uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Kaiyuan Lu

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

Master of Computer Science

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Data Distribution Management Schemes for HLA-compliant Distributed Simulation Systems

TITRE DE LA THÈSE / TITLE OF THESIS

Azzedine Boukerche

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Wail Gueaieb

Dorina Petriu

Gary W. Slater

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES /
DEAN OF THE FACULTY OF GRADUATE AND POSTDOCORAL STUDIES

Data Distribution Management Schemes for HLA-compliant Distributed Simulation Systems

By

Kaiyuan Lu

A Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of

The requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institution for Computer Science

School of Information Technology and Engineering

University of Ottawa

Your file Votre référence
ISBN: 0-494-14924-8
Our file Notre référence
ISBN: 0-494-14924-8

The following publications by the author are relevant to this thesis:

Journal

- Azzedine Boukerche, Kaiyuan Lu. "An Enhancement Towards Dynamic Grid-based DDM Protocol Using Multiple Levels of Data Filtering". *Journal of Parallele Computing*, Elsevier. (To appear in 2006).

Conference

- Azzedine Boukerche, Kaiyuan Lu. "Optimized Dynamic Grid-Based DDM Protocol for Large-Scale Distributed Simulation Systems". In *Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium(IPDPS 2005)*, 4-8 April 2005, Denver, CO, USA.

- Azzedine Boukerche, Nathan J. McGraw, Caron Dzermajko, Kaiyuan Lu. "Grid-filtered Region-based Data Distribution Management in Large-Scale Distributed Simulation Systems". In *Proceedings of 38th Annual Simulation Symposium (ANSS-38 2005)*, pages 259-266, 4-6 April 2005, San Diego, CA, USA.

- Azzedine Boukerche, Kaiyuan Lu. "A Novel Approach to Real-Time RTI Based Distributed Simulation System", In *Proceedings of 38th Annual Simulation Symposium (ANSS-38 2005)*, pages 267-274, 4-6 April 2005, San Diego, CA, USA.

- Azzedine Boukerche, Kaiyuan Lu. "Design and Performance Evaluation of a Real-Time RTI Infrastructure for Large-Scale Distributed Simulations", In *Proceedings of 9th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2005)*, pages 203-212, 10-12 October 2005, Montreal, Canada.

Abstract

Data Distribution Management (DDM), one of the six services provided by High Level Architecture and Run-Time Infrastructure, provides an efficient and scalable mechanism for data routing among hosts in distributed simulations. Traditional, DDM schemes are classified into two main types, region-based methods and grid-based methods. Currently, the time, computation and communication overhead of DDMs are still issues for large-scale simulations. We proposed two new DDM schemes addressing these issues. Our first algorithm, which we refer to as optimized dynamic grid-based DDM scheme, aims at further reducing irrelevant data that might be received by simulation entities in dynamic grid-based approach [11], by enforcing a second level of sender-side data filtering mechanism. Our second algorithm, which we refer to as grid-filtered region-based DDM, uses a threshold value of coverage percentage to determines if exact matching is necessary. In this thesis, we present and discuss the implementation of our proposed DDM algorithms, and report on their performance based on an extensive set of simulation experiments. Last but not least, we present the preliminary work we have done on real-time enabling scheme to RTI for HLA-compliant simulations.

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Azzedine Boukerche for his confidence in my abilities, his encouragement and precious comments that he shared with me during my research and writing this thesis. He has been always more than generous in both his time and his knowledge.

I would like to thank for Nathan J. McGraw and Caron Dzermajko for their collaboration with the work of Chapter 5.

Finally, I would also like to thank my parents for their unconditional support and constant encouragement. I will never forget the inspiring words they gave to me when I felt frustrated.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, using of modeling and simulation (M&S), especially distributed simulations, is becoming more pervasive in both the military and commercial worlds. So, what is a computer simulation? It refers to simulating the behavior of another system along the time axis using computer program(s), and a distributed simulation refers to executing computer simulations over computing systems with multiple processors [20], which might be closely coupled multiprocessors or geographically distributed computers. The benefit of computer simulations is twofold. On the one hand, it provides an efficient and cost-effective tool to analyze the targeted system in various ways. On the other hand, it can be used to create virtual environments into which humans and/or hardware devices are embedded [21] and such environments are widely used for training, entertainment, virtual shopping mall, and test or evaluation of hardwares. The main advantage of distributed simulation is that it provides scalable performance while achieves relatively comparable execution speed for bigger targeted systems being simulated by using more CPUs. Another attractive property of distributed simulation is that it allows geographical distribution of simulation components. In this chapter, we first give an overview on High Level Architecture (HLA), which is originated by U.S. DMSO and accepted as an IEEE standard for distributed simulation framework. Then, we present the motivation of this thesis and outline its organization.

1.1 High Level Architecture

The High Level Architecture (HLA) is a standard framework that can be used to construct simulations composed of different individual simulation components. HLA was originated by the U.S. Defense Modeling and Simulation Office (DMSO) of the Department of Defense (DoD) to meet the needs of defense-related projects [2]. In September 2000, HLA was approved as an open standard through the Institute of Electrical and Electronic Engineers (IEEE) as IEEE Standard 1516 - high level architecture for distributed simulations. HLA is now increasingly being used in other application areas than military ones, such as air traffic control and virtual interactive environments.

The main goal of HLA is to promote the reusability and interoperability of individual heterogeneous simulation components [6, 18]. Reusability means that the same simulation components can be reused in different simulation scenarios and applications. Closely related to reusability is interoperability, which means that the reusable components can be incorporated with other components without the re-coding [4]. One way to achieve interoperability is letting different simulation components confirm to uniformed interfaces.

As standard framework for distributed simulations, HLA considers a complex simulation as a hierarchy of components of increasing levels of aggregation [19]. At the lowest level is individual HLA-compliant simulation components, referred to as a federate. Higher level of HLA simulations are made up of a number of HLA federates and are called federations. The component-oriented nature of HLA-based simulations allows individual federates to join and resign from the federation during the simulation run time [4]. This nature of HLA also facilitates the reusability and interoperability of individual simulation components.

In the standard, HLA is defined by three interrelated components [5]:

1. *HLA Rules*

    - Ensure proper interaction of simulations in a federation.
    - Describe responsibilities of federation and federates.

2. *Interface Specification*

   - Defines Run-Time Infrastructure(RTI) services.
   - Identifies "callback" functions each federate must provide.

3. *Object Model Template*

   - Provides a common method for recording information.
   - Establishes the format of key models, including
     - Federation Object Model (FOM)
     - Simulation Object Model (SOM)
     - Management Object Model (MOM)

A) HLA Rules

At the highest level, the HLA consists of a set of 10 HLA rules which must be obeyed if a federate or federation is to be regarded as HLA compliant [6]. The HLA rules are divided into two groups consisting of 5 rules for HLA federations and 5 rules for HLA federates.

Rules for federations [5]

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

Rules for federates [5]

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

B) Interface Specification

A key component of HLA is the Interface Specification (IF), which identifies how federates will interact with federation and, ultimately, with one another [4]. HLA/IF defines two set of standard services, one set of services are implemented by the individual simulations, and the other are implemented by the Run-Time Infrastructure, or RTI.

C) Run-Time Infrastructure

HLA is only the standard framework, while RTI is the software that conforms to HLA/IF. RTI is responsible for providing software services that are necessary to support an HLA-compliant simulation. The HLA/RTI services provided are divided in to six management areas [5]: federation management, declaration management, object management, ownership management, time management and data distribution management. Different versions of the RTI are possible. One version is available from DMSO [2].

As illustrated in Figure 1.1 [4], RTI software is comprised of three components, the RTI Executive process (RtiExec), the Federation Executive process (FedExec) and libRTI library.

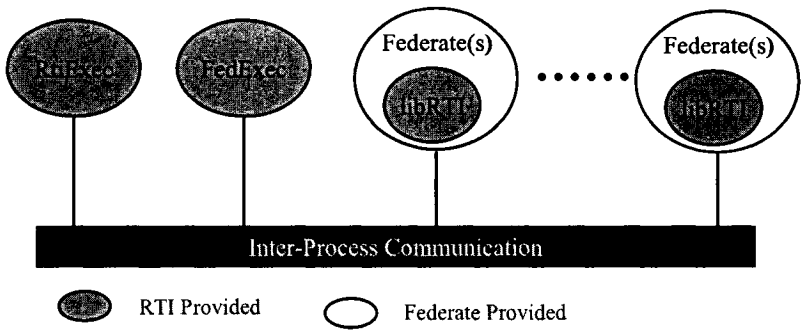

Figure 1.1: Logical view of RTI components

All of the components shown in Figure 1.1 are part of a single federation except for the RtiExec. RtiExec is a globally known process that manages the creation and destruction multiple federation executions, meanwhile directs new arrival federates to join appropriate federation and ensures that each FedExec has a unique name. Each application communicates with RtiExec to initialize RTI components. A globally unique RTI Initialization Data (RID) file is provided with control parameters needed to run an RTI. Each executing federation is characterized by a single, global FexExec. FexExec manages multiple federates within the federation execution. It allows federate to join and resign, and facilitates data exchange between participating federates. A FexExec process is created by the first federate which successfully invokes the "*createFederationExecution*" service for a given federation execution name. Each federation execution needs a Federation Execution Data (FED) file, with contains information derived from FOM and used by RTI during runtime.

Federates use the libRTI to invoke HLA/RTI services. User's code for a federate is linked by Local RTI Component Code (LRC) in libRTI to form a complete federate. LRCs provide services to federate through communication with RTIexec, FedExec and other federates [4]. The components of a single federate are shown in Figure 1.2. Federate code can request RTI services by calling member functions of class *RTIAmbassador*, which is contained in the LibRTI. RTI sends messages and responses to federate code by calling functions implemented in federate as a subclass of *FederateAmbassador*, which are known as callback functions.

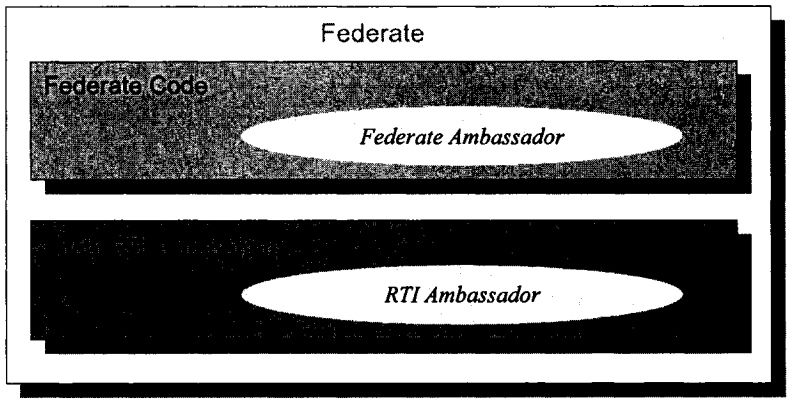

Figure 1.2: Components of a single federate

There are six groups of services defined in the HLA/RTI [2]:

1. Federation Management - is responsible for creation, dynamic control, modification, and deletion of a federation execution.

2. Declaration Management - allows federates to declare their intention to generate or receive information.

3. Object Management - is responsible for registration, modification, and deletion of object instances and the sending and receipt of interactions.

4. Ownership Management - transfers ownership of attributes among federates.

5. Time Management - controls advancement of federates along federation time axis.

6. Data Distribution Management - further refines the services provided by Declaration Management.

B) Object Model Template

Reusability and interoperability require that all objects and interactions managed by a federate, and visible outside the federate, should be specified in detail and with a common format [4, 6]. The Object Model Template provides a standard for documenting HLA object model information. In other word, OMT specifies how to document important information about

simulations and federations. HLA separates data and architecture, therefore, it prescribes that OMT objects and interactions defined according to OMT can be constructed and exchanged with no adjustments to HLA-derived software.

The OMT is used to create SOMs and FOMs. SOM lists the data requirements, objects classes, and other information about a particular federate. A FOM is concerned with federation-wide information, and every federation has a FOM. It includes an enumeration of all object and interaction classes pertinent to the federation along with a specification of the attributes or parameters that characterize these classes [5].

Since the OMT presents a standard format for the FOM and SOM, software tools can automate the creation and maintenance of these documents. Potential automation of the process is one of the benefits of OMT [4]. Another advantage of OMT is format consistency among federates and also among federations, promoting the ease of understanding for federation developers and users [6].

1.2 Motivation

Data Distribution Management is one of the six services provided by HLA/RTI. It is responsible for providing efficient, scalable, interest-based mechanisms for distributing state updates and interaction information in HLA based distributed simulations. Specifically, DDM will throttle the number of messages that are being placed on the network and relief the processing burden on the simulation entities (or federates). While designing DDM algorithms for large-scale systems, more issues need to be considered, such as, dynamic grouping of participants according to their communication interests, efficient data multicasting with bounded end-to-end network latency, and optimized signaling overhead on multicast group membership management.

Several DDM methods have been introduced in recent years, but time performance, message volume and resource usage continue to be factors in the practical application of these

methods, as we shall see in chapter 3. In this thesis, as an effort to offer a more efficient and more scalable solution to DDM, we propose two novel DDM algorithms, referred to as optimized dynamic grid-base DDM and grid-filtered region-based DDM respectively.

1.3 Contributions

In this thesis, we have three main contributions. One is an Optimized dynamic grid-based DDM algorithm, it tries to make up the weaknesses of the dynamic grid-based DDM, and enhance its network performance. We achieve that by using a second level of send-side data filtering mechanism to make sure publishers only send out data of absolutely necessary.

The second is a Grid-filtered region-based DDM scheme. It utilizes a grid overlay on the routing space, determines the percentage of grid covered by the subscription or publication region and further filters, based on a percentage threshold.

Last but not least, we designed a preliminary real-time RTI architecture to facilitate the use of HLA in time critical applications. This design is based upon a multi-threaded process model together with a global scheduling service, that address the weaknesses of HLA/RTI in terms of real-time applications.

1.4 Thesis outline

The remainder of this thesis is organized as follows:

- Chapter 2 briefly introduces the Data Distribution Management, including why DDM mechanism is necessary in large scale distributed interactive simulation, how does DDM mechanism works? and terminologies used in DDM domain.

- Chapter 3 describes previous and related work in DDM. Alternative DDM approaches, such as region-based, fixed grid-based and dynamic grid-based DDM etc., are introduced. We also discuss the Pros and Cons of these DDM strategies.

- Chapter 4 proposes an enhanced DDM algorithm, which we refer to as optimized dynamic grid-base DDM algorithm, and presents the performance evaluation based on an extensive set of experiments.

- Chapter 5 presents the Grid-filtered Region-based DDM, which is another new DDM algorithm, and reports its performance based on an extensive set of experiments.

- Chapter 6 discusses an initial work of real-time RTI architecture design, based upon a multi-threaded process model together with a global scheduling service, that address the weaknesses of HLA/RTI in terms of time critical applications.

Chapter 2

Data Distribution Management

Data Distribution Management (DDM) is one of the six services provided by HLA/RTI as complementarities of Declaration Management (DM) to provide efficient, scalable, interest-based mechanisms for distributing state updates and interaction information in large scale distributed simulations [5]. Specifically, the primary goals of DDM is to (1) cut down the number of messages that are being injected into the transmission network; (2) reduce the processing burden on the computers or hosts taking part in the simulation. In the rest of this chapter, we will give answers to two FAQ questions: why DDM mechanism is necessary in large scale distributed interactive simulation, and how does DDM mechanism works? Furthermore, the terminologies used in DDM domain are introduced which are fundamental to understand the different DDM schemes in later chapters.

2.1 Why DDM is necessary

DDM is the fundamental and challenging problem for the parallel and distributed simulation community. This is mainly due to the distribution of individual simulation components of a high level simulation among multiple computers which might be geographically located. The technologies used for distributed computational systems are essentially different from the traditional sequential simulations [13], which permit direct access between simulated entities since all the entities reside on the same computer and share the ubiquitous CPU clock. However, in a distributed simulation environment, data access between distributed simulation entities must be closely regulated for three reasons: (1) distributed simulation objects can be at different logical times; (2)inappropriate and/or excessive data access put heavy burden on the underlying network; (3)it is difficult to reference remote objects with certain relations in the virtual world.

Therefore, it is essential to have a mechanism responsible for providing efficient, scalable services for distributing state updates and interaction information in parallel and distributed simulations. Such a mechanism is referred as Data Distribution Management or DDM. Specifically, DDM will throttle the number of messages that are being placed on the network and relief the processing burden on the simulation entities (or federates).

The intuitionistic observation that real world objects might be interested in only a fraction of the objects surrounding them is the fundamental of DDM. The problems of which object/entity and what kinds of object's/entity's characteristics will be of interest for an 'observer' are greatly depending upon the observer's sensing capability. For example the scenario shown in Figure 2.1, which is a war game simulation where there are two opposite teams, red and blue. An object in this scenario is primarily interested only in other objects that are close to it, however, it probably does not concern with any other objects that might be too far away, this is mainly due to its limited sensing range. Furthermore, because the two team are antagonistic to each other and intestine information exchange might take place within a team, an object of red team is not allowed to detect the private information (such as the available fire

powder and the next action to be taken of an object) exchanged within the blue team and vice versa, which is the result of application-specific properties of the war game simulation. While systems without DDM mechanism[1] do not take into account of this real world phenomenon of limited interest, and all participating entities broadcast message about their updated states and activities to all other entities, even though most of these messages are of no interest at all, resulting in an explosion of network transmission load. Moreover all the entities will receive a copy of all the data injected into the network, which force each entity to filter out the received data that are not intent to be received so that the correctness of the simulation application could be maintained.

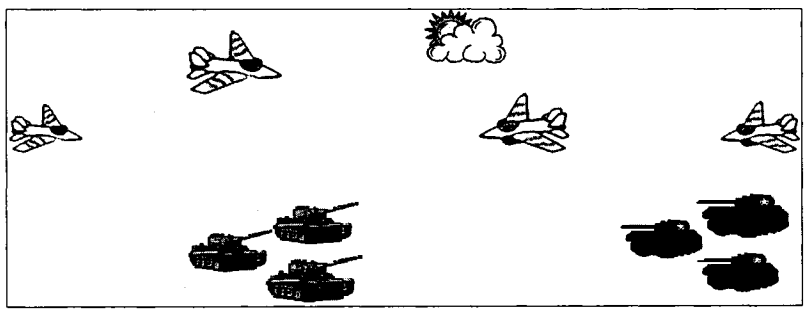

Figure 2.1: Limited interests of real world objects

Apparently, the cost of transmitting irrelevant data is very high, especially for large-scale distributed simulation systems, which require higher communication performance. Sending out data of no interest at all puts a heavy traffic on the underlying transmission network. Furthermore, the hosts receiving large amount irrelevant data will waste a significant amount of time and system resources while receiving and processing those needless information. Both the two factors would degrade the performance of large scale distributed simulation, and limit its utilization in reality. For small scale simulations such waste is relatively unimportant. However, the broadcast scheme becomes unworkable as exercises increase beyond a relatively small size. Therefore, DDM mechanism is of absolutely necessary in order to limit the network traffic and reduce the processing burden of participating hosts in large scale distributed interactive simulations. In HLA Interface Specification, it seems like Declaration Manage-

[1] Distributed Interactive Simulation (DIS) protocol is an example of this category.

ment (DM) is also designed for this purpose to some extent. However, DM and DDM are different. In general, many DDM services are simply more specific versions of a DM service. The specificity provided by DM and DDM service allows the federate programmer tools to segregate and limit the flow of data to and from federates within a federation. To achieve this, a compliant RTI design must support both service sets and provide the requisite inter operation of federates using the two sets of services [28].

2.2 How does DDM mechanism work

DDM mechanism is based on simulated entities' interest of data at large. In other words, every entity is required to express the data it can make available to the world[2], and the data it want to receive from the world. We refer this problem as "who can see whom"[13]. An obvious observation for the problem is that: the DDM mechanism should provide an interface letting the simulated entities to express their data interests, which however is not a trivial task and we will come back to this issue later.

Once interest are specified by federates using the standard interfaces provided by DDM mechanism, the two data sets (the data that can be seen and the data that wanted by others) are matched to compute the intersections of interests. The resulting intersections are then used to determine whether update or interaction information should be delivered to a participating host. This process is called filtering and it is one principle part of DDM mechanism. To date, several well-known methods to perform filtering functions are developed. Depending on different filtering schemes, DDM mechanism can be categorized into three main schools, namely region-based methods, grid-based methods and hybrid approaches. The region-based scheme deploys the exhaustive matching, thus its filtering processing gains the highest level of accuracy. The grid-based methods use a multidimensional grid system overlying on the virtual world in which the simulation is running, which makes its filtering processing tends to be less accurate, but the computational overhead is optimized compared with region-based scheme.

[2]World refers to the virtual environment of the simulated system.

The Hybrid approach attempts to combines the simplicity of the grid-based methods with the complex matching operation of the Region-based method, achieving certain trade-off between the two. We would introduce the various DDM mechanisms for HLA-compliant distributed simulations developed in recent years in Chapter 3.

The next step of a DDM mechanism is the data transferring phase. Communication is established based on the results of filtering, through the established communication channel, the participating federates can exchange update and interaction information with one another. In the HLA, three types of communication pattern are allowed, which are:

- *Unicast*

  There is one sender, and message is received by only one destination.

- *Broadcast*

  There is one sender, and message is received by all destinations[3] in the networks.

- *Multicast*

  There is one sender, and message received by multiple (but not necessarily all) destinations. Its operations include joining a multicast group, leaving a multicast group and sending messages to a multicast group. It can be implemented by various protocols such as unicast, or network multicast(such as IP multicast and TCP/IP sockets).

The communication pattern of multicast groups are used by most DDM schemes. And it is fundamental to all DDM implementations.

2.3 What are challenging while designing DDM

While designing and implementing a DDM algorithm, a number of requirements need to be satisfied. Some of the requirements are listed below. The challenging is to achieve all the requirements at the same time.

[3]The network nodes that are connected with the source node either directly or indirectly.

- *Correctness*

  Complete and unerring implementation that confirm to HLA Interface Specification.

- *Efficiency*

  An algorithm accomplish the desired functionalities while introduce as little overhead as possible.

- *Scalability*

  Important to help evaluate future applicability and capability of DDM services, especially for large-scale distributed simulations.

- *Performance*

  Desirable and necessary, but achieving the highest performance possible is not of paramount importance, especially if there are significant costs to clarity. In reality, there is always certain trade-offs between the performance and clarity [27].

- *Modularity*

  Easy interposition into component-based architectures like HLA. And for the purpose of reusability and interoperability.

Besides, as I mentioned in previous subsection, DDM mechanisms should provide an interface letting the simulated entities to express their data interest. However, it is not a trivial task, due to one property of large scale distributed simulations, which is that there is usually no relations between an entities' physical location in the network and its logical location in the virtual environment being simulated [27]. Figure 2.2 illustrates those positions of simulated entities in the virtual world and their potential locations in the network environment. Notice that some entities simulated on the same node or LAN are far apart in the virtual environment, while entities simulated far apart on the same WAN may be very close in the virtual environment. Thus, there are no obvious relations between the simulated entities' data interests and virtual world. To solve this problem, the HLA provides the notion of Routing Space and

Region which can be use by federates to express their interests, in the next subsection, we will introduce these terminologies.

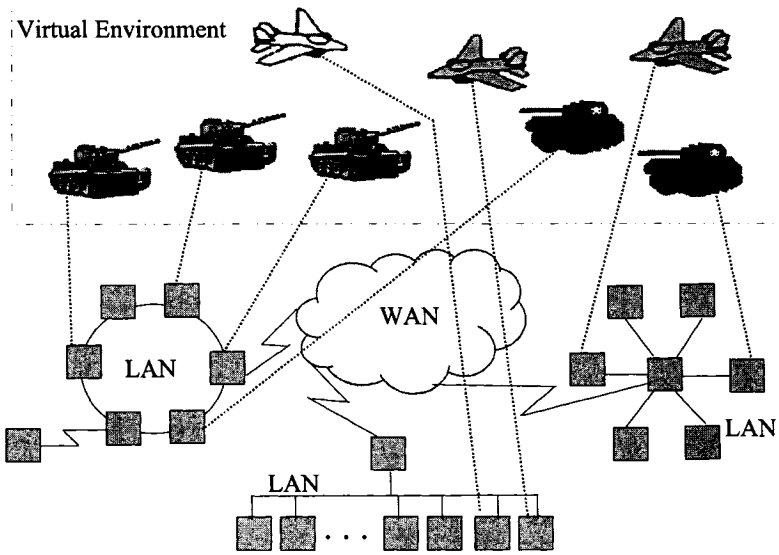

Figure 2.2: Virtual positions vs. physical location

Another challenge in designing DDM protocol is the filtering mechanism. We will explore the various DDM filtering mechanisms for HLA-compliant distributed simulations developed in recent years later. Each DDM method is discussed under the proper category.

2.4 Terminologies

As discussed before that DDM mechanisms should provide interfaces that allow the simulated entities to express their interest of data (either sending or receiving). A number of abstractions/terminologies of the HLA DDM services are introduced for this purpose. In this section, we will describe several terminologies used in the DDM literature, and which will be the fundamental to the discussion in the remainder of this thesis.

2.4.1 Routing space

A fundamental abstraction of the HLA data distribution management services is the routing space. A routing space is a multidimensional coordinate system [27]. It represents the virtual world in which the simulation occurs. For example, the war game scenario in Figure 2.1 might be represented by a four-dimensional routing space, where the dimensions are "altitude", "east-west", "north-south" and "team". If a more accurate representation of the simulated environment is required, more dimensions can be added to its routing space, such as extend the routing space of the war game simulation to six dimensions where the fifth and sixth dimension could be velocity and firepower of a participating object. The fourth to sixth dimensions allow the detection of entities that were on a specific team, traveling at a particular speed and carrying a specified compliment of weapons. As you might notice that, in practice, not only the geographical locations can be chosen as routing space dimensions, but also other object attributes might be used to represent dimension, such as the velocity and firepower in the above example. The number of dimensions may vary from one up to a maximum value limited only by the RTI implementation and the underlying technology. There is no conceptual limit to the number of dimensions in a routing space. This permits arbitrarily complex filtering to be specified via routing spaces.

Choosing how many dimensions a routing space will have and what they will represent are very important decisions that must be made carefully when designing a simulation that will use DDM in the system-design phase. Too few dimensions will lead to impreciseness and ambiguity in the simulating process, while too many dimensions will increase the complexity of the targeted system. Furthermore, misuse of dimensions can result in systems that do not perform adequately, or which exhibit undesirable behaviors. One simulation application may define multiple routing spaces, each with different characteristics and employed for different purposes.

In a distributed simulation environment, the participating hosts need to agree on three pieces of information about a routing space in use. They are the number of dimensions of the

routing space, the "real life" meanings corresponding to each dimension and the mapping of the routing variables[4] to routing space coordinates.

2.4.2 Publication region and subscription region

A region is defined as a subset of routing space. It is specified by a set of extents, one per routing space dimension [27]. An extent is an ordered pair denoting the maximum and minimum value of a region along a dimension [13]. If the routing variables for a dimension are discrete values, instead of extent, a point is used to express the value for that dimension, such as the fourth dimension of the war game simulation, has red and blue as its routing variables, therefore, for a specific region, its value for this dimension is either red or blue.

A subscription region is an abstraction referring to the set of data in which a simulated entity is interested. A publication region is analogous construct that refers to the set of data that the simulated entity is making available to the world [29]. Usually, interest region is used to refer to both publication and subscription. In general, publication regions tend to be very small, sometimes even a single point, whereas, subscription regions can be small or very large depending on the detecting capabilities of a simulated entity [13].

Subscriber (or publisher) refers to the host where the local federate has expressed an interest (or availability) and has generated the subscription (or publication). Publishers are the senders or producer of data, and subscribers are receivers or consumers [11, 13, 36]. A publication region is also known as an update region, since the publisher may need to send updates of its state information to its subscribers.

2.4.3 Intersection region

Intersection region refers to an area where subscription and publication regions overlap. Regions overlap if and only if the corresponding extent sets overlap [13]. If a given subscription and publication do not overlap, we say that there is no intersection. If an intersection

[4]Routing variables refer to the permitted values for a dimension.

region exists between a subscription and a publication, then data exchange occurs between the corresponding subscriber and publisher.

Actually, the intersection detection is similar to the concept of filtering mentioned in the previous section. Identifying the intersection and subsequent exchange of states update and interactions is not a trivial process in large scale distributed simulations, since the participating federates might be located far away from each other, even in different continents possibly. As a consequence, there are two fundamental problems any DDM approach must solve: (a) Precisely how is the intersection region identified with low cost, and (b) How does the data exchange between subscriber and publisher take place effectively? There are still some other issues that need to be considered while designing DDM algorithms for large-scale systems, such as, dynamic grouping of participants according to their communication interests, efficient data multicasting with bounded end-to-end network latency, and optimized signaling overhead on multicast group membership management [27].

2.4.4 Multicast group communication

Multicast group is a communication pattern referring to the concept of single-source, multiple destination transmission [34, 37]. It allows simulation host to send data to specific network addresses reserved for certain groups of hosts rather than broadcasting their update and interactions to all hosts [31]. In this manner, the simulation hosts can reject traffic not addressed to them more efficiently, at a lower level in the operating system, rather than running a function at the application level to ascertain the relevance of the data. Therefore, this type of communication is fundamental to all DDM implementations, and it can be implemented by various protocols, such as IP multicast and TCP/IP sockets. Multi-cast group communications are used by most DDM schemes.

However, multicast network addresses, or multicast groups, are limited system resources, and there are significant overhead associated with managing the groups, such as changing group membership and creating new groups. Therefore, it is critical to make use of multicast

groups in the most effective manner [31].

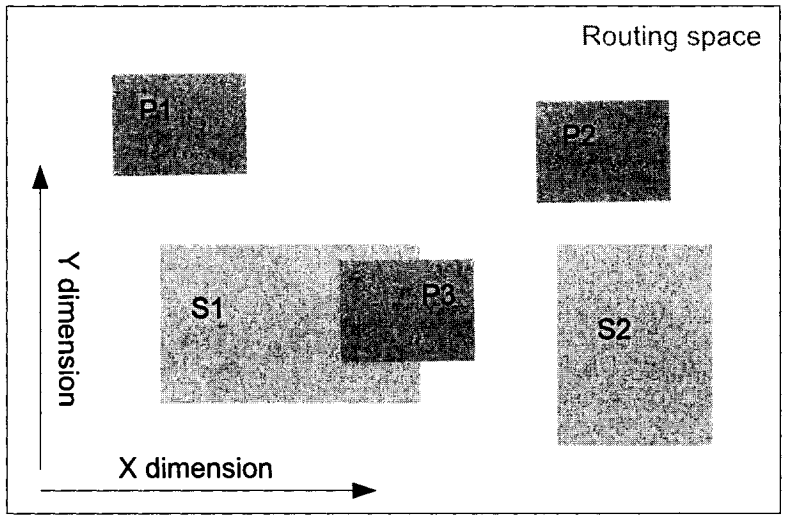

Figure 2.3: Routing space and regions

Figure 2.3 illustrates the above notions used in DDM. Note that this example has been simplified and abstracted for the sake of understanding - some details, such as the notion of extent, are omitted. This Figure shows that five federates have specified one region each in the two dimensional routing space shown. The first to third federates respectively specify update regions P1, P2, and P3 that define data the federates wish to send. The fourth and fifth federates specify subscription regions S1 and S2, respectively. These subscription regions define data these two federates wish to receive. DDM detects that region S2 overlaps with region P3 and creates an intersection region. Necessary network connectivity is established so that the data sent by the federate that specified P3 is delivered to the federate that specified S2. In contrast, no overlap is detected between between P3 and S1, so the same data does not need to be delivered to the federate that specified S1.

In the above example, a two dimensional routing space is used for two reasons. First, it is the least complex case that fully illustrates the concept of routing space; second, it simplifies the presentation of multidimensional space on paper medium. However, the practical simulation use routing spaces with at least three dimensions, and most of real world applications use more than three dimensions [13].

Chapter 3

Literature Review on DDM

Data distribution management have been deeply investigated on an ongoing basis. Traditionally this research area was referred to as Interest Management (IM) in DIS protocol. The concept of DDM comes with HLA/RTI. During the past decade, several groups of researchers have been working on DDM schemes for large-scale distributed simulation systems. These studies are seminal and substantial; the preliminary findings are quite promising.

Based on how to perform the matching to detect intersection regions between publication regions and subscription regions of all participating simulation entities, current DDM methods can be divided into three categories, region-based DDM methods, grid-based DDM approaches and hybrid ones [17, 26, 40]. Depending on the manner of multicast group assignment, grid-based scheme can be further divided into fixed grid-based DDM and dynamic grid-based DDM. Besides the three main categories, other DDM approaches using totally different paradigm are developed as well, such as agent-based and sort-based DDMs.

In this chapter, we are going to introduce the alternative DDM approaches developed in recent years. Each DDM method is discussed under the proper category. Our aim is to help better understanding of the current DDM approaches and point out the advantages and disadvantages of these approaches.

3.1 Region-based DDM method

The region-based DDM approach[1], which was implemented in the RTI 1.3 [28], is the most straightforward DDM approach. It just simply compares the subscription and publication directly in order to find which ones overlap. This process is called matching. As is known, in the worst case, the exhaustive matching used by region-base DDM requires every subscription to be compared with every publication, leading to $O(N^2)$ of the worst case upper bound computational complexity.

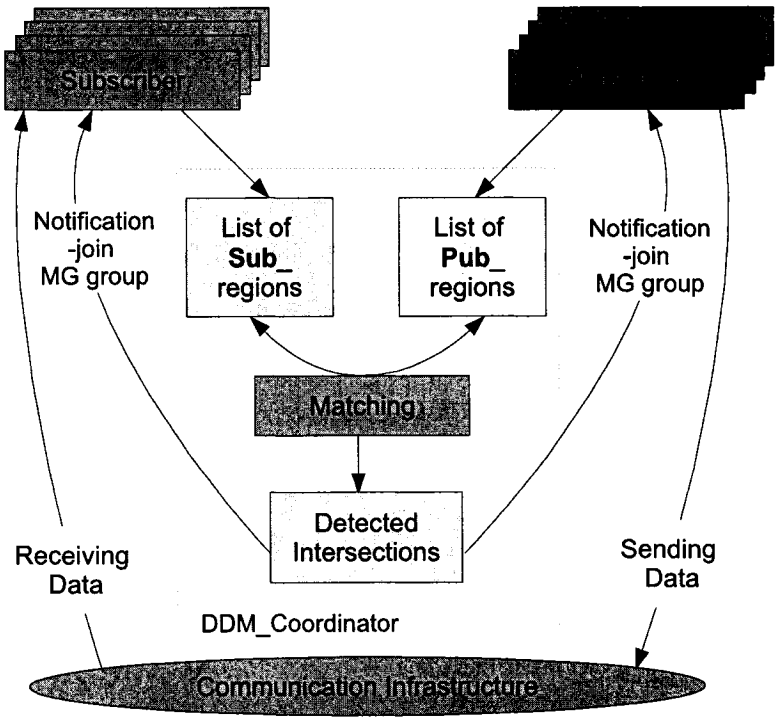

Figure 3.1: Conceptual abstraction region-based DDM

Figure 3.1 illustrates the conceptual abstraction the region-based DDM. There are a set of notional producers of data (the "Publishers") and a set of notional consumers of data (the "Subscribers"). Data[2] are transferred through a communications infrastructure as shown at the bottom of the Figure. Region-based DDM provides mechanisms through which federa-

[1]Region-based DDM is also referred to brute force approach as it employ the exhaustive matching scheme.
[2]Data refers to object attributes or interaction parameters

tions may control the distribution of data from publishers to subscribers. In order to establish the necessary connectivity, publishers and subscribers report their data requirements, in terms of publication regions and subscription regions, to the centralized DDM_Coordinator. As Figure 3.1 shows, databases of Sub_regions and Pub_regions are maintained at DDM_Coordinator locally. A matching operation between Sub_regions and Pub_regions is performed to detecting intersections at each time step. The result of matching are a set of intersections together with sets of subscribers. A subscriber set identifies the owner of each subscription (i.e., the "subscriber") that certain Pub_region matches. The subscriber set is essentially a listing of the recipients to which data in Pub_region must be delivered. The subscriber set is used to establish connectivity in the communications infrastructure, as indicated in Figure 3.1. As a result of matching, DDM_Coordinator will trigger publisher and subscribers to join or leave multicast groups. Thus, in the region-based DDM method, there are two types of communication among participating hosts: (1) the interaction between federates and the centralized DDM_Coordinator, federates report their publication regions and/or subscription regions to DDM_Coordinator, who will then compute the intersections of the two lists; (2) Once intersections are determined, the connectivity for a second type of communication will be established, this type of data transmission is between different federates or their the local RTI components from publisher to subscriber. The type of the first communication is unicast. While the second kind of inter-host communication is generally performed using multicast technology mentioned in Section 2.4.

Now, let us look at how the region-based scheme assign multicast groups and manage the membership of multicast groups. Suppose that the matching operation has determined that a publication made by federate F to region p is part of an intersection. Then, region p is assigned with a multicast group mg-p. Federate F is triggered to join mg-p and begin sending data on that group. Subscriptions that overlap with p are also triggered to join mg-p and thereby receive any data sent on that group. Thus, exchanging data has been facilitated between federate F and all of the other federates interested in the data that F is publishing in

region p. After additional matching has been performed due to the addition, modification or deletion of regions, region p might be no longer part of an intersection. Then, all federates that joined that mg-p are triggered to leave the group, terminating the data exchange between publisher and subscribers of previous intersection.

There are two overheads associated with the region-based scheme, which are computation and communication costs. The computational overhead occurs because matching can happen frequently (i.e., at each time step in discrete time simulation). Whenever a publication region is modified, it must be matched with all of the subscription regions. Publication region modification can be very common occurrence due to the fact that publishing objects, such as planes or tanks, are usually moving entities that repeatedly change their positions during every time step in discrete time simulations. Similarly, each time a subscription region is created or modified; this information must be matched to all of the publication regions on all participating federate.

The overhead of the inter-host communication needed to compare regions of different federates, is the second cost associated with region-based matching. Each federate creates and modifies its publication and subscription region locally on its own host, but the matching must be performed among regions federation-wide. So, every change to a publication or subscription has to be reported to the centralized DDM_Coordinator. This may result in an explosion of the incoming messages to the centralized DDM_Coordinator. A consequence is that the connectivity established by DDM is dynamic: connections are set up and broken down because regions go in and out of scope. There are clearly costs associated with each of these operations. Such costs must be accounted for not only in the design approach and its implementation but also in how DDM capabilities are employed in a federation. Moreover, the use of the centralized coordinator limits the scalability of region-based DDM scheme, affecting the performance and causing potential bottleneck in large scale distributed simulations.

3.2 Grid-based DDM method

Alternatively, the grid-based DDM approach tries to reduce the computational and communicational overheads of the region-based method [11, 30, 31]. In this method, the routing space is overlaid by a multi-dimensional grid, and the grid will generally have the same number of dimensions as the routing space.

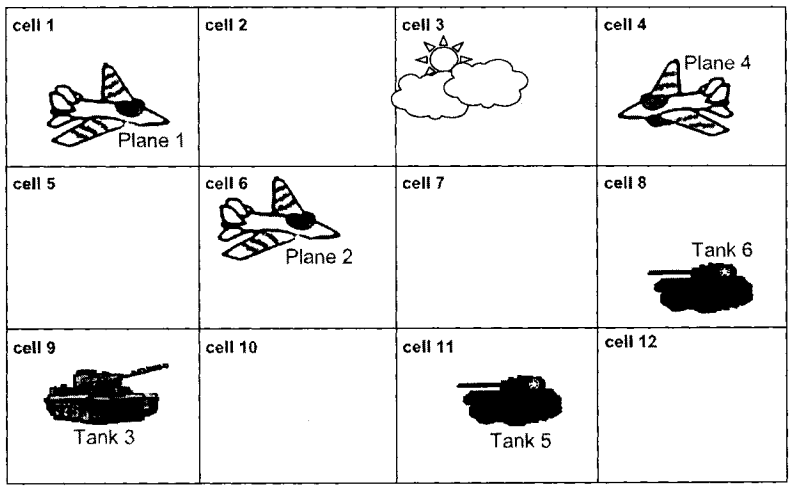

Figure 3.2: Grid overlaying on routing space

Figure 3.2 illustrates how a grid is overlaid on the simulation terrain (the war game scenario) and is used to determine intersection regions for grid-based DDM method. In this example, the two dimensional routing space is configured to have four cells along one dimension and three along the other. The subscription of the spy plane 2 is mapped to cell 6, 7, 10 and 11. The publication of tank 3 is mapped to cell 9, similarly for tank 5 and cell 11, tank 6 and cell 8. Cells that are part of both a publication and subscription represent an intersection. In our case, subscription region S2 overlaps with publication region P5 in cell 11. The conceptual abstraction of the same scenario is shown in Figure 3.3.

Compared to the region-based method, grid-based filtering approaches provide relatively simple mechanisms for establishing sufficient connectivity for the RTI to route relevant attributes and interactions from producers to consumers [27]. The steps of how a grid may be

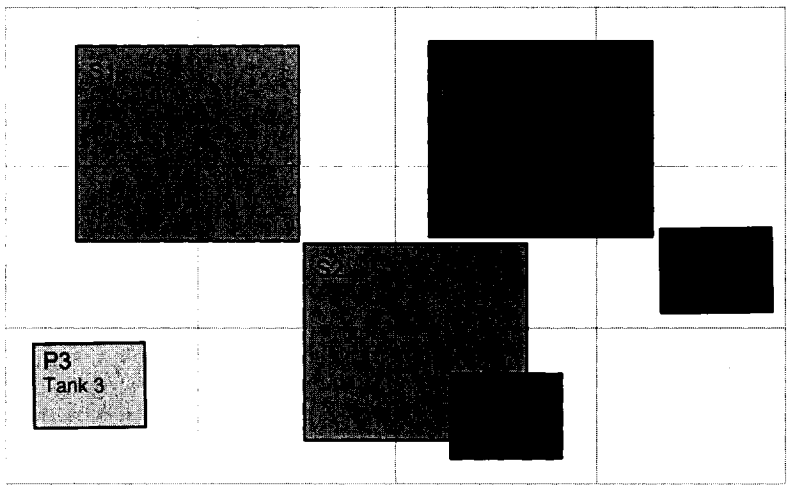

Figure 3.3: Conceptual abstraction of grid vs. routing space

used to determine which updates and interactions to route to which receivers are as follows:

- Subdivide each routing space into an array of cells and assign a multicast group to each cell.
- Determine overlapped subscription regions to a cell and trigger the subscribers as data receivers to join the multicast group assigned the cell. This procedure is applied to every cell.
- Determine overlapped update region to a cell and trigger the publisher as data sender to join the multicast group assigned the cell. This procedure is applied to every cell.
- Publishers Send updates for attributes and interactions associated with their update region to the corresponding groups.
- Federates receive data sent to groups they have joined (relevant data) but do not receive data sent to groups they have not joined (irrelevant data).

From the steps we can see that the grid-based technique avoids these extensive and costly operations in the region-based approach which necessitates matching between all publications and all subscriptions. Instead of explicitly comparing publication and subscription region,

grid-based style maps each interest region onto a multidimensional grid. The process of overlaying a grid on the terrain can be performed by the local RTI component (LRC) on each host. Each one performs the region-to-cells mapping independently of the other hosts since the number and size of the grid cells are determined before the simulations begin and remains constant throughout the execution of the simulation. Another advantage of grid-based method is that no centralized DDM_Coordinator is needed, making the grid-based method relatively scalable.

The major drawback of the grid-based method is that its matching accuracy tends to be lower [13], when compared to the region-based method. The mapping of the interest regions to the grid cells may not be exact. Indeed, region boundaries might not fall precisely on cell boundaries, in which case the area covered by the cells representing the interest region will be slightly larger than the region itself. This may cause superfluous intersections in some cells, and the publishers sending to subscribers' unneeded data.

Now we have known that how intersections are determined by the grid-based scheme, that is, by mapping interest regions to cells. Several grid-based variation have been proposed by now. They all differ in the way they use multicast communication strategies to efficiently transfer data between the hosts that produce data and the hosts that consume it. Specifically speaking, they deploy different strategies to manage the grid system and thus gain different results of intersection detection, which in turn results different multicast groups assignments. In the subsections, we will introduce some of the grid-based DDM variations and summarize the strengths and weaknesses of the different strategies.

3.2.1 Fixed grid-based DDM method

The fixed grid-based method, also known as the RTI prototype [36], was the initial implementation of the HLA-RTI. This approach is a fairly straightforward one. First, a multicast group is assigned to each grid cell at the system initialization phase, which means there is a bit of a start-up cost involved, but this pays off in a reduction of computational and com-

munication costs during the simulation run time. As the simulation progresses, the local RTI component running on each federate maps the publication regions or/and subscription regions to grid cells. At each federate, its local DDM component identifies all the cells[3] with which its subscription regions overlap, and the federate joins itself to all the multicast groups pre-assigned to Sub_Cells. At the same time, each federate's local DDM component identifies all the cells[4] with which its publication regions overlap, and starts sending data to all the multicast groups pre-assigned to Pub_Cells. So the connectivity is established automatically between publisher and subscribers that overlap with the same grid cell via the multicast group assigned to the cell at the beginning of simulation.

Table 3.1 shows how the fixed grid-based DDM system maps the grid cells to multicast groups and publication/subscription regions. It also shows which entities join which groups, for the scenario presented in Figure 3.3. Note that this scenario is just a snapshot of the simulation at one of the time steps. In reality, the war game simulation is progressing, therefore membership in any multicast group is changing as the publications and subscriptions are deleted, modified, added and/or changed during simulation run time. We assume six federates are participating in the war game simulation, and the plane and tank objects are distributed among them as follows: plane 1 is simulated at federate 1, and federate 2 to federate 6 are simulating plane 2, tank 3, plane 4, tank 5 and tank 6, respectively. The spy planes are subscribing to the terrain within its radar range. Subscription region of plane 1 is mapped to cell 1, 2, 5 and 6. Subscription region of plane 2 is mapped to cell 6, 7, 10 and 11. Subscription region of plane 4 is mapped to cell 3, 4, 7 and 8. The tanks are publishing there current positions. Publication region of tank 2 is mapped to cell 9. Publication region of tank 5 is mapped to cell 11. Publication region of tank 6 is mapped to cell 8. Therefore, federate 2 joins multicast group 6, 7, 10 and 11 (MG6, MG7, MG10 and MG11), that have been assigned to cell 6, 7, 10 and 11 respectively. Federate 5 joins multicast group 11 (MG11), that has been assigned to cell 11. The other participating federates following the similar step to join the right multicast

[3] A set of grid cells refers to Sub_Cells

[4] A set of grid cells refers to Pub_Cells.

group(s).

Federates	Entity type	Interest region	Cells in groups	Assigned
Fed1	Spy Plane 1	Subscription	1, 2, 5, 6	MG1, MG2, MG5, MG6
Fed2	Spy Plane 2	Subscription	6, 7, 10, 11	MG6, MG7, MG10, MG11
Fed3	Tank 3	Publication	9	MG9
Fed4	Spy Plane 4	Subscription	3, 4, 7, 8	MG3, MG4, MG7, MG8
Fed5	Tank 5	Publication	11	MG11
Fed6	Tank 6	Publication	8	MG8

Table 3.1: Fixed grid-based grouping

The simplicity of grid approaches makes them attractive. They are relatively easy to implement and robust because in order to establish connectivity between senders and receivers, intersection regions detection are not necessary, even indirectly. Matching of subscription and publication regions is performed implicitly on a grid local to both the senders and receivers. Only the number of cells in each dimension and the algorithm for assigning multicast groups to cells must be shared. Despite the fact that intersections are not considered by the algorithm, whenever an intersection is present, data will be properly transferred between the publisher and subscriber because they both will have joined the same multicast groups. These groups are the ones that correspond to the cells in the intersection, which in our example is multicast Group 11 (MG11).

Despite these good features, fix grid-based approach has a number of drawbacks [27, 13] that make it inadequate or unacceptable for many applications. These include:

- Large numbers of multicast groups are needed if the number routing space dimensions is large. Multicast groups tend to be a scarce resource whose over use has performance implications in the network infrastructure and in host I/O performance. Use of a larger number of groups also implies an increase in group change rate with more bursty traffic profiles as an undesirable consequence [29].

- Multiple transmissions may be required for extended update regions that are not points in routing space (one for each grid overlapped by the update region) or excessively large

interest extents, thereby reducing efficiency. While most vehicles and individual combatants can be adequately dealt with by point update regions, some simulated objects are inherently volumetric. Examples include weather, smoke, dust, electromagnetics, nuclear, chemical, and biological effects [27].

- No explicit use of information about what data is relevant is exploited, making fixed grid-based approach suboptimal. Data is sent even if it is relevant to no other federate, such as MG9 and MG12 in the above example. Also, the number of groups used can be in excess of what is actually needed provide the necessary connectivity, such as MG1-MG10 and MG12. This is because senders have no idea what receivers are listening vice versa.

- Irrelevant data is delivered because of the inherent quantization of the grid. Addressing this problem by reducing the grid size (increasing the number of cells along any routing space dimension) can vastly increase multicast group usage. In the RTI prototype, update regions are included in each update to permit irrelevant data to be optionally discarded through filtering at the receivers [27].

In summary, grid approaches are adequate for applications that do not need to push the limits of scalability, performance, and cost. Relative ease of implementation induced the RTI development team to select a fix grid-based approach as the initial approach in the RTI prototype [27, 13]. But better approaches are needed to achieve the required scalability of simulation systems supported by the RTI.

3.2.2 Dynamic grid-based DDM method

Similar to the fixed grid-based method, we define the cells by overlaying the terrain within a grid in dynamic grid-based approach too. However, unlike the fixed grid-based scheme which statically assigns a multicast group to each grid cell at system initialization phase, multicast groups are dynamically allocated during simulation run time, based on the current sta-

tus of publication regions and subscription regions in the system. The main purpose of this strategy is try to minimize the usage of multicast groups, which is a critical system resource associated with fairly large amount of overhead on their operations. In order to achieve the goad, dynamic grid-based DDM allocates a multicast group to a grid cell at a given time step if and only if there is at least one federate publishing and at least one federate subscribing to the cell. As a result, federates join and leave the appropriate groups upon receiving trigger messages sent by the grid system. One cell manager is specified to each grid cell in this scheme to manage the publisher(s) and subscriber(s) to the cell, and allocate multicast group and maintain membership list of the group whenever applicable.

The basic steps of this algorithm when a federate F publishes or subscribes to a region containing cell Ci is as follow. According to the algorithm, if there was previously a publisher and a subscriber whose regions overlapped with cell Ci, there should be a multicast group already assigned to that cell. Consequently, federate F simply joins that multicast group and performs the final step of this scheme, which is to record the new subscription region or publication region to the cell manager and update the membership of the corresponding multicast group. Otherwise, if the publication or the subscription made by federate F creates a matching[5] between the publisher(s) and subscriber(s) region in that cell, a multicast group is then allocated to that cell. Federate F, as well as all of the other federates that are publishing or subscribing to cell Ci, must join that group, followed by the final step. In either cases the final step must be carried out because if cell Ci is not currently assigned to a multicast group, it may be assigned one in the future, in which case federate F will need to be told to join that multicast group according to the information recorded by the cell manager in this step.

By applying dynamic grid-based DDM algorithm to the scenario show in Figure 3.3, we arrive at Table 3.2, which shows how the grid cells are mapped to publication/subscription regions and multicast groups. It also shows the membership of a multicast group.

This technique has the dual advantage of (1) preventing senders from transmitting data

[5]At least one publication region and at least one subscription region co-exist in cell Ci

Federates	Entity type	Interest region	Cells in groups	Assigned
Fed1	Spy Plane 1	Subscription	1, 2, 5, 6	none
Fed2	Spy Plane 2	Subscription	6, 7, 10, 11	MG11
Fed3	Tank 3	Publication	9	none
Fed4	Spy Plane 4	Subscription	3, 4, 7, 8	none
Fed5	Tank 5	Publication	11	MG11
Fed6	Tank 6	Publication	8	none

Table 3.2: Dynamic grid-based grouping

needlessly; and (2) reducing the number of multicast groups that a federate needs to join, since groups are only joined and data are only sent when co-exist of publication and subscription in a grid cell is detected. For example, consider the scenario from our previous example that was used to illustrate the fixed grid-based method. Using the dynamic approach, an intersection would be detected in Cell 11, and Fed2 and Fed5 would then be triggered to join MG11, as shown in Table 3.2. In this simple example, the dynamic method used only one multicast group (MG11) as compared to the twelve groups (MG1 to MG12) needed by the fixed method. For a large-scale simulation with many federates and objects, the savings in multicast groups usage by the dynamic grid-based scheme, as opposed to the fixed method, is tremendous as shown by the experimental results [13].

Another benefit of this scheme is that intersection detection and triggering mechanism are performed by local RTI components on each host. This may be viewed as a collection of distributed DDM-Coordinators where each coordinator is responsible for keeping track of a specific set of grid cells [13]. There is no central database or coordinator, thereby making our dynamic approach more scalable than the region-based scheme which uses a centralized DDM_Coordinator.

3.3 Hybrid DDM method

Both region-based and grid-based algorithms have their own advantages and disadvantages. Some researcher [25] proposed to combine the two independent ideas into a hybrid

DDM approach, which inherits the merits of both region-based and grid-based algorithms.

3.3.1 Simple hybrid DDM approach

Simple hybrid approach seeks to reduce the matching cost associated with the region-based approach by only performing matching between publications and subscriptions that are part of an intersection [12]. The algorithm is as follow. All participating federates send its interest information to a single centralized DDM_Coordinator, as in region-based method. The coordinator then determines intersections by mapping the interest regions onto a grid, as in the grid-based approaches. Finally, the DDM_Coordinator performs matching only on those publications and subscriptions that are mapped to the overlapping cells.

This approach limits the amount of matching that the DDM_Coordinator must perform, which is its main advantage over the region-based scheme. However, this scheme still requires a federation-wide DDM_Coordinator, as in a region-based scheme. The use of a centralized coordinator tends to limit the scalability of the DDM system. This is a drawback compared to grid-based approaches, which don't have a central coordinator.

3.4 Intelligent agent-based DDM

Agent-based DDM is another novel algorithm proposed by a research group in Singapore [43]. In this method, intelligent mobile agents are launched to publishers by its owner subscriber to fetch data, perform data filtering and send back the exact state updates and interactions information to its subscribe_owner.

The design structure of the agent-based DDM is show in Figure 3.4, in which a totally distributed RTI structure is employed, namely ARTI. In the HLA, federates define subscriptions to declare their interest of data that other federates own. At system initialization phase, if a federate declares a subscription, sub-children (agents) are launched to all associated publishers (where the data of interests can be found most likely) according to the internal record of the

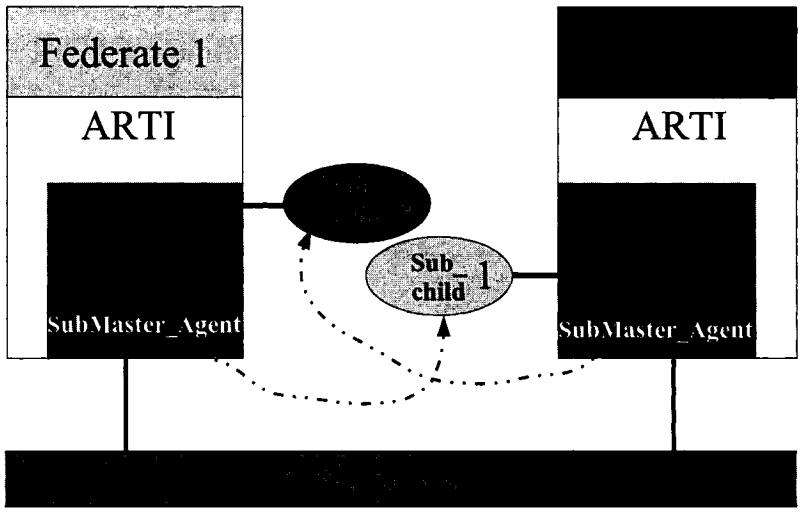

Figure 3.4: Design structure of the agent-based DDM

system's publication information provided by declaration management (DM). A SubMuster-agent is built to manage those Sub-children. When Sub-children have reached the publishers, they communicate directly with the federate environment. Thus, at each publisher, whenever update happens, the agent associated with it will fetch the data, perform data filtering and then send the subscriber exactly what it wants. At the same time, each agent keeps its internal filtering parameters updated by receiving notification from its owner (the subscriber launched it) for each subscription region modification. Sub-children employ the new filtering parameters from the beginning of the next processing cycle. As a result, only the information that falls into the new subscription region will be routed to the subscribing federate.

It should be noted that no update regions exist in the agent-based DDM. Since the agents deal with all the updated data and deliver only the data exactly fallen into the subscription region of its owner, no update region is needed to clarify the area of updates. Furthermore, compared with the other data filtering mechanisms, incorporating agent technology in DDM reduces the processing demand of filtering at the physical location of senders, minimizes the network traffic by filtering out all irrelevant data.

However, the agent-based DDM mechanism has some disadvantages. Firstly, the internal table of other federates' publishing interest costs storage space. Another disadvantage of the

agent-based DDM occurs in large-scale simulations. For a federation with large numbers of federates, hundreds or thousands of filtering agents may be sent out to one federate as a result of the initial matching operation based on the DM services. These large numbers of filtering agents, residing on one federate machine and performing filtering from time to time, will cause a heavy load to the federate machine. The agent-based DDM mechanism will become very inefficient in such cases.

3.5 Sort-based DDM

Sort-based DDM [49, 48] itself is not a whole solution to DDM, rather it essentially provides a way to compute the intersections among publication regions and subscription regions. However, it can be integrated into the original region-based DDM schemes to optimize their computational overhead introduced by the exhaustive matching. Experimental results show that the improvement on performance of sort-based matching ranges between 30% and 99% over the brute force and hybrid approaches [39].

Basically, in this algorithm, extents of different regions are sorted before computing the intersections as illustrated in Figure 3.5. This simple scenario shows four regions with their extents located in a two-dimensional routing space. The algorithm is to ascertain their overlap information. The interesting point of the algorithm is that the intersection is acquired dimension-by-dimension. Thus, each extent is first projected onto the x-axis, y-axis, etc,. in the way shown in Figure 3.5.

Now let us exam how does this matching algorithm work. Assume we have two region $R1$ and $R2$, their value ranges, $x.R1$ and $x.R2$, on dimension Di are $x.R1.1$ to $x.R1.2$ and $x.R2.1$ to $x.R2.2$ respectively. For each dimension Di of a routing space, if $R1$ and $R2$ intersect on dimension Di, there are only two possible cases as shown in Figure 3.5. In the first case, neither of $x.R1$ nor $x.R2$ is contained in the other. In the second case, one of them ($x.R2$ in this case) is contained in the other extent ($x.R1$ in this case). These endpoints, located in the other value

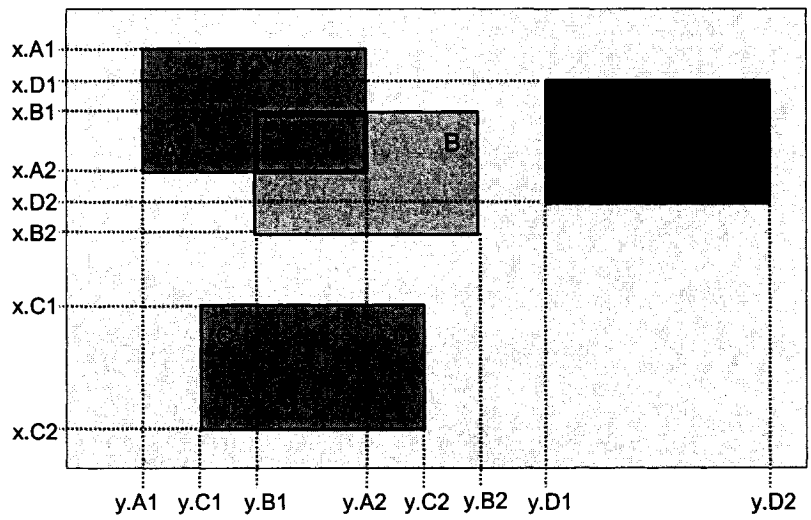

Figure 3.5: Basic scenario for the sort-based algorithm

range, are called internal endpoints. Other endpoints are external endpoints. Actually, overlap information can be obtained at the internal endpoint of each extent. For the first case, it is quite simple because each extent has exactly one internal endpoint. However, in the second case, both internal endpoints belong to $x.R2$. Fortunately, the overlap relationship is symmetrical, that is if $x.R1$ overlaps with $x.R2$, it is true that $x.R2$ overlaps with $x.R1$. Thus, the overlap information of $x.R1$ can be concluded at the endpoints of $x.R2$. This idea is adopted to solve the scenario as shown in Figure 3.5. Table I summarizes the overlap information of the x-dimension. After processing all the points of regions in the x-dimension, the information that $x.R1$, $x.R2$ and $x.R3$ overlap with each other can be obtained. After processing one dimension, this procedure is repeated for all the other dimensions in the routing space. The overall overlap information can be obtained by combining the information of each dimension: two region overlap, if and only if they overlap in all dimensions.

The detailed steps of sort-based matching is given as follows. First, construct a list for each dimension of the routing space. Each list contains the coordinates of all the regions for the associated dimension. For the scenario given in Figure 3.5, two lists are constructed.

$x-dimension$: $\{x.A.1, x.A.2, x.B.1, x.B.2, x.C.1, x.C.2, x.D.1, x.D.2\}$

$y-dimension$: $\{y.A.1, y.A.2, y.B.1, y.B.2, y.C.1, y.C.2, y.D.1, y.D.2\}$.

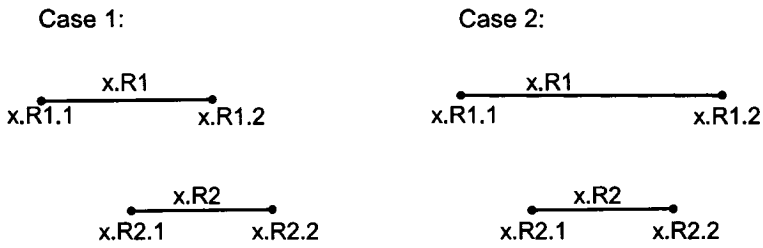

Figure 3.6: Overlap cases for two regions in one dimension.

Point	Point Overlap Information	Extent Overlap Information
x.A.1	/	/
x.D.1	x.D.1 overlaps with x.A	x.A and x.D overlap
x.B.1	x.B.1 overlaps with x.A and x.D	x.A, x.B and x.D overlap
x.A.2	x.A.2 overlaps with x.B and x.D	x.A, x.B and x.D overlap
x.D.2	x.D.2 overlaps with x.B	x.B and x.D overlap
x.B.2	/	/
x.C.1	/	/
x.C.2	/	/

Table 3.3: Overlap Information of x-dimension

Second, sort all the lists according to their coordinates in ascending order. In this example, the two lists will be

$x - dimension : \{x.A.1, x.D.1, x.B.1, x.A.2, x.D.2, x.B.2, x.C.1, x.C.2\}$

$y - dimension : \{y.A.1, y.C.1, y.B.1, y.A.2, x.C.2, y.B.2, y.D.1, y.D.2\}$.

Finally, scan each list from the left side to obtain the overlap information of each dimension. In the algorithm, a set S is employed to store extents currently overlapping with the point being processed. When processing a lower bound point, the extent is inserted into S. When processing an upper bound point, the extent is removed from S, and then the overlap information is written down [39].

3.6 Comparison of the alternative DDM strategies

From theoretical point of view, we can deduce that the region-based DDM algorithm is usually not very scalable because of the centralized DDM_Coordinator, high computational

and communicational overhead. When the total number of regions increases, its performance degrades heavily. The only case where it performs well is when there is a high probability of finding a pair of regions that intersect [29].

The fix grid-base DDM seems much more scalable compared with region-base DDM and has less computational and communicational overhead, however, its static multicast group assignment makes it inflexible and inextensible. More specifically speaking, it fails to manage the multicast groups in an efficient way, which causes bottleneck problem when the number of grid cells is large [31]. On the contrary, the dynamic grid-based DDM management multicast groups carefully, which makes it more scalable than fix grid-based method. Moreover, it also achieved better performance in terms of processing time, message overhead [13].

The hybrid algorithm is more scalable than both region-based and fix grid-based method. When the region size is much smaller than the grid size, it has the best results. However, the hybrid approach has a severe drawback: the size of the cell seriously affects its behavior. Even worse, it performs very poorly in situations where the size of the cell is relatively much smaller than the size of the region [39].

Both sort-based[6] and agent-based algorithms have better performance when compared with the region-based, fix grid-based and hybrid DDM under most situation. Their disadvantages is difficult to implement.

It is hard to compare the dynamic grid-based, agent-based and sort-based DDM approaches theoretically due to the complexities of the algorithms. In order to compare them, we need to implement each of them and run the same simulation using each method respectively to get accurate and practical results. However, this job is still leave to be finished in DDM community.

To sum it up, there are a number of viable DDM schemes. However, there is no "one-fit-for-all" solution, because each DDM solution has its own advantages and also disadvantages. Therefore, we can not elect the best one without specifying the simulation environment and

[6]Sort-based method ,itself, is a matching algorithm, and it can be combined with either region-based or grid-based DDM. Each way of combination might yield different performance result.

simulation application catalogues. It becomes necessary to clarify the differences between these DDM approaches discussed above. Since we have stated the pros and cons of the various approaches in the previous sections separately, we are not going to repeat here. Instead, I summarized them in Table 3.4 in a brief way so that we can gain a concise global view of the comparison of the alternative strategies.

DDM method	Filtering Techniques	Attributes
Region-based	Exact matching	High CPU processing load
	Centralized coordinator	Less irrelevant data transmission
		Limited scalability/ Potential bottleneck
Fixed grid-based	Grid overlay	High message overhead
	Implicit matching	Too many multicast group assignments
		Simple implementation
Dynamic grid-based	Grid overlay	Lower message overhead
	Distributed cell-manage	Fewer multicast groups
		Lower CPU processing
Simple hybrid	Grid overlay	Lower CPU processing
	Exact matching	Potential bottleneck
	Centralized coordinator	Less irrelevant data transmission
Agent-based	Intelligent agents	Low message overhead
	Exact Matching	Difficult implementation
Sort-based	Sort region extends	Low message overhead
	one dimension by one dimension	Difficult implementation

Table 3.4: Comparison of the alternative DDM strategies

Chapter 4

Optimized Dynamic Grid-based DDM Algorithm

As we already discussed in Chapter 3 section 3.2.2, the dynamic grid-based DDM approach exhibits a few appealing characters when compared to other DDM methods, especially region-based and fix grid-based schemes. However, when we examine this algorithm carefully, we find that a considerable amount of irrelevant data is injected into the networks for transmission under certain circumstance. Certainly, the DDM network performance will be compromised to some extend due to the extra data transmission burden. In order to cater for large-scale distributed simulations which demand high network performance, we wish to enhance its performance by enforcing it with a second level of sender-side data filtering mechanism. In this chapter, we will first discuss the weaknesses of the dynamic grid-base DDM approach. Afterward, we will propose the enhanced algorithm, which we refer to as optimized dynamic grid-base DDM algorithm. Then we present the implementation details of this DDM scheme and report on our set of experiments we have carried out to evaluate its performance.

4.1 Weaknesses of dynamic grid-based DDM algorithm

In this section, we wish to highlight some of the weaknesses of the dynamic grid-based DDM algorithm before we propose a solution to overcome these weaknesses.

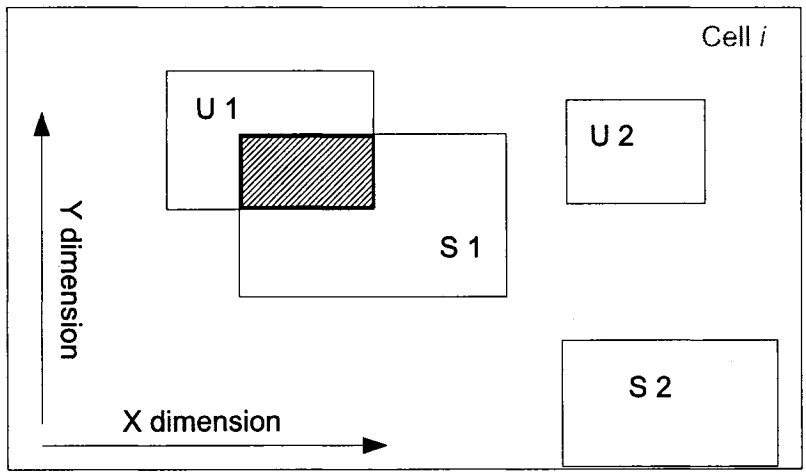

Figure 4.1: A grid cell with multiple regions

Consider the situation of a grid cell i shown in Figure 4.1. Besides the overlapped publication region $U1$ and subscription region $S1$, there still irrelevant publication region $U2$ and subscription region $S2$. According to dynamic grid-based approach, a multicast group is assigned to this cell in that publishing and subscribing regions co-exist in the same grid cell, therefore, all the federates that publishing or/and subscribing to the cell must be triggered joined the multicast group at the current time step. Under this circumstance, two types of irrelevant data transmission will occur:

1. Irrelevant federates are forced to join the multicast group to participate with the data transmission. For scenario in Figure 4.1, the publisher of $U2$ has to send data even though no data consumer is interested in $U2$ and the subscriber of $S2$ has to receive data from both region $U1$ and $U2$ which are totally useless and have to be discard at this federate locally.

2. Within the overlapping regions, needless data are being sent and received. Such as the data presented by the blank area of region $U1$. Even there is an overlapped area between

$U1$ and $S1$, the blank area of $U1$ is out of scope. It is just a waste of network bandwidth of send out this part of data.

Apparently, sending and receiving the needless data increase the communication (in case 1 & 2) and computation (in case 2) overhead to DDM services, and in turn degrade the performance of DDM in large-scale distributed simulations.

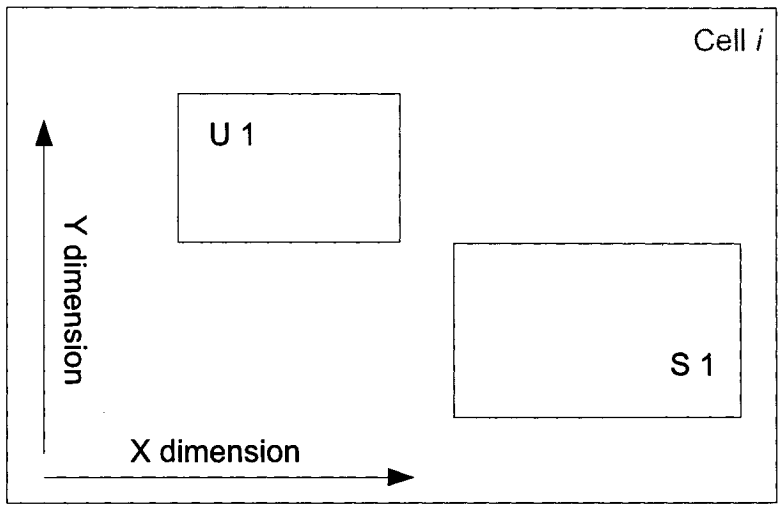

Figure 4.2: A cell with sub/pub but no intersection

Furthermore, the dynamic grid-based approach assigns unnecessary multicast groups, such as the case shown in Figure 4.2. Since the publishing and subscribing regions ($U1, S1$) coexist in this grid cell i, a multicast group is assigned to this cell even there is no intersection existing in this cell, which means the communication bandwidth is occupied with no data transmission. As we already know that multicast address is a relative rare network resource, such an abuse of it might cause problems in a system where the phenomena shown in Figure 4.2 is common. Such as in a simulation where objects have small sensing range and the objects are sparsely distributed. This counts for another drawback that needs to be eliminated for large-scale distributed simulations. We need to provide a mechanism to reduce system resource consumption, improve scalability and increase efficiency of the dynamic grid-based DDM approach.

4.2 Proposed solution

Realizing the weaknesses discussed above, we make a modification to the dynamic grid-based approach to limit and control these needless data from sending and prohibit the unnecessary multicast group from creating. Our solution is to incorporate a second level of precise filtering mechanism into the dynamic grid-based DDM scheme.

Generally speaking, filtering can occur in two general locations: at the sender or at the receiver. If a publisher knows that another federate is not interested in an update, then that publisher could avoid sending the update to the federate and thus save the expense of sending the data. This is sender-side filtering. If the publisher does not know whether or not a federate needs an update, then the publisher must send the data and the subscribing federate's LRC must determine locally whether to deliver the update to the federate or not. This is a receiver-side filtering. The sender-side filtering paradigm is more desirable but is also more difficult and costly to compute. Often, DDM services using sender-side filtering gain higher performance. When updates and interactions are filtered out before they are sent, there are savings involved in the cost of sending the data (e.g., sender processing and network bandwidth), costs in receiving the data (e.g., receiver processing and filtering the data at the receiver), and finally the callback to the federate. When an update is filtered out at the receiver's side, only the callback to the federate is saved. Therefore, our modification is about sender-side using a more precise filtering mechanism.

In our enhanced DDM algorithm, each cell has an owner responsible for collecting publication regions and subscription regions information like the region-based method, and stores them in a special data structure. The information includes which federates are publishing or subscribing to the cell and the corresponding regions, which are used to detect the intersection regions. Cell owner allocates multicast group to a cell if and only if an intersection exists, thus reduces the number of multicast group created. This is different from the original dynamic grid-based approach, which assigns multicast group to a cell when the pub/sub regions coexist in the cell. The information of detected intersection regions and the related federates are

stored in a local publish/subscribe table. Whenever a change to the table occurs during the run time, the cell owner will send out join/leave message to trigger the corresponding federates to join or leave the multicast group. This method, which is complementary to the weakness mentioned above of dynamic grid-based approach, makes DDM only send out messages that are absolutely necessary and optimizes the utility of network resources.

4.3 Description of the optimized DDM algorithm

We are going to present the optimized dynamic grid-based DDM algorithm in details in this section. The local matching process for each cell, multicast group assignment and specific data structure to be used are introduced respectively.

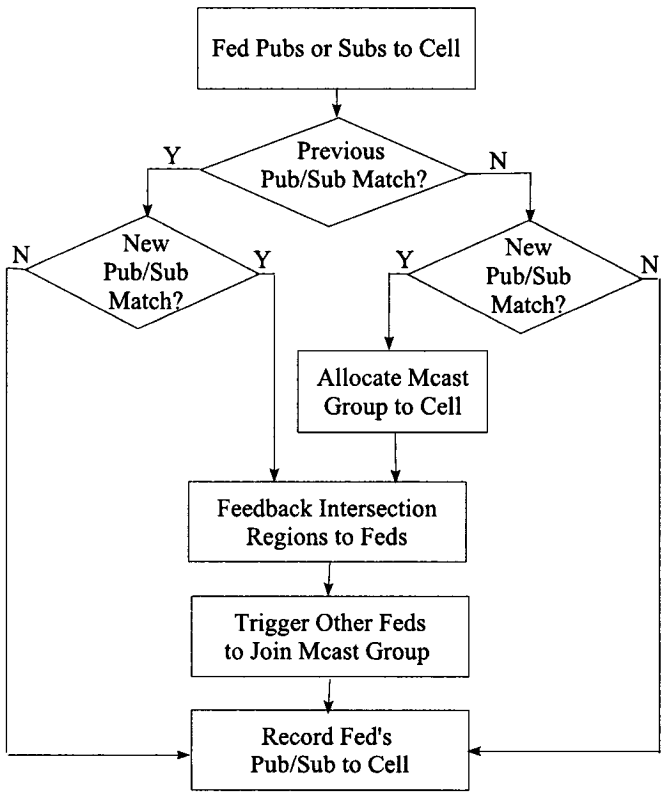

Figure 4.3: Execution steps of the proposed algorithm

4.3.1 Matching step of the optimized algorithm

Firstly, cells are evenly distributed among the nodes that are participating in the simulation. The node (or processor) to which a cell is distributed is said to be the owner of that cell. Cell owner keeps track of federates that are publishing and subscribing to each cell during the simulation execution.

The major change to the execution steps of the enhanced algorithm from the dynamic grid-based DDM, as shown in Figure 4.3, lies in the implementation details of the matching process. It adopts the exact matching like in the region-based DDM in every cell. To implement the exact matching in a cell, the cell owner needs the information of publishing and subscribing regions. We use two bit-arrays and a pointer list *Interest_Region* list to store those data at the cell owner node like shown in Figure 4.4.

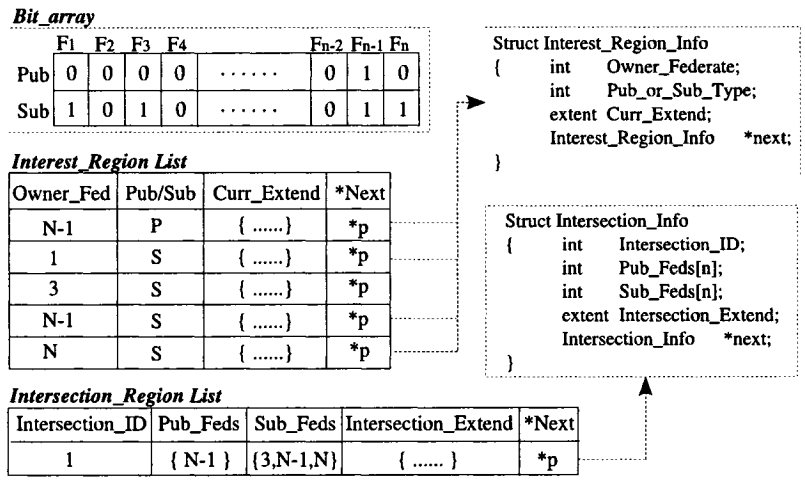

Figure 4.4: The data structure used in our DDM

The cell owner is responsible for matching the publishing regions and the subscribing regions in the pointer list to find intersection regions. Once intersection region (or regions) is found, the related information is store in the data structure *Intersection_Info* list shown in Figure 4.4, which will be used in the next step. The information stored in *Intersection_Info* includes intersection region ID, publisher, subscriber(s), the extend of this intersection, and the pointer to the next intersection of the list.

4.3.2 Multicast group assignment

The guideline is that a multicast group is allocated to a cell if and only if there is at least one subscription region overlaps with a publication region in that cell. At each time step, a cell owner is responsible for calculating the intersection regions when a change of regions in the cell occurs and stores the information of intersection regions such as the publishers, subscribers, and the extents of a certain intersection region in a local table, of which the *intersection region_ID* should be the index like data structure shown in Figure 4.4. A multicast group is allocated to a cell if and only if there is at least one intersection, and no more than one multicast group can be allocated to the same cell. After additional matching has been performed, if no intersection region exits any longer, the cell owner will release the assigned multicast group.

Once the cell owner detects any publisher/subscriber intersection in a cell, it first checks if there used to be an intersection in this cell in the previous time step.

- If there was no intersection:

  Cell owner creates a multicast group and assigns it to the cell. All federates publishing or subscribing to the intersection region(s) must be told to join the group.

- Otherwise, no need to create a new multicast group.

  According the result of new matching, a cell owner just updates the membership list of the multicast group already assigned to the cell in the previous time step.

The cell owner notifies federates to join a group by sending them a *join_group* message. The *join_group* messages for publisher and subscribers are different. In order that publisher only send out data of absolutely necessary, publisher need knowledge of the extend of detected intersection region. Therefore the extend of intersection region is in the *join_group* message to a publisher. Whereas the *join_group* message sent to subscribing federate is purely a notification. Upon receiving a *join_group* message, the local DDM manager will join itself to the multicast group specified in the message. Similarly, when a cell owner detects the cessation

of a publisher/subscriber intersection due to unpublisher/unsubscriber, it sends a *leave_group* message. Each time a *join_group* or *leave_group* message is sent, the cell owner will also update membership list of corresponding multicast group.

As we can see that by feed backing publisher the extend of intersection region, the optimized algorithm will send out data of absolutely necessary. The first weakness of dynamic grid-based DDM is overcome. Since principle of multicast group assignment to a cell is that if and only if there is at least one subscription region overlaps with a publication region in the cell, the second weakness is avoided too.

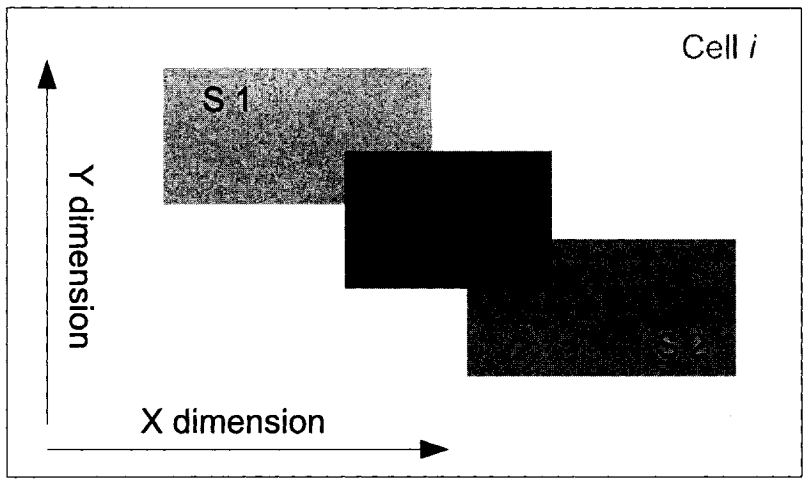

Figure 4.5: A cell with more than one intersections

Even though the optimized algorithm makes publishers send out data of absolutely necessary, it does not necessarily mean that no data filtering is needed at the subscriber's side. For example the case shown in Figure 4.5, there are two intersections associated with update region $U1$, thereby, publisher of $U1$ will send data presented in both of the two intersections. As a result, subscribers of $S1$ and $S2$ will receive all the data send out by the publisher. As a matter of fact, subscriber of $S1$ only need data in intersection 1, and same to subscriber of $S2$. Therefore, at receiver side, filtering is still necessary. However, the processing load of receiver-side filtering is less in the optimized algorithm compared with the dynamic grid-based DDM.

4.4 Performance evaluation

In this section, we present the simulation experiments we have carried out to evaluate the performance of our optimized dynamic grid-based DDM algorithm. During our experiments we wish to be able to study closely the behavior of the optimized protocol through comparing it with the original dynamic grid-based DDM and the region-based DDM which has beens implemented in RTI 1.3 [28].

4.4.1 Simulation environment

The optimized algorithm is implemented and incorporated with Grid-Kit [11], which was previously developed by our research lab using the Georgia Tech RTI-Kit [22]. The RTI-Kit contains a set of libraries designed to support the development of the run-time infrastructures (RTIs) for parallel and distributed simulations, especially federated simulations running on high performance computer platforms, such as multi-processor and clusters via myrinet. The Grid-Kit is a separate component integrated with the RTI-Kit performing DDM services with several DDM schemes, namely region-based, fixed grid-based, dynamic grid-based and the newly added optimized dynamic grid-based methods. It is envisioned that new DDM algorithms can be added to the Grid-Kit making it more flexible to different DDM requirements of various simulations.

We simulated the Tank dogfight scenario, which is typically employed in researches on data distribution management in parallel and distributed simulations. Initially, two groups (red vs. blue) of the objects are randomly placed in the two-dimensional battlefield. Each tank moves with a constant velocity towards a randomly chosen direction at each time step. All tank objects are both publishers and subscribers.

For the experiments, we run the simulations on 3.0GHz Pentium IV IBM computers connected by 100Mb Etherent, running Linux Fedora Core 4. Six federates are spawned in each simulation run, and each federate simulates the same number of tank objects. Half federates

Parameters	Values
Number of federates	6
Battlefield size(km^2)	500
Sensor range(km)	8.0
Number of routing space dimensions	3
Object moving speed($km/timestep$)	4.0
Number of time steps	100

Table 4.1: Constant parameters for the simulations

model red tank objects and half model blue ones, therefore, the routing space has three dimensions: North-South, East-West, and Team. The battlefield is 500 km^2. Simulations consisted of 100 timesteps. Table 4.1 summarizes the parameters that were held constant throughout our experiments.

We have evaluated our optimized DDM protocol using the following performance metrics:

Metrics	Description
DDM time	The total time consumed by the DDM services
DDM messages	The total number of messages generated to perform the task of DDM scheme
Multicast groups	The number of multicast groups created by the DDM scheme used during the simulation

Table 4.2: Performance metrics

4.4.2 Experimental results and analysis

In the experiments described in the previous subsection, we wish to investigate the performance of our optimized dynamic grid-based DDM algorithm by comparing it with region-based and the initial dynamic grid-based DDM. Here we use the terminology UPSD (the number of units per spatial dimension) to express the division of the routing space. In our experiments the following equation holds:

$$Number\ of\ grid\ cells = UPSD^2 * 2$$

Basically, we conduct the experiments in two stages. In the first stage, we assign a constant number 200 to UPSD which mean we divided the routing space to 80000 cells, while vary the number of tank objects in each simulation run from 100 to 1000. Figure 4.6- 4.8 give the experimental results of the first stage simulations.

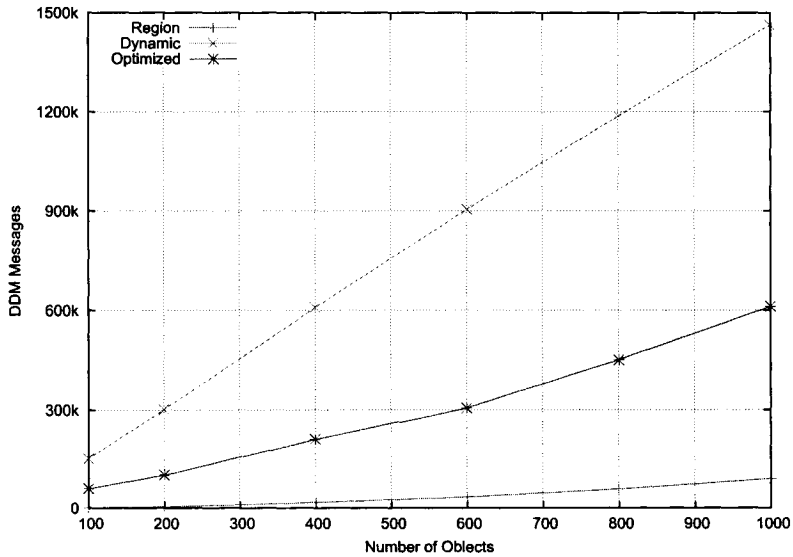

Figure 4.6: Number of messages vs. Number of objects

As shown in Figure 4.6, our optimized scheme has lower message overhead than the dynamic approach. This is the result of the introduction of the light-level-matching in the optimized scheme, which filters out further the unnecessary data to be transmitted on the network. Region-based DDM has the lowest message overhead due to its exact matching mechanism between publications and subscribe regions, which however causes higher time overhead. Figure 4.7 shows DDM time of the three schemes increase as the number of object grows. Region-based DDM has better time performance when the number of objects is below 400, while dynamic and optimized schemes outperform region-based DDM. The DDM time of the optimized scheme is slightly higher than that of dynamic DDM. As to the number of multicast groups, our optimized DDM lies between the dynamic and region-based ones as shown in Figure 4.8.

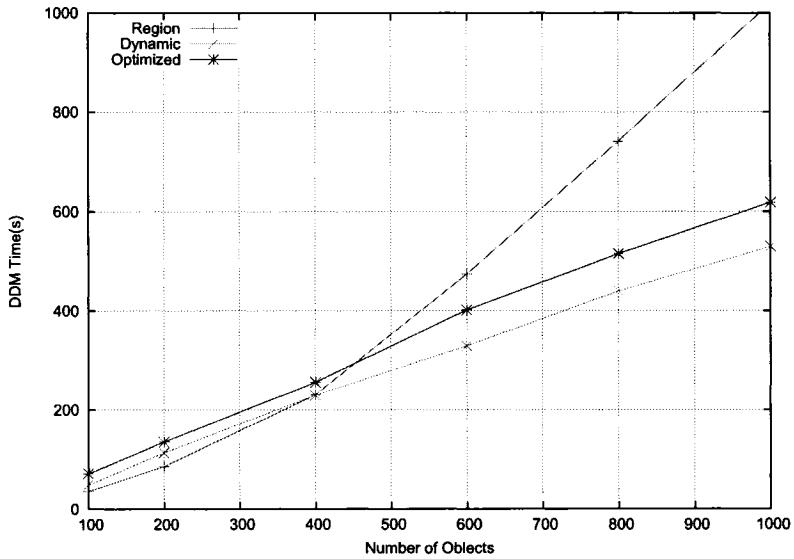

Figure 4.7: DDM time vs. Number of objects

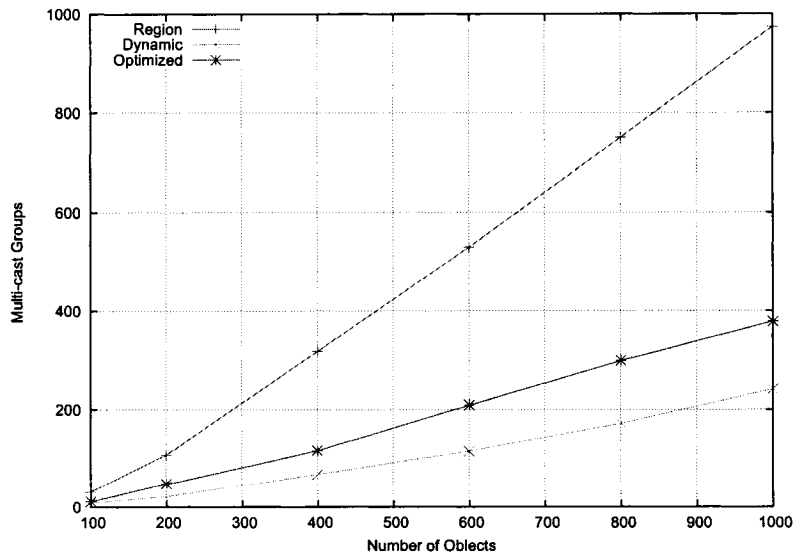

Figure 4.8: Multicast groups vs. Number of objects

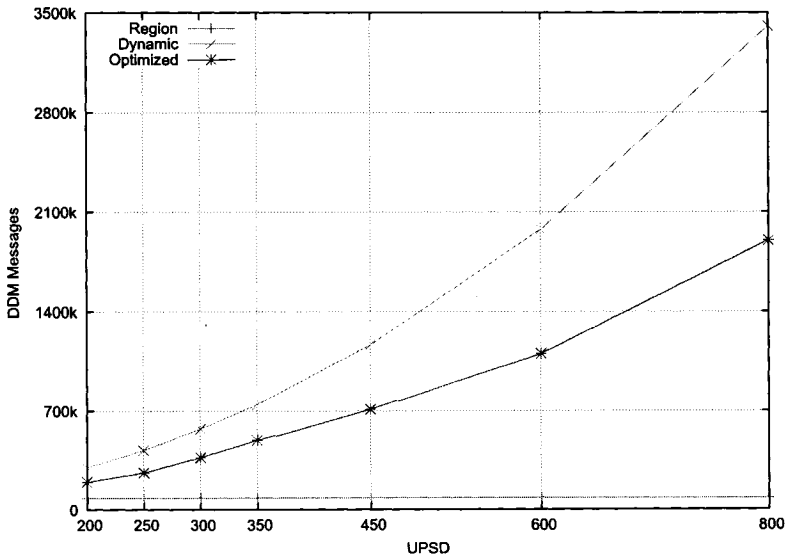

Figure 4.9: Number of messages vs. UPSD

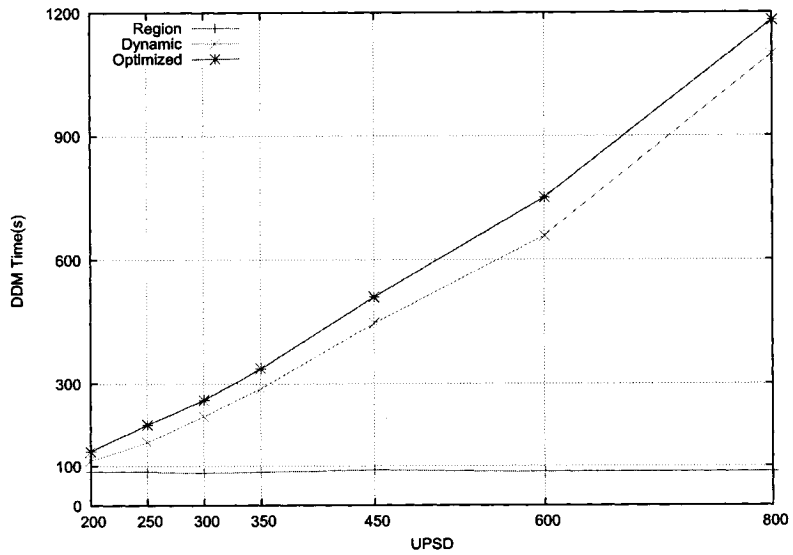

Figure 4.10: DDM time vs. UPSD

The second stage of experiments is as following: the number of UPSD is changing from 100 to 800 for each simulation run, while the number of tank objects is fixed at 200. We are trying to investigate the effect of changing number of grid cells on the performance of our optimized DDM. Figure 4.9- 4.11 give the experimental results of the second stage simulations.

From Figure 4.9- 4.11, we notice that varying the number of grid cells do not affect the performance of the region-based DDM scheme. For optimized and dynamic DDM, two metrics, DDM time and DDM message increase in a linear pattern showing that the two DDM schemes scale in terms of number of grid cells. The Multicast Groups used decrease somewhat in both optimized and dynamic DDM. This behavior is not surprising, because the decrease of cell size allows greater accuracy in detecting intersections, meanwhile, fewer intersections detected, and fewer multicast groups created. We can also observe that similar to the first set of experiments, the performance of our optimized DDM lies between that of region-based and dynamic schemes.

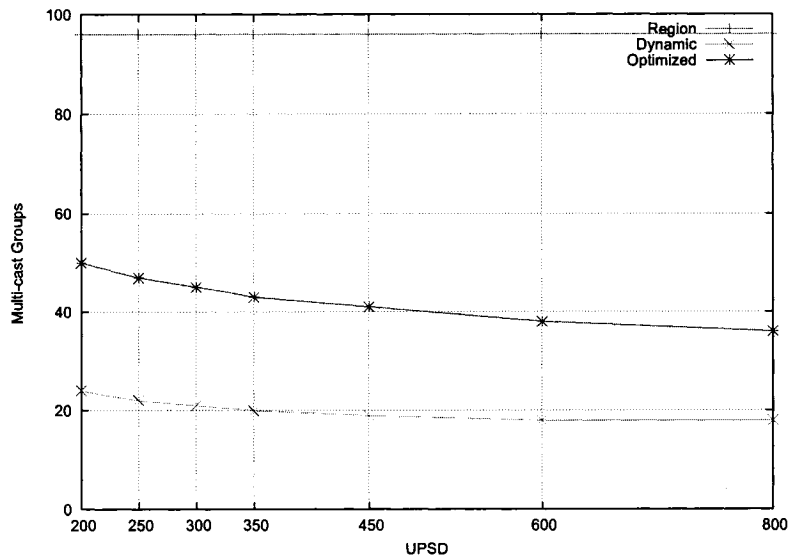

Figure 4.11: Multicast groups vs. UPSD

The experimental results from both the two stages demonstrate that our proposed optimized dynamic grid-based DDM algorithm has lower message overhead compared with the dynamic grid-based DDM approach, while lower time overhead and less multicast groups cre-

ated when compared with the region-based DDM. This is due to that a light level of exact matching is employed in the optimized algorithm, which makes it achieve certain trade-off between the two DDM schemes, namely region-based and dynamic grid-based DDM.

Chapter 5

Grid-filtered Region-based DDM Algorithm

Several DDM methods have been introduced in previous chapters, but time performance, message volume and resource usage continue to be factors in the practical application of these methods. In an effort to offer a more efficient and more scalable solution to DDM, we propose another DDM algorithm which we refer to as Grid-filtered Region-based DDM, that utilizes a grid overlay on the routing space, determines the percentage of grid covered by the subscription or publication region and further filters, based on a percentage threshold, using a matching technique, like that of the Region-Based DDM scheme. We present the implementation details of this DDM scheme and report on our set of experiments we have carried out to evaluate its performance.

5.1 Overview

Data sharing and management, across a large-scale distributed simulation environment, requires message passing between processors and the Run-time Infrastructure, to coordinate causality and state information throughout a simulation execution. Simulating many entities across many different hosts can increase communication across a network, on the scale of $O(N^2)$, where N is the number of processors or hosts [42, 29]. Data distribution management is a service that seeks to control the volume of messages exchanged during a simulation, thereby decreasing the workload on the simulation hosts. The key to reducing message volume in a large-scale distributed simulation is to find an efficient way to determine what data is relevant to which simulation hosts and relay state updates and interactions only to those applications that require them [9, 14, 10, 13, 11, 42, 20, 29, 27, 16, 43].

The key to efficient DDM is finding a way to limit the messages received by a host, to only those messages containing data regarding entities of interest to that host. Efficiency in data exchange is key to meeting this objective. In a system with no DDM, all hosts will receive update messages anytime participants change position, regardless of the proximity of the participant to other users in the simulated environment. The transmission of this unnecessary data is costly, in terms of resource expenditure. A host receiving a high volume of unwanted data, wastes valuable processing time receiving and reading that data. Likewise, sending this unnecessary data burdens the network with useless messages and taxes the sending host with sending and tracking those messages [13, 11].

We are developing and implementing a method of DDM that combines the low cost processing benefits of Grid-Based DDM with the high accuracy of Region-Based DDM, adding a percentage threshold concept to improve efficiency. This implementation is on our mini-RTI Kit and will be tested using a tank dogfight scenario, running many simulations that vary the number of federates, objects in the simulation and the number of grids.We analyze the results of these simulations, utilizing the Region-Based, Fixed Grid-Based, Dynamic Grid-Based and our new Grid-Filtered Region-Based DDM, in an effort to find a more efficient and

scalable DDM method.

The objective of this chapter is to compare results of the Grid-Based and Region-Based methods of DDM, currently implemented in various RTIs and present a new method of DDM that combines elements of Grid-Based and Region-Based DDM, in a distributed simulation environment. The Grid-Filtered Region-Based DDM method uses matching, rather than a DDM_coordinator, to determine intersection regions. The following section presents this new algorithm and describes the differences and advantages of this algorithm over the Region-Based DDM method.

5.2 Grid-filtered region-based DDM

We propose a new method of DDM we call Grid-Filtered Region-Based DDM. This scheme utilizes a grid overlay on the simulated environment, like all grid-based DDM [13, 31], but determines the percentage of the grid covered by the publication or subscription regions of federates responsible for objects positioned within that grid. Based on a percentage threshold, which is a concept we will describe in more detail a little later in this section, the DDM mechanism determines if matching should be performed to further filter data before a message is sent from the host simulating the publishing entity, to the host simulating the subscribing entity. In the subsections that follow, we will define the data structures used to implement this DDM method and discuss the potential advantages over a Region-Based DDM implementation.

Before we proceed further, let us define the parameters we shall use in our implementation.

- **threshold**: Cut off value to trigger matching. If a region covers a percentage of a cell higher than threshold, the region will automatically match the entire cell. Otherwise, each entity in the cell is explicitly checked against the region for overlap.
- **Cells**: Two dimensional array of cells. Indexed by row and column
- **cell$_{ij}$.ID**: unique integer identifying cell at position (i,j) in grid Cells.

- **cell$_{ij}$.P_fullcoverage**: Set of publishers whose coverage value exceeds threshold.
- **cell$_{ij}$.S_fullcoverage**: Set of subscribers whose coverage value exceeds threshold.
- **cell$_{ij}$.P_partcoverage**: Set of publishers whose coverage value is less than threshold.
- **cell$_{ij}$.S_partcoverage**: Set of subscribers whose coverage value is less than threshold.

5.2.1 Initialization

The initialization phase distributes the responsibility for managing cells among the available federates. This is done by using a mod operation. For example, if there are N federates, federate 0 will be responsible for managing cells 0, N, 2N, etc.

For each $c_{ij} \in$ *Cells*

Make federate (c_{ij}*.ID mod NumFederates) the manager of cell* c_{ij}

Initialize Simulation

5.2.2 Object publication/subscription updates

Object O_i on federate F_m updates its publication or subscription region in routing space when a simulated entity's publication or subscription region changes, and the set of cells that entity subscribes or publishes to changes as well. The cell manager has to be updated with the proper information including the percent of cell covered by the region. When object O_i on federate F_m updates its publication or subscription region in the routing space, the following comparisons are made.

First comparison of cell boundaries and region boundaries determine the set of cells, Rcells, where an overlap occurs. Secondly, the percent of cell area overlapped by the region is calculated. This is done by finding the smallest area defined by the region and cell boundaries. The calculation assumes a coordinate system such that (0, 0) is the low left corner of the simulation space. The calculation determines the vertical range by using the minimum of the upper coordinate of the region and the upper corner of the cell and the maximum of the

lower coordinate of each. The horizontal range is found using the minimum of the right edge of both cell and region boundaries and the maximum of the left edge of each. These ranges are multiplied to find then area of the overlap which is then divided by the cell area[2]. Once this is calculated, a notification of cell entry is sent to the federate managing the cell including the percent covered and the region's boundaries.

F_m:

Map region to set of cells Rcells such that

if any part of the region overlaps a cell c_{ij}, c_{ij} *is an element in Rcells.*

R_lft = region.leftbound; *R_rght = region.rightbound;*

R_upr = region.upperbound; *R_lwr = region.lowerbound;*

For each $c_{ij} \in Rcells$

{

$cell\_area_{ij} = (c_{ij}.upper - c_{ij}.lower) * (c_{ij}.right - c_{ij}.left);$

$P11 = (min(c_{ij}.upper, R\_upr) - max(c_{ij}.lower, R\_lwr));$

$P12 = (min(c_{ij}.right, R\_rght) - max(c_{ij}.left, R\_lft));$

$percent\_covered_{ij} = \frac{P11*P12}{cell\_area_{ij}};$

Identify owner F_k *of* $cell_{ij}$;

Send to F_k

cell_entry (R_lft, R_rght, R_upr, R_lwr, c_{ij}, $percent\_covered_{ij}$, *type, and* F_m);

}

5.2.3 Cell entry update

When a federate receives notification of an entity entering a cell, it uses the information sent with the notification to add the entity to one of four lists. If the type is publisher it adds the entity to one of its two publisher lists. If the type is subscriber its added to a subscriber list. The two publisher lists divide the entities based on percent of the cell covered by the region. If the

[2]Note: IF cell areas are uniform and constant, this last division is unnecessary. Threshold can be measured in space rather than percentage.

percent exceeds the threshold value, it is added to the list indicating full coverage, otherwise its added to the partial coverage list. The same logic is applied to subscribers. The following describes the algorithm when federate F_k receives cell_entry(R_lft, R_rght, R_upr, R_lwr, c_{ij}, percent_covered$_{ij}$, type and F_m).

F_k:

if type is publisher

{

new publisher Pub = (F_m, R_lft, R_rght, R_upr, R_lwr);

if percent_covered ≥ threshold

c_{ij}.P_fullcoverage = c_{ij}.P_fullcoverage + Pub;

else

c_{ij}.P_partcoverage = c_{ij}.P_partcoverage + Pub;

}

else if type is subscriber

{

new subscriber Sub = (F_m, R_lft, R_rght, R_upr, R_lwr);

if percent_covered ≥ threshold

c_{ij}.S_fullcoverage = c_{ij}.S_fullcoverage + Sub;

else

c_{ij}.S_partcoverage = c_{ij}.S_partcoverage + Sub;

}

5.2.4 Matching

When a federate must perform matching on its cells, it performs the following steps. For each cell it first takes each subscriber in its partial covered list and performs an exact matching algorithm against each publisher in its partial covered list of publishers. It concatenates the returned list of matches with all elements in its full coverage list of publishers. It then triggers

these federates to join the proper multicast groups. The algorithm assumes multicast groups are uniquely identified by the publisher writing to the group. Then the algorithm creates a matched list of all publishers in the cell and instructs all subscribers in its full coverage list to join multicast groups for each publisher in the matched list. The code that follows describes matching on federate F_k.

```
For each c_ij managed by F_k
{
   For each s ∈ S_partcoverage
   {
      matched = match(s,P_partcoverage)+P_fullcoverage;
      s.F.trigger_federate_join(matched)
   }
   matched = P_partcoverage + P_fullcoverage;
   For each s ∈ S_fullcoverage
   {
      s.F.trigger_federate_join(matched);
   }
}
```

5.2.5 Match(s,P_partcoverage)

This function performs an exact matching of subscriber s and publishers in P_partcoverage. If an overlap between regions exists, the publisher is added to a list which is returned when all matches are found. If a match exists there are three cases:

- The publication region completely encloses the subscription region.
- The subscription region completely encloses the publication region.
- At least one edge of the publication region will intersect with at least one edge of the

subscription region. This is the case when there is only partial overlap between the two regions.

```
If exact_match = ∅
For each p ∈ P_partcoverage
{
  if (p.left ≤ s.left) ∧ (p.right ≥ p.right) ∧ (p.upper ≥ s.upper) ∧ (p.lower ≤ s.lower)
    exact_match = exact_match + p;
  else if (s.left ≤ p.left) ∧ (s.right ≥ p.right) ∧ (s.upper ≥ p.upper) ∧ (s.lower ≤ p.lower)
    exact_match = exact_match + p;
  else
  {
    if ((s.left ≤ p.left ≤ s.right) ∨ (s.left ≤ p.right ≤ s.right))
    {
      if (p.lower ≤ s.lower) ∧ (p.upper ≥ s.upper)
        exact_match = exact_match + p;
      else if (s.lower ≤ p.lower ≤ s.upper) ∨ (s.lower ≤ p.upper ≤ s.upper)
        exact_match = exact_match + p;
    }
    else if ((s.lower ≤ p.lower ≤ s.upper) ∨ (s.lower ≤ p.upper ≤ s.upper))
    {
      if (p.left ≤ s.left) ∧ (p.right ≥ s.right)
        exact_match = exact_match + p;
      else if (s.left ≤ p.left ≤ s.right) ∨ (s.left ≤ p.right ≤ s.right)
        exact_match = exact_match + p;
    }
  }
}
```

5.2.6 s.F.trigger_federate_join(matched)

This function causes the federate modeling subscribers to join multicast groups indicated by the matched set. s.F indicates the federate controlling entity s. The algorithm assumes multicast groups are named based on the publisher.

For each publisher p in matched

s.F.join_multicast_group(multicast$_p$)

5.3 Experimental platform

For the experiments, we run the simulator on 4 and 10 Linux Fedora Core 4 machines, to measure performance differences, based on the number of participating federates.

While the grid-filtered region-based algorithm (Hybrid) is the one of primary interest, we repeated all experiments, using the dynamic grid-based (Dynamic), fixed grid-based (Fixed), and region-based (Region) approaches, for comparison. All the grid-filtered region-based trials use a constant threshold value of sixty percent, to determine whether a grid match is enough to trigger a subscription or whether the match needs to be confirmed with region-based matching. This value was selected to help differentiate the grid-filtered region-based approach from the region-based in trials with large cells, and from the grid-based approaches when the cells are small.

All experiments are of the Tank dogfight scenario type [13, 11], so all objects on all processors are both publishers and subscribers. Half the processors model red objects and half model blue objects.

5.3.1 Mini-RTI kit

In this section, we wish describe the relationships between our mini-RTI and the Georgia Tech RTI Kit [22]. The primary purpose of our mini-RTI Kit is to serve as an interface to the Grid-Kit [13]. For this reason, our mini-RTI Kit supports only the data distribution man-

agement service. We implement a component called Grid-Kit that integrates with the RTI Kit and performs DDM using the Grid-Based DDM schemes. A user may call upon the various components of the RTI Kit together, or individually, depending on their simulation needs. In this context, the user is an RTI builder or researcher.

Our mini-RTI (based upon Georgia Tech RTI Kit) uses the Grid-Kit that we developed and utilizes, for the purpose of this research, the region-based DDM and the grid-filtered region-based DDM methods to communicate data in a collaborative virtual environment, simulating a shopping mall setting. In addition to using the Grid-Kit, the mini-RTI also uses certain components of the RTI Kit [22]. Of the several RTI Kit components completed, the one that is most important to our work is the MCAST library (MCAST-Lib).

5.3.2 Multicast group communications

The MCAST-Lib component of the Georgia Tech RTI Kit [22] provides our mini-RTI with group communication services, using the multicast paradigm. Currently the MCAST-Lib implements multicasting in software, meaning that the network does not need to be multicast-enabled in order to use it. MCAST-Lib achieves communication over the network as follows. It uses another component of the Georgia Tech RTI-Kit, the FM-Lib, which is communication layer software, API based on the Illinois Fast Messages (FM). FM-Lib can be compiled using various transport protocols. We use the TCP/IP in our implementation. Note that optimizations made to the MCAST-Lib will not affect the mini-RTI, as long its API remains unchanged. If additional functionality is added to the MCAST-Lib, we can modify our mini-RTI accordingly to take advantage of these properties. *Delete* multicast groups is foremost among the enhancements to the MCAST-Lib that could benefit our work. There is currently no way to indicate that a particular multicast group is no longer being used.

To circumvent the inability to delete multicast groups once they are no longer needed, the mini-RTI recycles such groups. Once an allocated group is no longer in use, the mini-RTI places it on a free list. Before creating a group, the mini-RTI checks the free list and removes

	Federate	Mini-RTI	Grid-Kit
Responsibility	Simulates Objects	Maps Regions to Cells	Manages Cells
Steps (Objects)	1-Performs Operation with Region	2-Performs Operation for Each Cell Region	3-Updates Data Structure Based on Operation

Table 5.1: Component interface

a group from that list and reuses it. A new group is only created and initialized if the free list is empty.

It is the mini-RTI, by way of the Grid-Kit, that determines when federates should join and leave groups. However, the MCAST-Lib manages the actual membership lists for each group. In order to test the Grid-Kit, we develop a mini-RTI that uses the Grid-Kit to perform DDM. This RTI does not perform all of the services specified by HLA, and is not meant to be a complete RTI [13].

In what follows, we describe how the mini-RTI uses the Grid-Kit to carry out DDM. Recall that a Grid-Kit component is present on each node on which the mini-RTI runs. Since the mini-RTI links to each federate executable program, the mini-RTI, and the Grid-Kit, runs on all nodes that participate in the simulation. One of the primary functions of the Grid-Kit is to represent the grid cells and distribute these cells among the Grid-Kit components running on each node. There is no centralized representation of the grid, which is one of the strengths of our approach.

Note that both the cells that the Grid-Kit manages and the federate simulated entities reside on each node. However, the federate is not aware of the grid cells, since a federate should not make assumptions about how an RTI is performing DDM. Just as the federate is designed to work with different RTIs, we design the Grid-Kit to easily integrate into different RTIs. For this reason, the Grid-Kit is not aware of individual entities, only federates. The RTI is responsible for managing information about the federate simulated entities, if desired, and the

mini-RTI performs this task. Federates express interest in terms of regions and the mini-RTI translates these regions into the cells they overlap. For each cell, the mini-RTI sends a request to the Grid-Kit managing that cell, directing it to publish, subscribe, unpublish, or unsubscribe that federate to or from that cell. We represent the interface between the federate, mini-RTI, and the Grid-Kit on a particular node in Table 5.1. The Grid-Kit is technically part of the mini-RTI, although, in this context, the term mini-RTI refers to all mini-RTI components, except those that are part of the Grid-Kit. Referring again to Table 5.1, the federates and mini-RTI involved in steps 1 and 2 are both on the same Node X. However, the Grid-Kit, that performs step 3, is not necessarily on Node X. For example, the region used in step 1 is mapped to one or more cells in step 2. These cells are either managed locally, by the Grid-Kit on Node X, or remotely, by the Grid-Kit on other nodes. For remote cells, the mini-RTI on Node X sends a message to the remote node requesting the appropriate operation be performed on the specified cell. For local cells, the mini-RTI simply makes a function call that invokes the local Grid-Kit [13].

5.4 Experimental results

Again the tank dogfight scenario is used here. Simulations consisted of 100 timesteps and the parameters that were held constant throughout our experiments are the same as chapter 4. we looked at our grid-filtered region-based DDM method (Hybrid), compared to the region-based (Region), dynamic grid-based (Dynamic) and fixed grid-based (Fixed) DDM methods, when we varied the number of participating federates, the number of objects simulated, the grid size over the terrain and the percentage threshold. Specifically, we looked at the number of messages generated during the simulation time, the number of multicast groups used and total processing time to see which DDM method produced a reduction in resources used and efficiency in processing. The results of our experiments showed favorably toward a reduction in the number of messages, number of multicast groups used and more efficient processing

time using the grid-filtered region-based DDM method, compared to the region-based, dynamic grid-based and fixed grid-based DDM methods. An analysis of the following figures supports this assessment.

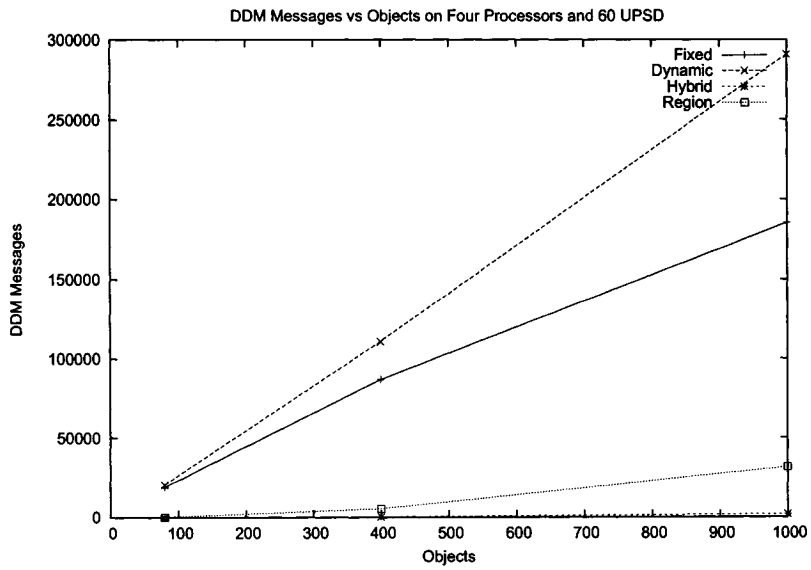

Figure 5.1: Number of messages vs. Number of objects with 4 federates

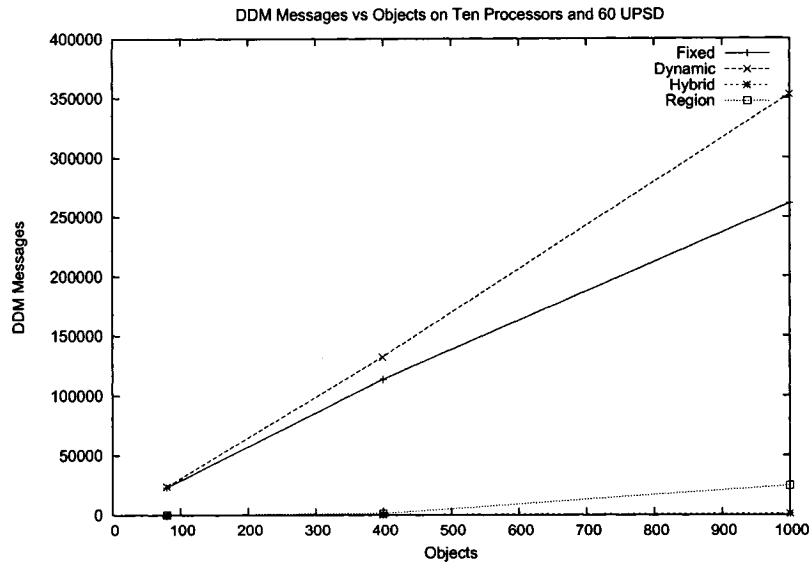

Figure 5.2: Number of messages vs. Number of objects with 10 federates

First we will discuss the results of our comparison between the number of DDM messages when we vary the number of objects and the number of processors. Figure 5.1 shows the

results of simulation runs with the UPSD set to 60, varying the objects from 80, to 400 and to 1000 and measuring the number of DDM messages generated during the simulation. Reviewing Figure 5.1, we see a steady increase in the number of DDM messages generated when the region-based, and grid-based DDM methods are used, while the line representing the grid-filtered region-based DDM method has very close to a slope of zero. Figure 5.2 shows similar and slightly improved results using ten processors, as we increase the number of objects.

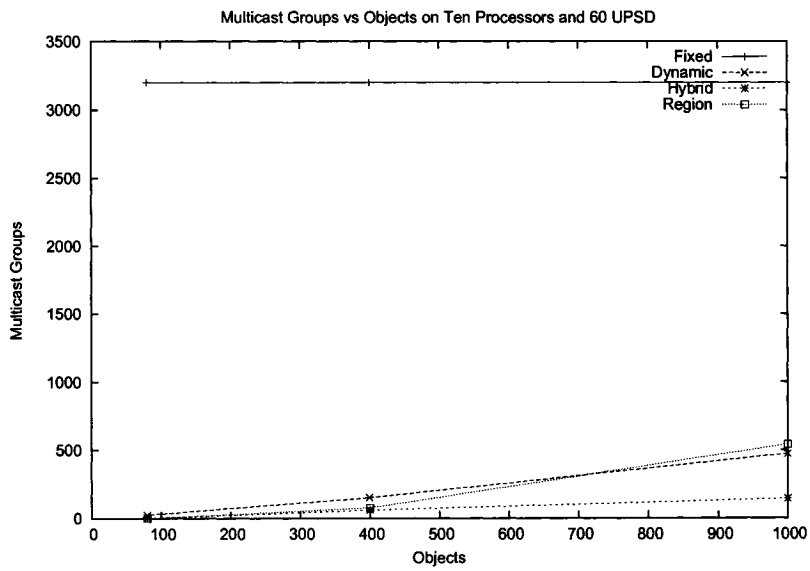

Figure 5.3: Multicast groups vs. Number of objects with 10 federates

Figure 5.3 and 5.4 show respectively the results of multicast groups consumptions and time overhead while varying the objects from 80, to 400 and to 1000 under the settings of 10 federates and 60 UPSD for each simulation runs. It is demonstrated that the fixed grid-based DDM is outperformed tremendously by the other three DDM algorithms including grid-filtered region-based DDM method, in term of both multicast groups consumptions and time overhead. Whereas grid-filtered region-based, dynamic grid-based and region-based DDMs have similar performance of both the two metrics according to the simulation results.

In summary, the first two figures together suggest that an increase in the number of processors and the implementation of the grid-filtered region-based DDM method will improve on the number of DDM messages generated by more accurately filtering data to subscribing fed-

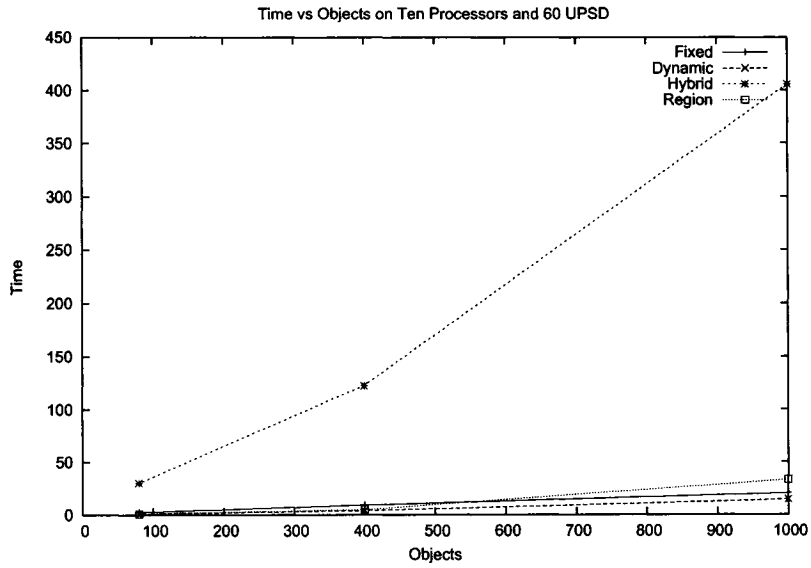

Figure 5.4: DDM time vs. Number of objects with 10 federates

erates, thereby reducing the number of spurious and unnecessary messages in a distributed simulation. When it comes to multicast groups consumptions and time overhead, grid-filtered region-based, dynamic grid-based and region-based DDMs have similar performance and are much more better than fixed grid-based method.

Chapter 6

Real-time Enabling Scheme to RTI for HLA-compliant Simulations

HLA, a DoD high level architecture, is emerging as an IEEE standard platform for large-scale distributed interactive simulation systems. It mainly emphasizes on the reusability and interoperability of different simulation components. Run Time Infrastructure is the middleware software that conforms to HLA Interface Specification (HLA/IF) by providing services necessary to support HLA-compliant simulations. HLA was well designed and has been proved to have the ability to guarantee the desirable properties of reusability and the interoperability in distributed simulation systems from several aspects. However, it shows apparent weaknesses when it comes to distributed time critical applications. Indeed, there are no specific rules provided with respect to real-time applications in HLA specifications. In this chapter, we propose a real-time RTI design, based upon a multi-threaded process model together with a global scheduling service, that address the weaknesses of HLA/RTI in terms of real-time applications. We discuss our real-time mechanism, and highlight the main design issues of our real-time RTI with an emphasis on the real-time RTI end-system.

6.1 Motivation

Run Time Infrastructure (RTI) is the middleware software that conforms to HLA Interface Specification (HLA/IF) by providing services necessary to support HLA-compliant simulations. To the best of our knowledge, there are no specifications addressed in HLA dealing with real-time distributed simulations, which is one of the many factors that limit the deployment of HLA/RTI in applications with real-time requirements. With the recent development of networking technology coupled with fast computers, real-time distributed and interactive simulations are about to enter the main stream. Obviously, developing the corresponding HLA/IF specification for time-critical applications is imperative.

The term "real-time", as it relates to simulation, requires that the computer program execution of a modeled and dynamic process must occur in the real-world time (i.e., not faster nor slower). With respect to the real-time RTI, it must have the ability to behave in a predictable way, and provide services to simulation components within required boundaries of response time [38]. This can only be achieved when all parts of an HLA simulation system behave deterministically [8, 47], and should be integrated in a predictable fashion in order to meet the time constraints throughout the distributed simulation exercise.

In this chapter we propose a novel approach to real-time RTI architecture, which enables RTI with real-time features to satisfy the requirements of time critical simulation applications by three means: *real-time RTI end-system* (including dynamic multi-threaded processing model, global scheduling service, and fixed-priority dispatching mechanism), *real-time enhancement of RTI services*, and *communication QoS assurance*. Our preliminary production on this project is real-time RTI end-system, referred to as RTrti. Therefore, for the time being, we focus upon RTrti component of the global design. Future work will be directed to deal with the two other issues.

The rest of this chapter is organized as follows. In section 2, we introduce briefly previous and related work. In Section 3, we outline the basic requirements for a real-time RTI, and present a global view of our proposed real-time RTI architecture. In Section 4, we shall de-

scribe in depth the main component of our real-time RTI, including the RTI internal processing model, the scheduling scheme and the fixed priority based dispatching which represent one of the most important module-RTrti in our architecture.

6.2 Previous and related work

With the advances of very powerful computers, and the increasing demands of powerful real-time simulations while ensuring the reusability and interoperability of different simulation components in both military and civilian applications, real-time distributed simulation is about to enter the main stream [46]. HLA has been proved to be a reasonable solution when developing large-scale distributed interactive simulation systems. However, HLA has no specific rules addressed with respect to real-time simulations. Therefore, there is no commonly accepted real-time implementation specification for HLA/RTI. Several efforts have been carried out to extend the HLA Interface Specification by including explicit support for real-time capabilities within the RTI. These studies are substantial and preliminary findings are quite promising for future real-time HLA/RTI based distributed simulation systems.

	Our proposed real-time RTI	McLean and Fujimoto [35]	Zhao and Georganas [50]	Bruzman and others [15]
Network QoS	Yes	Yes	Yes	No
RTI Multi-threaded asynchronous process	Yes	Yes	Yes	No
Preemptive priority scheduling	Yes	No	Yes	No
Globally Schedule Service	Yes	No	No	No
Real-time optimized RTI services	No	Yes(TM)	No	No
Special purpose transmission protocol	No	No	No	Yes

Table 6.1: Comparison of the techniques used in real-time RTI

Several ongoing studies [41, 35, 23, 15, 7] have proposed the use of special purpose transmission protocols within the RTI, thereby enhancing the RTI communication performance,

which is one of the critical requirements for a real-time RTI. However, in all of these studies, very few techniques have been employed to make RTI services more predictable in order to be suitable to real-time based applications. Basically, most of the proposed techniques fall into the following six categories: Network QoS [7], RTI multi-threaded asynchronous process, Preemptive priority scheduling [41], Globally scheduling service, Real-time optimized RTI services [35], and Special purpose transmission protocols [15]. In Table 6.1, we highlight the main differences among the different versions of real-time RTIs that have been proposed by now. In what follows, we shall present these real-time RTI based techniques and discuss their advantages and disadvantages as well.

- *Network QoS:* is a general concept that is used to specify the behavior of a network service. Designing service behavior by means of QoS control offers the advantage that the application developer indicates only "what" is wanted rather than "how" this QoS should be achieved [7]. Generally speaking, QoS is comprised of several QoS policies, each of which is an independent description. A list of independent QoS policies gives rise to more flexibility and more efficient resource utilization [24].

- *RTI Multi-threaded asynchronous process:* In this approach, RTI is usually incorporated with the use of multi-threaded processing paradigm, which is a basic mechanism for parallel processing in multi-processor platform and a fundamental part for preemptive priority scheduling to achieve high usage of system resources, and gain high scalability for the whole system [50, 1, 33]. This technique provides higher predictability of the RTI execution. Thus, it allows the RTI not only to run independently, instead of waiting to the federate's tick, but it also allows the local RTI component to execute different requests concurrently.

- *Preemptive priority scheduling:* Scheduling plays an important role in real-time RTI. In real-time applications, the RTI services should be predictable as well as be efficient in terms of resources' usage. Since it is not feasible to infinitely extend the required

resources [41], it is therefore important to be able to specify the available resources and provide policies that shall allow the real-time RTI to assign the resources to the most critical requirements. Preemptive priority scheduling might be a good policy for this type of purposes.

- *Global scheduling service:* In distributed real-time simulations, it is essential to provide a mechanism to allocate limited system resources the multiple requests to meet their time constraints and high performance requirements. As we shall later, our real-time global schedule service is designed for this purpose. Such a service library is responsible for allocating efficiently the system resources in order to meet the real-time scheduling requirements of the targeted application.

- *Real-time optimized RTI services:* the basic idea is to trim the RTI services making them suitable for real-time applications. Fujimoto et. al [35] have addressed the suitability of their real-time time management algorithm in a real-time RTI.

- *Special purpose transmission protocols:* are basically designed to meet the time constraints of real-time data transmission. There is no doubt that special purpose transmission protocols can satisfy the real-time application requirements within the RTI infrastructure. However, this is costly and may not be suitable for many applications [15].

6.3 Real-time RTI architecture

When designing real-time systems, time is a critical component in these systems. Therefore, it is essential to guarantee that the timing constraints of the system are met. Timing constraints are often expressed as deadlines at which processes have to complete their execution. Guaranteeing their timely behaviors requires that the system can provide services in a predictable fashion. Failure to met these constraints can lead to a degradation of the service[1] or

[1] As it may happen in soft real-time systems

a catastrophic damage to the whole system (as it may happen in hard real-time systems) [33]. Furthermore, it is highly desirable that the system attains a high degree of system resources utilization while satisfying these timing constraints [33].

6.3.1 Basic real-time RTI requirements

Recall that the RTI is responsible for providing HLA services to federate/federation and exchanging the updates and interactions among the federates transparently. With respect to the real-time RTI, it must have the ability to behave predictably and provide services to federates and federations within a required bound of response time. This can only be achieved when all of the HLA/RTI based simulation components behave deterministically and are integrated in an efficient way that can help to meet the time constrains throughout the distributed simulation exercise [50]. Based on the above discussion, we outline two basic requirements when designing real-time RTI-based distributed simulation systems:

1. The real-time RTI, acting as the middleware and providing HLA services, should support the policies and mechanisms for real-time applications to specify their end-to-end real-time requirements in a distributed environment. This is a pre-requisite for real-time simulations to meet certain timing constraints and mimic the targeted real-time application; and

2. Real-time RTI is responsible for enforcing those real-time requirements specified by upper level applications.

6.3.2 Proposed real-time RTI architecture

In order to guarantee the requirements discussed earlier, our proposed real-time RTI architecture is based upon the following three components. Figure 6.1 gives the global view of real-time RTI architecture. Every two components connected by bi-directional line can communicate with each other.

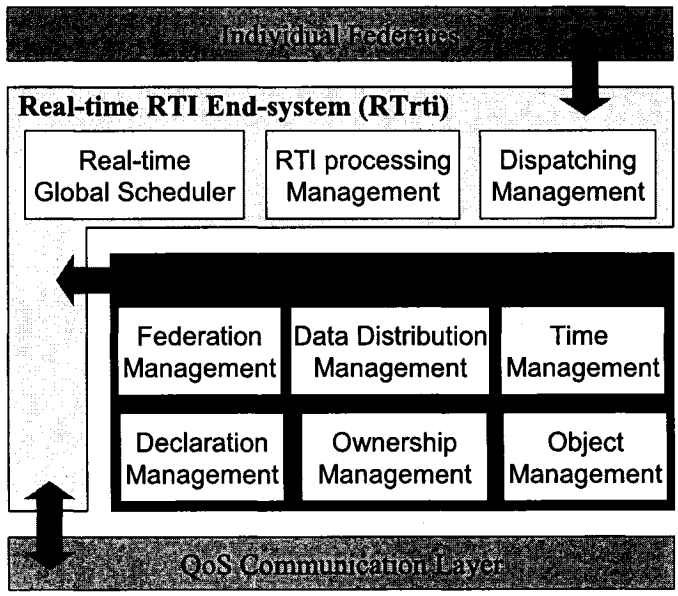

Figure 6.1: Global view of real-time RTI architecture

- *Providing predictable services.* Previous studies have mainly focused on the design and the implementation of the RTI services. We argue that preserving the predictable behavior of the HLA defined services is also very important, and it requires the optimization of all of the six RTI management services in order to reduce the unpredictable behavior of any real-time simulation. Fujimoto et. al. [35] have already addressed the limitation of the time management (TM) for real-time RTI based distributed simulation. They have made a modification to the previous TM LBTS calculation algorithm in order to ensure the effectiveness of the real-time simulation exercise. In our studies, we have proposed an optimized data distribution management [12] scheme based on dynamic grid-based DDM scheme [13]. Besides TM and DDM services, the remaining four HLA/RTI services should also be enhanced so that they could be more predictable.

- *Real-time RTI End-system.* A real-time RTI end-system, i.e., the local RTI component residing in federates, is necessary to manage the real-time RTI's process model. It is designed to provide standard interface specifications, and allow the simulation developer to specify the real-time requirements at a higher-level, in a more intuitive way rather than

having it within the OS and/or at the network level presentation. It is also responsible for scheduling the limited system resources, such as CPU, memory, I/O devices, to ensure that the real-time requirements of the applications are met, thereby, ensuring that all of the local and remote RTI services's requests can be accomplished within their deadlines. Thus, the real-time RTI end-system basically coordinates every component of the RTI so that these components cooperate in a predictable way in order to satisfy the real-time requirements as specified by the targeted application.

- *Communication QoS assurance.* To support real-time HLA/RTI based distributed interactive simulation, several performance metric have to be taken in consideration. Good throughput, low latency, and reliable message delivery requirements must be satisfied in any real-time systems, including real-time RTI. As the communication bridge among the involved federates, the real-time RTI must leverage a mechanisms in the underlying communication infrastructure that should provide a predictable network performance and thereby guaranteeing transportation upper latency boundary and high-bandwidth end-to-end for participating federates [50]. We propose to use a special transportation layer built upon the ACE communication framework [3], which supports adaptive transmission and priority-based connection.

Since the ACE communication framework is well designed and has been widely accepted by the real-time system community, the communication QoS assurance will not be our main concern in the current stage of this project. Thus, we shall assume that the communication QoS requirements of the real-time RTI is satisfied by just using the ACE communication framework within our real-time RTI architecture. Moreover, making RTI services to behave predictably is a big open issue in real-time HLA-compliant simulation community, thus it is out of the scope of this work. We currently focus on the design and implementation issues of real-time RTI end-system.

6.4 Real-time RTI end-system

In what follows, we shall present an overall of our proposed real-time RTI end-system (RTrti). The main components in RTrti are as follows: (*i*) multiple threads processing model using thread-pools, (*ii*) global scheduling services, and (*iii*) fixed-priority based dispatching mechanism. The real-time global scheduling services module provides the federate developers a high-level, and intuitive interface to specify their real-time requirements. It is also responsible for translating the application level requirements to the OS and network levels, which is one of the main priorities in our approach. Fixed-priority based dispatching module dispatches the requests according to their priorities. Finally, both scheduling and dispatching services are combined into a multiple threads processing model within the Local RTI Component (LRC) while providing to the RTI developers a strict control over the scheduling process, CPU, memory, and I/O resources in order to meet real-time simulation requirements.

6.4.1 Real-time RTI process model

Traditionally, HLA compliant simulation implementations use a single thread approach [44, 45]. In a single thread model, in order to interact with the RTI, a federate must transfer the control of the processor to the LRC by initiating the *tick()* function and regulate the execution time of the RTI by specifying the parameters within the *tick()* function. Without the *tick()* function's call, the LRC will never have a chance to run. However, it is not clearly defined when and where the *tick()* function can be called, which, in fact might significantly decreases the predictability of the RTI's execution, and thus it precludes the use of RTI in real-time distributed simulations. Furthermore, any federate that frequently calls *tick ()* in order to get updated and interact with LRC, will require significant system resources.

Current RTI's implementations use the two threads model [6, 23]. The second thread which is residing in the LRC component is basically responsible for receiving the messages on behalf of the low-level network [35]. Since the thread is called when a message is ready, and it

does not depend on the state of the federate thread (as opposed to the single thread model), the RTI behavior becomes more predictable when compared to the single thread model. However, the federates still need to call the *tick()* function in order to invoke RTI services and execute the callback functions. This means that the execution of RTI still depends on the state of the federate and thereby adds some unpredictability to the execution of the RTI.

However, these two approaches are not quite suitable for real-time based distributed simulation applications. In order to address their limitations and make the RTI behaving in a highly predictable fashion while dealing with the critical timing constraints, we argue that a fully multi-threaded model is necessary. Zhao and Georganas [50] have already suggested to employ a real-time RTI based upon a multi-threaded mechanism and a common thread-pool that manages the multiple threads. Some experimental results have shown that using the thread-pool mechanism exhibits a higher degree of predictability as well as a higher performance with a factor of 10 when concurrent tasks reached 1000 tasks [50]. Therefore, they have proven that the multi-threaded method is a promising paradigm to follow when designing real-time RTI based distributed simulation systems.

In what follows, we shall present our design of the internal processing model of real-time RTI end-system, which employs multi-threaded processing and thread-pool paradigm.

(a) A multi-threaded process model.

As in [50], we have decided to follow a multi-threaded paradigm to built our real-time RTI end-system, However, our design uses a different approach that was used by Zhao and Georganas [50]. In our scheme, at the federate's initialization phases, two mainstream threads are created and run simultaneously, where the first thread is responsible for retrieving the "down" call messages, that are requested by the local federates for RTI services, and the second thread is dedicated to polling the "up" call messages from the communication channel, which are the remote RTI service requests. Thus regardless of their respective priorities. Both "up" and "down" call requests will be handled equally. A request messages queue is assigned to each thread to store the arrival tasks.

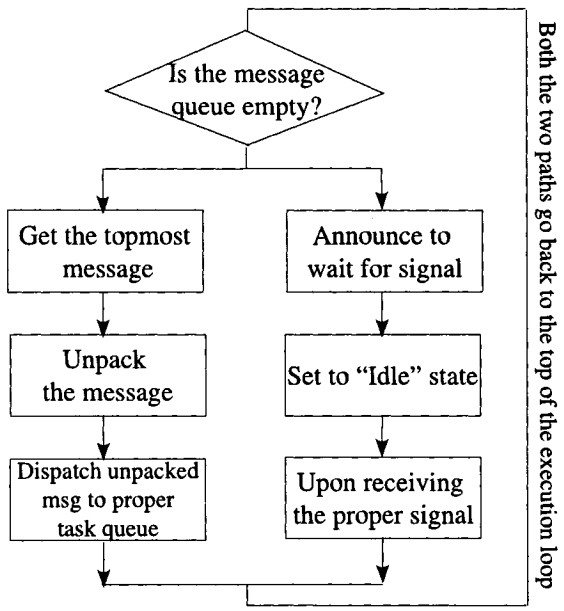

Figure 6.2: Process model of real-time RTI threads

The process model, as shown in Figure 6.2 is suitable for both type of threads. The main differences between them can be summarized as follows: (1) the execution of the "unpacking" block, mainly due to the policies for wrapping the "up" call and"down" call messages that could be different; (2) the unpacked messages are dispatched to different task queue; and (3) the threads basically serve their own waiting message queue.

Each thread keeps running within the execution loop except for external interruptions and exceptions. At the beginning, it checks whether its waiting message queue is empty. If the queue is empty, it will wait for a signal to get informed upon the arrival of a new request message, and then the thread is set to an "idle" state. However, before entering the idle state, a thread shall release all of the system resources that it is holding. Upon receipt of a new request message, a signal is sent to the thread, which will process the request message. On the other hand, if the queue is not empty, the thread will process the topmost request message from the queue, unpack the message according the predefined queue policy, dispatches the unpacked message to the corresponding task queue, and then repeat the same process all over again for the next arrival request message.

In this procedure, as we can see two additional resources are involved in, the invoking signal and unpacked message stack queue. In realizing signal mechanism, following the multi-threaded paradigm which allows threads to wait for some events to occur without wasting any CPU cycles while using the signal event feature, a conditional variable would be used. Note that the unpacked message stack queue is designed to store temporarily the unpacked messages of an RTI service request (during an"down" call message) or of an RTI initiated service that is implemented by the federate ambassador (during an "up" call message). In practice, the federate developers can pre-define a group of unpacked message queues to distinguish the different types of service requests, such as a queue for TM services, a queue for DDM services, etc., thereby facilitating the management of these unpacked request messages. The federate developers can also specify how to define the queue group according to their the targeted simulated system and its characteristics by writing the configuration to the RID file. Furthermore, developers can easily configure the system to spawn additional threads in addition to the two mainstream threads to serve these additional queues depending on the workload requirements, e.g., a single thread can monitor a single queue, or all of the queues based on a predefined priority discipline.

The multi-threaded processing model can help to optimize the resource usage quite efficiently when compared to the single thread model, and can handle concurrent threads by breaking up existing dependencies among the execution states of both the federate and the LRC component, thereby reducing the unpredictability that exists in either the traditional single thread or the two-threads processing model.

(b) Thread-pool.

As outlined earlier, in the multi-thread processing model, when a thread receives a high volume of tasks [2], such as the "down" call thread and the "up" call thread which are basically responsible for collecting the services' requests from local and remote federates in our real-time RTI process model, thread-pool is pre-allocated as opposed to the single thread model

[2]High volume of tasks refers to those occurring at a high rate and taking a relatively short execution time [41].

that handles itself the high rate events.

The thread-pool model is a collection of N threads, created at the beginning of an application and destroyed when the application terminates [32]. Real-time RTI should allow the configuration the RID file, so that thread-pool characteristics can be added when the pool is created. The thread-pool's configurable parameters includes several informations, such as (1) how many statically threads are be pre-allocated, (2) how many threads are allowed to be forked at run-time. When a request needs to be processed, a static thread of the pool will be assigned to handle it. If there are no more static threads available, a thread may be created to handle it. However, no additional threads can be created if the maximum numbers of threads have already been spawned. These statically pre-allocated threads will occupy the system resources even if that they are not used.

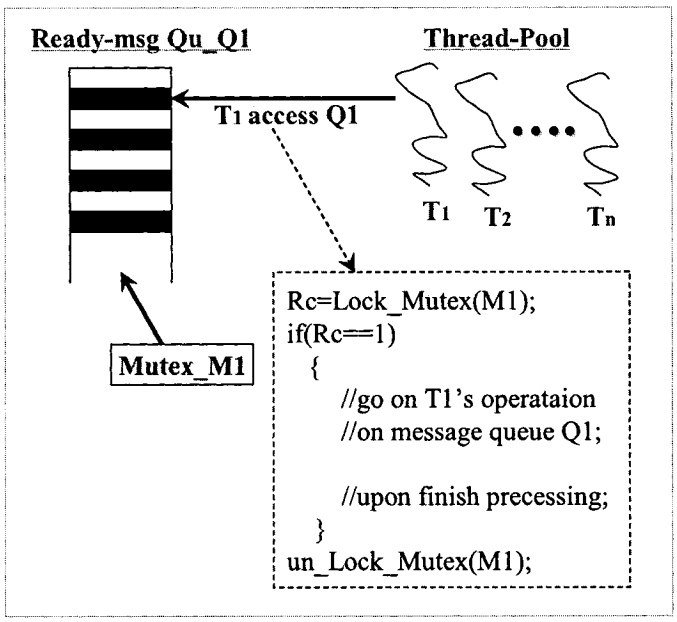

Figure 6.3: An illustrative example of thread-pol with Mutex

Note that the threads in a pool share the same memory space, the message queue, which demand us to assure exclusive access to the shared resources to maintain system consistency. A basic OS mechanism called "mutex" can solve this problem. Figure 6.3 illustrates how mutex works. A queue can be accessed by a thread only if it is locked successfully by that

thread. The thread should unlock the mutex after its operation on the queue. This mechanism ensures that each shared system resource can be accessed by no more than one thread at any given moment, maintaining a consistent system state.

6.4.2 Global scheduling service

The real-time RTI global scheduling service is responsible for allocating system resources to meet the real-time requirements of targeted simulation application. To fulfill this responsibility, the scheduling service should provide the federate developers a high-level, intuitive interface that can help them to specify their scheduling needs and the prediction of system resources' usage and be able to carry and pass on the application level requirements (e.g., deadlines, importance, worst case execution time, worst case communication delays, etc.) to the OS and network level presentations. Furthermore, the priority of message requests, in our design, is computed by the real-time RTI global scheduling service, and is based upon the real-time requirements, and system's resources that are priority based allocated.

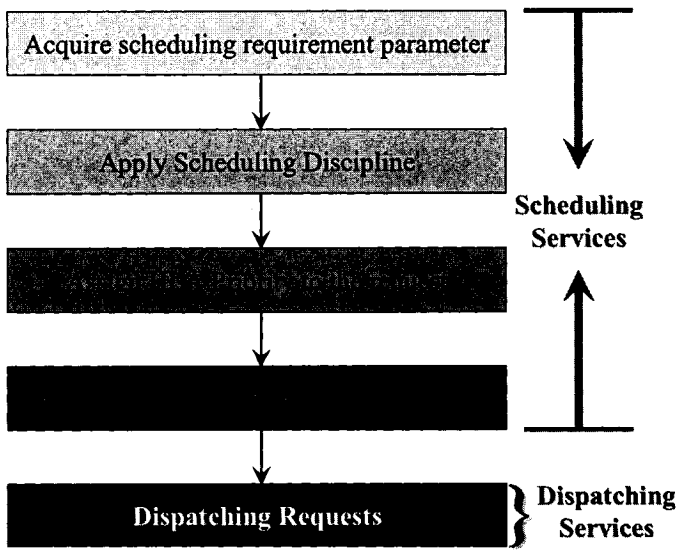

Figure 6.4: Steps of scheduling procedure

Several scheduling disciplines for real-time systems have been developed. These disciplines are to be implemented in the RTI schedule service library to cater for the scheduling

demands of different real-time simulation applications. Thus, specific information needed by the scheduling service will depend on which discipline is selected. While the request messages' priority discipline is used to sort these requests in the order they should be processed, the thread priority discipline will help to identify the ordering of the threads that will invoke the requests.

While taking advantage of the underlying operating system being used in our design, the scheduling service needs to map the HLA level thread priorities to the OS level priorities. The fixed-priority based dispatch mechanism, as we shall see later, is basically responsible for allocating the system resources to the request according to its HLA requests priority discipline and the OS level thread priority. Figure 6.4 illustrate the basic steps of the scheduling procedure.

The scheduling occurs independently on each federate, where remote requests will have to compete with the local system resources' requests according to their priorities that are assigned by the real-time RTI global scheduling service.

6.4.3 Fixed-priority based dispatching

The dispatching module shall dispatches first the request with the highest priority. Therefore, after computing the priority of the ready requests, a dispatching mechanism is used to dispatch the requests from the fixed-priority base. For instance, upon the arrival of a request, the dispatching service assign it to a thread within the "down/up" call thread-pool if there is an idle thread in the pool, otherwise it will insert it in the messages request's queue according to the highest priority predefined discipline.

However, to ensure that requests with the highest priority can always be processed first, when a request arrives with all of the threads in the thread-pool busy, the request with a lower priority which is being handled by a thread can be preempted, It is well known that preemption can lead to a high context switch overhead. In order to reduce the context switch overhead, we shall use Thread-pool-with-lane [32].

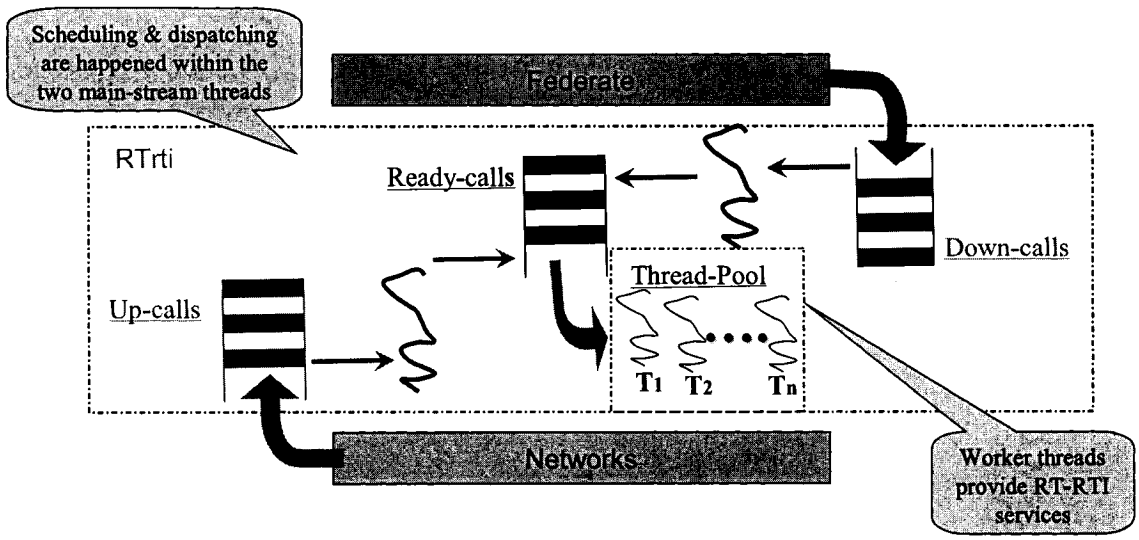

Figure 6.5: Main phases of RTrti execution

With the proposed multi-thread LRC processing model, global scheduling service and fixed-priority dispatching mechanism working together within RTI, we can have strict control over the scheduling and execution of CPU and memory resources and guarantee the completion of the real-time requests before their deadlines. In Figure 6.5, we illustrate the main phases that are to be run by our real-time RTI. Note that the fully multi-thread processing model in our real-time RTI architecture is not a "one-size-fit-all" based approach. It depends on the properties of different simulation applications. If the updates/interactions occur frequently, which is always the case in real-time applications, and the size of the distributed simulation is large, our multi-threaded approach is scalable and can provide higher system performance.

Chapter 7

Conclusions and Future Work

Data distribution management plays an important role in lager-scale distributed simulations. Therefore, it is desired to provide efficient, scalable and cost effective DDM mechanisms to meet the high performance requirement. Since various DDM schemes have been developed with their own pros and cons, the selection of an appropriate DDM scheme can be a critical factor in accomplishing efficient data exchange within a collaborative virtual environment.

As an outgrowth of our work in large-scale distributed interactive simulation, we have proposed an optimized dynamic grid-based algorithm, in which irrelevant information will be further filtered out using a sender-side-filter to effectively extend the filtering accuracy of dynamic grid-based DDM. It saves the network resources and reduces the receiving-side processing load, which optimizes the DDM services further to meet the high performance requirement in large-scale distributed simulation. The experimental results demonstrate that our optimized dynamic grid-based DDM achieves certain trade-off between the two DDM schemes, namely region-based and dynamic grid-based DDM,due to the use of the light level of exact matching in our new DDM scheme.

The second DDM approach is proposed, which is grid-filtered region-based DDM. It utilizes the concept of a percentage of a grid cell covered by the subscription/publication region to determine if further filtering is necessary. Experimental results show that it has better performance than region-based and grid-based DDMs, in terms of multicast groups, message and

time overhead.

There is still much to be learned by continued experimentation with various DDM techniques. Our plans for future research include a further simulation experimental study of the affects of aggregation and disaggregation [10] of federates with relation to various DDM approaches, and comparisons using agent-based and sort-based DDM schemes [43, 39].

As an ongoing project, in order to address the increasing needs for powerful real-time distributed simulation capabilities, we have proposed an efficient real-time RTI architecture in order to enhance HLA-based distributed real-time simulations. We have discussed our design and the implementation of the real-time RTI end system, which represents the core of our proposed architecture. We are currently at the design-wise stage of the project. We will later on implement our design of the real-time RTI end-system, to evaluate it real-time performance and gain close understanding of it behavior under different scenario.

In the future, we plan to extend our real time RTI towards large-scale distributed simulation over the Internet, and distributed collaborative virtual reality type of applications, (such as haptic virtual environments)

Acronyms

CPU: Central Processing Unit

DDM: Data Distribution Management

DM: Declaration Management

DMSO: U.S. Defense Modeling an Simulation Office

DoD: Department of Defense

FED: Federation Execution Data

FedExec: Federation Executive Process

FOM: Federation Object Model

HLA: High Level Architecture

IEEE: Institute of Electrical and Electronic Engineers

IF: Interface Specification

IM: Interest Management

LAN: Local Area Network

LRC: Local RTI Component

MG: Multicast Group

MOM: Management Object Model

M&S: Modeling and Simulation

OS: Operating System

RID: RTI Initialization Data

RTI: Run-Time Infrastructure

RtiExec: RTI Executive Process

RTrti: Real Time RTI End-system

SOM: Simulation Object Model

TM: Time Managment

UPSD: Number of units per spatial dimension

WAN: Wide Area Network

References

[1] "*CORBA Middleware*". In *http://www.cs.wustl.edu/schmidt/corba.html.*

[2] "*DSMO HLA Offcial Website*". In *https://www.dmso.mil/public/transition/hla/.*

[3] *"The ADAPTIVE Communication Environment"*. http://www.cs.wustl.edu/schmidt/ACE .html, Computer Science and Engineering Dept., Washington University in St. Louis.

[4] *"Tutorial on DSMO HLA"*. http://wwww.ecst.csuchico.edu/hla/LectureNotes, California State University, Chico.

[5] IEEE Std 1516-2000, 1516.1-2000, 1516.2-2000. In *"Standard for Modeling and Simulation High Level Architecture (HLA)"*. IEEE Society, 2000.

[6] DoD Defense Modeling and Simulation Office. In *"HLA Run-Time Infrastructure RTI 1.3-Next Generation Programmer's Guide Version 5"*, May 2002.

[7] *"Real-Time CORBA 2.0: Dynamic Scheduling Specification"*. http://www.omg.org/, November 2003.

[8] K. Balasubramanian, A. S. Krishna, E. Turkay, J. Balasubramanian, J. Parsons, A. Gokhale, and D. C. Schmidt. "Applying Model-Driven Development to Distributed Real-time and Embedded Avionics Systems". *International Journal of Embedded Systems, special issue on Design and Verification of Real-Time Embedded Software*, 2005.

[9] A. Berrached. "Alternative Approaches to Multicast Group Allocation in HLA Data Distribution Management". In *Proceedings of Spring Simulation Interoperability Workshop*, pages 98–SIW–184, 1998.

[10] A. Boukerche and C. Dzermajko. "Performance Comparison of Data Distribution Management Strategies". In *Proceedings of Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, pages 67–75, 2001.

[11] A. Boukerche and C. Dzermajko. "Dynamic Grid-Based vs. Region-Based Data Distribution Management Strategies for Large-Scale Distributed Systems". In *Proceedings of 17th International Parallel and Distributed Processing Symposium*, page 280, 2003.

[12] A. Boukerche and K. Lu. "Optimized Dynamic Grid-Based DDM Protocol for Large-

Scale Distributed Simulation Systems". In *Proceedings of 19th International Parallel and Distributed Processing Symposium(IPDPS 2005)*, April 2005.

[13] A. Boukerche and A. Roy. "Dynamic Grid-Based Approach to Data Distribution Management". *Journal of Parallel and Distributed Computing*, 62:366–392, 2002.

[14] A. Boukerche and A. Roy. "Dynamic Grid-Based Multicast Group Assignment in Data Distribution Management". In *Proceedings of Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications Workshop*, pages 27–34, 2003.

[15] D. Bruzman, M. Zyda, K. Watsen, and M. Macedonia. "Virtual Reality Transfer Protocol (vrtp) Design Rational". In *Proceedings of Sixth IEEE Int'l Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Distributed System Aspects of Sharing a Virtual Reality*, pages 179–186, June 1997.

[16] A. Buss and L. Jackson. "Distributed Simulation Modeling: a comparison of HLA, CORBA, and RMI". In *Proceedings of the 1998 Winter Simulation Conference*, pages 819–825, 1998.

[17] J. O. Calvin and D. J. V. Hook. "AGENTS: An Architectural Construct to Support Distributed Simulation". In *Proceedings of 11st Workshop on Standards for the Interoperability of Distributed Simulations (DIS)*, pages 94–11–142, Spet. 1994.

[18] J. S. Dahmann and K. L. Morse. "High Level Architecture for Simulation: An Update". In *Proceedings of the 2nd IEEE International Workshop on Distributed Simulation and Real-Time Applications (DiS-RT 1998)*, pages 32–40, 1998.

[19] S. Ferenci, K. Perumalla, and R. M. Fujimoto. "An Approach for Federating Parallel Simulators". In *Proceedings of Workshop on Parallel and Distributed Simulation*, pages 63–70, May 2000.

[20] R. M. Fujimoto. *"Parallel and Distributed Simulation Systems"*. Wiley Interscience, 2000.

[21] R. M. Fujimoto. "Advanced Tutorials: Parallel Simulation: Parallel and Distributed Simulation Systems". In *Proceedings of the 30th Conference on Winter simulation*, pages

819–826, 2001.

[22] R. M. Fujimoto and T. McLean. "The Federated-simulation Development Kit: A Source-Available RTI". In *Proceedings of Spring Simulation Interoperability Workshop*, pages 01–SIW–95, March 2001.

[23] R. M. Fujimoto, T. McLean, K. Perumalla, and I. Tacic. "Design of High Performance RTI Software". In *Proceedings of Fourth Workshop on Distributed Interactive Simulations and Real-Time Applications*, page 89, August 2000.

[24] L. Georgiadis, R. Guerin, V. Peris, and K. Sivarajan. "Efficient network QoS provisioning based on per node traffic shaping". *Journal of IEEE/ACM Transactions on Networking*, pages 482–501, 1996.

[25] G.Tan, Y.S.Zhang, and R.Ayani. "A Hybrid Approach to DDM". In *Proceedings of 4th International Workshop on Distributed Simulation and Real-Time Applications (DS-RT'00)*, pages 55–61, August 2000.

[26] F. G.Tan, W.N.Ngee. "Aggregation/Disaggregation in a HLA Multiresolution Distributed Simulation". In *Proceedings of Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT'01)*, pages 76–83, 2001.

[27] D. J. V. Hook and J. O. Calvin. "Approaches to RTI Implementation of HLA Data Distribution Management Services". In *Proceedings of the 15th Workshop on Standards for DIS*, pages 196–14–084, 1996.

[28] D. J. V. Hook and J. O. Calvin. "Data Distribution Management in RTI 1.3". In *Proceedings of the Simulation Interoperability Workshop (SIW)*, pages 196–14–084, 1998.

[29] D. J. V. Hook, J. O. Calvin, and J. E. Smith. "Data Consistency Mechanisms to Support Distributed Simulation". In *Proceedings of Twelfth Workshop on Standards for the Interoperability of Distributed Simulations*, pages 95–12–059, 1995.

[30] M. Hyett and R. Wuerfel. "Implementation of the Data Distribution Management Services in the RTI-NG". In *Proceedings of Simulation Interoperability Workshop*, pages 02S–SIW–044, 2002.

[31] R. S. J. and D. J. V. Hook. "Evaluation of Grid Based Relevance Filtering for Multicast Group Assignment". In *Proceedings of Fourteenth Workshop on Standards for the Interoperability of Distributed Simulations (DIS)*, pages 96–14–106, 1996.

[32] A. Krishna, D. C. Schmidt, R. Klefstad, and A. Corsaro. "Towards Predictable Real-time Java Object Request Broker". In *Proceedings of 9th IEEE Real-time /Embedded Technology and Applications Symposium (RTAS)*, page 49, May 2003.

[33] J. W. Liu. *"Real-Time Systems"*. Prentice Hall, 2000.

[34] M. Macedonia, M. Zyda, D. Pratt, and P. Barham. "Exploiting Reality with Multicast Groups: a Network Architecture for Large Scale Virtual Environments". In *Proceedings of Virtual Reality Annual International Symposium*, pages 2–10, 1995.

[35] T. McLean, R. M. Fujimoto, and B. Fitzgibbons. "Middleware for Real-Time Distributed Simulations". *Journal of Concurrency and Computation: Practice and Experience*, 16(15):1483–1501, November 2004.

[36] K. Morse and M. Petty. "Data Distribution Management Migration from DoD to IEEE 1516". In *Proceedings of the Fifth IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2001)*, pages 366–392, 2001.

[37] K. L. Morse, L. Bic, M. Dillencourt, and K. Tsai. "Multicast Grouping for Dynamic Data Distribution Management". In *Proceedings of 31st Society for Computer Simulation Conference*, page 47, 1999.

[38] J. P. Nalepka, T. Dube, G. Williams, R. B. Bryant, and T. Danube. "Real-Time Simulation Using Linux". In *Proceedings of AIAA Modeling and Simulation Technologies Conference*, pages 6–17, 2001.

[39] C. Raczy, G. Tan, and J. Yu. "A Sort-based DDM Matching Algorithm for HLA". *Journal of ACM Transactions on Modeling and Computer Simulation*, (15-1):14–38.

[40] C. Raczy, J. Yu, G.Tan, S.C.Tay, and R.Ayani. "Adaptive Data Distribution Management for HLA RTI". In *Proceedings of 2nd European Simulation Interoperability Workshop*, pages 14–38, June 2002.

[41] D. C. Schmidt and F. Kuhns. "An Overview of the Real-time CORBA Specification". *Journal of IEEE computer Magazine: Special Issue on Object-oriented Real-time Computing*, pages 56–63, June 2000.

[42] I. Tacic and R. M. Fujimoto. "Synchronized Data Distribution Management in Distributed Simulations". In *Proceedings of the twelfth Workshop on Parallel and distributed simulation PADS '98*, number 28-1, pages 108 – 115, 1998.

[43] G. Tan, L. Xu, F. Moradi, and S. Taylor. "An Agent-based DDM for High Level Architecture". In *Proceedings of the fifteenth workshop on Parallel and Distributed Simulation*, pages 75–82, 2001.

[44] S. T.Bachinsky, L. Mellon, G.Tarbox, and R. Fujimoto. "RTI 2.0 Architecture". In *Proceedings of Simulation Interoperability Workshop, Spring*, pages 98S–SIW–150, 1998.

[45] S. T.Bachinsky, J. Noseworthy, and Frank.J.Hodum. "Implementation of the Next Generation RTI". In *Proceedings of Simulation Interoperability Workshop, Spring*, pages 99S–SIW–118, 1999.

[46] R. D. Wuerfel and C. R. Johnston. "Real-Time Performance of RTI Version 1.3". In *Proceedings of Fall Simulation Interoperability Workshop*, pages 98F–SIW–125, 1998.

[47] R. D. Wuerfel and L. S. G. Purdy. "A Comparison of HLA and DIS Real-Time Performance". In *Proceedings of Spring Simulation Interoperability Workshop*, pages 98S–SIW–042, 1998.

[48] J. Yu, C. Raczy, and G.Tan. "Evaluation of a Sort-based Matching Algorithm for DDM". In *Proceedings of 16th Workshop on Parallel and Distributed Simulation*, pages 68–75.

[49] J. Yu, C. Raczy, and G. Tan. "Evaluation of Sort-based DDM Matching Algorithm for DDM". In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PAD'02)*, pages 68–75, 2002.

[50] H. Zhao and N. D. Georganas. "HLA Real-time Extension". In *Proceedings of Fifth International Workshop on Distributed Simulations and Real-Time Applications*, pages 12–21, 2001.