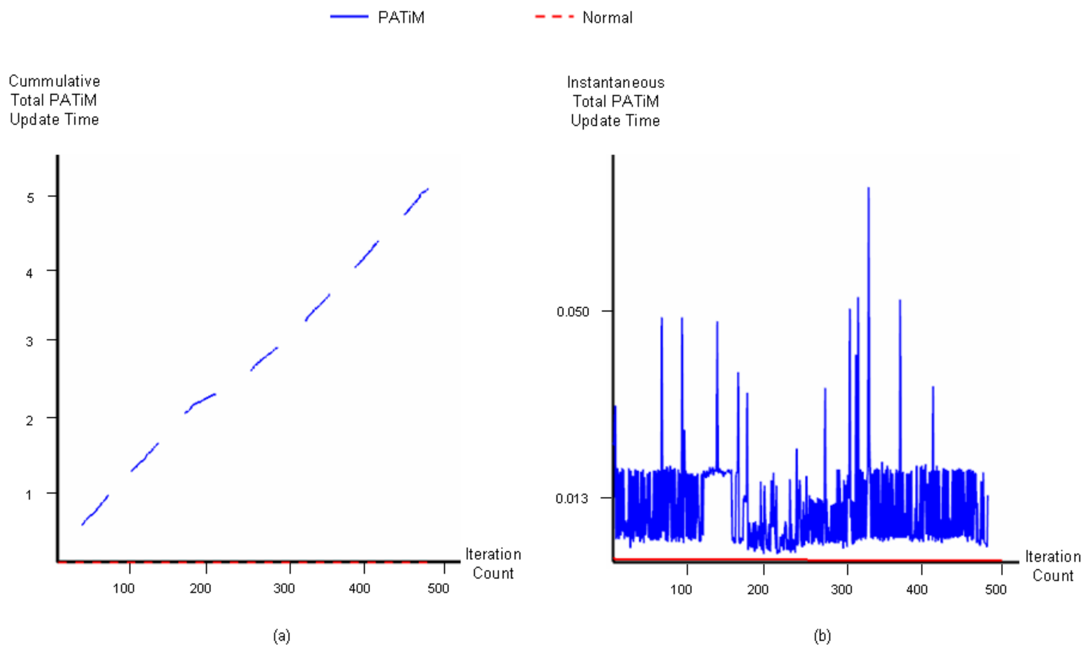


Figure 5.13: PATiM Update times for 7 federates

On the graphs in figure Figure 5.13, the execution times of the update method of the PATiM mechanism are shown. In part (a) of the figure, cumulative values are shown. In part (b), instantaneous values for the iterations are shown. As it can be seen from the graphs, the update time values for the normal time management mechanism is zero because there is not any clustering in this method. In the graph in part (b), there are some instant pick values in some iterations of the PATiM mechanism. In fact, execution times for the update method are very small. These small values are very sensitive to context switches that are caused by the operating system. These pick values are results of these context switches and can be omitted.

Overhead time for the dynamic cluster formation inside the update method includes different components: intra-cluster distance calculation and inter-cluster distance calculation. As mentioned in section 3.3, distance calculations are required to determine the relationships between the federates.

Inter-cluster distance calculation includes the determination of the position of each cluster in the virtual space and the sorting of these positions in order to find the distances between the clusters. The position of the cluster is found by taking the union of regions of individual federates in the cluster. For one federate, this operation has complexity of $O(k)$ where k is the number of the federates in the cluster. If all clusters are considered, total position finding operation complexity will be $O(n)$ where n is the number of federates in the federation. A modified Sweep-and-Prune algorithm is used to sort the calculated distances. Because of temporal coherence, the list will be nearly sorted and complexity of this sort operation will be $O(c)$ where c is the number of clusters. As c is a smaller value than n, , the total complexity of the inter-cluster distance calculation is $O(n)$. Another factor related to the federate clusters is the Approaching Speed (AS). Finding the AS value for two clusters is done by finding the distance change and the logical time change of these two clusters. Distance change is found using the current and previously sorted lists, which are already available.

Intra-cluster distance calculation includes sorting of positions of federates inside all clusters using the Sweep-and-Prune algorithm. If the average number of the federates inside a cluster is k, sorting for one cluster will has a complexity of $O(k)$. Having c number of clusters, total intra-cluster sorting complexity is $O(c.k)$ which is equal to $O(n)$.

To detect transitions from local synchronization periods to resynchronization periods, check operations should be performed for each cluster pair. These operations are controlling the distances and approaching speeds for all of the cluster pairs. Complexity of these checking operations is $O(k \log k)$ where k is the average number of clusters. In general, the number of clusters is very small when compared to the number of the federates. This fact leads that this $O(k \log k)$

complexity does not affect the overall overhead too much as the number of the federates increases.

The complexity of operations required when the algorithm detects that one cluster is divided into two clusters or two clusters are joined into one cluster, has constant values and does not affect the overall complexity of the dynamic cluster formation algorithm. In general, the overall complexity of dynamic cluster formation algorithm is $O(n) + O(k \log k)$ where n is the number of the federates and k is the average number of the clusters.

As mentioned before, the main component of the update method is the distance calculations, and these calculations are based on the modified Sweep-and-Prune algorithm. This algorithm is based on the idea of temporal coherence. This means that, in a simulation environment, locations of federates and therefore clusters are not changed dramatically. For this reason, previously sorted lists will be nearly the same in current sort operations. By keeping previous lists and using a quick sort algorithm, complexity of the sorting operation will be $O(n)$ in average. However, if the federates change their positions dramatically during the execution, complexity will become $O(n \log n)$, which is the worst case performance of the PATiM dynamic cluster formation algorithm. In fact, temporal coherence algorithm is an already required condition for the PATiM mechanism as if federates jump around virtual terrain randomly, this algorithm could not determine meaningful Approaching Speeds for the federate clusters. The physical simulations like the traffic simulations or war simulations are good examples, in which the federates do not change their locations dramatically in a single iteration of the simulation. This requirement of the PATiM mechanism makes the average complexity applicable in most of the iterations during the execution.

In Figure 5.14, there are PATiM update time graphs for an execution containing 20 federates. In part (a) and (b) of the figure, cumulative and individual update times of this execution are shown respectively. In part (c) of the figure, the graph of the federate cluster count of each iteration is shown. The first inference from this graph is that these update values are not much bigger than the ones for a federation containing 7 federates, given in Figure 5.13 (b). Another inference is that the update method execution times are very affected by the cluster count.

In the graph given in part (b), between the iterations i_1 and i_2, the cluster count decreases to 2 and the update execution time also decreases to smaller values, in comparison to values obtained with 3 clusters.

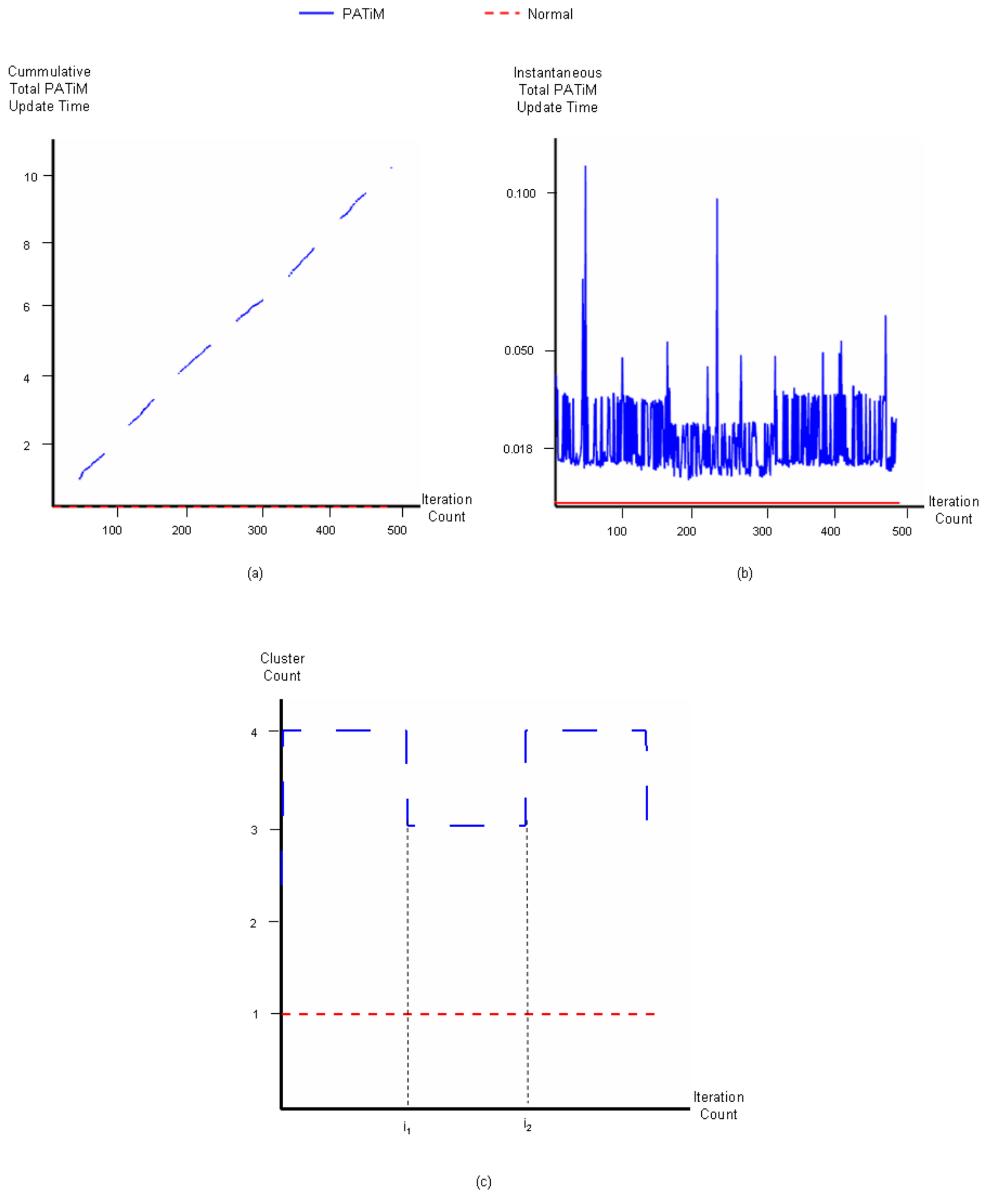


Figure 5.14: PATiM update times for 20 federates

In Figure 5.15, there are PATiM update time graphs for the execution containing 50 federates. In part (a) and (b) of figure, the cumulative and individual update times for this execution are shown. In part (c) of the figure, the graph of the federate cluster count in each iteration is shown. Execution time values for the update method are very close to the values of Figure 5.14 (b). This shows that increased federate counts do not affect the update method execution time significantly. However, as stated before, the number of clusters clearly affects the update execution time. The changes in cluster counts at iterations i_1, i_2, i_3 and i_4 directly changes update execution times.

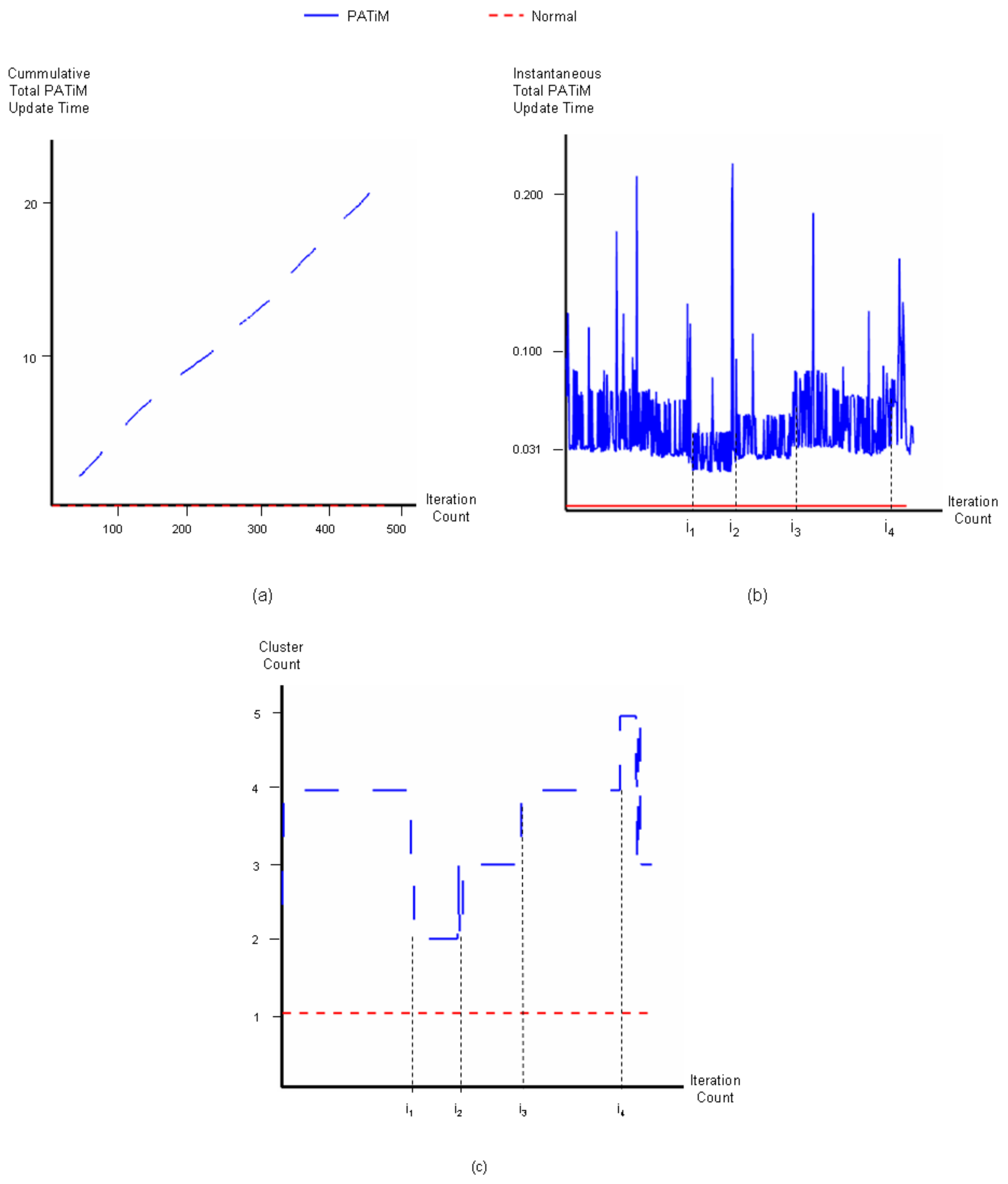


Figure 5.15: PATiM update times for 50 federates

Simulation Step Execution Time

In order to compare the overall execution time performance of the PATiM over the normal time management mechanism, execution times of simulation steps are analyzed. In part (a) of the Figure 5.16 step execution times are shown with respect to the iteration numbers. As it can be seen from the figures, these values are highly dependent on the cluster count. The main factor in determining the step execution time is the working iteration amount in that step, because the execution time increases when there are more working federates. It was mentioned previously that higher cluster counts result in increase in the working iteration amount (see Figure 5.9 and Figure 5.10).

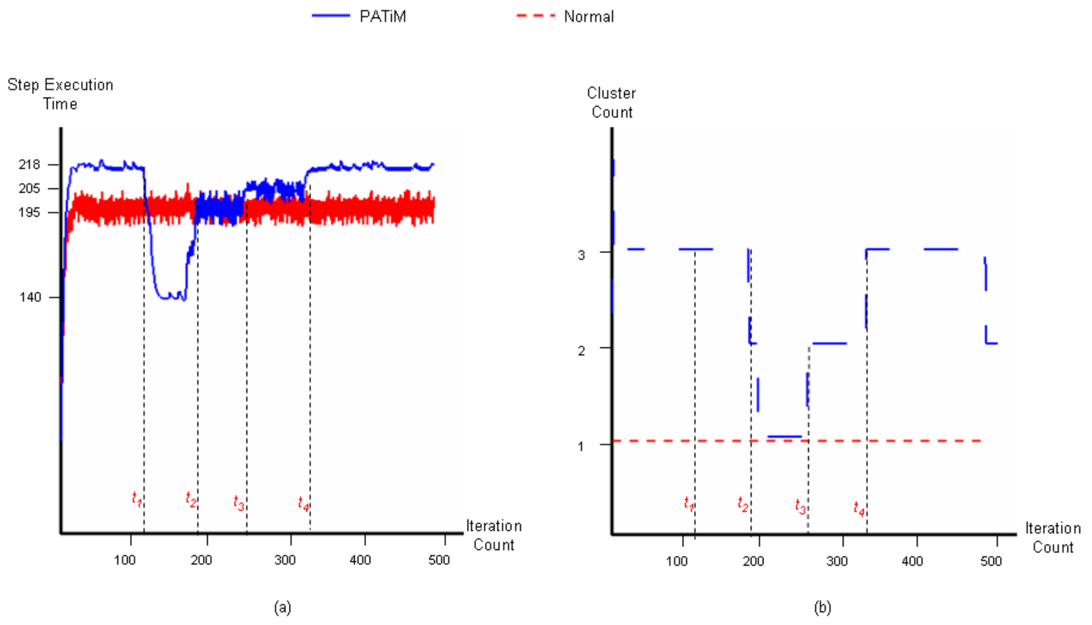


Figure 5.16: Step Execution Times

In these graphs, there are three clusters before t_1 and the PATiM generates higher execution times. Between t_1 and t_2 the federate clusters enter the resynchronization period and working iteration counts drop dramatically during this period because of the blocking of the faster federate. This results in relatively lower execution times for the PATiM. Between t_2 and t_3, there is only one cluster in the PATiM and execution times of both mechanisms are the same. After t_3 cluster counts and execution times increase again.

5.2 Transportation Simulation

Intelligent Vehicle/Highway Systems (IVHS) is aimed to increase the highway capacity and decrease the travel time without building new roads [68]. One promising strategy towards this goal is to organize traffic in platoons of closely spaced vehicles. The design and implementation of the control tasks needed to realize such an IVHS system will require a structured approach that uses control, communication and computing technologies both to maintain the position and the speed of a vehicle within a platoon and to coordinate platoon maneuvers.

The control tasks are arranged in a three layer hierarchy as shown in Figure 5.17. There is a single link layer for a long segment of the highway that extends to several sections. Each section may be between 50m and 500m long. The link layer has two functions. It assigns a path to each vehicle that enters the highway and continuously determines the platoon optimal speed (denoted as *optspeed*) and the optimal size (*optsize*) for each highway section. The values of *optspeed* and *optsize* are selected to maintain smooth traffic flow and to reduce congestion. The link layer functions are proposed to be implemented in a centralized manner [29, 3].

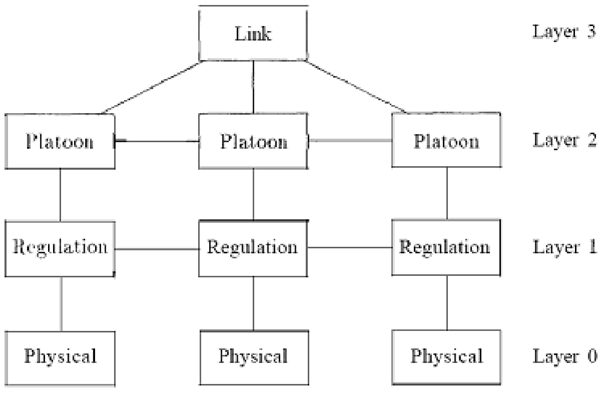


Figure 5.17: Control Hierarchy

The remaining control tasks are proposed to be implemented in a distributed manner. There is one platoon layer, one regulation layer, and one physical layer per vehicle. Each platoon layer of a vehicle plans a sequence of maneuvers and issues that correspond commands to the regulation layer, so that the trajectory of the vehicle follows the assigned path closely, and the platoon speed and size track the optspeed and optsize values. Each regulation layer of a vehicle executes the commands issued by its platoon layer by implementing corresponding pre-computed feedback control laws, which continuously determine the throttle, braking, and steering actions of the vehicle. Finally, the physical layer of a vehicle is a model of its dynamic behavior, against which the feedback control laws are designed.

Traffic is organized in platoons of vehicles in IVHS as shown in Figure 5.18. The size of a platoon is between 1 and 20, depending on the traffic flow. The headway within a platoon is small (about 1 m); the minimum headway between platoons grows with the platoon size; reaching about 60 m for platoons of size 20. The lead vehicle of a platoon is called its leader, and the rest are the followers. A single vehicle platoon is called a free agent. Protocol exchanges are always between leaders (including free agents) of neighbor platoons. If a follower wants to initiate a maneuver, it must send a request to its leader. It is further the task of the leader to track *optspeed* and *optsize*. The task of the followers is only to execute a feedback control law, which maintains the tight headway with the vehicle that is in front of it.

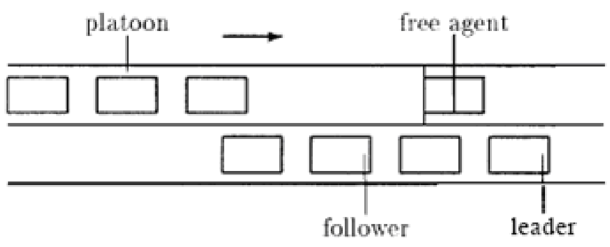


Figure 5.18: Definition of Platoon

The link layer is not implemented in this distributed simulation of IVHS because of its centralized nature. It collects all the information from all over the highway system to calculate the optimum speed and platoon size for the current traffic conditions. However, this continuous communication requirement does not allow the clustering in the PATiM. In fact, *optspeed* and *optsize* parameters can be taken as fixed values for different simulation times; for example, for week days and weekends. Additionally, the path assigning job of the link layer can be performed by the leader vehicles according to the current road conditions and the target locations.

The main focus of this example simulation is the platoon layer. During the course of simulation, each platoon layer continuously checks its internal state and the environment in order to keep its platoon size and speed close to optsize and optspeed. For this purpose, it uses three elementary maneuvers named as merge, split and change lane. The merge maneuver combines two successive platoons in the same lane into a single platoon, as shown in Figure 5.19. The merge is always initiated by the leader of the rear platoon, vehicle B. If the size of the platoon of B (*ownsize(B)*), is smaller than the *optsize*, B requests permission from A to merge. If A is not busy, and if this permission is granted, the platoon layer of B then requests from its regulation layer to accelerate and join the platoon of A.

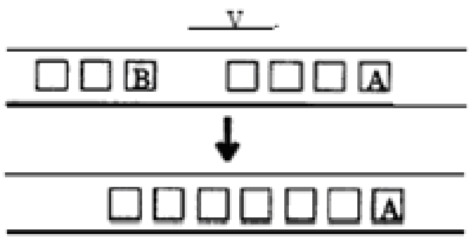


Figure 5.19: Merge

A split maneuver may be needed because a the size of a platoon may exceed the optsize, a vehicle in an adjacent lane requests a change lane maneuver, or a vehicle in a platoon initiates one or two splits in order to become a free agent.

As indicated in Figure 5.20, a split may be initiated either by a leader (vehicle A) or by a follower (vehicle B).

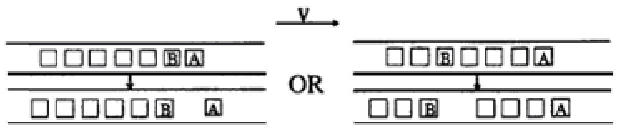


Figure 5.20: Split

Lane change maneuver can be initiated only by a free agent, i.e. a single vehicle platoon. If a vehicle in a multi-vehicle platoon needs to change lanes, it must first gain a free agent status by executing one split (if it is a leader) or two split maneuvers (if it is a follower).

For each vehicle of the simulation, a federate is implemented that contains platoon, vehicle and physical layers of the control hierarchy. Each vehicle has a group of state information containing road identifiers, position of vehicle, belonging platoon number and a busy flag. It publishes state information to other vehicles. Communication requirements of platoon layers, for example join requests, are implemented using the interactions of HLA.

5.2.1 Simulation Implementation and Results

IHVS simulation is realized by implementing each vehicle actor by a federate that contains a platoon and a regulation layer. The simulation program reads the highway structure and the vehicle properties from a configuration file and initializes the vehicle federates. The structure of the example implementation is shown in Figure 5.21. Each vehicle is implemented as a class inside the application process. Platoon and regulation layer is implemented inside the vehicle federate. There is no direct connection between federates and they only communicate through HLA services.

There are two critical parameters of the PATiM, which should be determined for each simulation application. The first one is the amount of clustering threshold

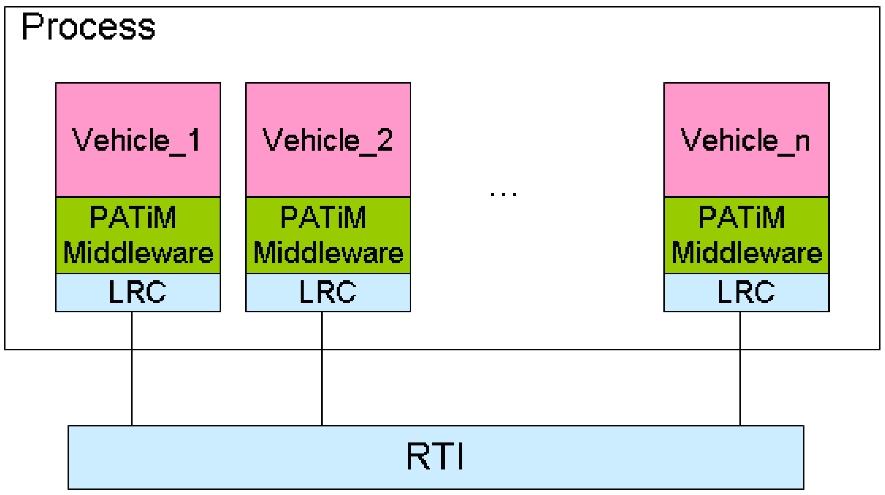
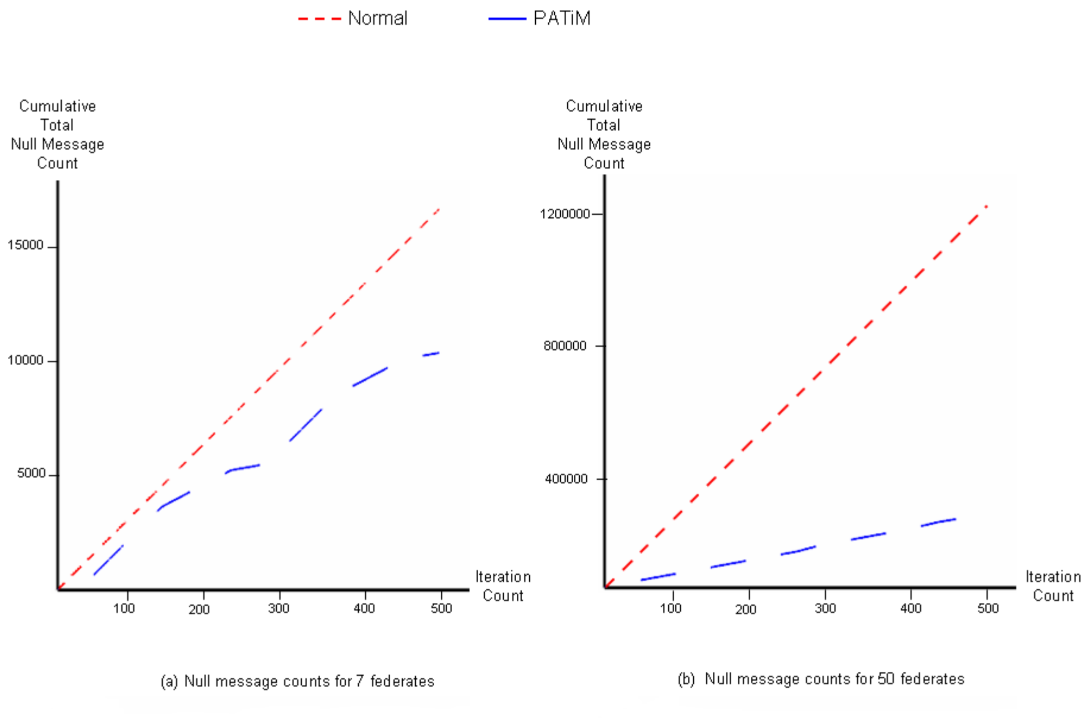


Figure 5.21: Example Structure

of the PATiM for this IVHS simulation. The critical information here is the optimal distance between two successive platoons. As mentioned before, this value should be 60 meters in most cases. This means that the leader vehicle should be able to monitor at least 60 meters distance. Further, the value of breaking the distance should be added to this value, because a vehicle should appropriately respond to an unexpected obstacle that appears in a 60 meters distance. This value is calculated 40 meters for a vehicle of maximum speed. A safety factor of 20 meters is further added in order to eliminate the ping-pong effect of clustering. The resulting threshold value is 120 meters. The second parameter is the maximum speed value used in the calculation of the approaching speed. In the traffic environment, maximum speed of vehicles can be accepted as 250 km/h in normal highways.

The null-message count, the LBTS calculation time and the PATiM update time comparisons are given in following figures. The graphs of Figure 5.22 show null-message counts of the normal and the PATiM time management algorithms. The parts (a), (b) and (c) are for simulations containing 7, 50 and 100 federates, respectively. As it can be seen from the figure, the number of null-messages generated by the PATiM is significantly smaller than those generated by the normal mechanism.

The graphs of Figure 5.23 show the LBTS calculation times of the normal



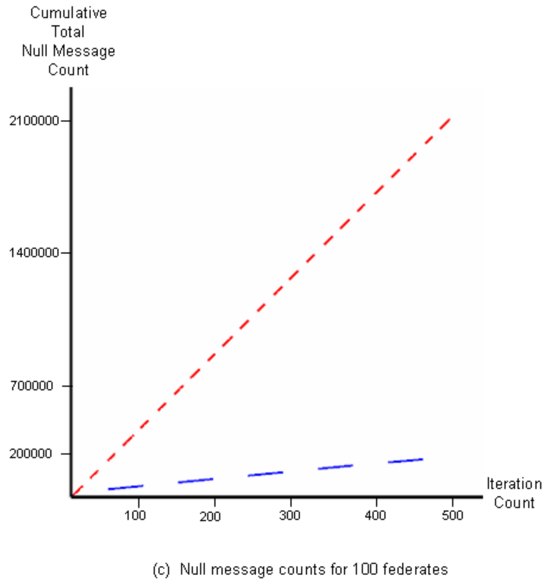


Figure 5.22: Null-message comparisons

and the PATiM time management algorithms. The parts (a), (b) and (c) are for simulations containing 7, 50 and 100 federates, respectively. As it can be seen from these graphs, PATiM LBTS calculation times are relatively less than those of normal time management mechanism.

Comparisons of PATiM update times for the PATiM time management mech-

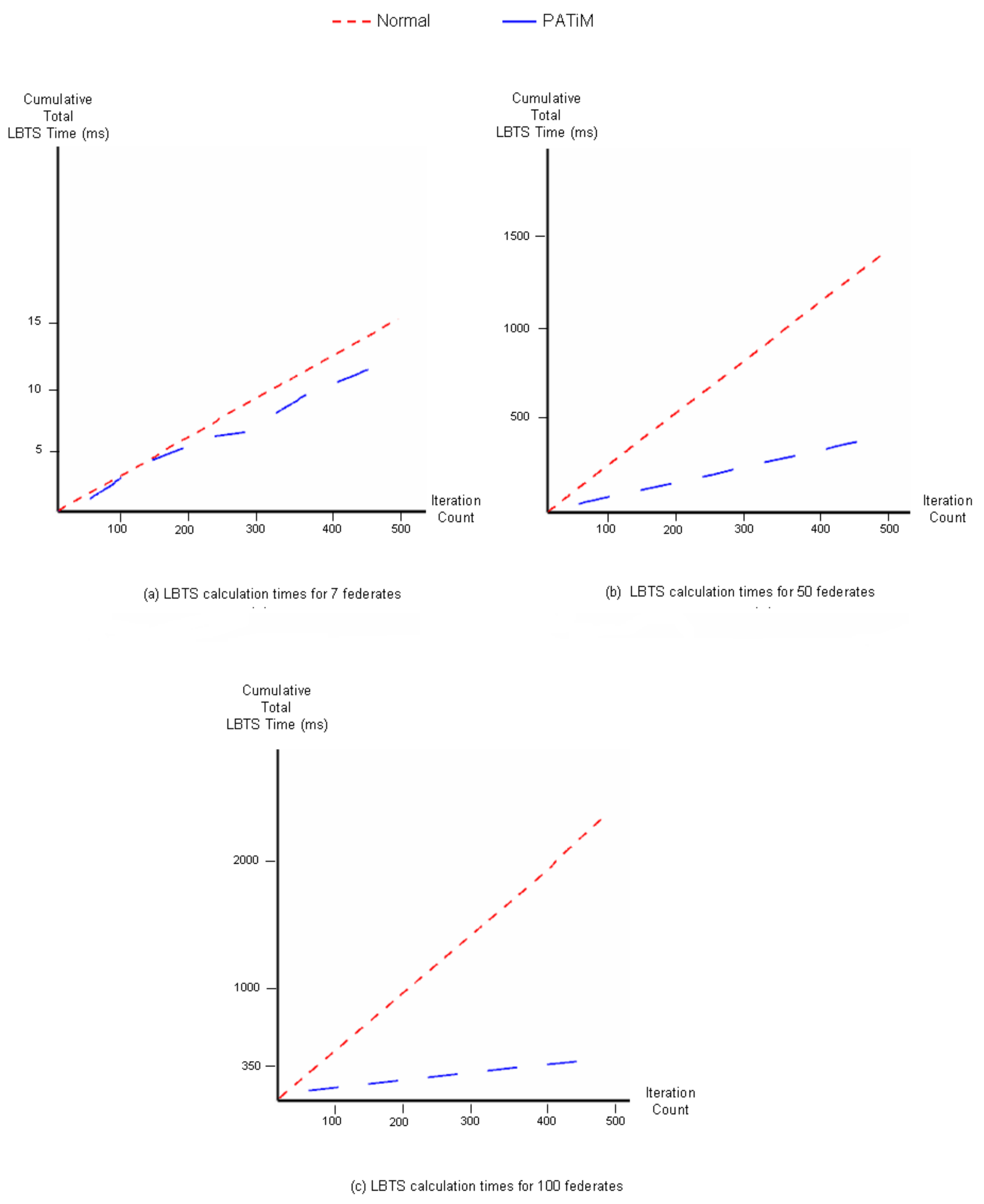


Figure 5.23: LBTS Calculation time comparison

anisms are given in Figure 5.24 for simulations containing 7, 50 and 100 federates. Update times for normal time management mechanism are not given because they are all zero. Plots show that the increased federate count and update overhead are linearly dependent.

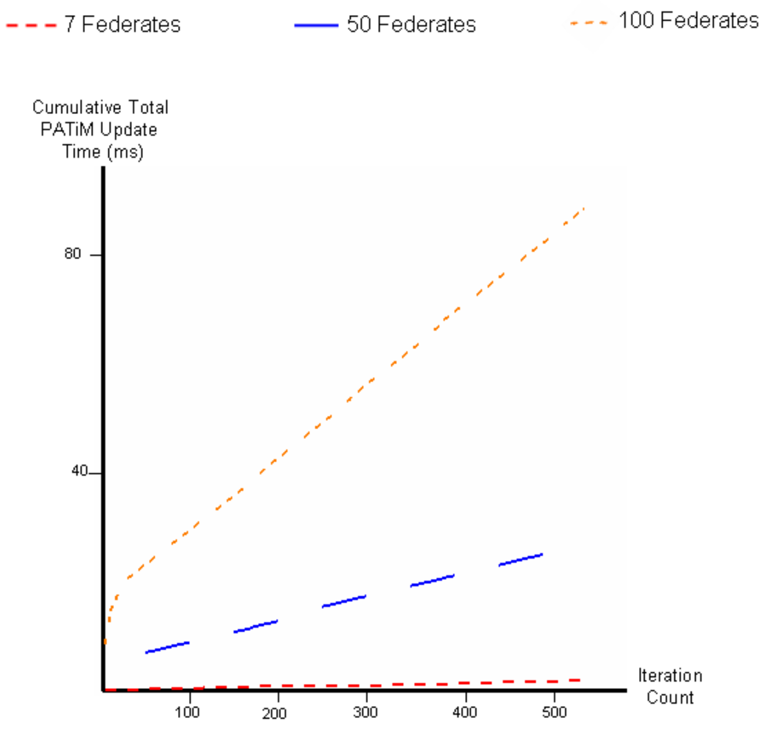


Figure 5.24: PATiM update time comparison

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

6.1 Summary

The High Level Architecture (HLA) provides a specification of a common technical architecture for use with a wide range of distributed simulation environments. Its main purpose is to support interoperability among different simulation systems, which is a similar requirement for distributed interactive simulation systems. Time management services are one of the most expensive services in the HLA, especially when there are enormous amount of federates in the federation. Improving the performance of the time management services is very critical in the distributed simulation domain.

In this study, a new mechanism to manage the logical time of federations by dynamically clustering federates is presented. The proposed PATiM mechanism is composed of two main parts, dynamic clustering and its time management. The aim of the clustering mechanism is to partition the entire simulation into federate clusters which contain proximity based logically related federates. The new term "proximity" for federates is represented by the distance between them within the virtual space. The key support to measure the proximity is given by the Data Distribution Management services of the HLA specifications, through subscribe and update regions. These regions are used to determine the position of the federates which can be used to measure the distance between the federates.

In PATiM, a relationship between time management and data distribution management is constructed in order to manage the logical times of federate clusters. This relationship is based on the idea that when a group of federates are distant to each other, there will be no interaction between them as long as they remain distant. Thus, there is no need to synchronize the logical times of such clusters as long as they are distant. These temporal logical time inconsistencies between clusters are utilized in order to increase the performance of the time management algorithm. In the proposed method, logical time dependencies between federate clusters are broken when they are distant, as there will not be any interaction during this period, thus there will not be any logical anomaly between them.

Logical times of federate clusters should be resynchronized when they become closer. For this purpose, PATiM runs a resynchronization period when it detects that the previously distant federate clusters are getting closer. After the resynchronization, these clusters are joined to form a single cluster.

In this study, a middleware approach is used to implement PATiM mechanism. The HLA 1.3 implementation, called RTI NG, is used as the HLA API. Otherwise, a new PATiM friendly RTI would need to be developed. At the moment, this is found not feasible.

6.2 Concluding Remarks

PATiM mechanism divides the federation into federate groups with a dynamic clustering algorithm. Breaking logical time dependencies between the federate clusters allows a group of performance improvements in the time management algorithms. These improvements can be listed as:

- Total number of exchanged synchronization messages,
- LBTS calculation overhead,
- Concurrency level

PATiM shows that the total synchronization related message exchanges are reduced by dividing the federation into federate clusters and breaking the logical time dependencies between these clusters. Another improvement is the reduction in execution time overhead of the LBTS calculation because this calculation is done on a smaller set. This improvement is applicable to both our distributed time management implementation and possible central implementation. Gains from these improvements are exponential in number of federates and also increase with higher number of clusters.

The concurrency of a distributed simulation is increased because the logical time dependency between clusters is broken when they are distant. The increase in the concurrency will be applicable only if there is a clustering in the federation so that the members of these clusters have different execution speeds from the members of other clusters. For example, consider a federation divided into two clusters. One of them contains only faster federates and the other one contains only slower federates. In PATiM mechanism, members of the faster cluster will not be blocked by slower clusters and the concurrency of the federation is increased in total compared to the concurrency in normal time management mechanism. However, if the cluster is formed so that both clusters have some faster and some slower federates, the overall concurrency of the federation will not change because faster clusters are again blocked by slower clusters, which is also shown by the example implementation.

Overhead of PATiM is the clustering calculations including measuring distances between the federates and the clusters. Complexity of intra-cluster distance calculations to detect cluster divisions is *O(n)* where n is the number of the federates. The complexity of the inter-cluster distance calculations performed to detect the start time of the resynchronization period and the cluster joining is $O(k \log k)$, where k is the average number of clusters. These overheads are relatively small in comparison to the advantages gained in the LBTS calculation time and the reduction in the total amount of synchronization related messages.

The PATiM mechanism is based on the detection of the approaching and drifting away federates. In order to detect such intervals, the simulation should use a time-stepped structure where the federate will not jump in the virtual space. Also actors in the simulation should spread away in the virtual space. Otherwise, all federates are always close to each other and there will not be any clustering.

In this work, we mainly focus on spatial simulation examples, methodology and performance metric analysis for other types is left as a future work.

In conclusion, the results show that the PATiM approach performs well in heterogeneous federation cases. When the federates in simulation have varying execution speeds and they spread around the virtual space, advantages of the PATiM become obvious. In a homogeneous federation, PATiM may not generate any clustering and it works as normal time management mechanism.

6.3 Future Directions

PATiM is currently designed in order to increase the performance of time management services in a federation. This mechanism can be applied to federation communities by modifying the federation connection components, especially when different federations become distant in the community. In this case, reduction in the synchronization related messages are much more critical because different federates are possibly on different LANs, and the communication between them is more costly.

Another current research topic is the ad hoc distributed simulations. Such a simulation is a collection of autonomous online simulations brought together to model an operational system [27]. In the PATiM mechanism, dynamic clustering algorithm can be applied in an ad hoc environment in order to manage possible communication availabilities and breakdowns. The proposed proximity relation can be used to construct the bottom-up structure of ad-hoc simulations.

REFERENCES

[1] IEEE standard for distributed interactive simulation. Technical Report IEEE Std 1278.2-1995, Communication Services and Profiles, 1995.

[2] High level architecture (HLA)—framework and rules. Technical Report IEEE Standard No.: 1516-2000, IEEE Standard for modeling and simulation, 2000.

[3] Sonia Sachs Ann Hsu, Farokh Eskafi and Pravin Varaiya. Technical report.

[4] Rassul Ayani, Farshad Moradi, and Gary Tan. Optimizing cell-size in grid-based DDM. In *PADS '00: Proceedings of the fourteenth workshop on Parallel and distributed simulation*, pages 93–100, Washington, DC, USA, 2000. IEEE Computer Society.

[5] Roberto Beraldi, Libero Nigro, Antonino Orlando, and Francesco Pupo. Temporal uncertainty time warp: An agent-based implementation. In *SS '02: Proceedings of the 35th Annual Simulation Symposium*, page 72, Washington, DC, USA, 2002. IEEE Computer Society.

[6] M. Schenk M. Schumann Bluemel, E. Distributed virtual worlds with HLA. In *Proceedings of 2002 Fall Simulation Interoperability Workshop, Paper 02F-SIW-031*, 2002.

[7] A. Boukerche. Time management in parallel simulations. *High Performance Cluster Computing*, 2:375–394, 1999.

[8] A. Boukerche and C. Dzermajko. Performance comparison of data distribution management strategies. In *Proceedings of the 5th IEEE International Workshop on Distributed Simulation and RealTime Applications.*

[9] Azzedine Boukerche and Sajal K. Das. Dynamic load balancing strategies for conservative parallel simulations. *ACM SIGSIM Simulation Digest*, 27(1):20–28, 1997.

[10] Azzedine Boukerche and Sajal K. Das. Reducing null messages overhead through load balancing in conservative distributed simulation systems. *Journal of Parallel and Distributed Computing*, 64(3):330–344, 2004.

[11] Wentong Cai, Francis B. S. Lee, and L. Chen. An auto-adaptive dead reckoning algorithm for distributed interactive simulation. In *PADS '99: Proceedings of the thirteenth workshop on Parallel and distributed simulation*, pages 82–89, Washington, DC, USA, 1999. IEEE Computer Society.

[12] Wentong Cai, Stephen J. Turner, and Boon Ping Gan. Hierarchical federations: an architecture for information hiding. In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, pages 67–74, Washington, DC, USA, 2001. IEEE Computer Society.

[13] Wentong Cai, Stephen J. Turner, Bu-Sung Lee, and Junlan Zhou. An alternative time management mechanism for distributed simulations. *ACM Transaction on Modeling and Computer Simulation*, 15(2):109–137, 2005.

[14] Wentong Cai, Percival Xavier, Stephen J. Turner, and Bu-Sung Lee. A scalable architecture for supporting interactive games on the internet. In *PADS '02: Proceedings of the sixteenth workshop on Parallel and distributed simulation*, pages 60–67, Washington, DC, USA, 2002. IEEE Computer Society.

[15] Christopher D. Carothers, Richard Fujimoto, Richard M. Weatherly, and Annette L. Wilson. Design and implementation of HLA time management in the RTI version f.0. In *Winter Simulation Conference*, pages 373–380, 1997.

[16] Christopher D. Carothers, Kalyan S. Perumalla, and Richard M. Fujimoto. Efficient optimistic parallel simulations using reverse computation. *ACM Transactions on Modeling and Computer Simulations*, 9(3):224–253, 1999.

[17] K. M. Chandy and J. Misra. Distributed simulation a case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, 1979.

[18] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(4):198–206, 1981.

[19] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav Ponamgi. I-COLLIDE: an interactive and exact collision detection system for large-scale environments. In *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–ff., New York, NY, USA, 1995. ACM.

[20] J.P. Oudshoorn M.J. Cramp, A. Best. Time management in hierarchical federation communities. In *Proceedings of 2002 Fall Simulation Interoperability Workshop, Paper 02F-SIW-031*, 2002.

[21] Judith S. Dahmann, Richard Fujimoto, and Richard M. Weatherly. The department of defense high level architecture. In *Winter Simulation Conference*, pages 142–149, 1997.

[22] Om. P. Damani and Vijay K. Garg. Fault-tolerant distributed simulation. *ACM SIGSIM Simulation Digest*, 28(1):38–45, 1998.

[23] R. Fujimoto. Time management in the high level architecture. *SIMULATION Special Issue on High Level Architecture*, 71(6):388–400, 1998.

[24] R. Fujimoto and R. Weatherly. HLA time management and DIS. In *14th Workshop on Distributed Interactive Simulation*, 1995.

[25] Richard Fujimoto. Exploiting temporal uncertainty in parallel and distributed simulations. In *Workshop on Parallel and Distributed Simulation*, pages 46–53, 1999.

[26] Richard Fujimoto. Parallel and distributed simulation. In *Winter Simulation Conference*, pages 122–131, 1999.

[27] Richard Fujimoto, Michael Hunter, Jason Sirichoke, Mahesh Palekar, Hoe Kim, and Wonho Suh. Ad Hoc distributed simulations. In *PADS '07: Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation*, pages 15–24, Washington, DC, USA, 2007. IEEE Computer Society.

[28] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems.* Wiley-Interscience, 1996.

[29] Datta N. Godbole and John Lygeros. Longitudinal control of the lead car of a platoon. *IEEE Transactions on Vehicular Technology*, 43(4):1125–1135, 1994.

[30] Len Granowetter. RTI interoperability issues: API standards, wire standards, and RTI bridges. In *Proceedings of 2003 European Simulation Interoperability Workshop*, 2003.

[31] D. Van Hook, J. Calvin, M. Newton, and D. Fusco. An approach to DIS scaleability. In *11th Workshop on Standards for the Interoperability of Distributed Simulations*, pages 347–356, 1994.

[32] D. Van Hook and J. O. Calvin. Data distribution management in RTI 1.3. In *Proceedings of 1998 Spring Simulation Interoperability Workshop, Paper 98S-SIW-206*, 1998.

[33] Jiung-Yao Huang, Ming-Chih Tung, Lin Hui, and Ming-Che Lee. An approach for the unified time management mechanism for HLA. *Simulation*, 81(1):45–56, 2005.

[34] Jack Hung, Mark Torpey, and Wayne Civinskas. Performance cost of using time management services. In *Proceedings of 2002 Spring Simulation Interoperability Workshop, Paper 02S-SIW-071*, 2002.

[35] Jorn W. Janneck. Generalizing lookahead behavioral prediction in distributed simulation. *ACM SIGSIM Simulation Digest*, 28(1):12–19, 1998.

[36] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[37] Vikas Jha and Rajive Bagrodia. Simultaneous events and lookahead in simulation protocols. *ACM Transactions on Modeling and Computer Simulation*, 10(3):241–267, 2000.

[38] Yu Jun, Come Raczy, and Gary Tan. Evaluation of a sort-based matching algorithm for DDM. In *PADS '02: Proceedings of the sixteenth workshop on Parallel and distributed simulation*, pages 68–75, Washington, DC, USA, 2002. IEEE Computer Society.

[39] Ajay D. Kshemkalyani and Mukesh Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distributed Computing*, 11(2):91–111, 1998.

[40] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[41] Bu-Sung Lee, Wentong Cai, and Junlan Zhou. A causality based time management mechanism for federated simulation. In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, pages 83–90, Washington, DC, USA, 2001. IEEE Computer Society.

[42] Jong S. Lee and Bernard P. Zeigler. Space-based communication data management in scalable distributed simulation. *J. Parallel Distrib. Comput.*, 62(3):336–365, 2002.

[43] W.T.Cai B.S.Lee G.Y.Li L.Liu, S.J.Turner. DDM implementation in hierarchical federations. In *Proceedings of 2002 Fall Simulation Interoperability Workshop, Paper 02F-SIW-033*, 2002.

[44] David L. Mills. A brief history of NTP time: memoirs of an internet timekeeper. *ACM SIGCOMM Computer Communication Review*, 33(2):9–21, 2003.

[45] K. Morse, L. Bic, M. Dillencourt, and K. Tsai. Multicast grouping for dynamic data distribution management. In *Proceedings of the 31st Society for Computer Simulation Conference (SCSC '99)*, 1999.

[46] K. L. Morse and J. S. Steinman. Data distribution management in the HLA. In *Proceedings of the 1997 Spring Simulation Interoperability Workshop*, 1997.

[47] Clark D. Myjak, M. and Lake. RTI interoperability study group final report. In *Proceedings of 1999 Fall Simulation Interoperability Workshop*, 1999.

[48] M.D. Myjak and S.T Sharp. Implementations of hierarchical federations. In *Proceedings of 1999 Fall Simulation Interoperability Workshop, Paper 99F-SIW-180*, 1999.

[49] David M. Nicol and Richard M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, (53):249–285, 1994.

[50] Aydin Okutanoglu and Muslim Bozyigit. Proximity-aware synchronization within federation communities. In *DS-RT '06: Proceedings of the 10th IEEE international symposium on Distributed Simulation and Real-Time Applications*, pages 185–192, Washington, DC, USA, 2006. IEEE Computer Society.

[51] Aydin Okutanoglu and Muslim Bozyigit. Time management in dynamically clustered federation communities. In *Proceedings of 2006 Fall Simulation Interoperability Workshop*, 2006.

[52] Kalyan S. Perumalla. Parallel and distributed simulation: traditional techniques and recent advances. In *WSC '06: Proceedings of the 38th conference on Winter simulation*, pages 84–95. Winter Simulation Conference, 2006.

[53] M. D. Petty and K. L. Morse. Computational complexity of HLA data distribution management. In *Proceedings of the 2000 Fall Simulation Interoperability Workshop*, 2000.

[54] J. Porras, V. Hara, J. Harju, and J. Ikonen. Improving the performance of the chandy-misra parallel simulation algorithm in a distributed workstation environment. In *Proceedings of the SCSC'97*, pages 657–662, 1997.

[55] Jari Porras, Jouni Ikonen, and Jarmo Harju. Applying a modified Chandy-Misra algorithm to the distributed simulation of a cellular network. In *Workshop on Parallel and Distributed Simulation*, pages 188–195, 1998.

[56] B. Preiss, W. Loucks, and I. Macintyre. Null message cancellation in conservative distributed simulation. In *Proceedings of the SCS Multiconference on Advances in Parallel and Distributed Simulation*, pages 33–38, 1991.

[57] Francesco Quaglia. A scaled version of the elastic time algorithm. In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, pages 157–164, Washington, DC, USA, 2001. IEEE Computer Society.

[58] C. Raczy, G. Tan, and J. Yu. A sort-based DDM matching algorithm for HLA. *ACM Transactions on Modeling and Computer Simulation*, 15(1):14–38, 2005.

[59] Yu J. Tan G. Tay S. C. Raczy, C. and R Ayani. Adaptive data distribution management for HLA RTI. In *Proceedings of 2002 European Simulation Interoperability Workshop, Paper 02E-SIW-043*, 2002.

[60] George F. Riley, Richard Fujimoto, and Mostafa H. Ammar. Network aware time management and event distribution. In *Workshop on Parallel and Distributed Simulation*, pages 119–126, 2000.

[61] Bradley C. Schricker and Sonia R. von der Lippe. Using the high level architecture to implement selective-fidelity. In *ANSS '04: Proceedings of the 37th annual symposium on Simulation*, page 246, Washington, DC, USA, 2004. IEEE Computer Society.

[62] Reinhard Schwarz and Friedemann Mattern. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing*, 7(3):149–174, 1994.

[63] Jeffrey S. Steinman, Craig A. Lee, Linda F. Wilson, and David M. Nicol. Global virtual time and distributed synchronization. *ACM SIGSIM Simulation Digest*, 25(1):139–148, 1995.

[64] I. Tacic and R. Fujimote. Synchronized data distribution management in distributed simulation. In *Proceedings of the 1997 Spring Simulation Interoperability Workshop*, 1997.

[65] Gary Tan, Rassul Ayani, YuSong Zhang, and Farshad Moradi. Grid-based data management in distributed simulation. In *SS '00: Proceedings of the 33rd Annual Simulation Symposium*, page 7, Washington, DC, USA, 2000. IEEE Computer Society.

[66] Georgios Theodoropoulos and Brian Logan. An approach to interest management and dynamic load balancing in distributed simulation. In *Proceedings of the 2001 European Simulation Interoperability Workshop (ESIW'01)*, pages 565–571, Harrow, London, UK, June 2001. Simulation Interoperability Standards Organisation and Society for Computer Simulation.

[67] Neville Thomas. Dynamic grid-based multicast group assignment in data distribution management. In *DS-RT '00: Proceedings of the Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, page 47, Washington, DC, USA, 2000. IEEE Computer Society.

[68] P. Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.

[69] Ronald C. De Vries. Reducing null messages in Misra's distributed discrete event simulation method. *IEEE Transactions on Software Engineering*, 16(1):82–91, 1990.

[70] Xiaoguang Wang, Stephen John Turner, Malcolm Yoke Hean Low, and Boon Ping Gan. Optimistic synchronization in HLA-based distributed simulation. *Simulation*, 81(4):279–291, 2005.

[71] D.D. Wood. Implementation of DDM in the MAK high performance RTI. In *Proceedings of 2002 Spring Simulation Interoperability Workshop, Paper02S-SIW-056*, 2002.

[72] D.D. Wood. Developing the fault tolerance support extensions for HLA evolved. In *Proceedings of 2005 European Simulation Interoperability Workshop, Paper 05E-SIW-019*, 2005.

[73] Stephen Zabele. Interest management using an active networks approach. In *Proceedings of 2000 Spring Simulation Interoperability Workshop, Paper 00S-SIW-030*, 2000.

[74] Suiping Zhou, Wentong Cai, Stephen J. Turner, and Francis B. S. Lee. Critical causality in distributed virtual environments. In *PADS '02: Proceedings of the sixteenth workshop on Parallel and distributed simulation*, pages 53–59, Washington, DC, USA, 2002. IEEE Computer Society.

[75] Suiping Zhou, Wentong Cai, Stephen J. Turner, Junhu Wei, and Wenbo Zong. Flexible state update mechanism for large-scale distributed wargame simulations. *Simulation*, 83(10):707–719, 2007.

VITA

Personal Information

Surname, Name: Okutanoğlu, Aydın

Nationality: Turkish (TC)

Date and Place of Birth: 26 December 1977, Aydın

Marital Status: Maried

Phone: +90 312 2979058

email: aokutanoglu@meteksan.com.tr

Education

Computer Engineering Department, METU, MS Education, 1999-2001.

Computer Engineering Department, METU, BS Education, 1994-1999.

Aydın Lisesi, High School Education, 1991-1994.

Work Experience

Meteksan Sistem A.Ş. Simulation and Visual Systems Group, Served as software engineer. Worked on especially server side technologies and simulation technologies. From February 2001 to now.

Dept. of Computer Engineering at METU, Served as system administrator of department. From June 1999 to February 2001.

Publications

- Aydın Okutanoğlu and Müslim Bozyiğit. "Proximity-aware synchronization within federation communities". In DS-RT '06: Proceedings of the 10th IEEE international symposium on Distributed Simulation and Real-Time Applications, pages 185-192, Washington, DC, USA, 2006. IEEE Computer Society.
- Aydın Okutanoğlu and Müslim Bozyiğit. "Time management in dynamically clustered federation communities". In Proceedings of 2006 Fall Simulation Interoperability Workshop, 2006.