SPACE-BASED DATA MANAGEMENT

FOR HIGH PERFORMANCE DISTRIBUTED SIMULATION

By

Jong Sik Lee

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2001

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgement of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: _____

ACKNOWLEDGEMENTS

I thank my advisor Bernard P. Zeigler for having provided me with much needed support and guidance in the Ph.D. program. Besides my advisor, I would like to thank the rest of my dissertation committee: Ralph Martinez and Salim Hariri for their helpful comments, suggestions and encouragement during the course of writing this dissertation.

A special thanks goes to the people at the AIS Lab, Hessam S. Sarjoughian, Young Kwan Cho, and Sunwoo Park. They were always available to talk about my ideas and to ask me good questions to help me think through my problems.

Most of all, my parent, brother, and sister are the ones who have been with me and I always remember their constant love and great support for me. Also, I thank my friends in Korea whose friendship and care helped me.

This research was supported by grants from the DARPA Contract N6133997K-0007 and partially supported by National Science Foundation Grant: "DEVS as a Formal Modeling Framework for Scaleable Enterprise Design" and by Lockheed Martin Space System Grant: “Object Oriented HLA Interface Design for Military Simulations.”

TABLE OF CONTENTS

TABLE OF CONTENTS ? Continued

LIST OF FIGURES

LIST OF FIGURES ? Continued

LIST OF FIGURES ? Continued

LIST OF FIGURES ? Continued

LIST OF FIGURES ? Continued

LIST OF TABLES

LIST OF TABLES ? Continued

ABSTRACT

There is a rapidly growing demand to model and simulate complex large-scale distributed systems and to collaboratively share geographically dispersed data assets and computing resources to perform such distributed simulation with reasonable communication and computation resources. Interest management schemes have been studied in the literature. In this dissertation we propose an interest-based quantization scheme that is created by combining a quantization scheme and an interest management scheme. We show that this approach provides a superior solution to reduce message traffic and network data transmission load.

As an environmental platform for data distribution management, we extended the DEVS/HLA distributed modeling and simulation environment. This environment allows us to study interest-based quantization schemes in order to achieve effective reduction of data communication in distributed simulation. In this environment, system modeling is provided by the DEVS (Discrete Event System Specification) formalism and supports effective modeling based on hierarchical and modular object-oriented technology. Distributed simulation is performed by a highly reliable facility using the HLA (High Level Architecture). The extended DEVS/HLA environment, called DEVS/GDDM (Generic Data Distribution Management), provides a high level abstraction to specify a set of interest-based quantization schemes.

This dissertation presents a performance analysis of centralized and distributed configurations to study the scalability of the interest-based quantization schemes. These

results illustrate the advantages of using space-based quantization in reducing both network load and overall simulation execution time. A real world application, relating to ballistic missiles simulation, demonstrates the operation of the DEVS/GDDM environment. Theoretical and empirical results of the ballistic missiles application show that the space-based quantization scheme, especially with predictive and multiplexing extensions, is very effective and scalable due to reduced local computation demands and extremely favorable communication data reduction with a reasonably small potential for error. This realistic case study establishes that the DEVS/GDDM environment can provide scalable distributed simulation for practical, real-world applications.

1 INTRODUCTION

Distributed systems approaches are being applied to a growing variety of systems including process control and manufacturing, military command and control, transportation management, and so on. To model and simulate these distributed systems, the development of a distributed modeling and simulation environment has drawn the attention of many modeling and simulation researchers [10, 44, 47]. Distributed simulation is characterized by numerous interactive data exchanges among multiple simulation entities over a network. Thus, in order to provide a reliable answer in reasonable time with limited communication and computation resources, a methodology for reducing the interactive data exchanges is required in a distributed modeling and simulation environment. In this dissertation, a novel, interest-based quantization scheme is proposed to promote the effective reduction of data communication in a distributed simulation environment.

The DEVS/GDDM (Generic Data Distribution Management) modeling and simulation environment was developed in order to perform complex and large-scale distributed modeling and simulation with reasonable communication and computation resources with the interest-based quantization scheme. In the DEVS/GDDM environment, system modeling is provided by the DEVS (Discrete Event System Specification) formalism and the distributed simulation is performed by the HLA (High Level Architecture) Interface. The scalability of the interest-based quantization scheme is investigated in a pursuer/evader example testbed; and through a real application (e.g.

multiple ballistic missiles), the usefulness of the DEVS/GDDM environment is demonstrated.

1.1 Modeling formalisms

For discrete event system modeling and simulation, Zeigler [1] provides the system formalisms and the corresponding system theoretic framework. The provided system formalisms are the Differential Equation System Specification (DESS), Qualitative System Specification (QSS), the Discrete Time System Specification (DTSS), and the Discrete Event System Specification (DEVS). Figure 1.1 depicts the System Specification Formalisms.

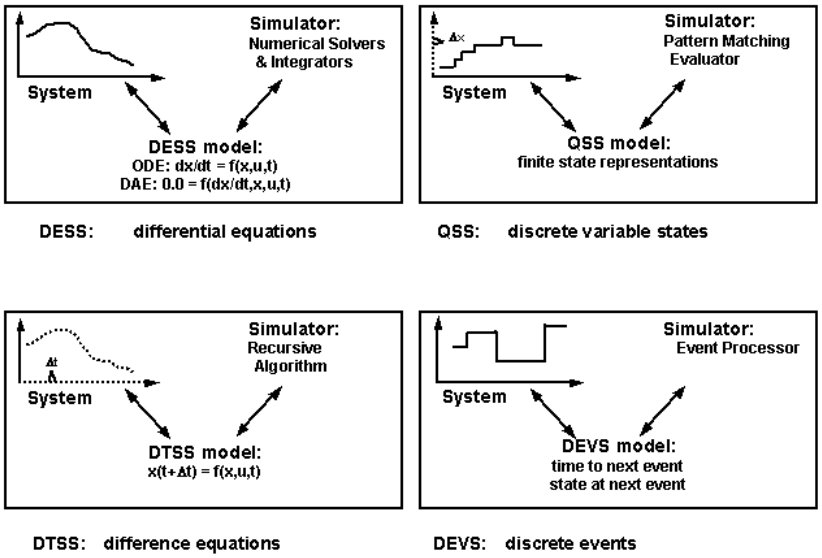


Figure 1.1 System Specification Formalisms

In order to connect these various levels of specification and work across them, Zeigler [1] suggests a homomorphism concept and develops the DEVS models related to the homomorphism concept. A system modeler can develop a valid simplified homomorphic (lumped) model of a complex (base) model with the homomorphism concept. While a specified attribute of a base model is mapped to that of a lumped model, the behavior of the lumped model's attribute mapped to that of the base model is reflected by the behavior of the attribute of the base model with a certain condition provided by the homomorphism concept.

1.2 Agent-based system

An agent-based system is a working system based on autonomous software and/or hardware components (agents) that cooperate to perform tasks. The agent-based system includes agents and environments within an environment. An agent is a system component with the capability of perception, decision, and mobility. Also, an agent is autonomous since it has the ability to generate its own goals and the inbuilt desires (or preferences) determined by the agent system developer. The environment indicates the computational system which any agent inhabits. An environment can contain a particular agent and can include other agents whose environments are disjointed or partially overlapped with it. The desires or goals of the autonomous agent are sensitive to the current state of both the agent and the environment. An agent can effectively change states of a given environment or other environments by moving from one part of the environment to a part of the other environment. For example, if an agent moves a bomb from one environment to another environment, the topology of both the environments is thereby changed.

In an agent-based system, effective communication plays a important role. There are three kinds of communications: environment-to-environment, environment-to-agent, and agent-to-agent. To perform the communications, an agent generates any output event and receives any external input. Without the external input event to the autonomous agent, the agent can produce the output event in response to the autonomous process

within the agent. When the agent receives the input event, it then changes its states, desires or goals.

The agent-based system gives great promise in advancing a new technology for developing complex system which has been blocked by the limitation of current development tools and methodologies. In this case, the agent-based system is especially appropriate in applications where independently developed components inter-operate with each other in a heterogeneous environment (e.g. telecommunications, business modeling, control of mobile robots, and military simulations [2, 3]). The agent-based system, which includes many agents within an environment or many environments, is called multi-agent system. Generally, multi-agent system is extremely complex [4], so that the verification of the multi-agent system is difficult. Simulation plays a key role in the development of the complex multi-agent system [5, 6, 7, 8]. The use of simulation facilitates the replication of results in the multi-agent system with a very difficult or impossible-fielded system.

The main problem of the simulation of multi-agent systems is that the simulation requires a lot of computation resources. Actually, each agent is a complex system to perform its own right (e.g. sensing, planning, movement, and so on), and many agents investigate the behavior of the other agents and the states of many environments. These behaviors of the multi-agents system require a lot of computation resources. Therefore, the solution to limited computation resources is to apply a high degree of parallelism in a multi-agent system.

In a recent study, as the network technology to perform the effective data communication has been advanced, most of the multi-agent system designers or researchers noticed the multi-agent system distributing the agents over a network of processors interacting via some various communication protocols. This distributed simulation of the multi-agent system has the same concept of a high degree of parallelism in order to reduce the computation resources required.

Meanwhile, a lot of communication resources are also required to perform the distributed simulation of a multi-agent system in order to exchange the data among agents of distributed hosts over a network. Most applications of the multi-agent system are the network-based applications, and the developer and the researcher of the multi-agent system have to solve the problem that is caused by a large amount of communication resources adding a burden to the computation resource shortage problem. Actually, telecommunications, computer games, and military simulation applications are typical multi-agent applications which need very interactive data communication over a network. In the multi-agent system, as the number of agents increases, the message exchanges among agents distributed in network end-hosts may increase quadratically, so that the numerous messages over a network cause the problem of scalability of a multi-agent distributed simulation. In this dissertation, in order to execute the complex, multi-agent distributed simulation with reasonable communication and computation resources, several message traffic reduction schemes are considered.

1.3 Distributed Simulation and its Environment

The demand for distributed simulation is rapidly growing to support a simulation of defense and industrial systems that are getting more complex and distributed in their computational infrastructure. Distributed simulation supports many practical application domains that require reliable communication linkage among multiple, geographically separated systems. In addition, through a distributed simulation, we can expect to improve computing power, access more memory, provide high scalability, and lower the simulation cost. Also, such a distributed simulation can share geographically dispersed data assets and computing resources collaboratively; thus, it can execute those complex simulations.

To support distributed computing, several software developments for distributed processes running on machines to interact across a network have been developed. The software development is called "middleware." Middleware provides communication across heterogeneous platforms and performs interoperability based on client/server architectures. Through the integration of heterogenous platforms, middleware provides efficient, cost-effective, flexible, and extensive information sharing. Most public middlewares are the High Level Architecture (HLA) [9, 10] of the Department of Defense (DoD), the Common Object Request Broker Architecture (CORBA) [11, 12] of the Object Management Group (OMG), and the Distributed Component Object Model (DCOM) [13] of the Microsoft company.

The middleware designed specially for a distributed simulation is the HLA. HLA is a technical architecture for DoD simulations and defines the functional elements, interfaces, and design rules needed to achieve a proper interaction of simulations in a federation or among multiple federations. There are two types of communication in HLA: attribute updating and interaction communication. Attribute updating is communication between an object in a federate and an object in another federate. Interaction communication is a non-persistent and time-stamped communication between two federates.

HLA also has two major components: the Object Model Template (OMT) and the RunTime Infrastructure (RTI). The OMT is a format to represent the information required by the HLA object model. RTI is a software component of HLA. RTI coordinates the interactions among the simulations of a federation and performs a basic mechanism for initializing, directing, and controlling the flow of data exchange among simulations. RTI provides services commonly required by simulation systems. These services include time management, ownership, objects, federations, data declaration, and data distribution. With the standard format of the OMT described by a simulation developer, RTI performs the attribute and interaction communications across different platforms.

1.4 Message Traffic Reduction Scheme

Recently, distributed systems approaches are being noticed for a growing variety of systems including process control and manufacturing, military command and control, transportation management, and so on. Such distributed systems are complex and large in their size. In fact, in order to model and simulate these complex and large-scale distributed systems, the development of a large-scale distributed modeling and simulation environment entities has drawn the attention of many modeling and simulation researchers.

In general, a large-scale distributed simulation requires achievement of real-time linkage among multiple and geographically distant systems, and thus has to execute complex large-scale simulation and to share geographically dispersed data assets and computing resources collaboratively. However, large-scale distributed simulations are characterized by numerous interactive data exchanges among simulation entities distributed between computers networked together. In the worst case, each entity interacts with all the others so that as the number of entities increases (e.g. the message exchanges may increase quadratically, greatly limiting the scalability of distributed simulation approaches). The methodology to support the reduction of the interactive messages among simulation entities is called a "message traffic reduction scheme". It is the goal of a message traffic reduction scheme that a large-scale distributed simulation is performed with reasonable communication and computation resources. To perform a message traffic reduction scheme reliably, flexibility and efficiency are required.

Flexibility does not indicate anything specific to any particular problem domain or technology, but rather indicates being general in nature. Efficiency requires the scaling of simulations from very small to very large along many dimensions including numbers of the simulated objects, complexity of interactions, fidelity of representations, and computational/network resources.

1.5 Quantization Theory

The Quantization theory [14, 15] is based on modeling formalism and system homomorphisms. As Figure 1.2 illustrates, a continuous trajectory with a finite number of values in a finite time interval is approximated. In order to obtain a discrete time system approximation, discretization of the time base is needed with a finite time interval. The finite number of values is then calculated from the partition of the trajectory into a finite number of segments (each of which has a finite computation). The partition of the trajectory with the finite number of values provides a way to quantize the value space, which is partitioned in every D interval (quantum), and the time space is partitioned in every T interval (time interval).

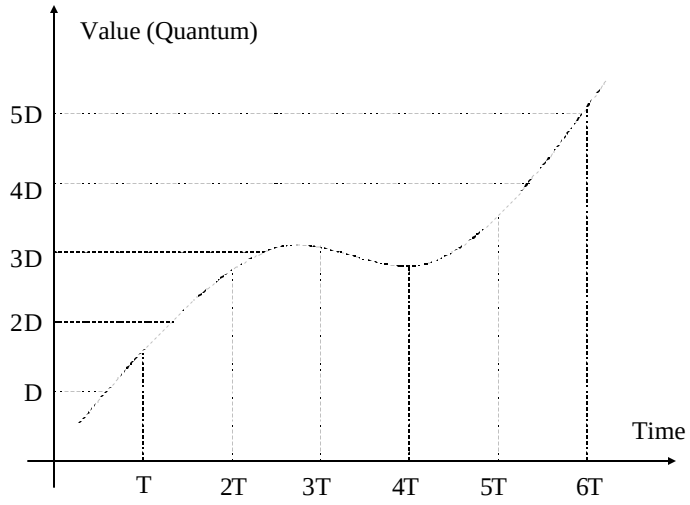


Figure 1.2 Time discretization and Quantization

In discrete event systems, we sample the time values at every quantum interval (D), use discrete values with continuous time, and send the quantum levels out after the sampled time interval. This is called the quantization based on the quantum D.

In a real application, the state trajectory is represented by the crossings of an equally spaced set of boundaries separated by the quantum interval (D). Using quantization, we checks a threshold crossing of output value of a sender whenever an output event occurs and sends the output value to a receiver only when the threshold crossing occurs. The effect of quantization is to reduce the number of messages exchanged between sender and receiver. We can expect to save the communication data and the computation of the receiver from the reduced messages through the message reduction. Considered with the scalability of a system, the quantization increases system

performance in various ways such as decreasing overall execution time or allowing a larger number of entities to be performed. In chapter 2, the actually realized quantization scheme is introduced as one of the message traffic reduction schemes.

1.6 Object-oriented Design

Object-orientation technology allows a particular system to be encapsulated by a system modeler and provides a common interface of the encapsulated system to the rest of the whole system. That is abstraction capability of the object-orientation technology. The DEVS/GDDM environment, based on the object-orientation technology and the DEVS formalism [1], has a portability of models across platforms at a high level of abstraction. Such portability enables a model to be developed and verified in a platform, and then easily ported across distributed platforms. Because the DEVS formalism is expressed as a collection of objects and their interactions with the details of the implementation hidden within the objects, and any DEVS component is shielded from the environment which provides any services to the DEVS component.

In the DEVS/GDDM environment based on the DEVS formalism, a system modeler can build a DEVS model in a hierarchical and modular fashion. Each DEVS model at a certain level of the DEVS hierarchy can see its one-level lower models and its one-level upper level models and the coupling among models of upper and lower level is considered in its modular fashion. The coupling in DEVS formalism allows two DEVS model (sender and receiver) to be coupled, then delivers a DEVS message from a sender

to a receiver model. This high level DEVS modeling provides the maintainability and reusability of a DEVS model in the DEVS/GDDM environment.

1.7 Dissertation Organization

Chapter 2 presents the discrete event system formalism and reviews existing message traffic reduction schemes. Chapter 3 presents the space-based quantization scheme as a more efficient means of message traffic reduction, and discusses the scalability of space-based quantization schemes in a distributed simulation. The DEVS/HLA-Interface is introduced and its functions are illustrated in chapter 4. Chapter 5 introduces the DEVS/GDDM simulation environment which uses the interest-based quantization scheme and discusses the network load reduction methods supported by this environment. In chapter 6, the DEVS predictive integrator model is developed as a basis for the predictive quantization scheme. Chapter 7 and chapter 8 present real world applications and show how the space-based quantization scheme is applied to these applications. The conclusion is in chapter 9.

2 BACKGROUND

2.1 Discrete Event System Formalism

The discrete event system specification (DEVS) is a formalism for the discrete event systems [1]. The DEVS formalism consists of two parts, base and coupled models. A basic model of a standard DEVS is a structure:

$$M = <X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta>$$

Where

X : set of external input events;

S : a set of sequential states;

Y : a set of outputs;

δ_{int}: S ? S : internal transition function

δ_{ext} : Q ? X^b ? S : external transition function

X^b is a set of bags over elements in X,

(where $\delta_{ext}(s,e,?) = (s,e)$);

δ_{con} : S ? X^b ? S : confluent transition function;

? : S ? Y^b : output function generating external events at the output;

ta : S ? Real : time advance function;

Where Q = { (s,e) | s ? S, 0 ? *e* ? ta(s) }, and *e* is the elapsed time since last state transition

Two major activities involved in *coupled* models are specifying its component models and defining the couplings, which create the desired communication links.

$$DN = <X, Y, D, \{M_I\}, \{I_I\}, \{Z_{I,j}\}>$$

Where

X : set of external input events;

Y : a set of outputs;

D is a set of components names;

for each I in D,

M_I is a component model

I_I is the set of influencees for I

for each j in I_I,

$Z_{I,j}$ *is* the I-to-j output translation function

A *coupled* model contains the following information

- the set of components
- for each component, its influencees
- the set of input ports through which external events are received
- the set of output ports through which external events are sent
- the coupling specification consisting of
- the external input coupling connects the input ports of the coupled to one or more of the input ports of the components

- the external output coupling connects the output ports of the components to one or more of the output ports of the *coupled* model
- internal coupling connects output ports of components to input ports of other components

2.2 Message Traffic Reduction Scheme

In this section, we provide an overview of the major message traffic reduction techniques, which are currently used in most entity-based virtual simulations. These techniques include dead-reckoning, interest management, Data Distribution Management (DDM) of HLA, and quantization.

2.2.1 Dead-Reckoning Scheme

As a scheme to reduce the number of state update messages, the dead-reckoning scheme is widely employed in distributed simulations [28, 29]. The state update messages are exchanged among each simulated entity to maintain the accurate state of the other remote simulated entities. Each federate maintains accurate information (position, velocity, acceleration) of its own simulated entity's movement with a high fidelity model. Also, each federate includes the dead-reckoning (inaccurate) models of all simulated entities including that of its own entity. As the simulation time passes, the states of dead-reckoning models are updated by working the second-order extrapolation with the last

updated message. The anticipated position of a simulated dead-reckoning entity is calculated by the second-order extrapolation below:

$$X(t + \Delta t) = X(t) + Vx(t)\Delta t + 0.5Ax(t)\Delta t^2$$
$$Y(t + \Delta t) = Y(t) + Vy(t)\Delta t + 0.5Ay(t)\Delta t^2$$
$$Z(t + \Delta t) = Z(t) + Vz(t)\Delta t + 0.5Az(t)\Delta t^2$$

$X(t), Y(t), Z(t)$ are the position coordinates of a simulated entity at time t. $Vx(t), Vy(t), Vz(t)$ and $Ax(t), Ay(t), Az(t)$ are the x, y, z components of the velocity vector and the acceleration vector at time t and $X(t + \Delta t), Y(t + \Delta t), Z(t + \Delta t)$ are the new coordinates predicted at Δt time unit from time t.

When the state of a simulated entity changes, the state of the high fidelity model of the simulated entity is updated and is compared to the state of the corresponding dead-reckoning model. If the position/acceleration of the dead-reckoning model of the simulated entity deviate from the exact position/acceleration of the high fidelity model of the simulated entity by more than a threshold value, the simulated entity creates a new message and sends it to the other remote federates. The remote federates, which receive the new message, correct the state of the corresponding dead-reckoning model and begin the new second-order extrapolation with the new position/acceleration. In the dead-reckoning scheme, reduction of the data issued by dead-reckoning models plays a role of a message traffic reduction scheme.

2.2.2 Interest Management

The interest management technique [32] was proposed as a method to avoid broadcast communication among agents. Generally, the interest management technique is a message filtering mechanism to enable execution with the reasonable communication and computation resources in real-time large-scale simulations. Interest management is based on interest expression between pairs of sender and receiver agents. The receiver agent expresses the interest to an attribute of the sender agent and the sender agent sends the value of the attribute interested to the receiver agent. The interest expression expresses a subset of all data exchanges of the all attributes of the sender agent. The expression of an attribute can be changed as the simulation time passes. As the number of agents and the number of the attributes in the agents increase, the interest expression may become complicated. A special entity to manage the interest expression and to enable the effective data exchange between a sender and a receiver agent pair is called the "interest manager".

Recently, several interest management techniques has been proposed and studied. In most application systems, IP multicast addressing [33, 34] is an example of the interest management technique. A multicast group is an example of the interest expression and is defined for each message transferred.

2.2.3 Data Distribution Management (DDM) of HLA

HLA provides the DDM service as an example of the interest management. In the DDM, the interest expression works with regions in a multi-dimensional parameter space. The multi-dimensional coordinate system is call the "routing space" and the routing space is subdivided into a predefined array of fixed sized cells. Each cell is assigned to a multicast group [36]. The DDM [37, 38, 39] service of HLA constitutes an interest-based message traffic reduction scheme. This service tries to filter out irrelevant data among federates. Each federate expresses the interest for the data to be sent and received by defining publication region and subscription region in the routing space. When a sender's publication region overlaps a receiver's subscription region, the RTI (RunTime Infrastructure), an implementation of the HLA specification, establishes network connectivity between the federates and makes data communication available. Communication overhead from region change notification due to moving agents negatively impacts the efficiency of the DDM filtering mechanism [36]. The efficiency is expressed by comparing the amount of useful data transmission compared to the total amount of data transmission including region change notifications.

2.2.4 Quantization schemes

Quantization, which is based on the quantization theory [14, 15], is an approach to distributed logical simulation in which the value space is quantized and trajectories are represented by the crossings of a set of thresholds. This is an alternative to the common approach which discretizes the time base of a continuous trajectory to obtain a finite

number of equally spaced sampled values over time. In distributed simulation, a quantizer checks for threshold crossings whenever an output event occurs and sends this value across to a receiver thereby reducing the number of messages exchanged among federates in a federation. In this section, we introduce three quantization schemes: 1) the baseline mechanism for quantization, called non-predictive quantization, 2) the more advanced form of quantization, called predictive qunatization, and 3) an approach to packaging individual data bits into a large message packet, called multiplexed predictive quantization.

2.2.4.1 Non-predictive Quantization

As Figure 2.1 illustrates, the non-predictive quantization [41, 42] applies when a sender component is updating a receiver component on a numerical, real-valued, state variable, which is a dynamically changing attribute. In the non-predictive approach, a quantizer is applied to the sender's output, which checks for threshold (boundary) crossings whenever a change in the variable occurs. Only when such a crossing occurs, a new value of the variable is sent across the network to the receiver. The non-predictive quantization reduces the number of messages sent (not their size) and incurs some local computation at the sender.

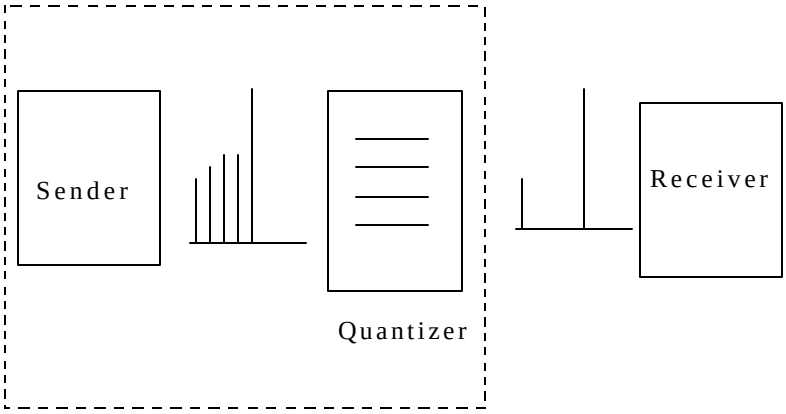


Figure 2.1 Non-Predictive Quantization

2.2.4.2 Predictive Quantization

As Figure 2.2 illustrates, a more efficient form of quantization is predictive quantization [40, 41, 42], where the sender employs a model to predict the next boundary crossing and the time this crossing will occur. Since the next boundary crossing is either one above or one below the last recorded boundary, the sender need not send the full floating point (double word) value to the receiver, so that it sends a one-bit message at crossings. The one-bit message represents whether the next higher or next lower boundary has been reached. In the predictive quantization approach, the main advantage over non-predictive quantization is that both the number of messages and their size can be reduced. A second advantage is that discrete event prediction can also greatly reduce the sender's state transition computation execution time and frequency if simple predictive models are used.

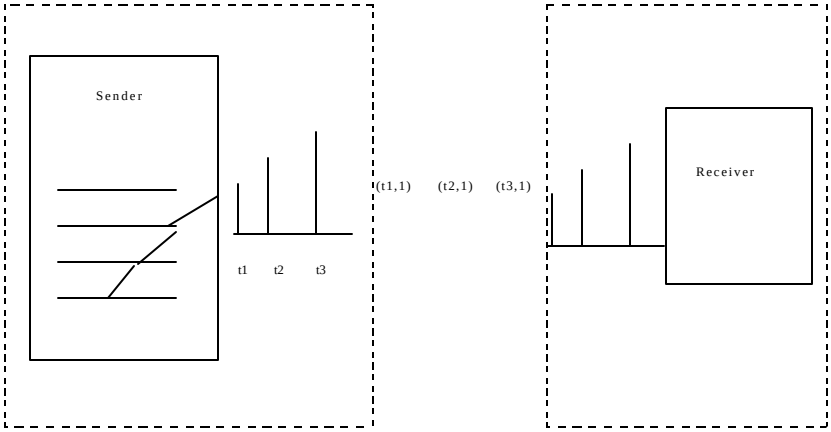


Figure 2.2 Predictive Quantization

2.2.4.3 Multiplexing Predictive Quantization

In simulations with a large number of entities, there will be many entities assigned to each federate. As Figure 2.3 illustrates, sender and receiver federates encapsulate a large number of similar component models. Each of these components has a predictive quantizer to produce a one-bit output of a variable. Then, at each event time, several components will be crossing their boundaries (a component is called *active* at a given event time if it has a boundary crossing at that time). The multiplexer encodes the joint output of the active components of the sender federate into a single message.

At the receiver federate, the de-multiplexer decodes the multiplexed packet in inverse fashion using a set of ghost components in a one-to-one correspondence with the sending components. There are two types of multiplexing: fixed and variable. Figure 2.3 illustrates the implementation of the fixed multiplexing predictive quantization [46, 57].

In fixed multiplexing, each pair of bits is examined. If the first bit of the source indicates *active*, then the receiver updates the appropriate variable of the counterpart (ghost) with the predefined quantum size, and increments the saved value of the tracked variable by the quantum in the direction (+1/-1) indicated by the second bit. Of course, sending and receiving federates must know the shared value of the quantum size and be informed of the new value should it be changed. In variable multiplexing, introduced in this dissertation, the size of the encoding message is directly related to the number of active components.

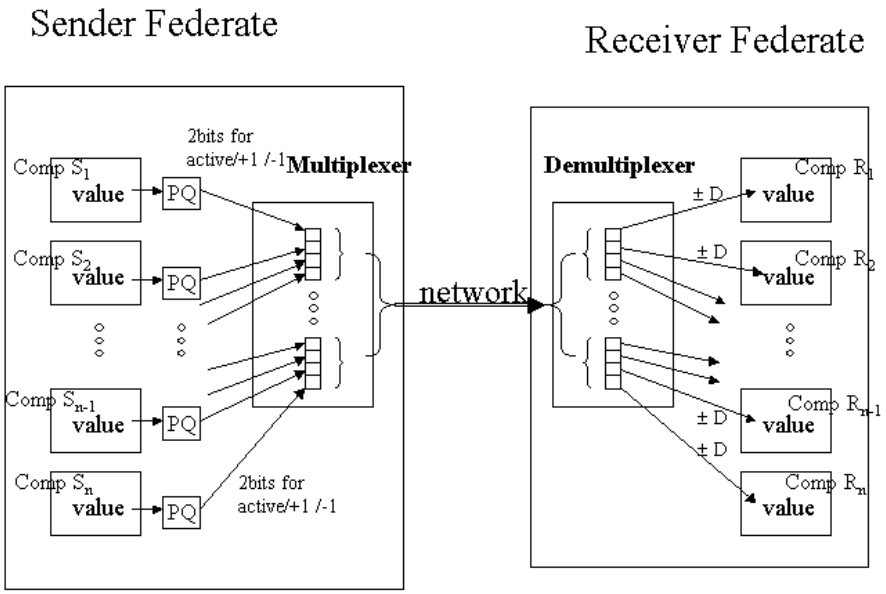


Figure 2.3 Implementation of the fixed multiplexed predictive quantization scheme

Table 2.1 Specialization of multiplexing and quantization schemes

Scheme		Predictive Dimension	
		Non-predictive quantization (send real value: 64 bit)	Predictive quantization
Multiplexing Dimension	Non-multiplexing quantization (1 message per output at a time instant))	Non-predictive quantization	Predictive quantization
	Multiplexing quantization (1 message for all component outputs at a time instant)	Multiplexing non-predictive quantization	Multiplexing predictive quantization (fixed, variable)

Table 2.1 shows the specializations of the multiplexing predictive quantization scheme using multiplexing and predictive quantization dimensions.

Table 2.2 Network load (bandwidth) requirements for fixed multiplexing and quantization schemes

(S_{OH} : the number of overhead bits for a packet; S_D: the non-quantized data bit size; N_{pair}: the number of component pairs; a: the ratio of active components)

Scheme	# bits required for N_{pair}	Ratio to Non-predictive quantization for large N_{pair}	Ratio for N_{pair} =1000 S_{OH} = 160 bits S_D = 64 bits
Non-predictive quantization	$a\ N_{pair}\ (S_{OH} + S_D)$	1	1
Predictive quantization (non-multiplexing)	$a\ N_{pair}\ (S_{OH} + 1)$	$(S_{OH} + 1) / (S_{OH} + S_D)$	0.74
Fixed multiplexing non-predictive quantization	$(S_{OH} + S_D * N_{pair})$	$S_D / a\ (S_{OH} + S_D)$	$0.28/a$
Fixed multiplexing predictive quantization	$(S_{OH} + 2\ N_{pair})$	$2 / a\ (S_{OH} + S_D)$	$.0096/a$

Table 2.2 analyzes network load requirements for the four combinations of fixed multiplexing and quantization types. It computes the ratio of the message size needed for a multiplexed predictive quantization to the number of bits needed for a non-multiplexed quantization with the same number of component pairs. Non-multiplexing cases send a fraction a of (larger) messages at each global event, while fixed multiplexing cases always send the same number of bits. From the table, we see that fixed multiplexing has high potential for data load reduction provided that a is high enough. However, since activity may not always be very high in arbitrary simulations, we introduced the above-mentioned variable multiplexing approach.

In this dissertation, we will discuss the influence of variation of a (activity ratio) in the variable multiplexing and the effectiveness of both fixed and variable multiplexing. In addition, the relationship between a and a time granule size will also be discussed. The time granule concept was introduced in [46] to enable boundary crossings within a time interval to be considered simultaneous.

3 SPACE-BASED QUANTIZATION SCHEME

3.1 Space-based quantization scheme

The space-based quantization scheme is created by combining the quantization scheme with an interest management scheme for monitoring the spatial encounters among agents. In the non-quantized spatial encounter monitoring scheme, there is only one critical distance to specify the communication relationship between two agents and, at any time, this holds or does not hold in all-or-none fashion. In contrast, in the space-based quantization scheme, there can be more than one critical distance between two agents thus allowing communication in a more tunable fashion. A quantum is assigned to each distance range created by the critical distances. The quantum size determines the rule for transferring or discarding messages from sender agent to receiver agent, and this rule is called a “filtering policy”. Figure 3.1 compares the change of quantum sizes based on the distance for the conventional spatial monitoring scheme and its space-based quantization extension. In the conventional approach there are, in effect, two quantum sizes: zero and infinity, corresponding to regions of interest or non-interest. The extended approach allows multiple quantum sizes thereby allowing communication frequency to be controlled as a smoother function of distance, as illustrated in Figure 3.2.

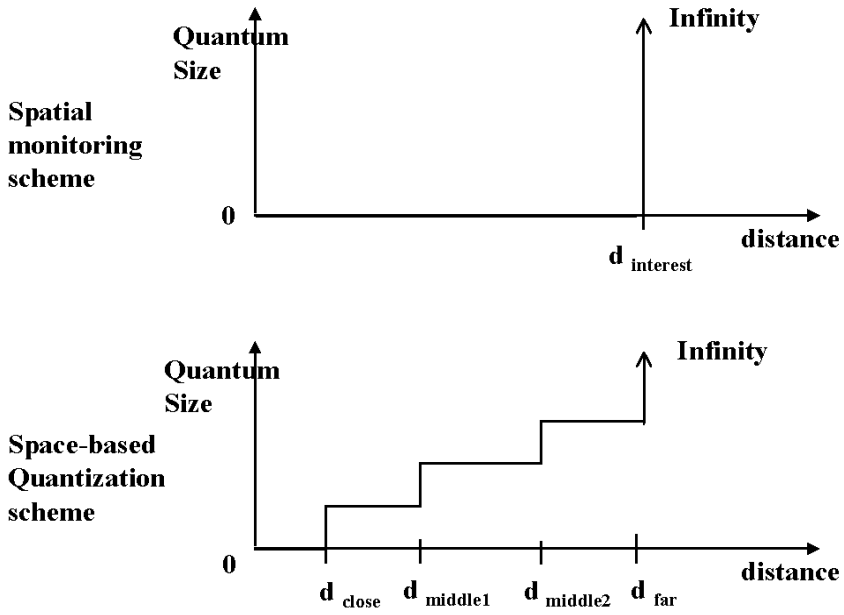


Figure 3.1 Change of quantum sizes based on the distance between Spatial Monitoring Scheme and Space-based Quantization Scheme

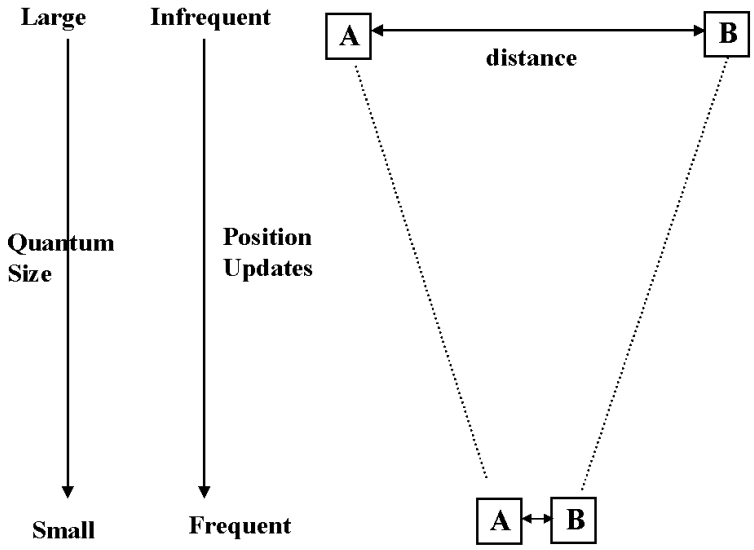


Figure 3.2 Space-based Quantization Scheme

In this dissertation, although we use the RTI for data communication among federates, we implement the space-based quantization scheme without using the DDM routing service of HLA. There are four disadvantages of the DDM in applying the space-based quantization scheme. First, DDM allows data to be exchanged among federates only in all-or-none fashion. There is no computation of the degree of overlap between publication and subscription regions. The second disadvantage, as noted before, is the large communication overhead required to notify the RTI of a region update whenever an agent moves. A third limitation is that the circular-shaped region necessary for the space-based quantization is not supported directly by RTI. The RTI supports the specification of only rectangular-shaped regions. To make a circular-shaped region, more complex areas must be defined by collecting multiple extents within a region. Unfortunately, the use of

multiple extents has a negative impact on system performance. Alternatively, retaining the smallest rectangular bounding region of a circular region, one can employ a two-layer filtering approach. In this approach, a federate must use additional information to discern if messages transferred with rectangular-shaped region are applicable or not. This approach also demands additional computation for the second filtering. The fourth disadvantage is the fact that many regions are created when multiple agents are assigned to one federate. A region is associated with one "Interaction" or one attribute of an "Object" which is used for communication between a pair of agents that respectively exist in separate federates. As the numbers of agents within federates increase, the number of regions that need to be created increases quadratically, heavily consuming local memories of the federates. For these reasons we did not employ the DDM routing service to implement the communication management data system working with the space-based quantization scheme.

3.2 Space manager

In the space-based quantization scheme defined here, the space manager provides filtering of the data communicated among agents. The main objective for using the space manager is the reduction of the data to be processed by the receiving agents as well as the data actually sent over the network. The space manager includes a spatial encounter monitor and a coupling operator. An agent perceives other agents using size and motion

detectors and decides its direction and speed based on this perception. Figure 3.3 illustrates an object model diagram of space manger and agents.

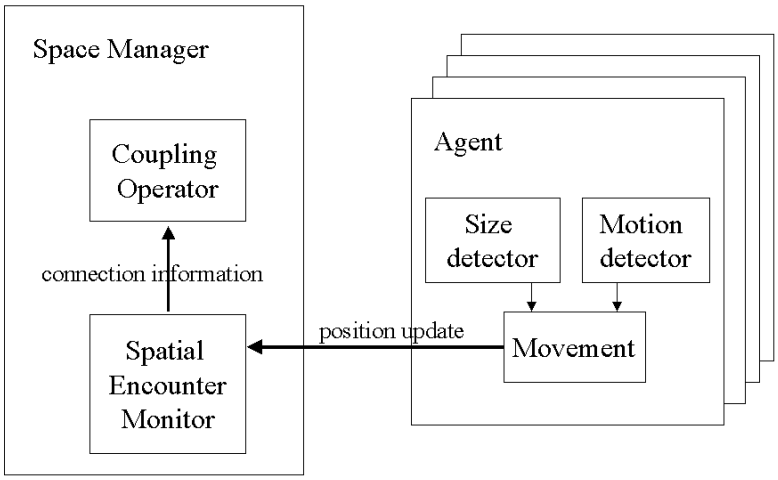


Figure 3.3 Object model diagram of Space Manger and agents

The spatial encounter monitor maintains objects, called “tuples,” to express the information for pairs of agents required to determine distance and assign new quantum values. The attributes of a tuple object include agent identities and their positions, distance between the agents, quantum sizes, connection information, etc. Employing position updates from agents, the spatial encounter monitor determines the spatial relationship among agents by calculating their separation distances. Using these spatial relationships, the spatial encounter monitor determines the connection information among agents. With this connection information, the coupling operator changes the coupling specification supported by DEVS modeling. However, unlike conventional schemes, the

space manager does not transmit spatial relationships to agents. Instead, the coupling operator of the space manager directly performs the filtering operation by adding or removing the couplings (network connections) among agents. Figure 3.4 illustrates the coupling operation supported by the DEVS modeling formalism. In Figure 3.4(a), a coupling exists between the "out" output port of component A and "in" input port of component B due to the coupling specification shown. In Figure 3.4(b), the coupling specification of Figure 3.4(a) is removed from the coupling specification. Adding a connection from the "out" output port of component A to the "in" input port of component C is performed by a new coupling as illustrated in Figure 3.4(c).

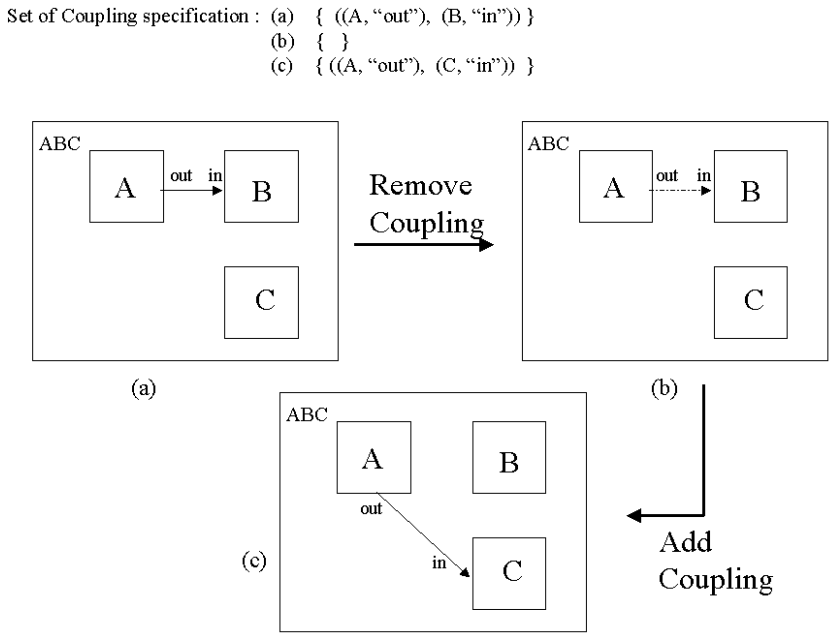


Figure 3.4 Coupling operation in DEVS Modeling

Updating agents' positions leads to communication overhead. Deciding connection information between agents leads to computation overhead. Both communication and computation overheads influence the performance of data management using the space manager. However, we will show that the reduction of communication data and agents' local computations made possible by the space manager can significantly outweigh the communication and computation overheads of the space manager.

3.3 Scalability of the space-based quantization scheme

To model and simulate a large-scale distributed system, we use the DEVS/HLA distributed simulation environment. DEVS/HLA is an HLA-compliant modeling and simulation environment that supports high-level federation development and execution using the DEVS formalism. The formalism provides a well-defined concept of system modularity and component coupling, which is supported and managed by the DEVS/HLA distributed simulation environment. We will discuss this support in more detail later. A large-scale distributed simulation is implemented in the DEVS/HLA distributed simulation environment using several local computers. Several federates are assigned to a local machine. A group of agents is assigned to a federate. In this dissertation, two approaches to supporting the scalability of the space-based quantization scheme in a large-scale distributed simulation are introduced. These approaches are

based on a centralized global space manager and distributed schemes based on local space managers.

3.3.1 Global Space Manager

With the global space manager, a fixed group of agents is assigned to each federate. The global space manager itself resides in a separate federate. All agents send their position updates to the global space manager over a network. The global space manager uses these position updates to determine the connection information among agents, which it then sends to the agent-holding federates. Each such federate has a coupling operator that adds or removes the coupling between agents, between federate input and agent input, and between federate output and agent output using the connection information from the global space manager. Through this coupling operator, traffic message filtering among federates and among agents in the same federate is achieved.

With the global space manager approach, there are two kinds of communication overhead. The first type of overhead results from the position update of each agent to the global space manager. The second type of overhead results from the distribution of connection information, as determined by the global space manager, to each coupling operator on each federate. Figure 3.5 shows the architecture of the global space manager approach with the pursuer/evader model in the DEVS/HLA distributed simulation environment. The global space manger is assigned to a particular federate. A fixed

number of pursuer agents is assigned to each pursuer federate and a fixed number of evader agents is assigned to each evader federate.

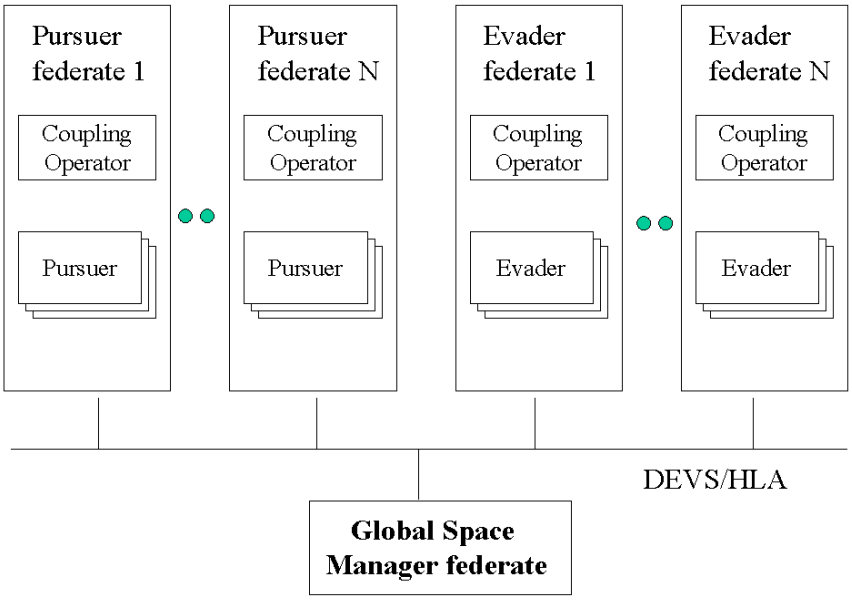


Figure 3.5 Architecture of the Global Space Manager Approach

3.3.2 Local Space Manager

With the local space manager approach, one local space manager and a number of agents are assigned to each federate. Each local space manager receives position information from local agents within the same federate and from external agents in other federates. It uses the updated positions to determine the connection information, which it

then employs to directly perform coupling operations in each federate. In contrast with the global space manager approach, the local space manager does not pass on the connection information to each federate. Nevertheless, it has a larger communication overhead than the global space manager approach because the positions of all agents in each local space manager in each federate must be updated. Figure 3.6 shows the architecture of the local space manager approach with the pursuer-evader model in the DEVS/HLA distributed simulation environment. Each local space manager is assigned to a federate. A number of pursuer agents is assigned to each pursuer federate, and a number of evader agents is assigned to each evader federate. The load balancing problem for both the global space manager and local space manager approaches is discussed in detail in the next section.

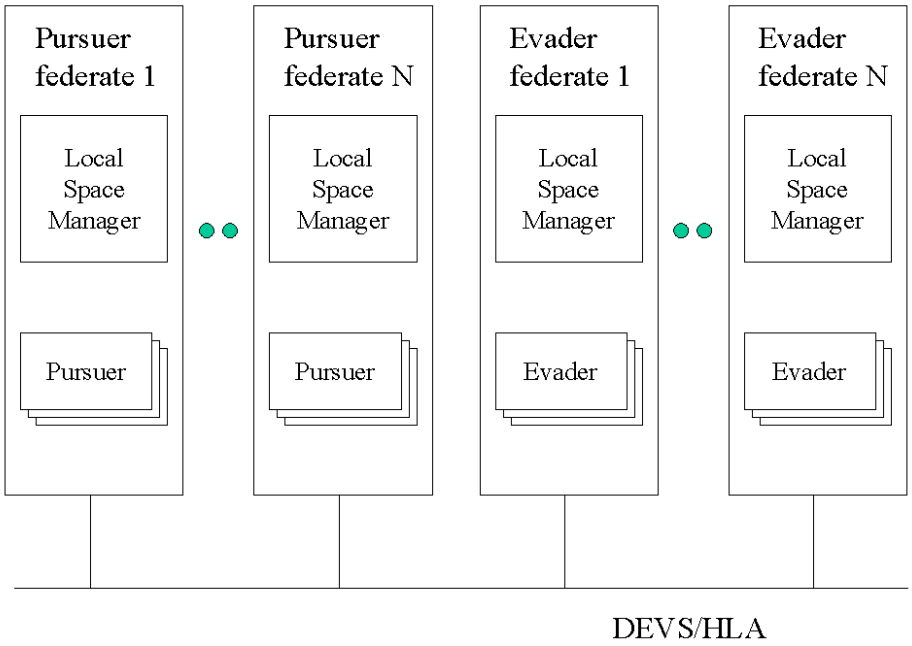


Figure 3.6 Architecture of the Local Space Manager Approach

3.3.3 Load balancing of Global and Local Space Manager Approaches

A federate and a computer have a limitation of CPU and memory usage. In large-scale distributed simulation, as the number of agents simulated increases, a certain number of agents has to be assigned to available federates and available computers. A computer can include several federates. The number of federates assigned to a computer depends on available physical and virtual memory of the computer and the computation load assigned to each federate. A federate can include a certain number of agents. For the pursuer/evader, we have found that 40 is the maximum number of agents that can be

assigned to a federate. Table 3.1 shows the load balancing approach used to assign agents to federates. Each federate includes the same maximum number of agents. As the number of agents increases, the number of federates almost exponentially increases. In Table 3.1, a "+1" refers to the global space manager federate. There is a limitation to the load balancing applied in this approach. The global space manager federate can have a memory shortage problem since the number of tuples it employs increases quadratically with the total number of agents. Further its computation of the connection information and its distribution also increase quadratically as total number of agents increases.

Table 3.1 Load balancing in the Global Space Manager Approach

Total # of Agents	40	80	160	320
# of federates	1	2 + 1	4 + 1	8 + 1
# of Agents in a federate	40	40	40	40

Table 3.2 shows the load balancing strategy used to assign agents to federates in the local space manager approach. As the number of agents increases, the number of agents that can be assigned to each federate decreases. This is so, since the local space manager on each federate has to handle a quadratically increasing number of tuple objects. Therefore, a larger number of federates is needed for this approach compared with the global space manager.

Table 3.2 Load balancing in the Local Space Manager Approach

Total # of Agents	40	80	160	320
# of federates	1	2	8	32
# of Agents in a federate	40	40	20	10

3.3.4 Analysis of message traffic reduction in Global and Local Space Manager Approaches

We expect both global and local space manager approaches to work well and to efficiently reduce message traffic in a large-scale distributed simulation. In this section, we will discuss the advantages and disadvantages of each approach and how they influence performance. The performance of the two approaches depends on the communication and computation overhead required to perform message traffic reduction among agents. There are two types of communication overhead. The first occurs as a result of position updates from each agent to the space manager. The second type of overhead results from the distribution of connection information, which is computed by the space manager, to the coupling operator on each federate. To reduce overhead, distribution of connection information is performed only when the connection information changes. The global space manager approach incurs both types of communication overhead. The first type of communication overhead is relatively small

because position update messages are transferred to the global space manager in a separate federate. However, the second type of communication overhead is very large because messages bearing connection information have to be transferred from the global space manager to the coupling operator in each federate. In contrast, in the local space manager approach the second type of communication overhead does not exist. However, the first type of communication overhead is larger in the local space manager approach because position update messages must be transmitted to each local space manager in each federate. Table 3.3 analyzes the message traffic reduction and the conditions under which we can expect performance improvement in the two approaches. For this analysis, we assume two conditions as follows:

a. There is no communication between pairs of pursuers or evaders.
b. The only communication is messages from pursuers to evaders.

These conditions focus on the inter-federate communication in a distributed simulation. Accordingly, there is no communication among agents in the same federate and only pursuer-to-evader inter-federate communication. As Table 3.3 shows, in the system without a space manager in operation, in a single global state transition the number of messages passed is N*(N-1)/2, since a message is broadcast to all other agents (N is the number of agents). In the systems with space manager, Overall Filtering Rate (OFR) and Connection Change Rate (CCR) are the critical factors to influence the performance in terms of the number of messages passed. Let

FRS: Filtering Rate at Sender federate

FRR: Filtering Rate at Receiver federate

H: Number of Agents to which a message does not have to be transferred

OFR is calculated as:

$$OFR = FRS + FRR \quad (3.1)$$

$$\text{Average of OFR} = OFR / \text{Number of messages sent} \quad (3.2)$$

$$FRS = H / (N/2), \text{ when } FRR = 0 \quad (3.3)$$

$$FRR = H / (N/2), \text{ when } FRS = 0 \quad (3.4)$$

As Equation (3.3) and (3.4) show FRS and FRR cannot simultaneously exist when a message from a sender is transmitted. If FRS > 0, then FRR = 0 (and vice verse). Therefore, in Table 3.3, we use the average of OFR calculated by Equation (3.2) to calculate the number of messages passed. H varies with three factors -- the number of agents, the critical distances for ordered pairs of agents, and the spatial distribution of agents. As the same number of agents is more spatially dispersed with fewer close encounters (i.e., with a greater "mean free path") we can expect that the messages exchange requirements diminish and so H (the number of non-recepients) increases. Conversely, under crowded conditions, H decreases. CCR is calculated as:

$$CCR = L / OC = L / N*(1 + N/2) \quad (3.5)$$

Overall Connections (OC)

= Connections corresponding to Filtering at Sender federate

+ Connections corresponding to Filtering at Receiver federate

$$= N + N^2/2 = N*(1 + N/2) \quad (3.6)$$

Overall Connections (OC) is the number of connections changeable and is calculated by Equation (3.6). OC varies with two factors -- the number of agents, number of quantum sizes employed. Equation (3.6) indicates the OC when the number of quantum sizes are two, such as zero and infinity. L is the number of connections actually changed and grows with the number of agents (N) and number of quantum sizes because L is a subset of OC. L mainly depends on a change in quantum size, which requires a change in coupling in the implementation we discuss here.

Table 3.3 Analysis of Message Traffic Reduction

(N: Number of Agents, M: Number of Federates, OFR: Overall Filtering Rate, CCR: Connection Changing Rate)

Approaches	Number of messages passed	Coefficient of N^2 as N ? ?	Condition for message traffic reduction
NO Space Manager	N*(N-1) / 2	1 / 2	
Global Space Manager	N + (CCR)* (N)*(1 + N/2) + (1 – OFR) * N*(N-1)/2	CCR/2 + (1 – OFR)/2	CCR + (1 – OFR) < 1
Local Space Manager	N * (M-1) + (1 – OFR) * N*(N-1)/2	(1 – OFR)/2	As (N/M) ? ? , (1 – OFR) < 1

The analysis in Table 3.3 reveals that, especially for large numbers of agents encapsulated into federates, we should expect the greatest message reduction to come from the local space manager approach. Figure 3.7 compares the computation time of the two approaches. With both approaches, computation overhead is necessary to regulate the connection information among agents in a space manager. The global space manager has a larger computation load than that of a local space manager because it must regulate the connection information among all agents in all federates. This large computation load causes a bottleneck problem. While the global space manager calculates the data, computation of agents within the other federates is delayed and, therefore, the logical time of the other federates is not advanced. The output operation time required for the distribution of connection information messages to the other federates exacerbates the

bottleneck problem. In contrast, when there is a local space manager in each federate, the computation load is reduced because each local space manager regulates the connection information only for those agents within its own federate. With the local space manager approach, the connection computation is divided up and pieces are assigned to local space managers for concurrent processing. In addition to the advantage of load partitioning, no output operation time is required for the distribution of connection information messages to each federate using the local space manager.

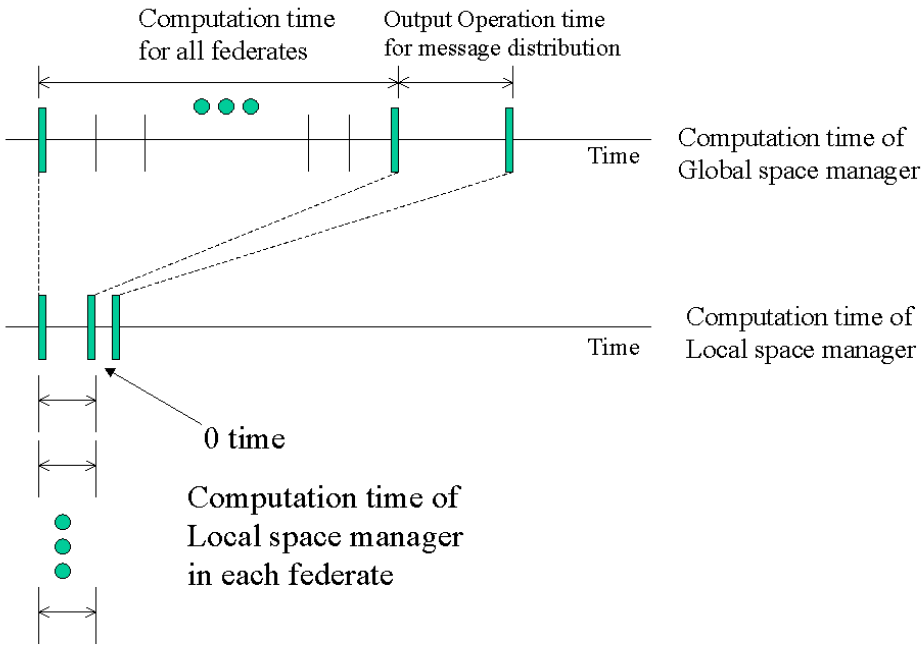


Figure 3.7 Concurrent processing in the local space manager approach

4 DEVS/HLA-INTERFACE

4.1 HLA-Interface

General purpose object-oriented HLA-Interface was developed at Lockheed Martin Space Systems' Advanced Simulation Center [58]. The HLA-Interface is complementary to the HLA/RTI, which is the standard DMSO HLA implementation. This HLA-Interface automates the declaration of HLA classes and the registration of all HLA objects. It also performs all the calls and callbacks to and from HLA.

Figure 4.1 shows the HLA-Interface layered structure. The HLA-Interface supports the modeling and simulation of the non-DEVS (general) models as well as the DEVS models. Non DEVS models directly access the functions of the HLA-Interface layer and takes a part in a distributed simulation on the HLA-Interface layer. To construct and simulate DEVS models, the DEVS/HLA-Interface layer is provided. The DEVS/HLA-Interface layer was developed by separating the HLA components out from the DEVS components in the DEVS/HLA distributed simulation engine developed at The University of Arizona. Then the separated HLA components were included into the HLA-Interface layer.

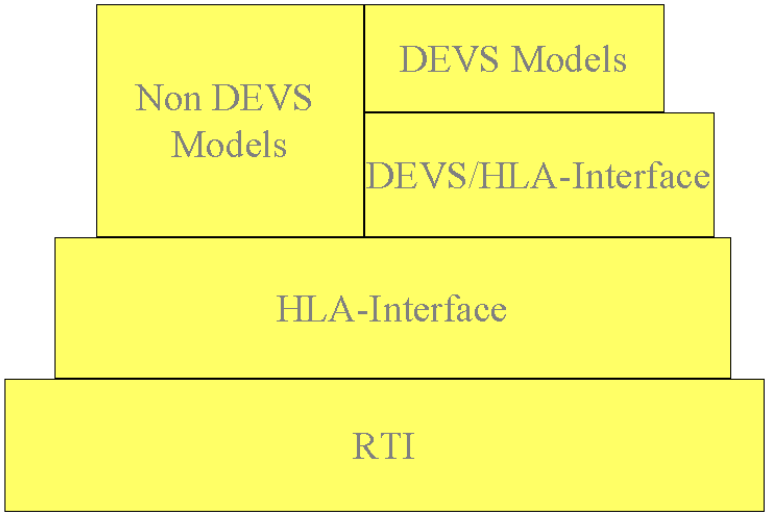


Figure 4.1 HLA-Interface layered structure

4.2 DEVS/HLA-Interface Environment

In the DEVS/HLA-Interface environment, a developer defines DEVS models in the DEVS model layer on top of the DEVS/HLA-Interface layer. To develop an HLA federate in the DEVS model layer, the developer creates a specified federate component, which is the top DEVS component in each federate. As Figure 4.2 shows, the top DEVS component in the DEVS model layer can access all methods in the Federate class of the DEVS/HLA-Interface layer which is hidden to the DEVS model developer.

The Federate class of the DEVS/HLA-Interface layer supports functions which allow it to connect to the HLA_Federate class in the HLA-Interface layer. The HLA_Federate class in turn provides services to work a distributed simulation of the HLA-Interface layer. The HLA-Interface layer is responsible for the inter-federate communications (RTI interaction and attribute communications) that transfer data to and from the DEVS/HLA-Interface layer.

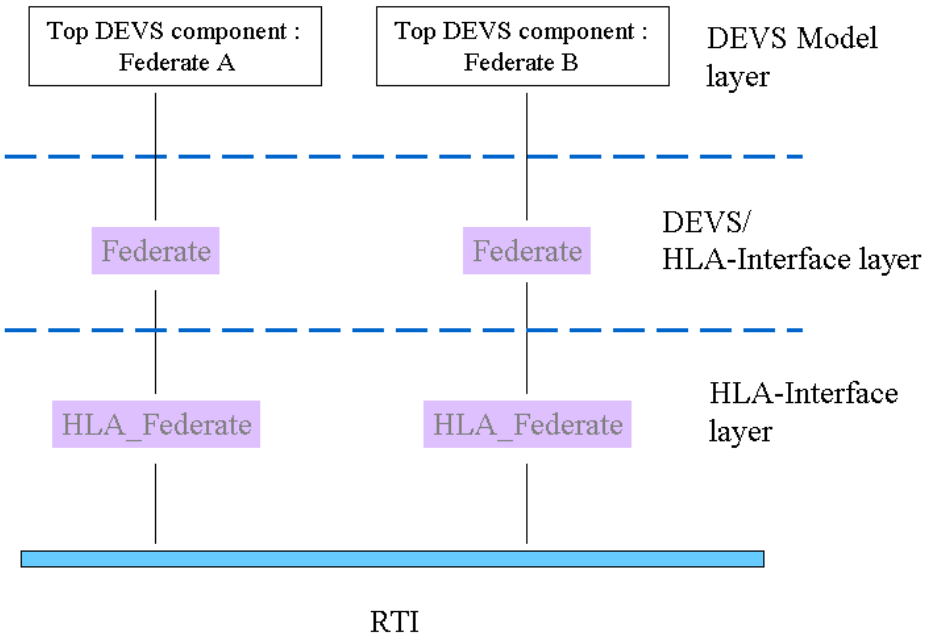


Figure 4.2 DEVS/HLA-Interface layered modeling

4.3 RTI communications

The attribute communication and the interaction communication are the two types of communications by provided the HLA-Interface layer. Figure 4.3 shows how the attribute communication performs in the DEVS/HLA-Interface layer and the HLA-Interface layer. In the sender federate, the attribute value of the attributeList component in the DEVS/HLA-Interface layer is transferred to the AttributeList component in the HLA-Interface layer and, using the updateAttributeValues() function, the value is sent to the RTI executive.

In the receiver federate, using the reflectAttributeValues() function, the appropriate callbacks from RTI are automated. To invert the sender's process, the received attribute value is transferred from the AttributeList component in the HLA-Interface layer to the attributeList of DEVS/HLA-Interface layer. For the DEVS time management in distributed simulation, the attribute communication and the quantizer component in the DEVS/HLA-Interface layer were used [50, 51]. The quantizer checks whether the simulation time advance has crossed over a certain time unit or not. If the simulation time has crossed, the quantizer sends the time value from the attributeList component in the DEVS/HLA-Interface layer to the AttributeList component in the HLA-Interface layer.

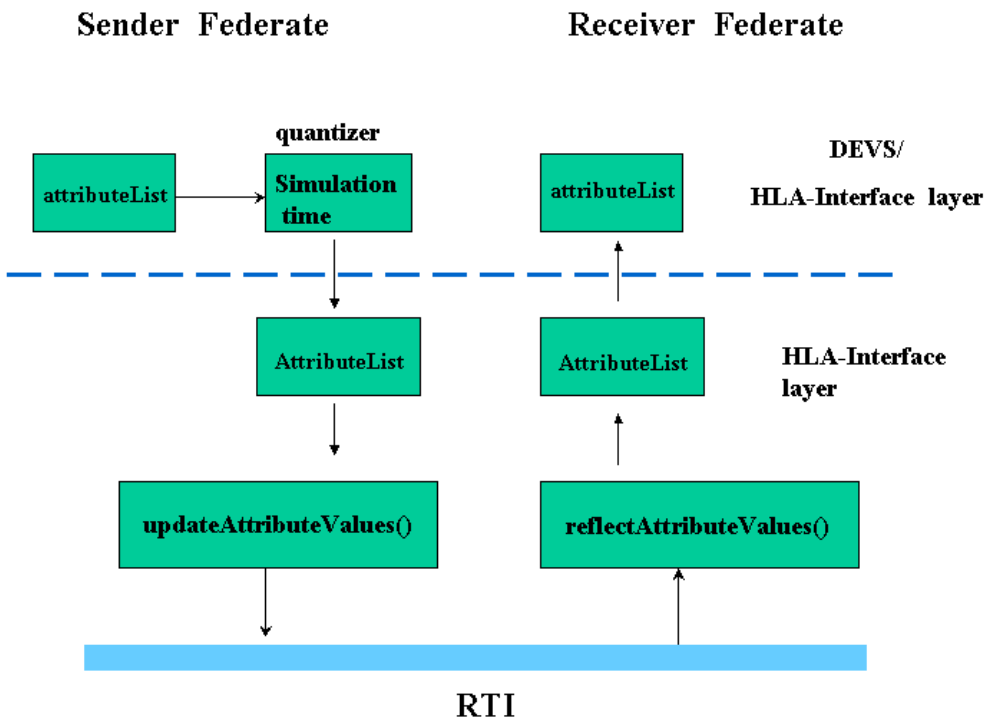


Figure 4.3 Attribute communication in the DEVS/HLA-Interface layer and the HLA-Interface layer

Figure 4.4 illustrates how the interaction communication works in the DEVS/HLA-Interface layer and the HLA-Interface layer. In the sender federate, the HLA_Interaction Interface component in the DEVS/HLA-Interface layer receives DEVS messages from DEVS models, extracts the data value from the DEVS message, and sends the data value to the HLA-Interface layer. In the HLA-Interface layer, with the SendInteraction() function, the data value from the DEVS message is sent to the RTI executive. In the receiver federate, the callbacks from RTI are automated with the

ReceiveInteraction() function. Conversely in the sender federate, the received data value is transferred from the HLA-Interface layer to the DEVS/HLA-Interface layer, in which a DEVS message with the received data value is created and transferred to the upper layer. The data communication, among the DEVS components distributed in federates, works with this interaction communication.

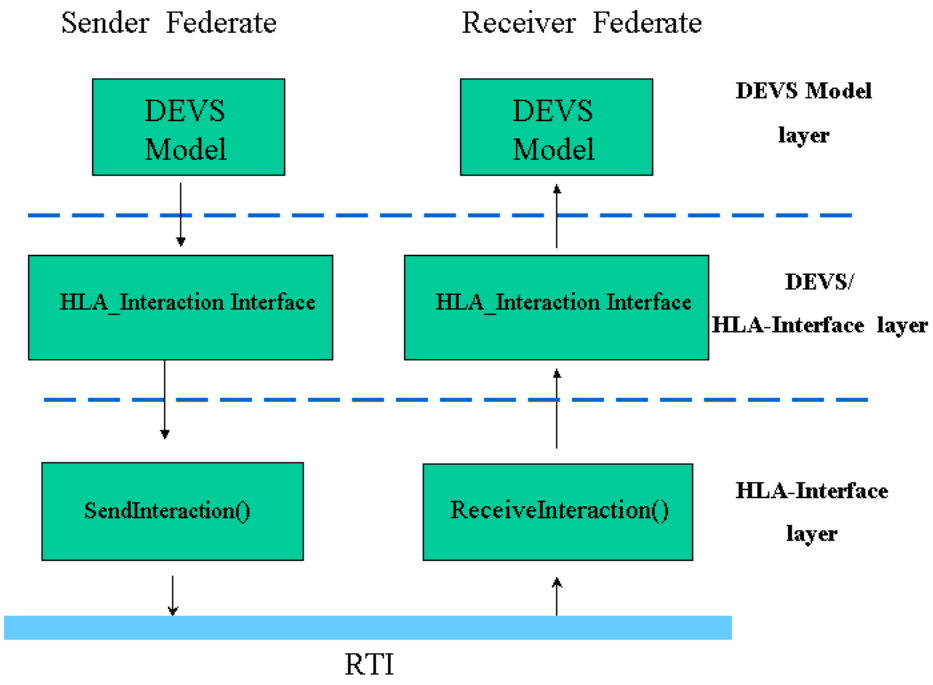


Figure 4.4 Interaction communication in the DEVS/HLA-Interface layer and the HLA-Interface layer

4.4 The upgraded DEVS/HLA-Interface Environment

The HLA-Interface developed at Lockheed Martin Space Systems' Advanced Simulation Center was implemented with C++. In this dissertation, the HLA-Interface was integrated to Java using code translated from the C++ code. Also, the Java-based DEVS/HLA-Interface environment upgrades the DEVS/HLA-Interface environment in C++ [58]. The upgraded DEVS/HLA-Interface includes three main differences from the DEVS/HLA-Interface with C++. These differences are user interface, data casting, and class hierarchy as described next.

4.4.1 User Interface

The HLA_Federate class, in the HLA-Interface layer in C++, has the communication protocol services to define inter-federate communications (interaction and attribute communications). However, the DEVS model developer cannot access the communication protocol services of the HLA_Federate class in the HLA-Interface layer. Therefore the DEVS model developer uses only the previously defined DEVS/HLA interaction and attribute communications for distributed simulation. In order to compensate for the disadvantage of the DEVS/HLA-Interface in C++, we allow the DEVS model developer to define and to set up the interaction and the attribute communications in the top DEVS component in the DEVS model layer. Thus, the DEVS model developer can access the communication protocol functions (for the interaction

and the attribute communications) in the DEVS model layer. Thus, the DEVS/HLA-Interface in Java provides an easier user interface than the DEVS/HLA-Interface in C++. Figure 4.5 and Figure 4.6 compare the definition and setup of interaction and attribute communications for the two DEVS/HLA-Interface environments.

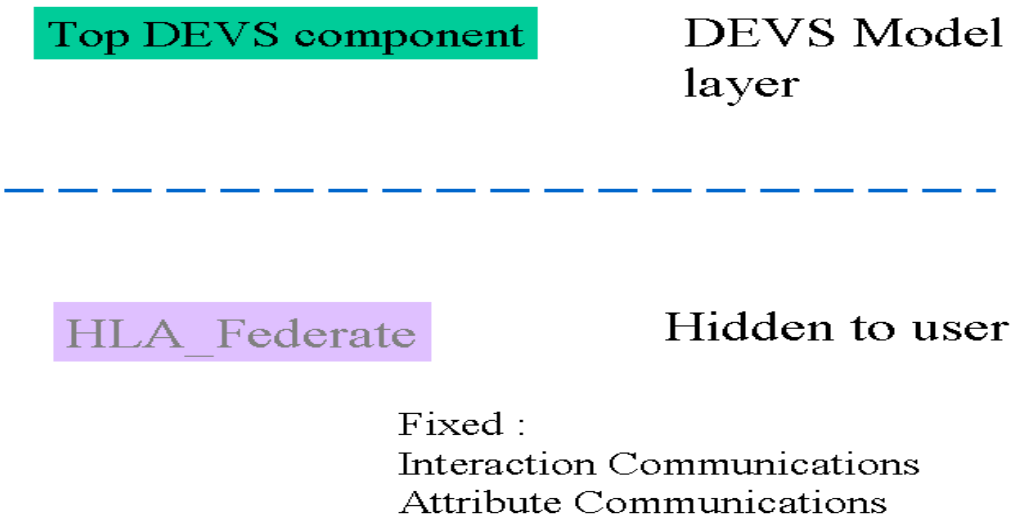


Figure 4.5 The definition and setup of interaction and attribute communications in the DEVS/HLA-Interface environment in C++

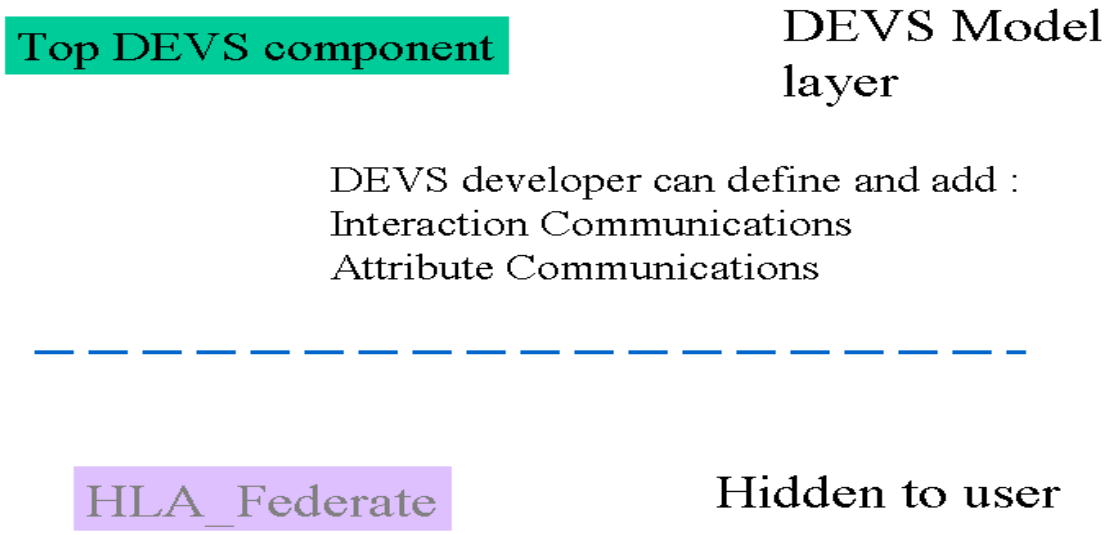


Figure 4.6 The definition and setup of interaction and attribute communications in the DEVS/HLA-Interface environment in Java

Furthermore, in the DEVS/GDDM environment to be introduced in the next chapter, the DEVS model developer does not ever have to define interaction and attribute communications. That is the developer needs only work with DEVS models. The DEVS/GDDM environment takes the inter-federate connection information from DEVS models and automatically defines and sets up interaction and attribute communications. Figure 4.7 illustrates the definition and setup of interaction and attribute communications in the DEVS/GDDM environment.

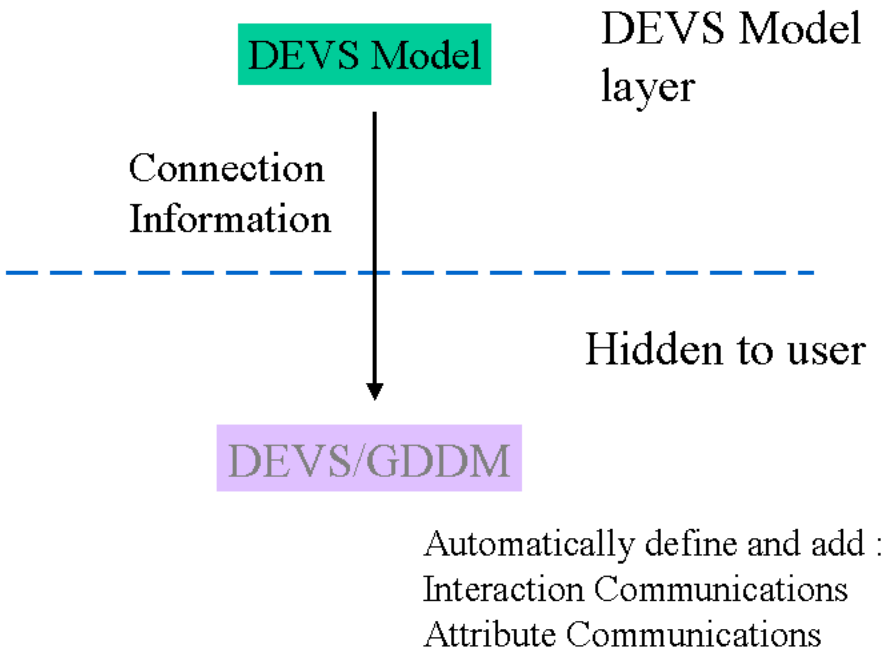


Figure 4.7 The definition and setup of interaction and attribute communications in the DEVS/GDDM environment

4.4.2 Data Casting

In the DEVS model layer in the sender federate, a sender agent (a DEVS model) outputs a DEVS message. A DEVS message includes data values of any type (double, float, integer, string, etc.). In the HLA-Interface layer in C++, the data value from the DEVS message is cast into its data type and is sent to RTI for interaction communication. To do exact by the same data type casting in the HLA-Interface layer, the type of the received data from the upper layer has to be previously known. Otherwise, an additional

operation to know the type of received data is needed and thus causes the overhead of system computation in the HLA-Interface layer. The same problem occurs in the HLA-Interface layer in the receiver federate. The type of the data value received from RTI callback function is unknown; therefore the additional operation to know the type of received data is needed in the HLA-Interface layer in the receiver federate. Figure 4.8 illustrates the data casting necessary to perform the RTI interaction communication in the DEVS/HLA-Interface environment in C++.

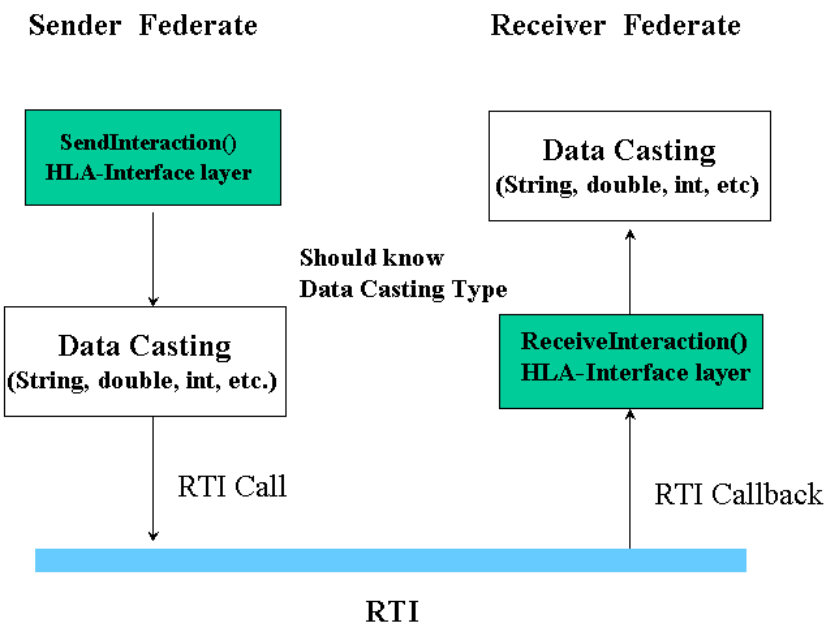


Figure 4.8 Data casting in the DEVS/HLA-Interface environment in C++

To fix this data casting problem, Java enabled byte stream data casting is used in the new HLA-Interface layer. In a sender federate we take any type of data value from a DEVS message and cast it into the byte stream. The byte stream is then transferred to RTI using interaction communication.

Figure 4.9 illustrates the data casting of the RTI interaction communication in the DEVS/HLA-Interface environment with Java. At the HLA-Interface layer in the receiver federate, the byte stream received from the RTI callback function enters into the entity object. The entity object is a general object that can contain any type of data value of DEVS messages. The DEVS message, which includes the entity object, is transferred to the upper layer. A receiver agent (a DEVS model) in the DEVS model layer receives the DEVS message, which includes the entity object (with byte stream), and casts the byte stream into the exact data type since both sender and receiver DEVS models know the exact data type sent and received. Byte stream data casting, in the DEVS/HLA-Interface environment with Java, provides data transparency between the HLA-Interface layer and the RTI, and obviates the additional operation for casting the exact data type.

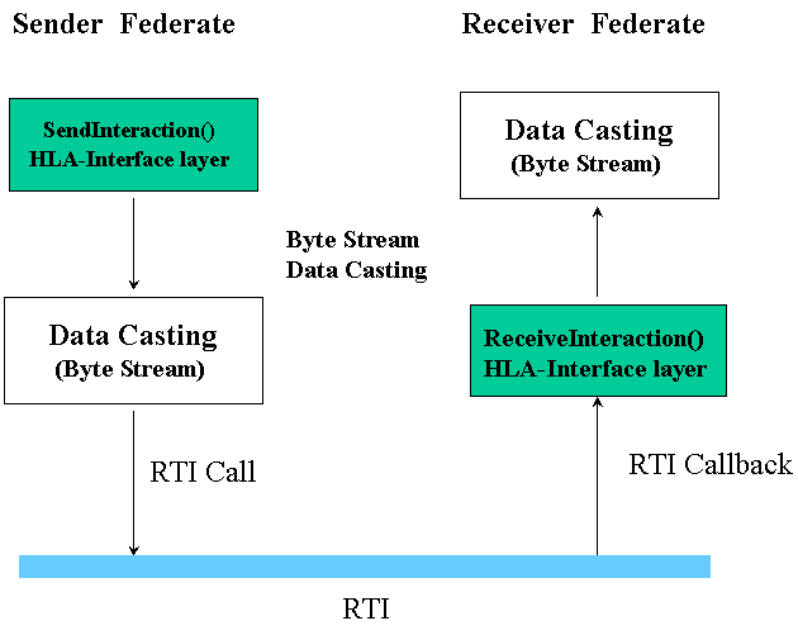


Figure 4.9 Data casting in the DEVS/HLA-Interface environment with Java

4.4.3 Data structure and Container sub-layer class hierarchy

The HLA-Interface layer has its own data structure (e.g., Element, Set, etc.) to support its class development and the functions of the classes of the HLA-Interface layer. The DEVS/HLA-Interface layer has a container sub-layer to support the DEVS modeling [59, 60]. In order to connect the data structure of the HLA-Interface layer to the container sub-layer of the Java-based DEVS/HLA-Interface layer, we made the top class (Entity class) of the data structure of the HLA-Interface layer inherit the top class (entity class) of the container sub-layer of the DEVS/HLA-Interface layer. Figure 4.10 illustrates the

class hierarchy of the data structure and the container sub-layer. Using the polymorphism in this class hierarchy, classes in the DEVS/HLA-Interface layer or the HLA-Interface layer can be developed using both the data structure and the container sub-layer.

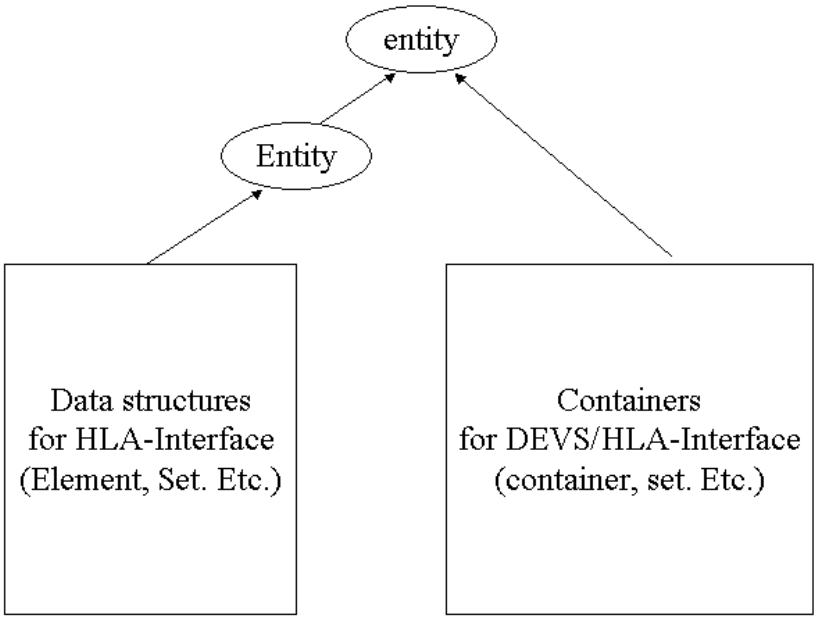


Figure 4.10 Class hierarchy of the data structure of the HLA-Interface layer and the container sub-layer of the DEVS/HLA-Interface layer

4.5 Summary of DEVS/HLA-Interface environment

We introduced a general purpose object-oriented HLA-Interface developed at Lockheed Martin. The HLA-Interface includes the DEVS/HLA-Interface environment

(extended from the DEVS/HLA developed by the University of Arizona) that allows to model and simulate DEVS models distributed at multiple federates over network. In this dissertation, we upgraded the Java-based DEVS/HLA-Interface environment from the DEVS/HLA-Interface environment in C++. The upgraded DEVS/HLA-Interface environment provides more useful user interface and efficient implementation (by changing data casting and class hierarchy).

In next chapter, using the upgraded DEVS/HLA-Interface environment, we will create the DEVS/GDDM modeling and simulation environment that provides GDDM (Generic Data Distribution Management) services in distributed simulation. The upgraded DEVS/HLA-Interface environment supports the system modeling facility (based on DEVS formalism) and the distributed simulation facility (using HLA) of the DEVS/GDDM environment.

5 DEVS GENERIC DATA DISTRIBUTION MANAGEMENT (GDDM) ENVIRONMENT

5.1 Motivation

A large-scale, distributed simulation is characterized by numerous interactive data exchanges among simulation entities dispersed among computers that are networked together. The development of message traffic reduction schemes to reduce the interactive messages among simulation entities has drawn the attention of many researchers as one means of achieving greater scalability. At present, with increasing demand for distributed simulation, message traffic reduction schemes for distributed simulation with reasonable communication and computation resources are needed more and more.

The major message traffic reduction schemes proposed for improved communication data management are the quantization scheme and interest management scheme. The quantzation scheme has two types: non-predictive and predictive. The interest management scheme includes the spatial encounter prediction scheme and the Data Distribution Management (DDM) service of High Level Architecture (HLA).

The Data Distribution Management (DDM) service of HLA tries to filter out irrelevant data among federates. However, as mentioned in chapter 3, the DDM of HLA has several disadvantages, which inhibits applying modeling and simulation to a large of variety of problems.

In this dissertation, we developed the DEVS/GDDM simulation environment that uses the interest-based quantization scheme (which combines the quantization scheme and the interest management scheme) and performs the effective message filtering between senders and receivers. This environment overcomes the disadvantages of DDM of HLA and performs a distributed simulation with reasonable communication and computation resources. Figure 5.1 illustrates how message filtering between senders and receivers is supported by the DEVS/GDDM simulation environment.

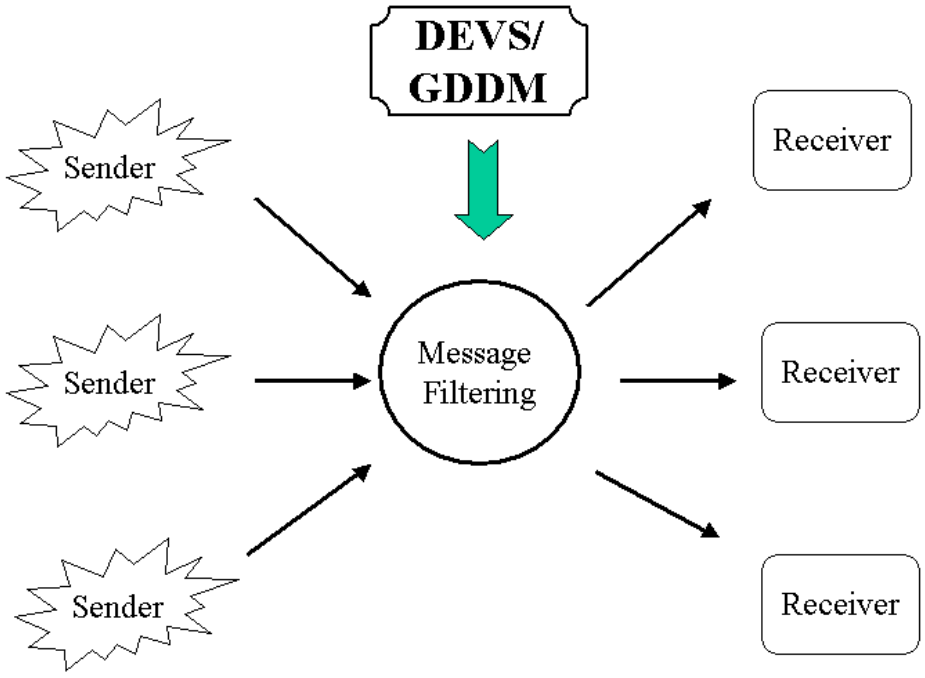


Figure 5.1 Message Filtering between senders and receivers

5.2 DEVS/GDDM Structure

As Figure 5.2 shows, the DEVS/GDDM environment, implemented as the upper layer of the DEVS/HLA-Interface layer, supports a portability of models across platforms at a high level of abstraction. Thus, DEVS models, based on object-oriented design, can be developed and reused on the DEVS/GDDM layer; and they can easily be ported across distributed platforms. Figure 5.3 summarizes the roles of each layer in DEVS/GDDM layered structure. The roles of the DEVS/HLA-Interface and the HLA-Interface layers were discussed in chapter 4. The only difference is that the DEVS/GDDM layer replaces the roles of the DEVS/HLA-Interface layer (setup of attribute and interaction communications) and adds DEVS/GDDM component specifications. It also supports more simple user modeling in the DEVS model layer.

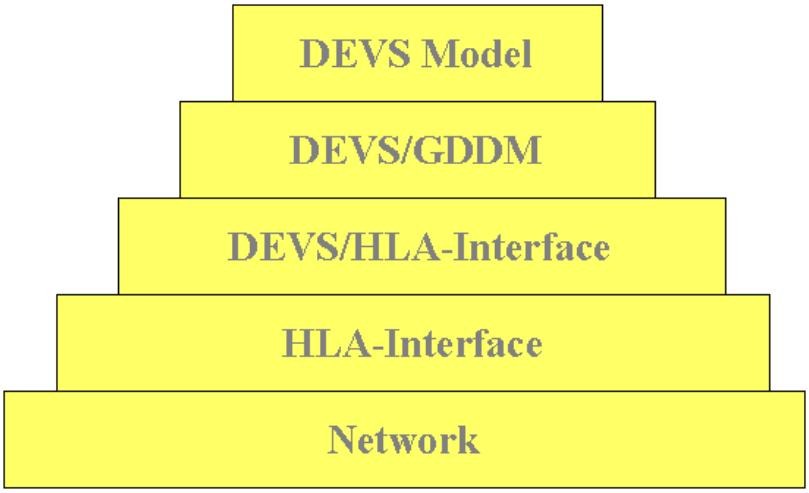


Figure 5.2 DEVS/GDDM layered structure

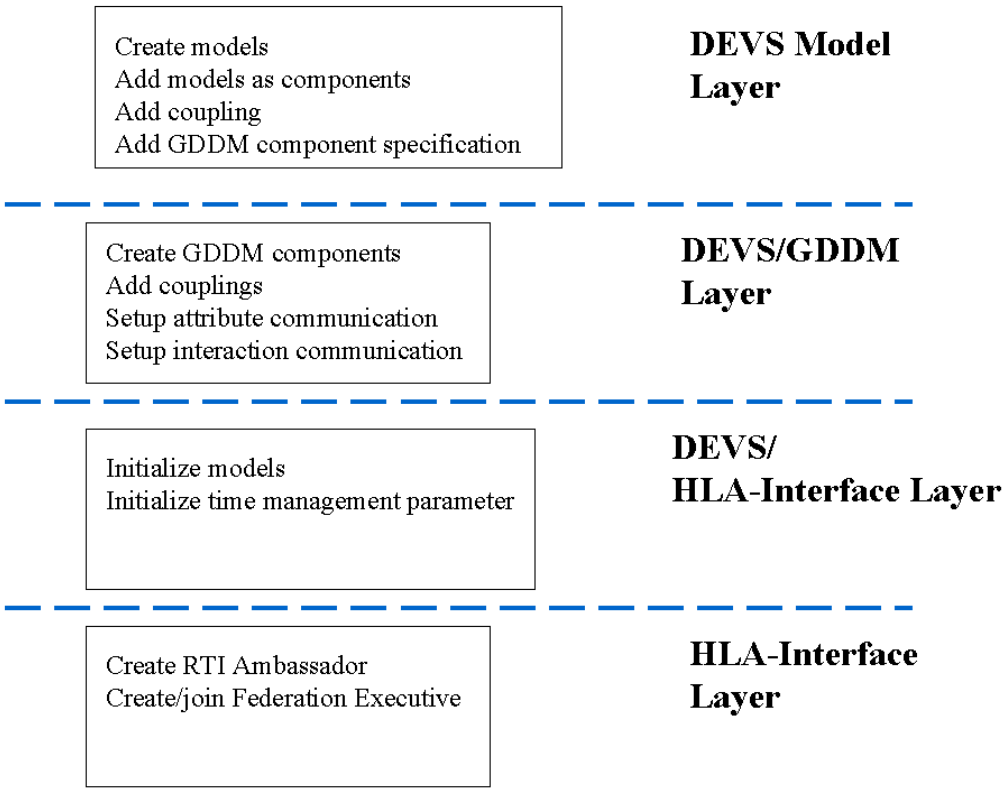


Figure 5.3 Roles in each layer of the DEVS/GDDM layered structure

Communication and data exchange among DEVS components distributed in multiple federates are supported by the DEVS/GDDM environment. The DEVS/GDDM layer takes the DEVS coupling information from DEVS models, automatically defines the HLA interaction communications using this coupling information, and performs HLA/RTI communications. Therefore the DEVS/GDDM environment provides a friendly user interface, and the developer only defines models on the DEVS model layer. Figure 5.4 illustrates the HLA interaction communication setting provided by the DEVS/GDDM environment.

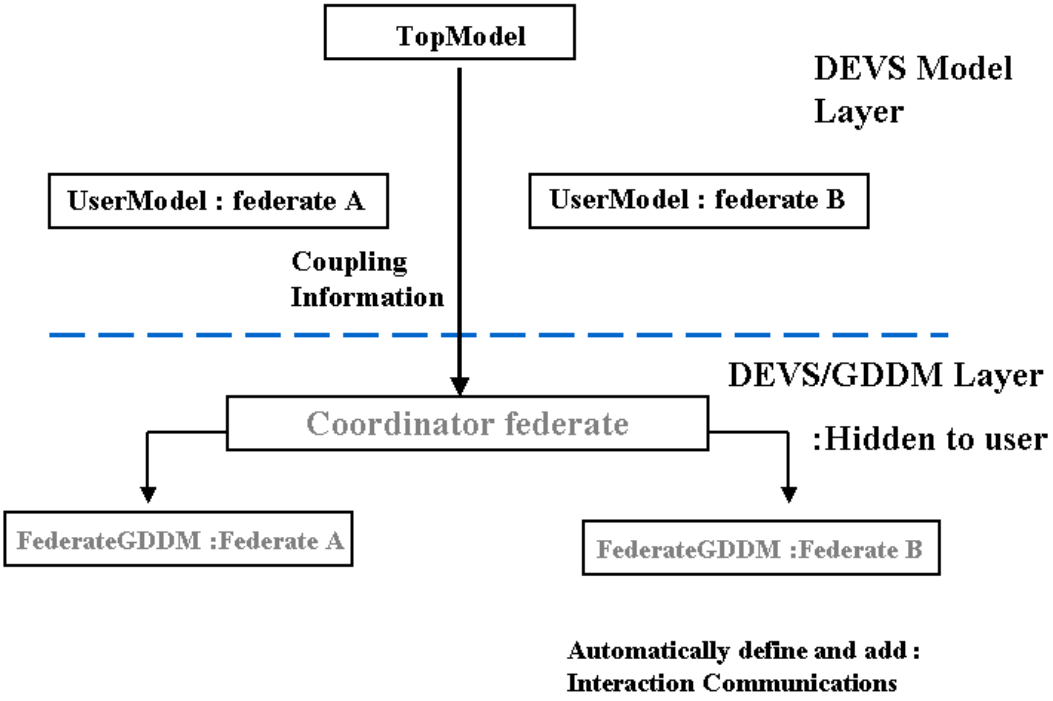


Figure 5.4 HLA Interaction communication setting in the DEVS/GDDM environment

5.3 DEVS/GDDM Components

To perform the interest-based quantization scheme in the DEVS/GDDM environment, several components are developed in this dissertation. The major components are initializer, space manager, and message handler.

The initializer allows a model developer to model any application easily. The initializer gets information from an application model needed to implement the interest-

based quantization scheme of the DEVS/GDDM environment. This information includes the number of agents, how they are distributed in the separate federates, and the coupling information among agents. The initializer creates the other DEVS/GDDM components (space manager and message handler) and creates the couplings between the user DEVS model as well as the other DEVS/GDDM components, and the couplings among all DEVS/GDDM components. Thus, the initializer sets up the message communication among all DEVS/GDDM components and the user model.

As Figure 5.4 illustrates, the coordinator federate takes the coupling information from the DEVS top model and sends it to the FederateGDDM components in the other federates. Each FederateGDDM component includes an initalizer component. Using this coupling information, the initializer creates the HLA interaction communications that are supported by the DEVS/HLA-Interface layer. Therefore DEVS modeling in the DEVS/GDDM environment is not different from DEVS modeling in the DEVS/JAVA [62] or in the DEVS/CORBA [61] environments. This means that the DEVS/GDDM environment provides DEVS modeling transparency with respect to other DEVS-based environments.

The space manager is the main component in the DEVS/GDDM layer. In order to work out the proper quantum size to allocate to sender and receiver agent pairs, the space manager has two sub-components: the tuple and the decision-maker. The tuple component maintains the data for deciding the proper quantum size allocations. This data is application dependent and may vary during run time.

In the pursuer/evader model that will be discussed in the chapter 7, the data for deciding the quantum size are the distances between sender and receiver agents. The tuple component receives and updates the data for deciding the quantum size from the agent as it changes its own attributes. For example, in the pursuer/evader model, whenever any agent changes its position, the space manager must collect the updated position and re-compute the distances between agents. Using the updated position of the tuple component, the decision-maker component determines the exact quantum size for all sender and receiver pairs. To make this decision, the decision-maker component employs a quantum decision table, which specifies how quantum sizes are related to data values in the tuple component. Using the quantum decision table, the decision-maker finds the new quantum sizes, and the space manager then sends the new sizes to the filtering components. These are the message handlers and user model components. Figure 5.5 illustrates the component diagram in DEVS/GDDM layer and the data flow among DEVS/GDDM components.

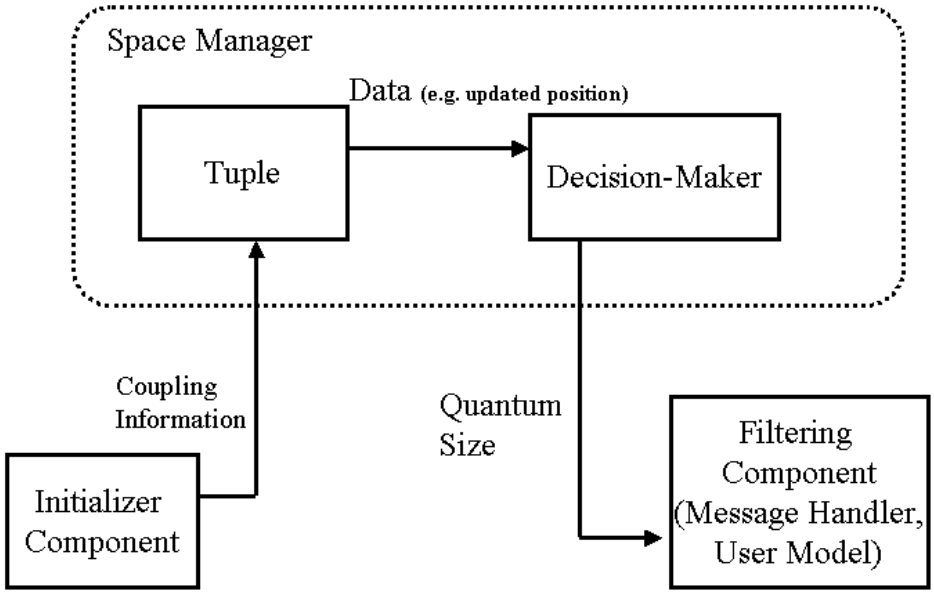


Figure 5.5 Component diagram in DEVS/GDDM layer

The message handler collects the output messages from a user model and distributes the received messages to the proper user models in the other federates. The DEVS/GDDM environment supports three methods employed by the interest-based quantization scheme. These are non-predictive, predictive, and multiplexing interest-based quantization. To perform the three methods, the message handlers in the DEVS/GDDM layer have the functions for performing quantization (non-predictive and predictive) and multiplexing. The message handler is specialized into several types: non-predictive, sender predictive, receiver predictive, sender multiplexing, and receiver de-multiplexing. The non-predictive message handler performs non-predictive quantization.

The sender predictive and the receiver predictive message handlers perform the predictive quantization. The sender multiplexing and the receiver de-multiplexing message handlers are used for the multiplexing. Another role of the message handler is to reduce the number of, and efficiently utilize the HLA interaction communications. This role can increase the scalability of DEVS/GDDM environment.

Figure 5.6 illustrates the information flow from users and their DEVS models to the DEVS/GDDM layer. The initializer gets the coupling information from a user model. The user informs the space manager the quantum decision table, the quantized variables, and the chosen interest-based quantization method. The message handler also gets the message type (e.g. message dimensions) from the user.

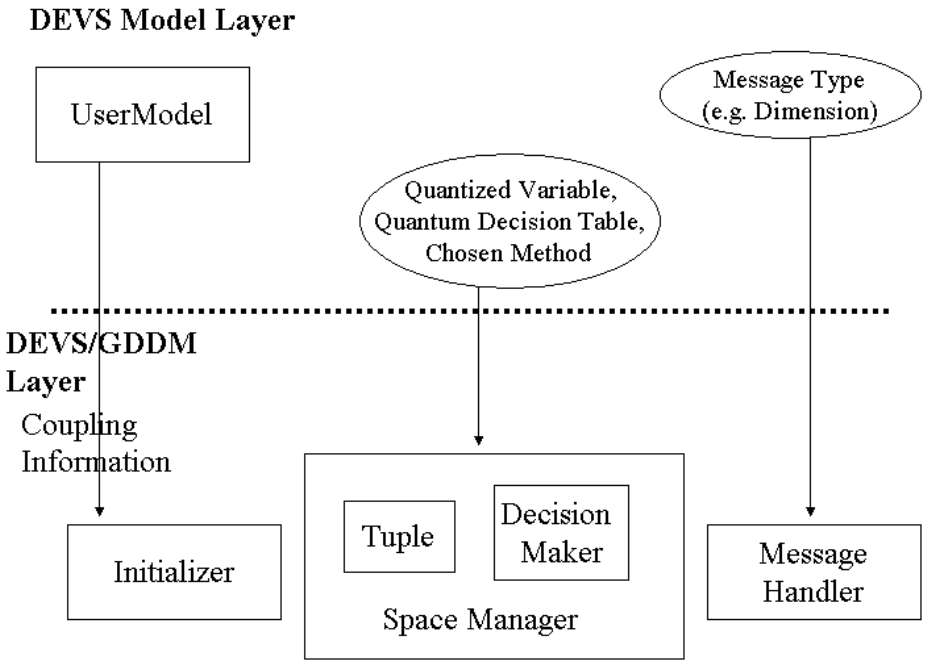


Figure 5.6 Information flow from the user and DEVS models to DEVS/GDDM layer

5.4 DEVS/GDDM Flow of Execution

Figure 5.7 illustrates the DEVS/GDDM flow of execution.

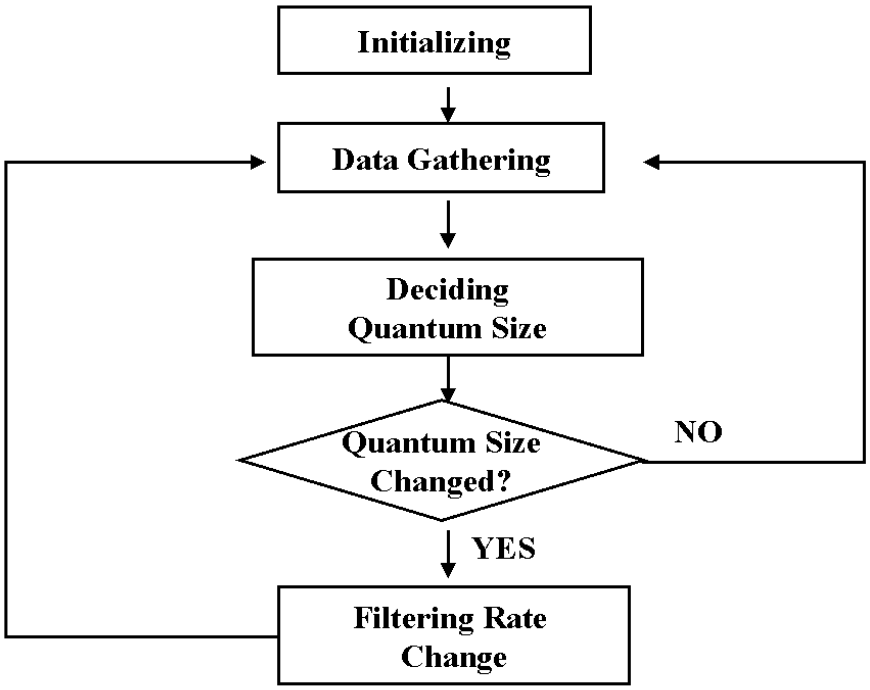


Figure 5.7 DEVS/GDDM Flow of Execution

Initializing

Initializing sets up the environment needed to perform the interest-based quantization scheme in the DEVS/GDDM environment. In this initializing step, the couplings among the DEVS models and the DEVS/GDDM components are created and the HLA interaction communications are setup between the DEVS/GDDM layers in the different federates. The space manager then collects the initial information from all agents needed to decide the initial quantum sizes. For example, it collects the initial positions of all agents.

Data Gathering

During simulation, the space manager gathers the data for deciding proper quantum sizes among sender and receiver pairs.

Deciding Quantum Size

As the data for deciding quantum sizes changes, the decision-maker finds the new quantum sizes from the quantum decision table.

Quantum Size Changed?

The space manager checks whether the current quantum size is different from the old quantum size. If the quantum size is changed, the flow follows the "YES" direction. If the quantum size is not changed, the flow follows the "NO" direction. If there is a "NO" direction flow, the execution goes back to the Date Gathering.

Filtering Rate Change

When the filtering component for quantization receives the new quantum size from the space manager, it changes the internal quantum size. Thus, its message filtering rate is changed and execution goes back to the Date Gathering phase and continues.

5.5 Interest-based Quantization Scheme in DEVS/GDDM

The DEVS/GDDM environment supports three methods performed by the interest-based quantization scheme. These are non-predictive, predictive, and multiplexing interest-based quantization.

5.5.1 Non-predictive Interest-based Quantization Method

The non-predictive interest-based quantization method filters the output messages of the sender agent by using a non-predictive message handler which includes the quantizers. The output message from the sender agent can contain multi-dimensional values (x, y, …). For example, if the output message represents the position in space, the three-dimensional values (x, y, z) are contained within the message. In order for the quantizer to quantize multi-dimensional values, it has several types of quantizers related to the dimensions. Thus, DEVS/GDDM layer includes distinct quantizers that support quantization of different dimensional messages. Figure 5.8 illustrates the operation of the non-predictive interest-based quantization method.

Recall that order to perform the non-predictive interest-based quantization method, the space manager collects the data from the sender and receiver agents to decide the quantum size. A certain quantum size is specified for each sender and receiver pair. The data for deciding the quantum size for each sender and receiver pair depends on each application. For example, in the pursuer and evader model, the space manager collects the

positions of the pursuer and the evader and uses the distance between the pursuer and the evader to decide the quantum size for the pursuer and evader pair. After deciding, the space manager checks whether the decided quantum size has changed or not. If the decided quantum size has changed, it is sent to the proper quantizer assigned to the proper sender agent from the space manager. When the quantizer receives the new quantum size, it changes to the new quantum size and changes its filtering rate.

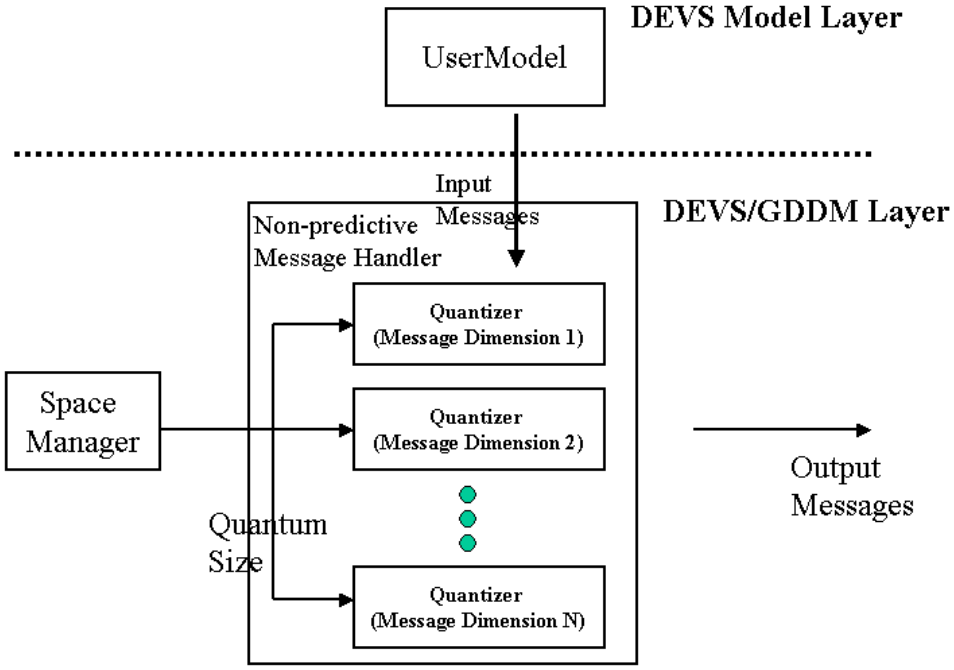


Figure 5.8 Operation of the Non-Predictive Interest-based Quantization method

5.5.2 Predictive Interest-based Quantization Method

The predictive interest-based quantization method, based on the predictive quantization discussed in chapter 2, filters the messages in the sender agent itself. To perform the predictive interest-based quantization method, a sender agent has a model to perform the predictive quantization. In this dissertation, the DEVS predictive integrator is used as the model for the predictive quantization. In the DEVS predictive integrator, the next crossing of the boundary is predicted, and the input value and the time step are variables. In contract, in order to calculate the output value in a simple integrator (such as a DTSS integrator), the varying input value is multiplied into a fixed time step. Chapter 6 discusses the DEVS predictive integrator in depth. Figure 5.9 illustrates the operation of the predictive interest-based quantization method.

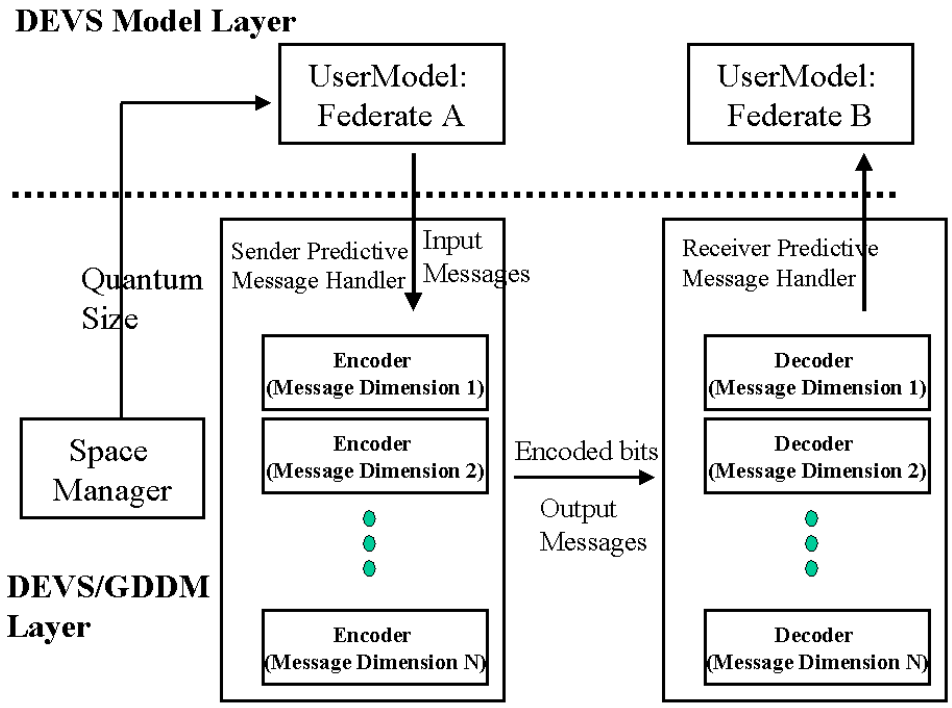


Figure 5.9 Operation of the Predictive Interest-based Quantization method

In the conventional predictive filtering method, a fixed quantum size is used to perform the predictive quantization. The predictive interest-based quantization method allows the space manager to change the quantum size. In predictive filtering, the message size can be reduced tremendously since both sender and receiver agents know the current quantum size, therefore, the sender agent sends only –1/+1 value and the receiver agent generates the original value of the sender agent using the –1/+1 value and the current quantum size. Hence, the data bit of the one-dimensional message is only one bit for a sender and receiver pair.

The DEVS/GDDM environment supports the communication of the message that contains the multi-dimensional values. In the DEVS/GDDM layer, a sender predictive message handler sends –1, 0, and +1 values for each dimension to a receiver predictive message handler. Note that 0 is needed since a sender may not have crossed a threshold at the time of sending the message. As the number of the message dimensions increases, the number of message alternatives, which are represented with –1, 0, and +1 values, also increases. The number of message alternatives is calculated by:

$$\text{Number of message alternatives} = 3^{D} \qquad D = 1,2,3,\ldots\ldots \text{ (\# of Dimensions)}$$

For example, when a message has three-dimensional values, the number of message alternatives is twenty-seven. The twenty-seven message alternatives can be represented within five bits ($5 > \log_2 27$). An encoder component in the DEVS/GDDM layer maps a message alternative to a unique bit pattern. In the receiver federate, a decoder inverts the received data bits into the corresponding proper message alternative.

The DEVS/GDDM layer includes a variety of encoders and decoders that support the encoding and decoding of the multi-dimensional message alternatives.

5.5.3 Multiplexing Interest-based Quantization Method

As the number of sender and receiver pairs in federates increases, the number of messages communicated among federates increases quite quadratically. The DEVS/GDDM environment supports the multiplexing interest-based quantization method to reduce the messages and data exchanged among many sender and receiver pairs. The multiplexing interest-based quantization method is then extended from the predictive interest-based quantization method. To perform this multiplexing interest-based quantization method, the DEVS/GDDM layer has sender multiplexer and receiver de-multiplexer components.

The sender multiplexer gathers the messages output from the sender agents at the same event time, encodes the data values from the messages, multiplexes the encoded bits into a large message, and sends the large message to a receiver de-multiplexer in some other receiver federates. The receiver de-multiplexer then separates the multiplexed message to smaller messages (using de-multiplexer), decodes the encoded bits to the original data values, and distributes the messages (including the original data values) to the proper receiver agents. Through this multiplexing method, a large number of data bits can be saved as the number of sender and receiver pairs increases. Moreover, many HLA interactions can be reduced to only one HLA interaction. To exchange the message

between a sender and receiver pair between two different federates, one HLA interaction is needed. As the number of the sender and receiver pairs increases, the number of the HLA interactions for the increased pairs also increases. The increased number of the HLA interactions causes memory and computation overhead in HLA/RTI communication. By reducing the number of HLA interactions, the multiplexing method in DEVS/GDDM environment is more effective in a large-scale distributed simulation. Figure 5.10 illustrates the operation of the multiplexing interest-based quantization method.

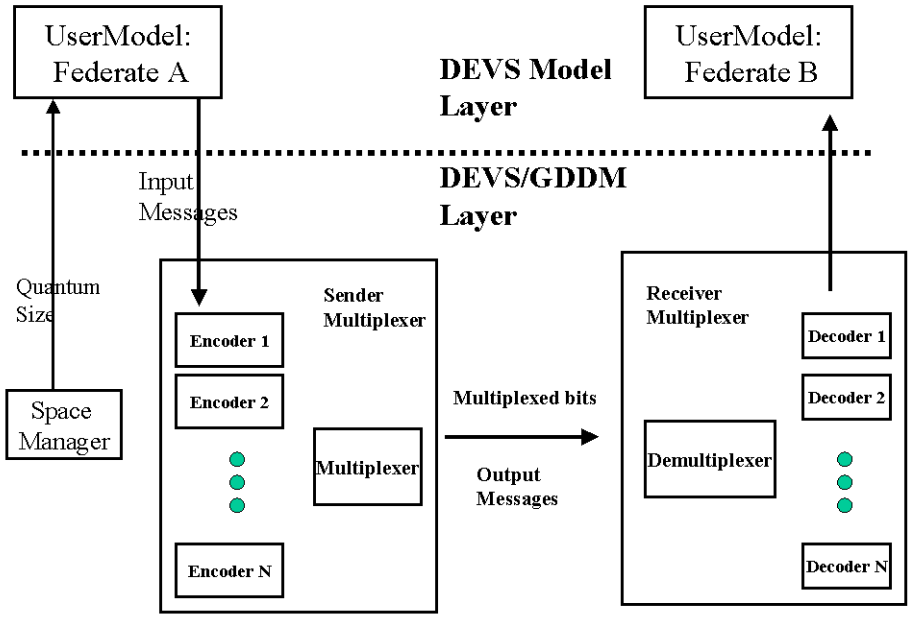


Figure 5.10 Operation of the Multiplexing Interest-based Quantization method

5.5.3.1 Fixed and Variable Multiplexing

There are two types of multiplexing interest-based quantization methods: fixed and variable. In fixed multiplexing, the multiplexed message size is constant while in variable multiplexing the size varies with the number of active senders.

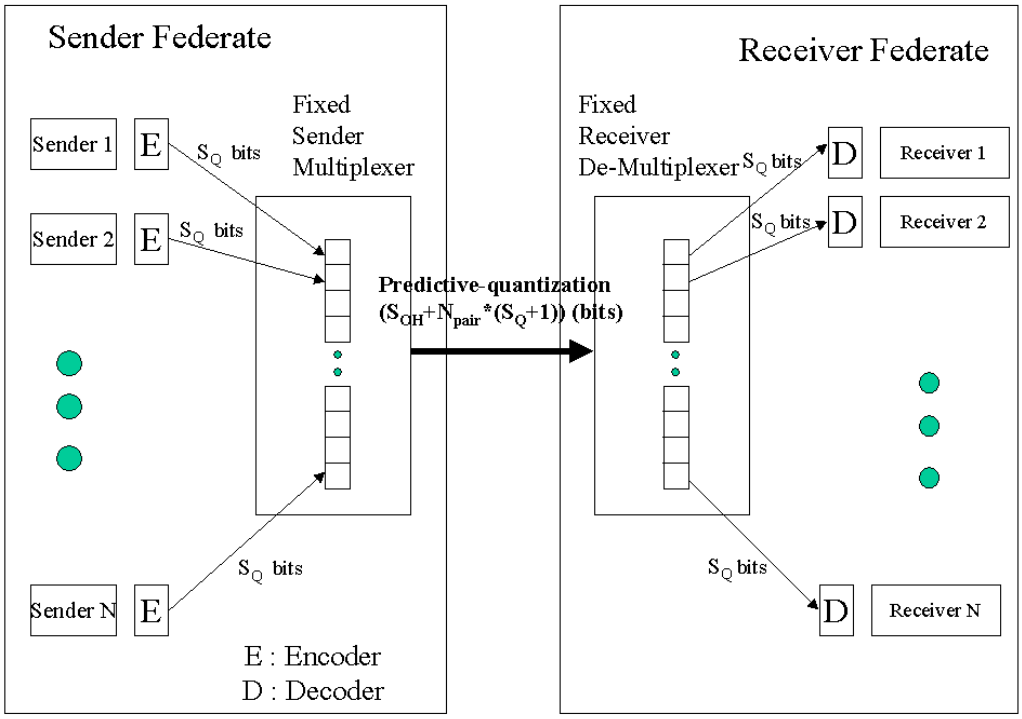


Figure 5.11 Implementation of the fixed multiplexing using the predictive quantization

(S_{OH} : the number of overhead bits for a packet; S_Q: the quantized and encoded data bit size; N_{pair} : the number of pair components; a: the ratio of active components**)**

Figure 5.11 illustrates the implementation of the fixed multiplexing using the predictive quantization. The fixed multiplexer collects the encoded bits and the active bits from each encoder. The encoded bits are the bits required to represent the message dimension alternatives. Let S_Q be the number of the encoded bits. Then by:

$$S_Q = \lceil D * \log_2 3 \rceil \qquad D = 1,2,3,\ldots\ldots \text{ (\# of Dimensions).}$$

$$= \lceil 1.7 * D \rceil$$

For example, if a message has three-dimensional values in the predictive quantization, five bits ($\log_2 3^3 < 5 = S_Q$) are required to represent the message dimension alternatives. The active bit indicates whether a sender is active or inactive. An active sender is one that has a boundary crossing at a given event time and generates an output event. A receiver de-multiplexer checks the active bit of each sender and sends the encoded bits of active senders to the respective decoders. In fixed multiplexing, for any global state transition of a sender federate at any given event time, the network loading is fixed and calculated by:

Network bandwidth requirement for fixed multiplexing

$$= S_{OH} + N_{pair} * (S_Q + 1) \text{ (bits)}$$

However, the bits assigned for inactive senders can be wasted in fixed multiplexing. The fixed receiver de-multiplexer knows which sender sends certain

encoded bits since the bit stream order in the multiplexed bits with fixed size follows a fixed ordering of the senders. Therefore, the additional bits representing which sender sends are not needed.

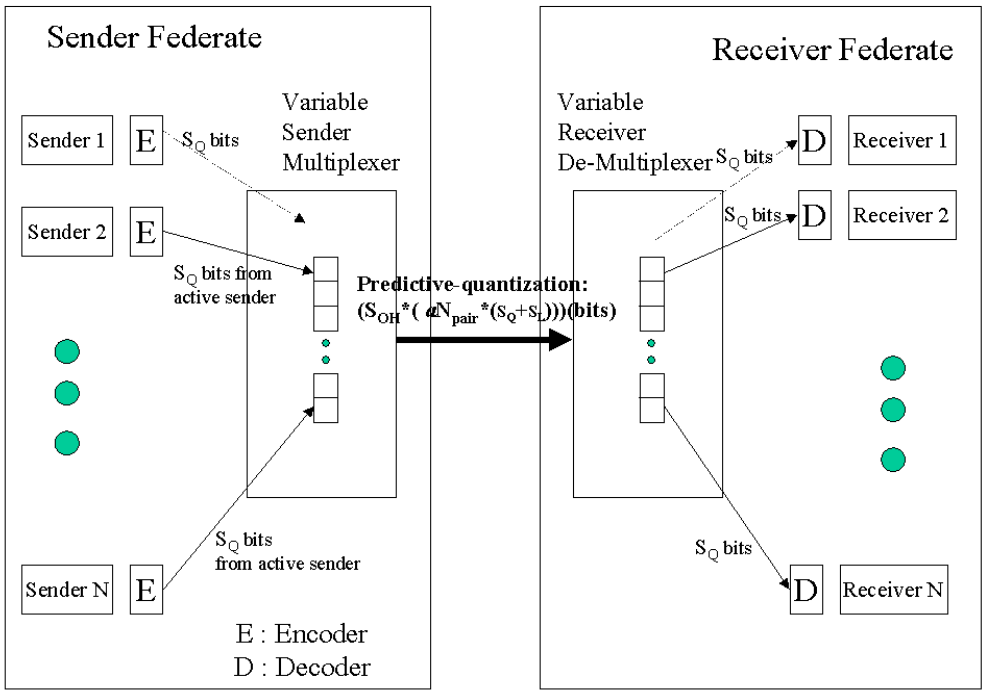


Figure 5.12 Implementation of the variable multiplexing using the predictive quantization

(S_{OH} : the number of overhead bits for a packet; S_Q: the quantized and encoded data bit size; S_L: the encoded data bit size for sender ID; N_{pair} : the number of pair components; a: the ratio of active components)

As Figure 5.12 illustrates, in variable multiplexing using predictive quantization, the variable sender multiplexer only collects the encoded bits from active senders. At a

given event time, the number of active senders varies and the number of transmitted data bits is not fixed. Different from fixed multiplexing, additional bits (S_L) are needed to represent active senders. The number of data bits for an active sender is calculated by adding the additional bits ($\log_2 N_{pair} < S_L$) and the encoded bits (S_Q). Usually, a is less than 1 since all senders are not active senders at any given event time. The network loading for any global state transition of a sender federate using variable multiplexing is:

Network bandwidth requirement using variable multiplexing

$= S_{OH} + a^* N_{pair} * (S_Q + S_L)$ (bits)

Figure 5.13 illustrates how the network bandwidth requirement in fixed and variable multiplexing depends on the ratio of active senders. For a low ratio of active senders, the variable multiplexing requires a small network bandwidth and is more effective than the fixed multiplexing. However, as the ratio of active senders increases, the network bandwidth requirement in variable multiplexing also increases; thus, when there is a high ratio of active senders, the fixed multiplexing is more effective. The crossover value (a_c) represents the ratio of active senders at the intersection point between the two lines. It separates the effectiveness of the two multiplexing schemes. At the intersection point, both fixed and variable multiplexing methods have the same network bandwidth requirement. When a is less than a_c, we can say that variable multiplexing is more effective than fixed multiplexing.

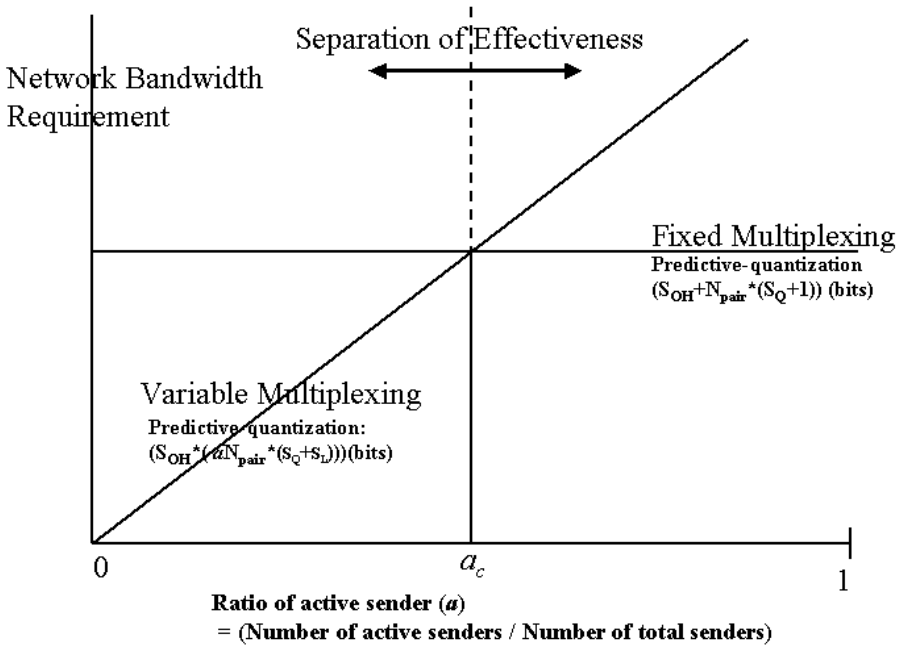


Figure 5.13 Network bandwidth requirement in fixed and variable multiplexing by varying the ratio (*a*) of active senders

Table 5.1 shows how the value of a_c is dependent on the number of message dimensions (D) and the number of component pairs.

Table 5.1 Analysis of ratio (a_c) of active senders at the intersection point

Scheme	# bits required for N_{pair}	a_c	a_c for $D = 3$, $N_{pair} = 80$
Fixed Multiplexing (Predictive quantization)	$S_{OH} + N_{pair} * (S_Q + 1)$	$(S_Q + 1) / (S_Q + S_L)$	0.78
Variable Multiplexing (Predictive quantization)	$S_{OH} + a * N_{pair} * (S_Q + S_L)$		

When the required network bandwidth needed to perform both fixed and variable multiplexing schemes is the same, a_c is calculated by:

$$a_c = (S_Q + 1) / (S_Q + S_L)$$

where

$$S_Q = \lceil 1.7 * D \rceil \qquad D = 1,2,3,\ldots\ldots \text{ (\# of Dimensions)}$$

$$S_L = \lceil \log_2 N_{pair} \rceil$$

The crossover value, a_c, approaches

$$a_c \to S_Q / S_L = \lceil 1.7 * D \rceil / \lceil \log_2 N_{pair} \rceil$$

for $1 << N_{pair}$ and $S_Q << S_L$

When D (number of dimensions) is 3 and N_{pair} is 80, a_c is 0.78. As Figure 5.13 shows, the high a_c indicates that there is a wider range in which variable multiplexing requires less network bandwidth than that of fixed multiplexing. Figure 5.14 illustrates how a_c varies with the number of message dimensions (D) and the number of component pairs (N_{pair}). Note that a_c increases as D increases, and a_c decreases as N_{pair} increases. Decreasing a_c indicates that fixed multiplexing acquires a wider effectiveness interval than variable multiplexing. However, a_c changes slowly as varying N_{pair} since it is proportional of $\log_2 N_{pair}$. As the number of pairs approaches infinity, fixed multiplexing is always preferred.

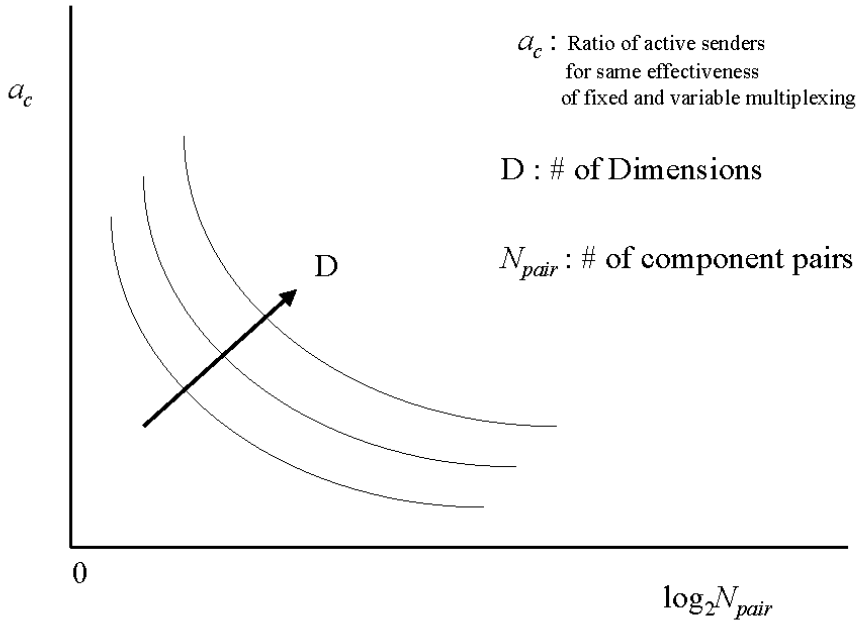


Figure 5.14 Variation of a_c in varying # of Dimensions and # of Component pairs

5.6 DEVS/GDDM Class Hierarchy

In implementing the DEVS/GDDM simulation environment on the DEVS/HLA-Interface layer, we have extensively used the object-orientation property of inheritance from DEVS object-oriented classes. This inheritance hierarchy is depicted in Figure 5.15. To realize the space manager, initializer, and message handler component classes, we implemented them as extensions of the *Atomic* class in the DEVS/HAL-Interface layer. The message handler has different types depending on the non-predictive, predictive quantization, or multiplexing methods performed in the DEVS/GDDM modeling and simulation environment. The FederateGDDM class was implemented as an extension of the *Digraph* class.

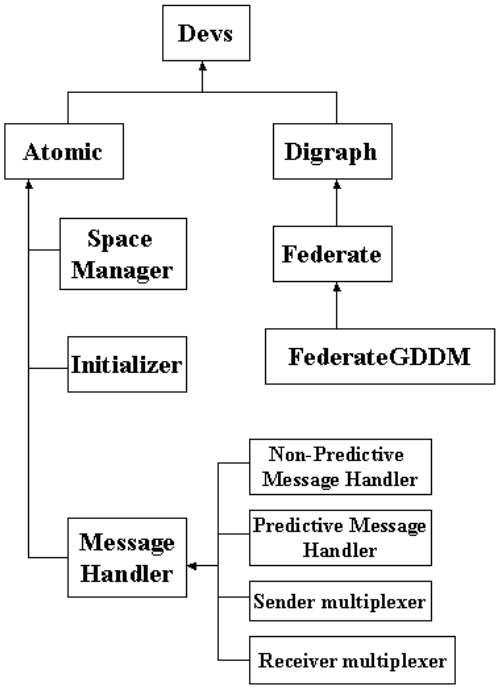


Figure 5.15 DEVS/GDDM class hierarchy

Within the DEVS/GDDM components, several inert or passive components are characterized (e.g. tuple, distance, decision maker, quantizer, message dimension, encoder, and decoder). To implement these DEVS/GDDM-specific passive components, we extended the DEVS *entity* class to create the needed classes in the same fashion as the DEVS container class library. Figure 5.16 illustrates the inheritance hierarchy for these passive DEVS/GDDM components.

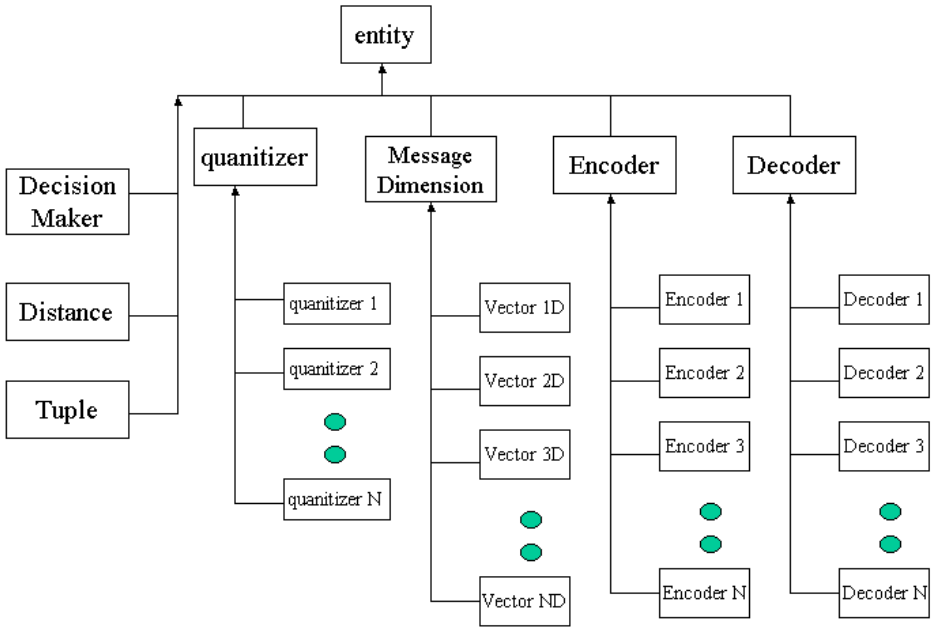


Figure 5.16 DEVS/GDDM container class hierarchy

5.7 User Interface of DEVS/GDDM

A developer defines a coupled model which includes all DEVS models simulated in a federate, and constructs a FederateGDDM component (the top DEVS component in a federate) containing the user model. Also, the developer defines a top DEVS model as usual. However, this top DEVS model is not directly used in simulation since it provides only the couplings among the DEVS models distributed in separate federates to the DEVS/GDDM environment. Using the coupling information from the top DEVS model, the DEVS/GDDM environment connects the federates together via the HLA interaction

communication. In subsequent sections, actual examples of the user interface in the DEVS/GDDM environment are introduced with two models: Projectile/Earth, Projectile/Missile.

5.7.1 Projectile/Earth model

A projectile model sends its position updates to an earth model. The earth model then uses the projectile position to calculate three parameters: gravity, atmosphere velocity, and atmosphere density. The projectile model needs the three parameters to calculate its next position, so that the three parameters are sent from the earth model to the projectile model. Figure 5.17 illustrates the passing of these attributes (position, gravity, atmosphere velocity, and atmosphere density) between the earth and projectile models, and the implementation codes of the top model of projectile/earth model in DEVS/GDDM environment. The FederateGDDM component of the projectile federate includes its own user model, which is the projectile DEVS component. The FederateGDDM component of the earth federate includes the earth DEVS component as its user model.

Sample code for a Top Model

```
Digraph Projectile = new Projectile("Projectile");
Digraph Earth = new Earth("Earth");
add(Projectile);
add(Earth);
add_coupling(Projectile,"position", Earth,"position");
add_coupling(Earth,"Gravity", Projectile,"Gravity");
add_coupling(Earth,"AtmoTangVel ", Projectile,"AtmoTangVel ");
add_coupling(Earth,"AtmoDensity", Projectile,"AtmoDensity ");
```

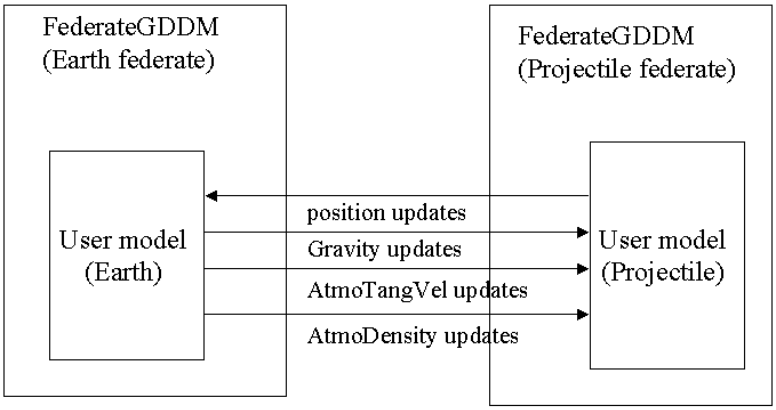


Figure 5.17 Implementation of the Top model of Projectile/Earth model in the DEVS/GDDM environment

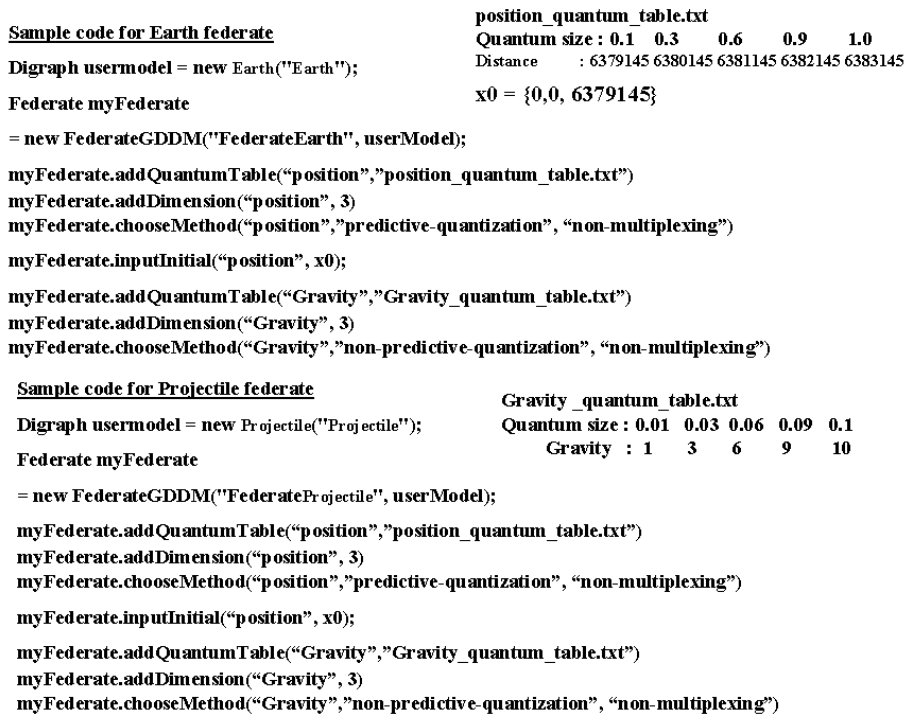


Figure 5.18 Implementation of the Projectile and Earth Federates in the DEVS/GDDM environment

In order to quantize the attributes passed between the projectile and the earth federates shown in Figure 5.18, a user chooses one of the methods supported by the DEVS/GDDM environment. The method for quantization includes non-predictive and predictive quantizations, and the method for multiplexing includes non-multiplexing and multiplexing. Therefore, the user can take one of the four combinations provided from the quantization and multiplexing methods. In Figure 5.18, considering the position attribute passed between the projectile and the earth federates, the user chooses the predictive quantization and non-multiplexing method and provides the information to the

environment to perform the method. The information includes a position quantum table, a position dimension, and initial values of position. For the gravity attribute, the user chooses the non-predictive quantization and non-multiplexing method.

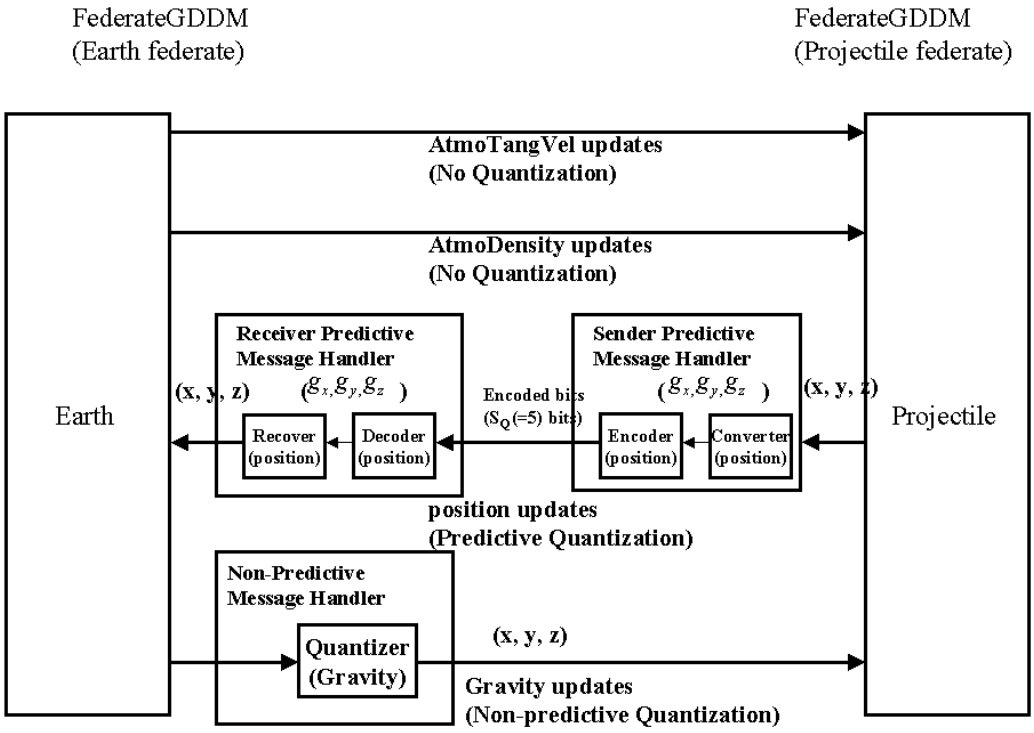


Figure 5.19 Data passing between the Projectile and Earth Federates in the DEVS/GDDM environment

A sender predictive message handler is used to perform the predictive quantization and non-multiplexing method. The sender predictive message handler has two sub-components (converter and encoder) which make the encoded bits (S_Q: five bits

for three dimensions ($5 > \log_2 3^3$)) that are passed to the receiver federate. The converter maps double precision position values (x, y, z) to integer values (g_x, g_y, g_z), where ??1,0,1?? g_x, g_y, g_z , and the encoder converts the integer values to the encoded five bits. The receiver predictive message handler has two sub-components (decoder and recover) which change the encoded five bits to the original double precision position values (x, y, z). The decoder changes the encoded five bits into the integer values (g_x, g_y, g_z), and the recover component changes the integer values (g_x, g_y, g_z) to the original double precision position values (x, y, z). In order to perform the non-predictive quantization and non-multiplexing method for the gravity attribute, the quantizer in the non-predictive message handler quantizes the gravity values and sends the double precision gravity values (x, y, z). For the rest of the attributes (atmosphere velocity and atmosphere velocity), no quantization method is provided. Figure 5.19 illustrates data passing between the projectile and earth federates in the DEVS/GDDM environment.

5.7.2 Projectile/Missile model

The projectile/missile model shows how the predictive quantization and multiplexing method performs in the DEVS/GDDM environment. In the projectile/missile model, a projectile sends its position updates to a specified missile (not another missiles), so that, in order to reduce the data bit for passing the attribute dependant to projectile/missile pairs, the predictive quantization and multiplexing method

is used. Figure 5.20 and Figure 5.21 illustrates the implementation codes of the projectile/missile model in DEVS/GDDM environment.

Sample code for a Top Model

```
Digraph Multi-Projectile = new Multi-Projectile("Multi-Projectile");
Digraph Multi- Missile = new Multi-Missile("Multi-Missile");
add(Multi-Projectile);
add(Multi-Missile);
add_coupling(Multi-Projectile,"position", Multi-Missile,"position");
```

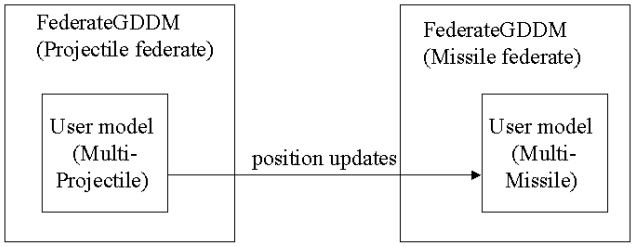


Figure 5.20 Top model codes of Projectile/Missile model in the DEVS/GDDM environment

```
Sample code for Projectile federate

Digraph usermodel = new Multi-Projectile(" Multi-Projectile");

Federate myFederate

= new FederateGDDM("FederateProjectile", userModel);

myFederate.addQuantumTable("position","position_quantum_table.txt")

myFederate.addDimension("position",3)  ;
myFederate.chooseMethod("position","predictive-quantization", "multiplexing")
myFederate.chooseNumOfAgent(80) ;


Sample code for Missile federate

Digraph usermodel = new Multi-Missile(" Multi-Missile");

Federate myFederate

= new FederateGDDM("FederateMissile", userModel);

myFederate.addQuantumTable("position","position_quantum_table.txt")
myFederate.addDimension("position",3)
myFederate.chooseMethod("position","predictive-quantization", "multiplexing")

myFederate.chooseNumOfAgent(80) ;
```

Figure 5.21 Projectile and Missile Federates' codes of Projectile/Missile model in the DEVS/GDDM environment

In the implementation of the projectile/missile model, a user models the multi-projectile model, which includes many projectile models. The user puts the multi-projectile model as a user model into the FederateGDDM component in the projectile federate. For the position attribute passed from the projectile federate to the missile federate, the user chooses the predictive quantization and multiplexing method and informs the position quantum table, the position dimension, and the number of projectiles to the DEVS/GDDM environment.

The message (ID, x, y, z) from each projectile includes the projectile ID with the three dimensional position values. The sender multiplexer has three sub-components (converter, encoder, and multiplexer) to pass a multiplexed message to the missile federate. The converter changes the double precision values (ID, x, y, z) to integer values (ID, $g_{x,}g_{y,}g_{z}$); and the encoder changes the integer position values to a properly encoded bits (S_Q: five bits for three dimensions ($5 > \log_2 3^3$)) and changes the projectile ID to a properly encoded bits (S_L). For example, if the number of projectiles is eighty, seven bits ($7 > \log_2 80$) are needed to represent the projectile ID. The multiplexer receives the encoded bits (S_Q and S_L), makes a large multiplexed message, and sends it to the missile federate. The receiver de-multiplexer has three sub-components (de-multiplexer, decoder and recover) to make the original double precision values (ID, x, y, z) from the multiplexed message. The de-multiplexer separates from the multiplexed message to each encoded bits (S_Q and S_L). The decoder changes the encoded bits (S_Q and S_L) to the integer values (ID, $g_{x,}g_{y,}g_{z}$), and the recover component changes the integer values (ID, $g_{x,}g_{y,}g_{z}$) to the original double precision values (ID, x, y, z). Figure 5.22 illustrates data passing between the projectile and missile federates in the DEVS/GDDM environment.

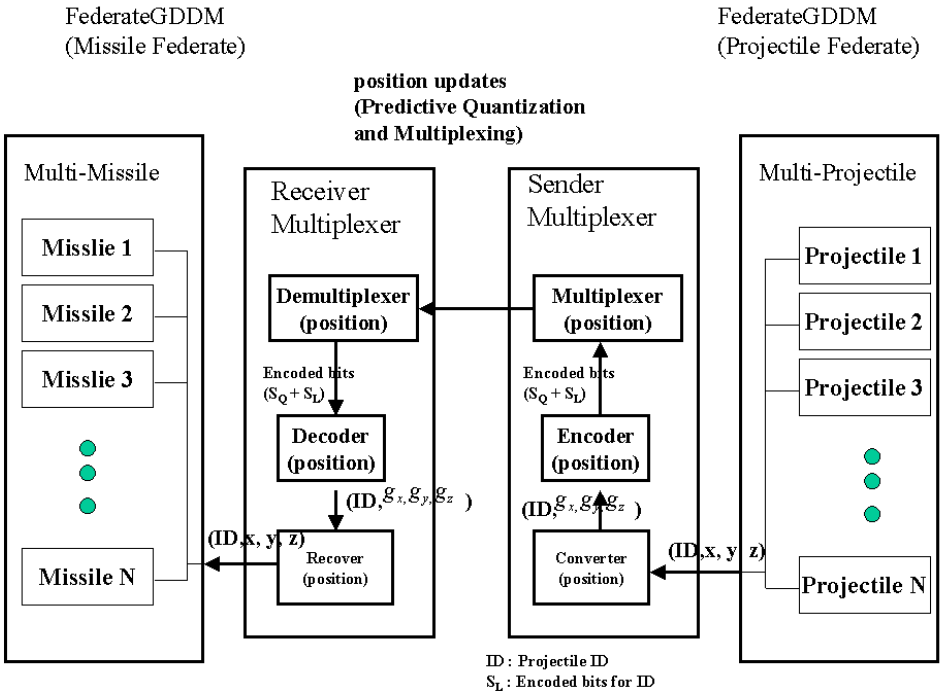


Figure 5.22 Data passing between the Projectile and Missile Federates in the DEVS/GDDM environment

6 DEVS PREDICTIVE INTEGRATOR

A theoretical and empirical study of the advantages of predictive quantization over non-predictive quantization is provided in [40, 43]. Using the predictive quantization, Zeigler [51] developed an example model (DEVS predictive integrator). The DEVS predictive integrator basically performs, as illustrated in Figure 6.1, linear extrapolation. The time to the next boundary crossing is the quantum size divided by the input (derivative). The boundary is predicted either to be one up or one down according to the sign of the derivative. When an input event is received, the state is updated using the old input before recalculating the predicted crossing, which provides an important correction for error reduction. A DEVS predictive integrator accepts DEVS input segments and produces quantized output.

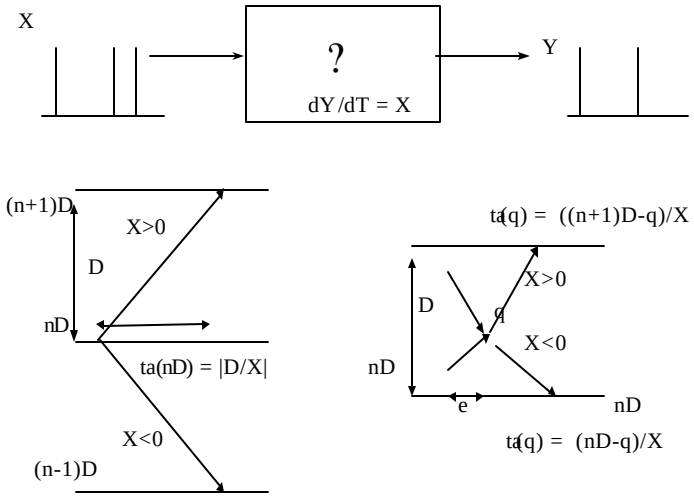


Figure 6.1 DEVS Predictive Integrator

DEVS representation of the DEVS predictive integrator is the following:

$$M = (X, Y, S, ?_{ext}, ?_{int}, ?, ta).$$

where $X = Y = R$ and $S = R ? R? I$ and

?? $?_{ext}((q,x,n),e,x') = (q+x*e,x', n)$

?? $?_{int}(q, x, n) = (nD + D*sign(x), x, n+ sign(x))$

?? $?_{con}((q,x,n), x') = (nD + D*sign(x), x', n+ sign(x))$

?? $?(q,x) = nD + D*sign(x)$

?? $ta(q,x,n) = ((n+1)D - q)/x,$ *if* $x > 0$ and $(n+1)D - q > 0$

$= (nD - q)/x,$ *if* $x < 0$ and $nD - q < 0$

$= |D/x|$ *if* $x ? ?0$ and none of the above

$= ? ????$otherwise *(i.e., x = 0)*

As Figure 6.1 illustrates, if we are on a boundary, the time advance computation merely divides *D* by the current input *x* (the derivative or slope). If we reach the upper boundary *(n+1)D* or lower boundary *(n –1)D*, we output and update the state accordingly. As long as the input remains the same, the time to cross the successive boundaries (*(n+1)D* or *(n-1)D*) will be the same. When a new input is received, we update the state

using the old input and the elapsed time. From this new state (q), the new time to reach either the upper or lower boundary is computed.

6.1 DEVS Representation with Hysteresis of DEVS Predictive Integrator

The necessity of hysteresis in a Quantized-State System (QSS) is presented by Kofman [56]. Without hysteresis of the quantized variable, a QSS can perform an infinite number of state transitions at the same time or within a finite time interval. At first glance, the DEVS predictive integrator does not include hysteresis and might suffer from the problem of an infinite number of transitions in a finite interval, called illegitimacy [27]. Actually, the DEVS predictive integrator developed by Zeigler [43, 51] includes the hysteresis properties discussed by Kofman. In this section, we express the hysteresis within the DEVS formalism. The operation of the DEVS predictive integrator with DEVS representation including the hysteresis is the following:

$$M = (X, Y, S, ?_{ext}, ?_{int}, ?_{con}, ??, ta).$$

where $X = Y = R$ and $S = R ? R ? I$, a typical state $(q, x, n) ? S$ represents the integrator state, q, stored input x, and multiple of quantum, n.

$$?? \quad ?_{ext}((q,x,n),e,x') = (q+x*e,x', n)$$

$$?? \quad ?_{int}(q, x, n) = (nD + D*sign(x), x, n+ sign(x))$$

?? $?_{con}((q,x,n), x') = (nD + D*sign(x), x', n+ sign(x))$

?? $?(q,x) = nD + D*sign(x)$

?? $ta(q, nD,x,n) = ((n+1)D - q)/x$, if $x > 0$ and $q > nD$

$= (nD - q)/x$, if $x > 0$ and $q < nD$

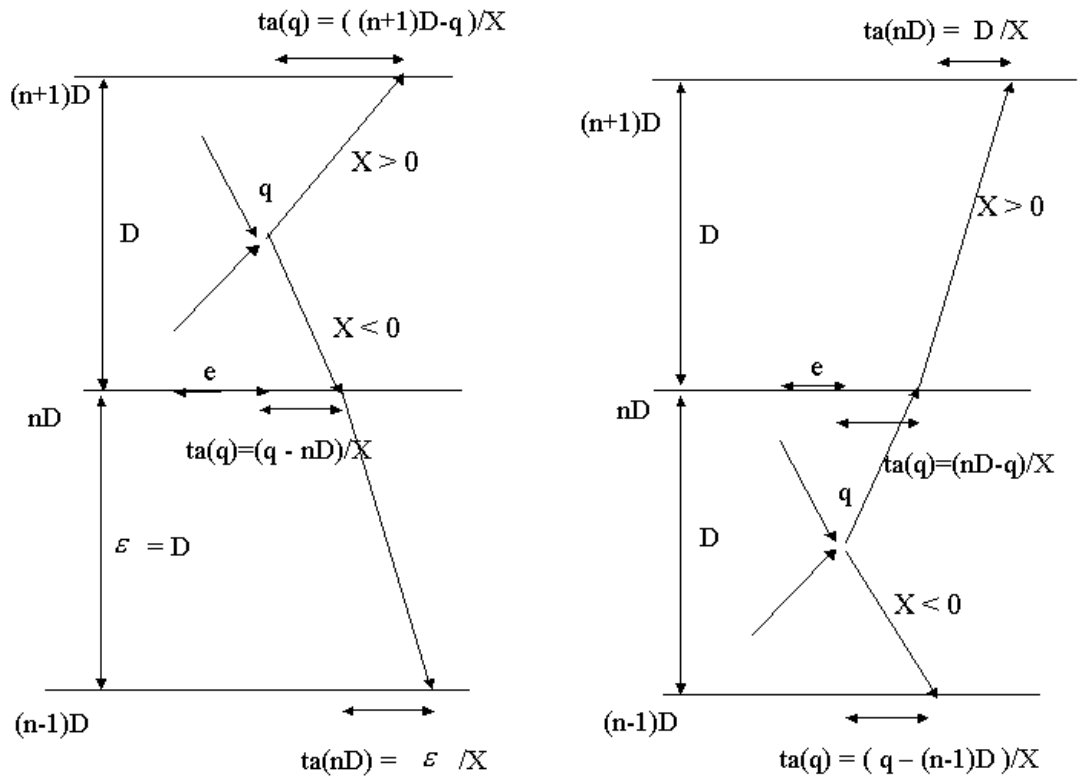
$= D/x$, if $x > 0$ and $q = nD$

$= (q - nD)/x$, if $x < 0$ and $q > nD$

$= (q – (n-1)D)/x$, if $x < 0$ and $q < nD$

$= |?/x|$, if $x < 0$ and $q = nD$

= ? ?????otherwise *(i.e., x = 0)*



(a) when increasing state direction (b) when decreasing state direction

Figure 6.2 Operation of the DEVS Predictive Integrator with Hysteresis

In Figure 6.2, nD indicates the current state boundary since n is the index of a current boundary and D is a certain quantum size. Here, (n-1)D and (n+1)D indicate next state boundaries reached from the current state (nD); and ? indicates the width of the hysteresis window and is the same as the quantum size (D).

In implementation, in order to indicate the direction of the state transition, we used lowerBound and nextLowerBound variables. The lowerBound (n) is the boundary index of the current state (nD) and the nextLowerBound (n-1 or n+1) is the boundary index of the next state ((n-1)D or (n+1)D). The values of lowerBound and

nextLowerBound variables indicate the direction of state transition. For example, when lowerBound is 1 and nextLowerBound is 2, the direction of state transition increases. When lowerBound is 2 and nextLowerBound is 1, the direction of state transition decreases.

Hysteresis is used when the DEVS predictive integrator receives decreasing derivative as its input (e.g. input value is less than zero (X<0)) in increasing state direction. As Figure 6.2(a) illustrates, in order to process an input of decreasing derivative with hysteresis when the state transition direction is increasing, there are two operations for changing the direction of the state transition: The first is to make nextLowerBound the same as lowerBound; and the second is to assign nextLowerBound for indicating the new direction (decreasing state direction from decreasing derivative) and to calculate the next state value (by subtracting the width of the hysteresis window (?) from the current state boundary value (nD)). These two operations are performed at the same time and the changed lowerBound and nextLowerBound indicate the new direction of state transition. The output value of the DEVS predictive integrator is related to the new direction of state transition and is calculated by multiplying the changed nextLowerBound and the quantum (D). Figure 6.2(b) illustrates the operation of the DEVS predictive integrator when it receives increasing or decreasing derivative as its input in decreasing state direction, and the DEVS predictive integrator does not hysteresis.

6.2 Kofman's DEVS Predictive Integrator with Hysteresis

Using hysteresis, Kofman modified the DEVS predictive integrator developed by Zeigler [43]. In this dissertation, we modeled the Kofman's DEVS integrator [56] using the DEVS formalism. The operation of the Kofman's DEVS integrator with DEVS representation including hysteresis is the following:

The DEVS representation is the following:

$$M = (X, Y, S, ?_{ext}, ?_{int}, ??, ta).$$

where $X = Y = R$ and $S = R ? R? I$ and

?? $?_{ext}((q,x,n),e,x') = (q+x*e, x', n)$

?? $?_{int}(q, x, n) = (q', x, n+ sign(x))$

where

$q' = (n+1)D$, if $x>0$

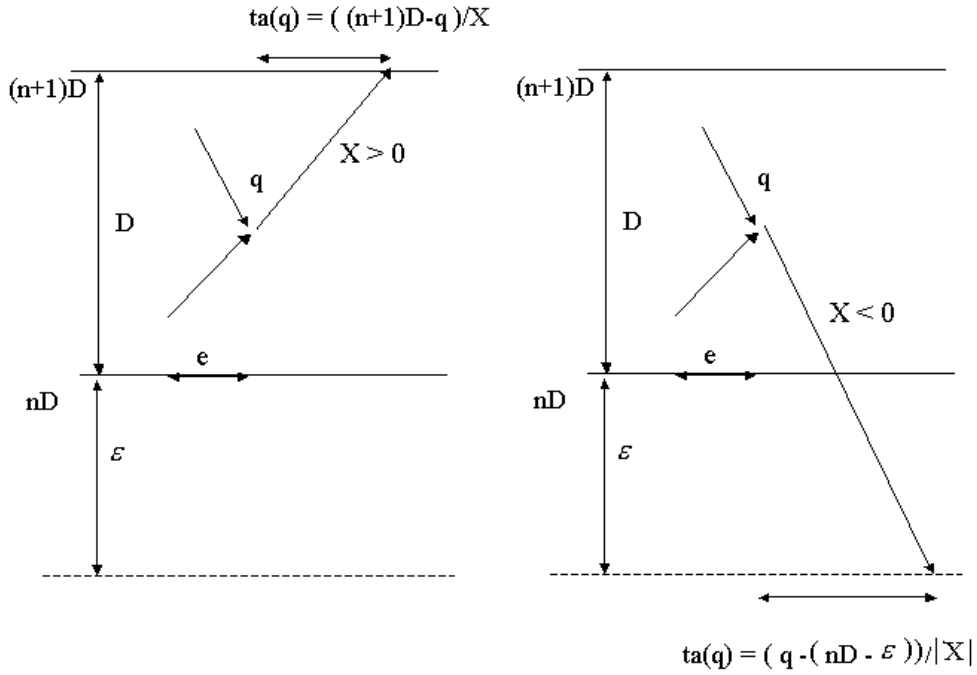
$= nD-?$, if $x<0$

?? $? (q,x) = nD + D*sign(x)$

?? $ta(q,nD,x,n) = ((n+1)D - q)/x$, if $x > 0$

$= (q – (nD - ?))/|x|$, if $x < 0$

$= ?$??????????????????????????otherwise *(i.e., x = 0)*



(a) when increasing derivative (b) when decreasing derivative

Figure 6.3 Operation of Kofman's DEVS Integrator with Hysteresis

Figure 6.3 illustrates how the Kofman's DEVS integrator uses the hysteresis. Like in the original DEVS predictive integrator, the hysteresis is used when the Kofman's DEVS integrator has decreasing derivative as its input (e.g. input value is less than zero (X<0)). In the implementation of the Kofman's DEVS integrator, we use the Actual_index (n) which indicates the boundary index of current state (nD). Without indicating the direction of state transition, the Actual_index only indicates the current state boundary.

As Figure 6.3(b) illustrates, the Kofman's DEVS integrator receives an input of decreasing derivative and uses the hysteresis to calculate the next state value by subtracting the width of the hysteresis window (?) from the current state value (nD). Unlike the original DEVS predictive integrator, the Kofman's DEVS integrator does not consider the direction of state transition; thus it needs only one operation for calculating the next state value and the time advance value for next event with the hysteresis. The next state value and the next time advance value of Kofman's DEVS integrator are only related to the current input value. If the current input value is greater than zero, the Actual_index is increased by one. If the current input value is less than zero, the Actual_index is decreased by one. The output value is calculated by multiplying the increased or decreased Actual_index and the quantum (D). Figure 6.3(a) illustrates the operation of Kofman's DEVS integrator when it receives increasing derivative as its input, and Kofman's DEVS integrator does not hysteresis.

6.3 Experimentation and Results

In order to illustrate the qualities of the DEVS predictive integrator with hysteresis, we chose a second order stiff system as a simulation example. The stiff system includes at least one integrator that frequently changes the direction of the state transition. Since hysteresis is only used when the direction of the state transition of a quantized variable changes, the stiff system is the proper example needed to show the operation of

the DEVS predictive integrator with hysteresis. The second order stiff system is represented:

$$\dot{x}_1 = \frac{1}{L} x_2$$

$$\dot{x}_2 = U - \frac{1}{C} x_1 - \frac{R}{L} x_2$$

$$y = \frac{1}{L} x_2 \qquad (6.1)$$

where L is 0.01, U is 100, C is 0.01, and R is 100.

The analytical solution of the second order stiff system is below:

$$y(t) = \frac{10000}{9999}(e^{-t} - e^{-10000t}) \qquad (6.2)$$

The error from this simulation was evaluated by comparing the simulation results to the analytical solution of (6.2).

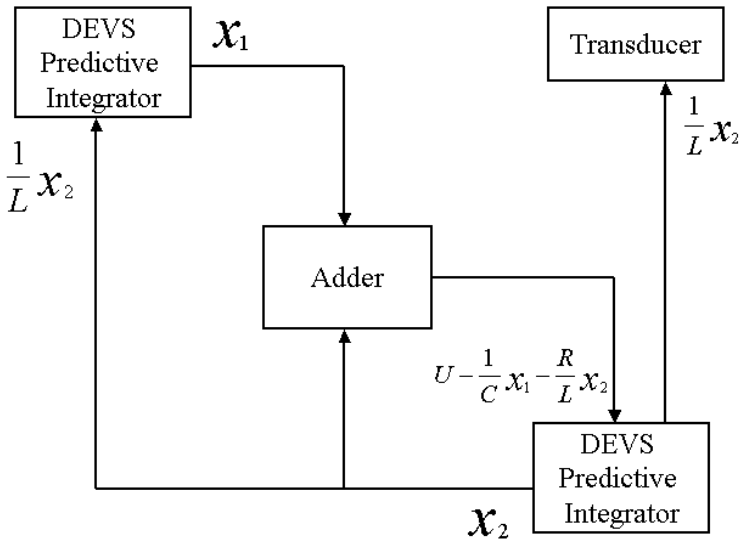


Figure 6.4 Component Diagram of Second Order Stiff System

To simulate the second order stiff system, we modeled the DEVS components: DEVS predictive integrator, adder, and transducer. The second order stiff system includes two DEVS predictive integrators, which generate x_1 and x_2. The DEVS predictive integrator is modeled by the DEVS predictive integrator representation in section 6.1. Also, we modeled the second order stiff system, which uses the Kofman's DEVS integrators modeled from the DEVS representations of section 6.2. The adder component collects the output values from the two integrators (which generates x_1 and x_2) and makes the derivative for the integrator that outputs x_2. The transducer component gathers the output values of the second order stiff system and shows occurred error. Figure 6.5 shows

the simulation result of the second order stiff system using the DEVS predictive integrators.

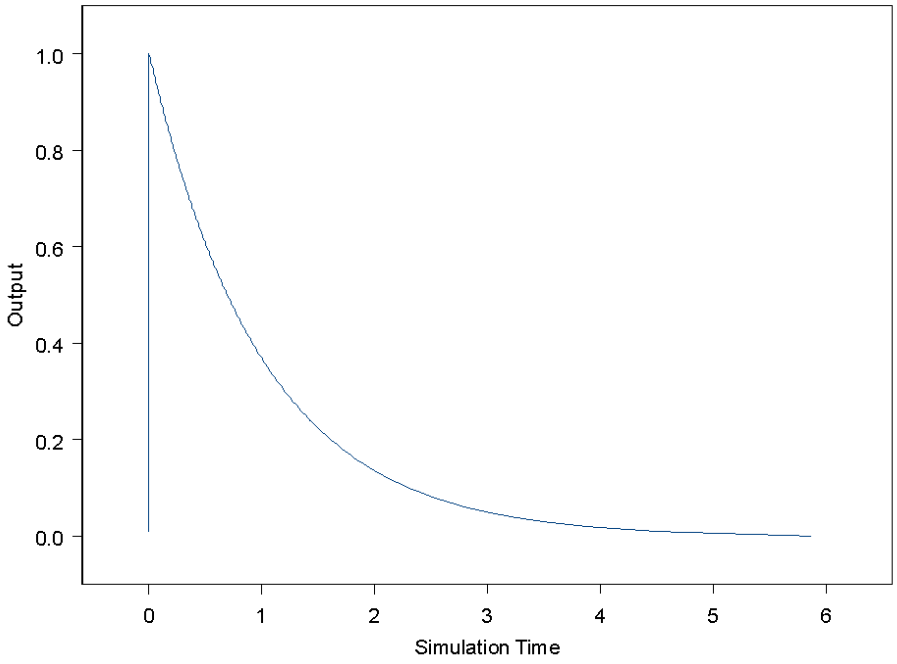


Figure 6.5 Output trajectory of the second order stiff system using the DEVS Predictive Integrators

In order to validate the second order stiff system using the DEVS Predictive Integrators, we investigated the error trajectory between the value from the stiff system simulation and the exact value of *y(t)* in (6.2). Figure 6.6 shows the error trajectory of the second order stiff system using DEVS Predictive Integrators. The greatest error was less than 10^{-2}, or 1.0 (%) of maximum value. After the simulation time is 3.498976, the state

value of the integrator that outputs χ_2 is below its quantum size (10^{-4}); therefore, the error is bounded within 10^{-2} , which is calculated by:

$$y \approx \frac{1}{L}\chi_2 \approx 10^{-2}$$

where L is 0.01.

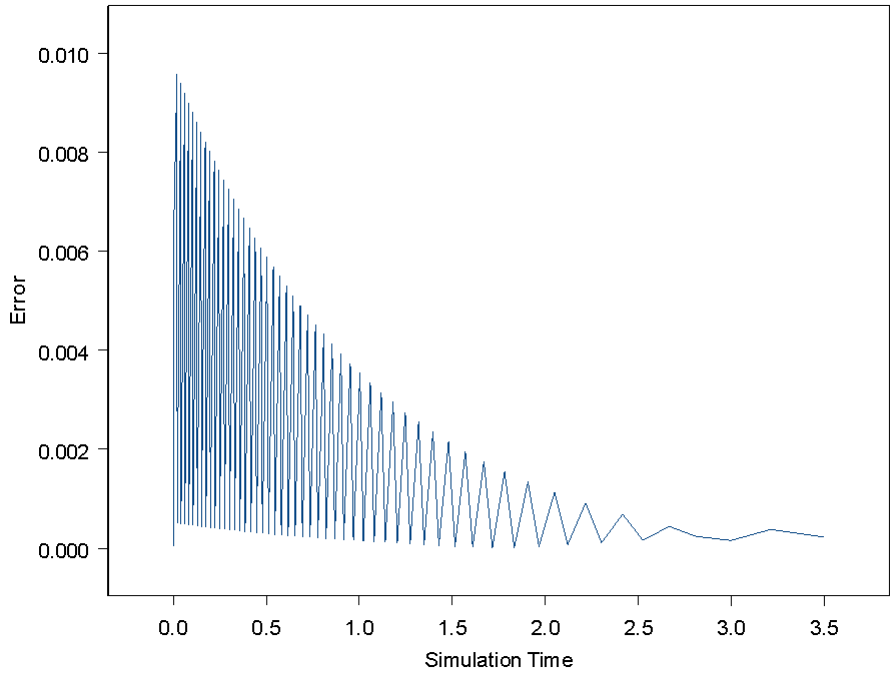


Figure 6.6 Error trajectory of the second order stiff system using the DEVS Predictive Integrators (Quantum sizes - X1: 10^{-2} , X2: 10^{-4})

In order to compare the quality between the DEVS predictive integrator and the Koffman's integrator, we checked the error of the simulation time when the value of *y(t)* in (6.2) was equal to 1.000 (exact value). Figure 6.7 illustrates the error check point to check the error of the simulation time, which is computed as:

$$\text{Error} = t_{app} - 9.2*10^{-4}$$

where t_{app} : approximated time to exact time ($9.2*10^{-4}$) for varying quantum sizes.

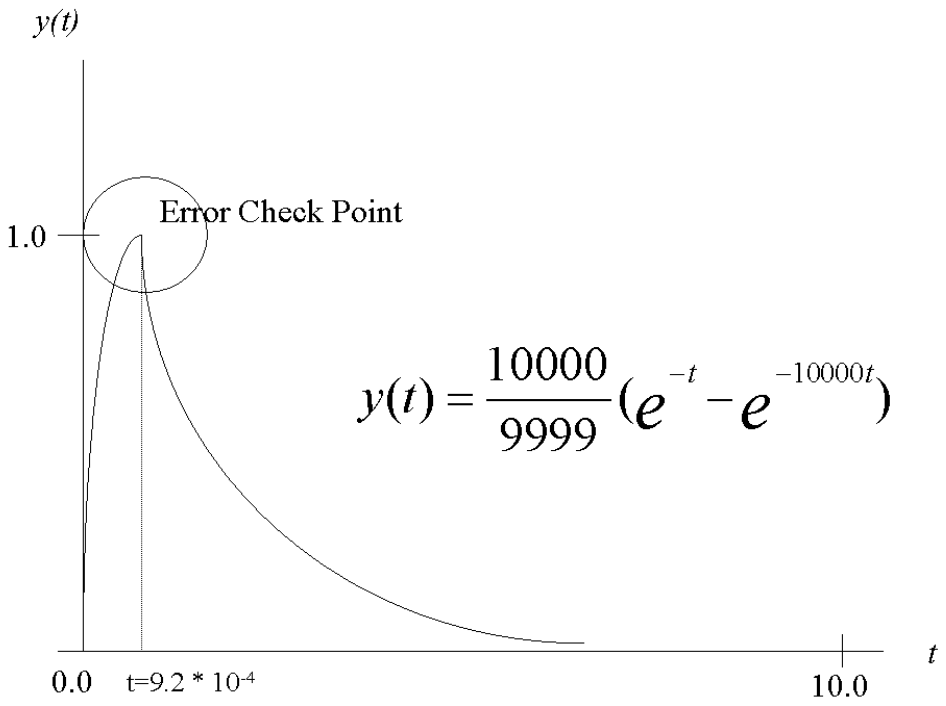


Figure 6.7 Error Check Point in Second Order Stiff System

In simulation, the hysteresis value (?) is equal to the quantum size (D). Figure 6.8 illustrates the errors from the DEVS predictive integrator and from the Kofman's DEVS integrator for varying quantum sizes. As the quantum sizes of the integrators of the second order stiff system increase, the incurred error also increases. Both the original DEVS predictive integrator and the Kofman's DEVS integrator show the same accuracy.

Figure 6.9 illustrates the number of internal transitions from the DEVS predictive integrator and from the Kofman's DEVS integrator in varying quantum sizes. When small quantum sizes are used, internal transitions from the Kofman's DEVS integrator are less than those from the DEVS predictive integrator.

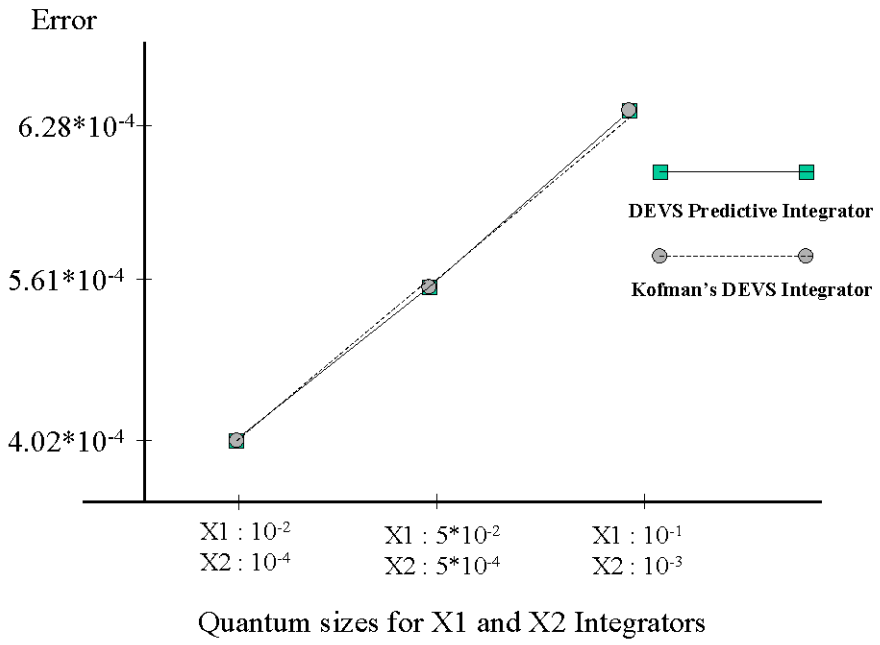


Figure 6.8 Error from the original DEVS predictive integrator and the Kofman's DEVS integrator in varying quantum sizes

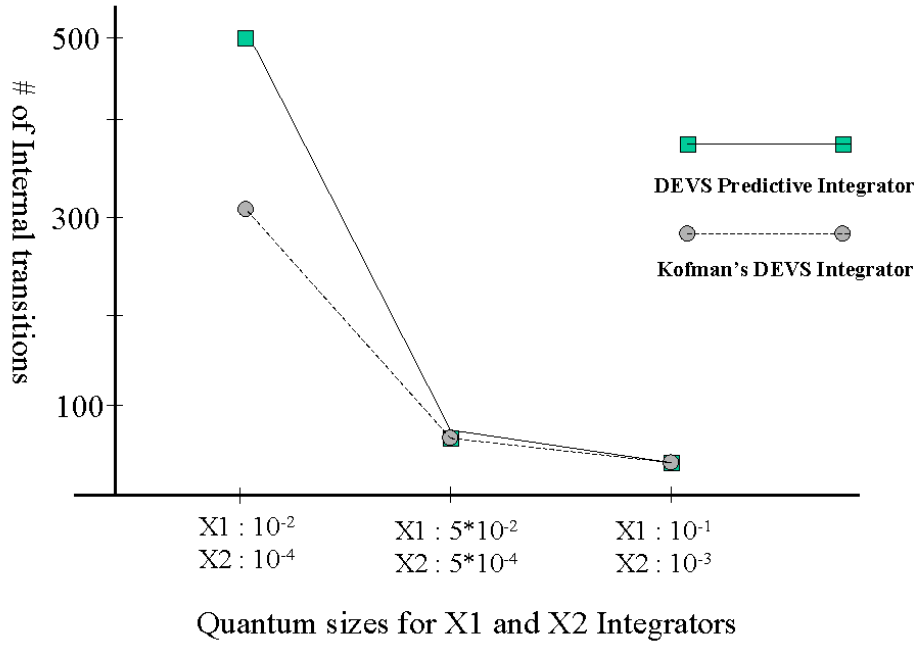


Figure 6.9 Internal transitions from the original DEVS predictive integrator and Kofman's DEVS integrator in varying quantum sizes

6.4 Discussion

Considering hysteresis, we compared the qualities of the original DEVS predictive integrator and Kofman's DEVS integrator. In simulation of the second order stiff system example, the errors (for varying the quantum sizes) of the two integrators (DEVS predictive integrator and Kofman's DEVS integrator) were not much different. Both integrators perform with the same accuracy, however the number of internal transitions of the original DEVS predictive integrator is greater than those of the

Kofman's DEVS integrator. When the direction of the state transition is changed, the original DEVS predictive integrator needs two internal transitions to perform hysteresis; meanwhile, only one internal transition is needed for the Kofman's DEVS integrator. Since the second order stiff system frequently changes the direction of the state transition, the second order stiff system simulation shows the difference in the number of internal transitions of the two integrators (the DEVS predictive integrator and the Kofman's DEVS integrator). However, in many real-world applications, the change of the direction of the state transition does not occur frequently.

7 PURSUER-EVADER MODEL

In order to evaluate the performance of the space-based quantization scheme in a distributed simulation environment, we introduce the pursuer-evader model and a federation executing on the DEVS/HLA distributed simulation environment. The federation contains two types of agents, pursuers and evaders, which move and interact with each other in a bounded region of two-dimensional space. There are two types of federates, pursuer federates and evader federates. Each pursuer federate contains an arbitrary number of pursuers while each evader federate contains an arbitrary number of evaders. Pursuers and evaders bounce in elastic fashion off the walls of the region in which they are confined. The pursuers chase evaders that come within close proximity and shoot at those within a smaller range. The evaders run away from pursuers they "notice" at some distance and freeze when detecting any within a closer range. Figure 7.1 illustrates the operation of the pursuer-evader model.

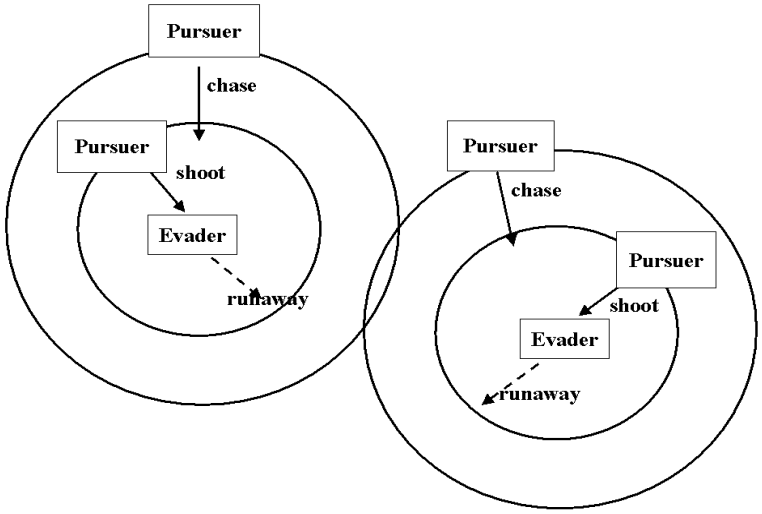


Figure 7.1 Pursuer-Evader Model

7.1 Distance-Dependent Sensitivity of Vision

Perception abilities of pursuers and evaders are modeled with a simple approach to distance-dependent sensitivity of vision. The ease with which one agent can detect another depends upon the latter's projection on the former's hypothetical retina. The projection is defined as the size of the agent divided by the distance between the two agents. The projection must be larger than a threshold value to be perceived. Since, in our model, a pursuer is bigger than an evader, with the same threshold on the projection, an evader can see a pursuer better than a pursuer can see an evader. Critical distance, D, is defined as the size divided by a threshold of projection. As Figure 7.2 illustrates, there are two critical distances, D_{see} and D_{notice}, corresponding to two thresholds for seeing and

noticing. An evader can detect the pursuer within D_{notice} of evader and can see the pursuer within D_{see} of evader (and vice versa).

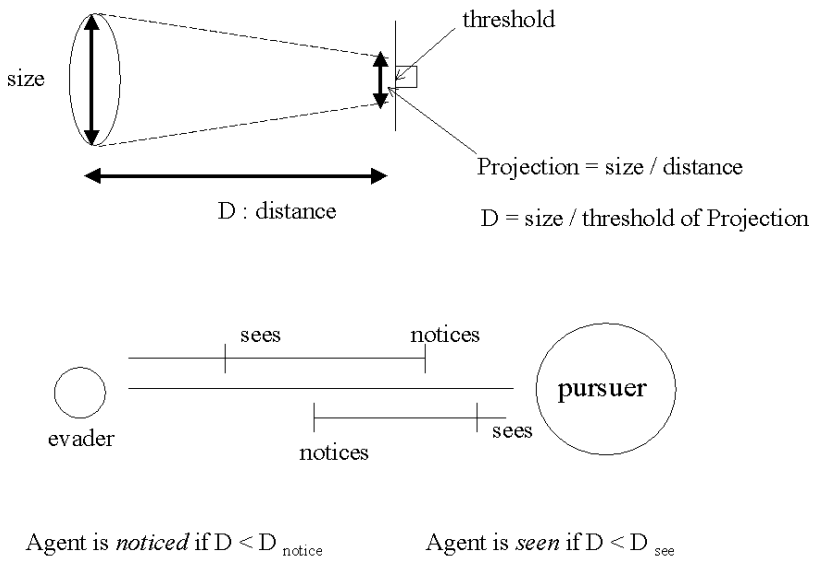


Figure 7.2 Modeling Distance-dependent Sensitivity of Vision in the Pursuer-Evader Model

At any moment, with this distance-dependent sensitivity, an evader may exist in one of four states, “move”, “run away”, “freeze”, or “dead”. These states change in response to the evader’s perception of the pursuers which is a function of the distance between each pursuer and the evader. Figure 7.3 illustrates a state transition diagram for the evader.

When a pursuer comes within the critical range, D_{notice}, of an evader, the evader can detect the pursuer. At this point, the evader switches to the "run away" state and runs away from the pursuer. However, if a pursuer comes within the critical D_{see} range, the evader can "see" the pursuer and changes to the "freeze" state. In the "freeze" state, the evader does not move hence does not output any position update messages. The evader changes from the "freeze" state to the "move" state when all pursuers are out of range determined by the critical D_{see} distance. The "freeze" state was introduced to provide interesting interactions. It can be used to keep the pursuers from quickly eliminating the evaders, thereby enabling long simulation runs.

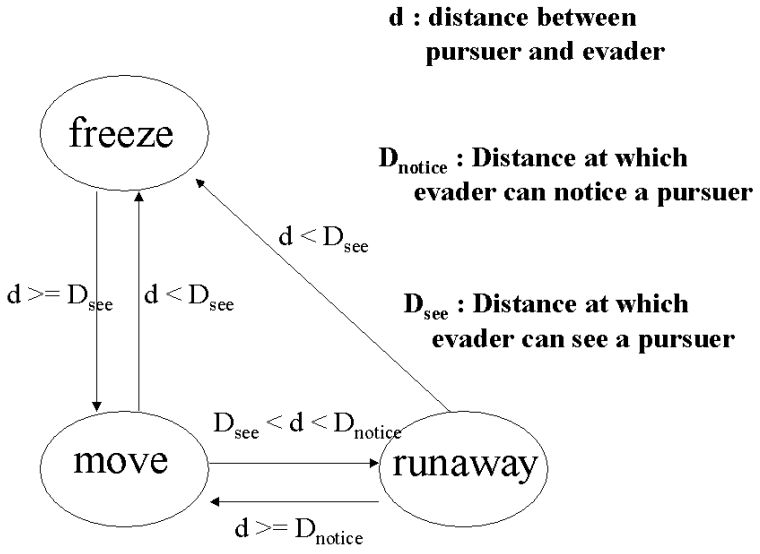


Figure 7.3 State Transition Diagram for Evader

7.2 Space-based Quantization with Distance-Dependent Sensitivity of Vision

The space-based quantization scheme is applied to the operation of the pursuer/evader model with distance-dependent sensitivity of vision. This scheme uses two critical distances, D_{see} and D_{notice}, which are determined by distance-dependent sensitivity of vision. Figure 7.4 illustrates how a quantum size is assigned according to the distance between pursuer and evader. In Figure 7.4(a), three quantum sizes are used with the two values of distance, D_{see} and D_{notice}. Two quantum sizes are used in Figure 7.4(b) and Figure 7.4(c). Note that filtering of messages is greater with the assignment of Figure 7.4(c) than that of Figure 7.4(b).

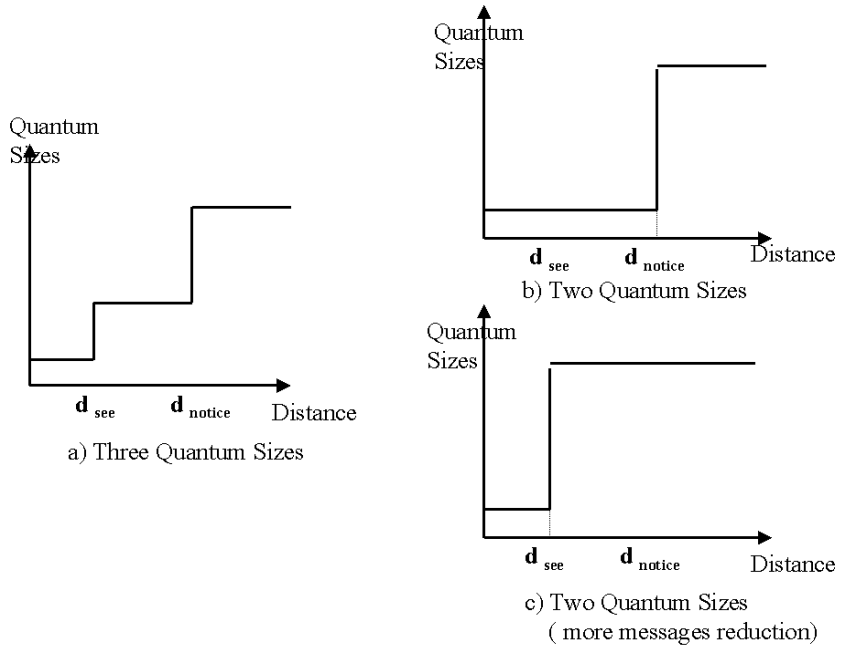
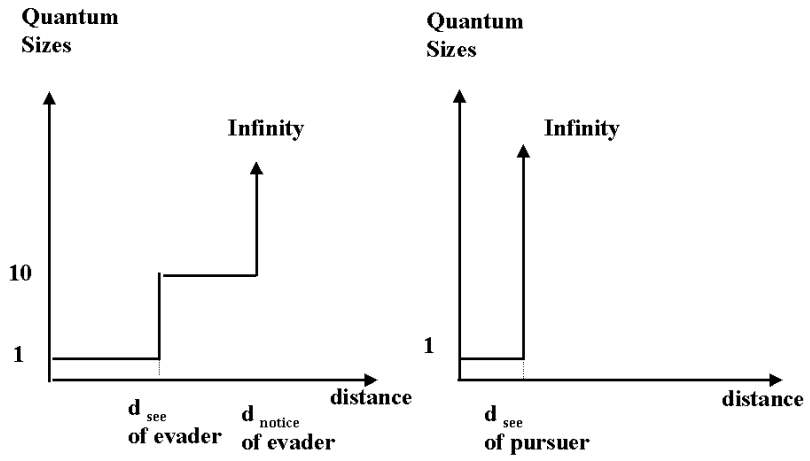


Figure 7.4 Assigning quantum sizes based on the distance



a) Pursuer >> Evader b) Evader >> Pursuer

Figure 7.5 Assigning quantum sizes based on the message direction and the distance

As Figure 7.2 shows, we have to consider four critical distances: the respective values, D_{see} and D_{notice} of pursuers and evaders. Figure 7.5 illustrates the assignment of quantum sizes with these four distances. Figure 7.5(a) illustrates the quantum size assignment for messages being transmitted from pursuer to evader while Figure 7.5(b) considers messages from evader to pursuer. In Figure 7.5(a), three quantum sizes are assigned, using a quantum size of 10 for distances between D_{see} and D_{notice} of evader. D_{see} and D_{notice} of evader are bigger than D_{see} and D_{notice} of pursuer respectively, since a pursuer is bigger than an evader. Thus perception is not symmetric: an evader can perceive a pursuer better than that pursuer can perceive it. In Figure 7.5(b) we assigned only two quantum sizes for greater message reduction. In this example, the pursuer cannot see the evader outside the range of 10 units. Therefore, quantum size 10 is

replaced by quantum size infinity. Of course, more quantum sizes with the space-based quantization scheme can be employed. The number of quantum sizes assigned is very dependent to each application.

7.3 Filtering operation

In this section we provide more detail on the filtering operations performed by the space manager with the pursuer/evader model in the DEVS/HLA distributed simulation environment. In the DEVS/HLA distributed simulation environment, there are two types of filtering operations. The first type is the filtering at sender federate. When a pursuer has a message to be transferred, if all evaders are too far from the pursuer, the space manager decides that the message from the pursuer does not need to be transferred and blocks the message. The space manger operates in the same manner when the message originates from an evader. This type of filtering blocks HLA inter-federate messages from entering a network. In a distributed simulation, network delay is a critical factor of system performance. By blocking a message from entering a network, filtering at sender federate can prevent the communication overhead that results from network delay. Therefore, with filtering at sender federate, we can greatly reduce system execution time as a result of network message reduction in a distributed simulation.

The second type of filtering operation is the filtering at receiver federate. When a pursuer has a message to be transferred, if some evaders are close by and some evaders are far away, the message has to be released outside the pursuer federate without filtering.

The space manager decides, according to the distance between the evaders and the pursuer, which evaders can receive the message and blocks the message to those evaders that are far from the pursuer. The filtering operation works in the same manner when the evader sends a message. This filtering operation filters DEVS messages, which are for communication within a federate. Figure 7.6 illustrates the two types of filtering operations controlled by the global or the local space managers in the DEVS/HLA distributed simulation environment.

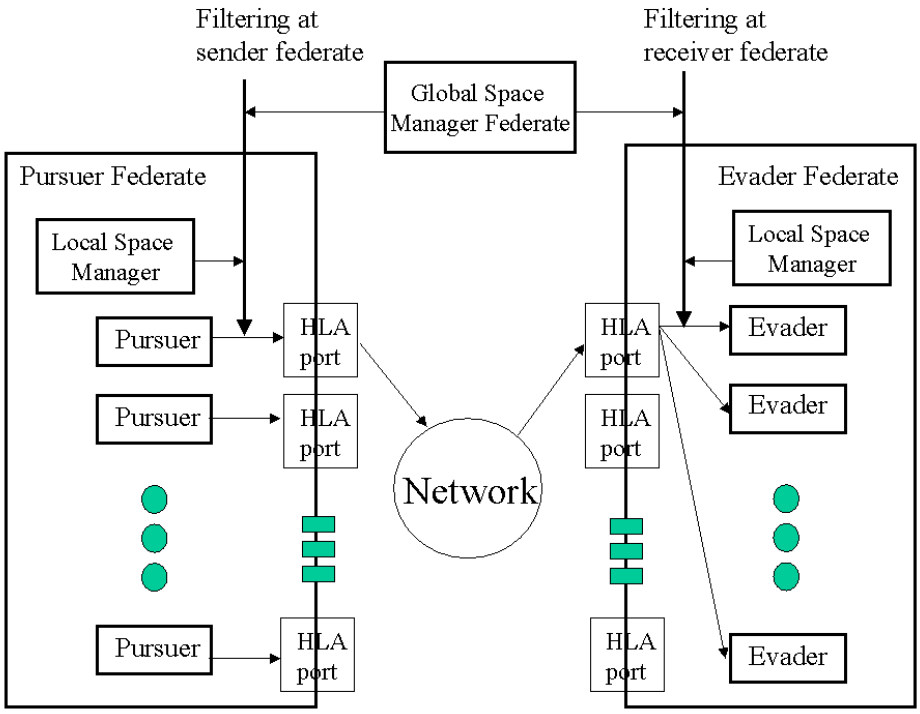


Figure 7.6 Filtering operations

In this dissertation, the direct filtering scheme supported by DEVS modeling and simulation is introduced and applied to these two types of filtering operation. In most conventional filtering schemes, each agent directly filters the message traffic. For example, with the spatial encounter prediction of the Joint MEASURE architecture [44], the space model sends the spatial relationship to each agent, and each agent performs the filtering operation. However, in the direct filtering operation, the space manager directly performs the filtering operation by changing the coupling specification supported by DEVS modeling, but does not inform the spatial relationship to each agent. With this coupling specification, a message can be transferred to any model. Each model can change the message transfer path among the models by adding or removing the coupling specification. With the space manager, the direct filtering is applied to these two types of filtering operations in the DEVS/HLA distributed simulation. With filtering at sender federate, the space manager performs direct filtering by changing the coupling specification between the sender agent's output and the output of the sender federate. With filtering at the receiver federate, the space manager performs direct filtering by changing the coupling specification between the receiver agent's input and the input of the receiver federate.

7.4 Experiment and Results

7.4.1 Effect of the Space-based Quantization Scheme

To evaluate the performance of the space-based quantization scheme, we developed the pursuer and evader DEVS/HLA models in two federates in the DEVS/HLA distributed simulation environment. One federate included pursuer DEVS/HLA models. The other federate had evader DEVS/HLA models. Each federate had its own local space manager. Each local space manager receives position updates from pursuers and evaders and performs the message filtering between pursuers and evaders. Whether or not message filtering is performed depends on the distance between pursuers and evaders. If pursuers and evaders are close together, message filtering may not be performed. Message traffic reduction is chosen as the performance measure of this experiment because, through message traffic reduction, the data to be processed by the receiving agents is reduced, as is the data actually sent over a network.

To represent the effect of the space-based quantization scheme, three different experimental conditions are introduced. In the first condition, there is no space manager. Messages are broadcast to all agents without the space manager operation. In the second condition, the space manager operation is used with two distance ranges. In this experimental condition, the distance between any two agents is stratified as two ranges, “close” and “far”, so that two different quantum sizes exist. If close, the quantum is 1 and the message can be transferred. If far, the quantum is Infinity and the message cannot be

transferred. In the third condition, the space manager operation is used with three distance ranges. In this condition, the distance between any two agents is stratified as three ranges, “close”, “middle” and “far.” If “close” or “far”, a message is transferred or filtered in the same manner as in the second condition. On the other hand, if the range is “middle,” the quantum is 10 and the filtering operation follows a filtering policy with distance-dependent sensitivity of vision in the pursuer-evader model. In middle distance range, a pursuer transmits its position update to an evader. The evader notices the pursuer and runs away. However, because the evader does not transmit its position update to the pursuer, the pursuer does *not* notice the evader. Through message filtering in the middle range, evaders can “run away” from pursuers. With this filtering policy, message communication depends on the direction of the message. When a pursuer has a message to be transferred, if the distance range between the pursuer and the evader is in the middle, the message can be transferred. However, when an evader has a message to be transferred, if the distance range between pursuer and evader is in the middle, the message cannot be transferred. Thus, assigning quantum size follows Figure 7.5.

Figure 7.7 shows the effect of the space-based quantization scheme on message traffic. The figure compares the number of transferred messages as a function of different space dimensions in the three experimental conditions. In this experiment, the total number of agents is fixed and the space dimension varies. With a fixed number of agents, the space dimension (size of bounding region) is a critical factor in comparing the number of messages because the filtering operation of the space manager is based on the distance among agents. As the space dimension increases, the space manager filters more

messages among agents with the distance-dependent sensitivity of vision in the pursuer-evader model. In the first condition (i.e. no space manager), as the space dimension increases, the number of messages transferred increases because the distances between pursuer and evader are skewed toward smaller values and many evaders stay in the "freeze" state. In the "freeze" state, the evader does not move and does not transfer any messages. Thus, as the space dimension increases, evaders have more messages to be transferred. This contrasts with performance in the second and third conditions (i.e. space manager with two distance ranges and space manager with three distance ranges, respectively). Here, as the space dimension increases, the number of messages transferred decreases because the space manager operation is based on the distances among agents. These results support the assertion that the space-based quantization scheme proposed and developed in this dissertation is an efficient means of message traffic reduction.

As illustrated in Figure 7.7, there is more reduction in message traffic when there are three distance ranges than when there are only two. In the former condition, the distance between pursuer and evader is more stratified than in the latter condition. Three different quantum sizes are assigned to these three distance ranges. The quantum sizes 1, 10, and Infinity are assigned to the close, middle and far distance ranges, respectively. If the quantum size is 1 or 10, when an agent crosses over the boundary of the area assigned for that quantum size, the agent transfers a message. However, if the quantum size is Infinity, the agent does not transfer a message. The existence of the middle distance in the third condition permits the application of distance-dependent sensitivity of vision in the pursuer-evader model, which in turn results in greater message traffic reduction. It

should be noted that the greatest benefit of the triple-quantum scheme occurs in the close encounter range where evaders are likely to have more pursuer detections of the “notice” kind than when their “mean free path” becomes large.

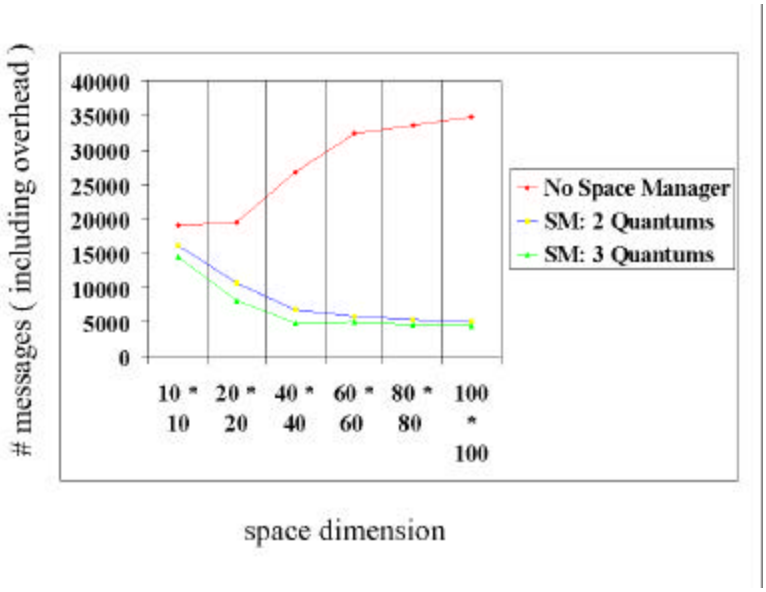


Figure 7.7 Traffic Message Reduction with the Space-based Quantization Scheme

Figure 7.8 shows the filtering rates of the different filtering types at sender and receiver federates. As stated previously, as the space dimension increases, the overall filtering rate increases. The overall filtering rate consists of filtering rates at the sender federate as well as filtering rates at the receiver federate. The filtering rate at sender federate shows how many HLA messages between federates are filtered through a network. The filtering rate at receiver federate shows how many DEVS messages within a federate are filtered. By reducing the network delay, network message reduction more

effectively decreases execution time in a large-scale distributed simulation than does non-network message reduction. At smaller space dimensions, the filtering rate at receiver is higher than the filtering rate at sender. However, at larger space dimensions, the filtering rate at sender increases further. In effect, as the space dimension increases, the increased filtering rate at sender improves system performance by decreasing execution time.

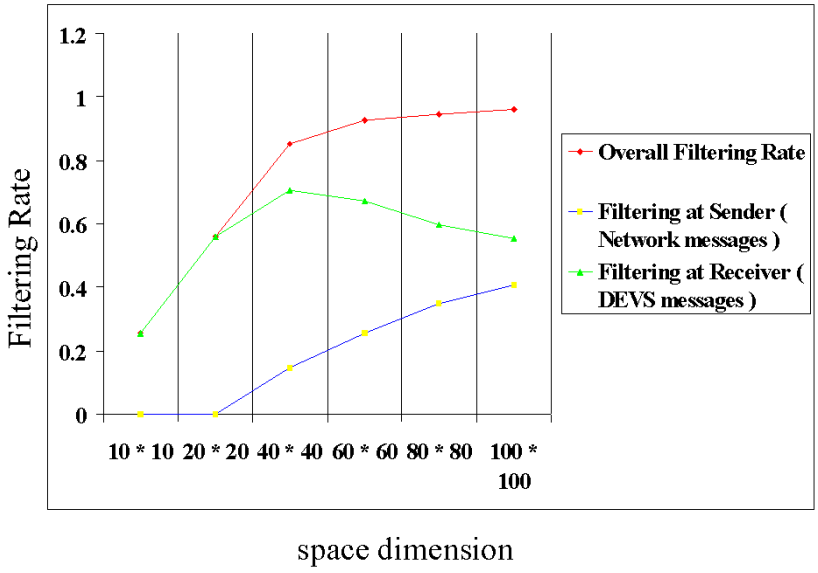


Figure 7.8 Filtering Rates with Filtering operations

7.4.2 Global and Local Space Manager Approaches

To evaluate the performance of the global and local space manager approaches in terms of the scalability of the space-based quantization in a large-scale distributed simulation, we applied the heavy load test, using over one hundred agents, to three different systems in the DEVS/HLA distributed simulation environment. In the first system, messages are broadcast to agents without the space manager operation. The second system filters messages among agents using the global space manager approach. The third system filters the messages using the local space manager approach. [1]

7.4.2.1 Message Traffic Reduction Using Global and Local Space Manager Approaches

In this section, we compare the performance of global and local space manager approaches in terms of message traffic reduction and discuss the reasons underlying the different performance of the two approaches. Figure 7.9 illustrates the number of transferred messages, including the overhead messages, in these two approaches as well as the number of messages broadcast to all agents without the space manager operation. This figure also shows the 99 (%) confidence interval of the number of messages passed. As Figure 7.9 shows, both approaches greatly reduce message traffic with the space manager operation. As the number of agents increases, the local space manager approach

[1] Each condition was executed with 5 replications and the averages were significantly different at the 99% confidence level [45].

reduces the number of messages more than the global space manager approach. Figure 7.10 shows the net message reduction using both approaches. The net message reduction is calculated by subtracting the overhead messages from the total number of messages transferred. The overhead messages in the global space manager approach are for position updates of agents to the global space manager and for distributing the connection information decided by the global space manager to each coupling operator on each federate. The overhead messages in the local space manager approach are for position updates of agents to each local space manager on each federate. Because the overhead messages that distribute the connection information in the global space manager approach are more numerous than the overhead messages for position updates of agents to each local space manager, the net message reduction of the local space manager approach increases much more than that of global space manager approach as the number of agents increases.

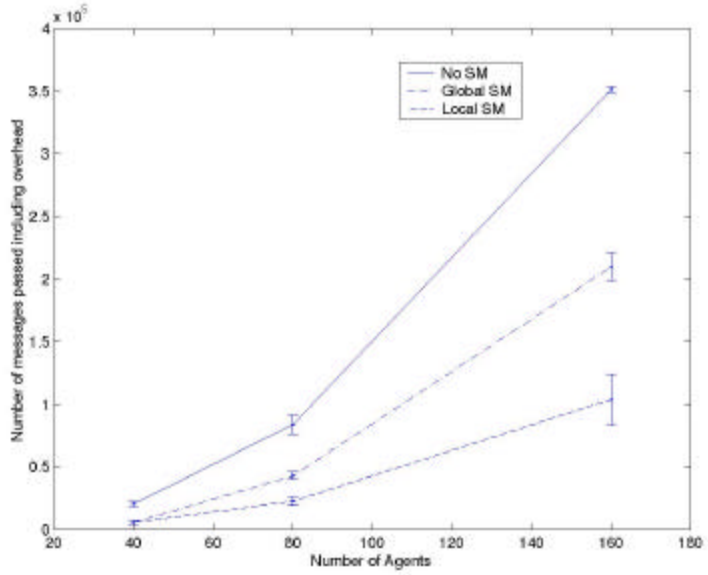


Figure 7.9 Message Traffic Reduction using Global and Local Space Manager approaches

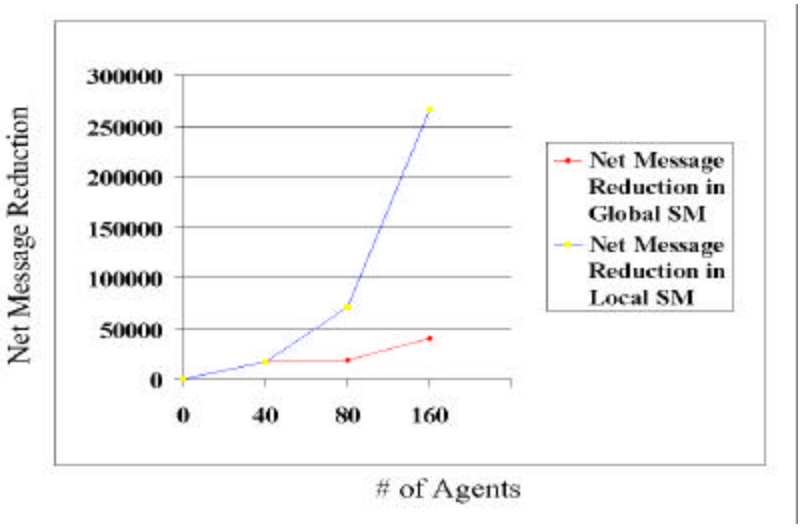


Figure 7.10 Net message traffic reduction using global and local space manager approaches

7.4.2.2 The Effect of the Space-Based Quantization in Global and Local Space Manager Approaches

In this section, we compare the performance of the global and local space manager approaches with the space-based quantization scheme. With this scheme, the distance between pursuer and evader is stratified, so that the quantum sizes related to the stratified distance are chosen. In this experiment, two and three quantum sizes are chosen and each quantum size depends on the distance between pursuer and evader.

Figure 7.11 shows the effect of the space-based quantization scheme when it is applied to the global space manager approach. Figure 7.12 shows the effect on message traffic reduction when it is applied to the local space manager approach. As the number

of agents increases, the space-based quantization scheme reduces more messages when three quantum sizes are chosen than when only two quantum sizes are chosen. Furthermore, the effect of space-based quantization on message traffic is greatest when the space-based quantization scheme is applied to the local space manager approach.

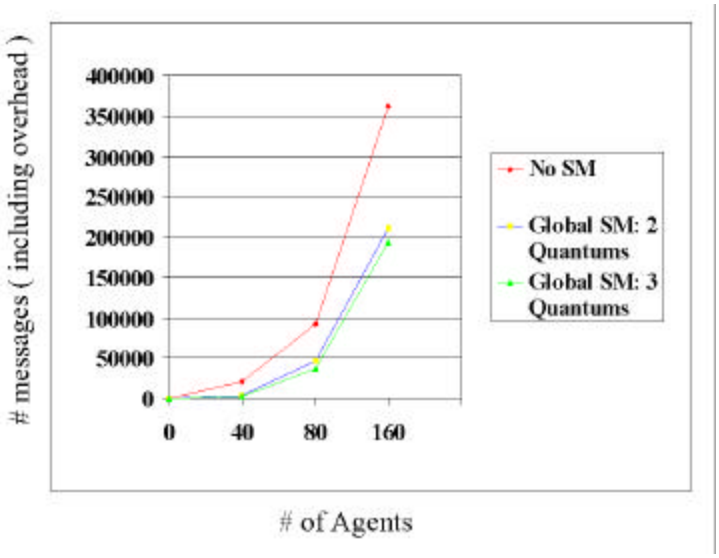


Figure 7.11 Message Traffic Reduction with the Space-based Quantization Scheme in Global Space Manager approach

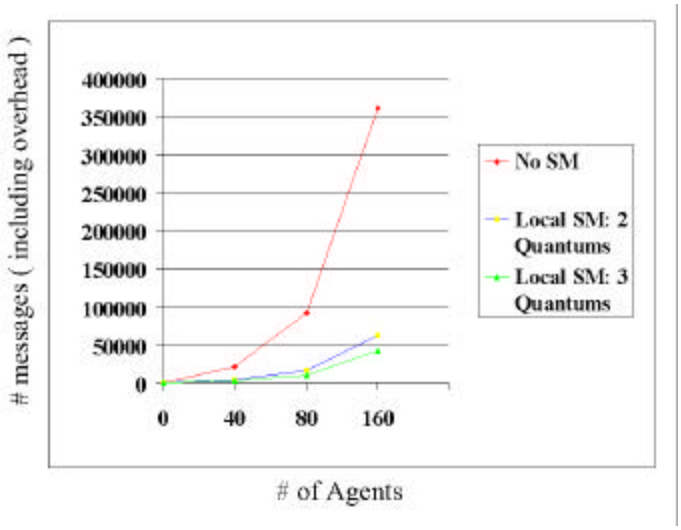


Figure 7.12 Message Traffic Reduction with the Space-based Quantization Scheme in Local Space Manager approach

7.4.2.3 Influence of Network Delay and Computation Load in Global and Local Space Manager Approaches

In this section we analyze how network delay and computation load influence system performance using system execution time as a performance measure. In order to analyze the influence of network delay in Window NT machines connected via a 10 Base T Ethernet network, we designed the experiment in such a way that there was a holding time before a message was sent from a simulation component output. This message holding time represented the network delay that occurs prior to a message being transferred over a network and the implementation of the message holding time operation is added in the DEVS/HLA simulation engine. In order to analyze the influence of system

computation time, we assigned a certain computation time to receiving agents. The system computation time represents the computation time originating from agents and the space manager.

Figure 7.13 illustrates the influence of network delay and computation load on system execution time in three system experiments including no space manager, the global space manager, and the local space manager. The execution time of the system operated without the space manager increases with a high slope as the network delay increases. The execution time of the system using the local space manager approach increases with a small slope. In the system using the global space manager approach, the system execution time increases with a somewhat higher slope than that of the local space manager approach. In all three systems, system execution time from network delay depends on the number of messages passed. The difference in performance of the three systems in terms of network delay would be clearer at the network saturation point.

In the very low network delay range, the difference noted between the performances of the system operated without the space manager and the system with the global space manager was less remarkable, although fewer messages were passed in the system using the global space manager than when no space manager was used. This is because, in the very low network delay range, the system execution time from network delay is very low. It, therefore, follows that the system execution time is due primarily to system computation time. The system computation time mainly results from the computation overhead of the global space manager approach, which is fairly high. However, the computation overhead of the local space manager is still low. Also, in the

very low network delay range, the local space manager approach has the lower system execution time than that when no space manager was used since the message reduction from the local space manager approach causes the reduced computation time of the receiver agents.

The computation time of the system operated without the space manager increases with a high slope as the agent computation time increases. The large number of messages broadcast among agents in this system causes a large local data processing time by the message-receiving agents, thus it increases the system computation time with a high slope. The computation time of the system using the local space manager approach increases with a low slope. In the system using the global space manager approach, the system computation time increases with a relatively higher slope than that of the local space manager approach. With both approaches, the local data processing time by the receiving agents is significantly reduced by the message filtering operation of the space manager though the computation overhead for the space manager operation exists. The system using the global space manager approach encounters a "bottleneck" during computation of the global space manager. However, the local space manager approach solves this problem using concurrent processing, which decreases computation overhead for the space manager.

In the very low agent computation range, the computation time of the system using the global space manager approach is not much different from that of the system operated without the space manager. This is because, in this range, the computation overhead of the global space manager is still fairly high although the local data

processing time by the receiving agents is very low. In this range, the computation time of the system using the local space manager approach is still low. To summarize, Figure 7.13 shows that as both network delay and computation load increase, the execution times of these three systems with no space manager, global space manager, and local space manager, increase with orderly different slopes. The best performance was accomplished by the local space manager approach proposed in this dissertation as both network delay and computation load increase.

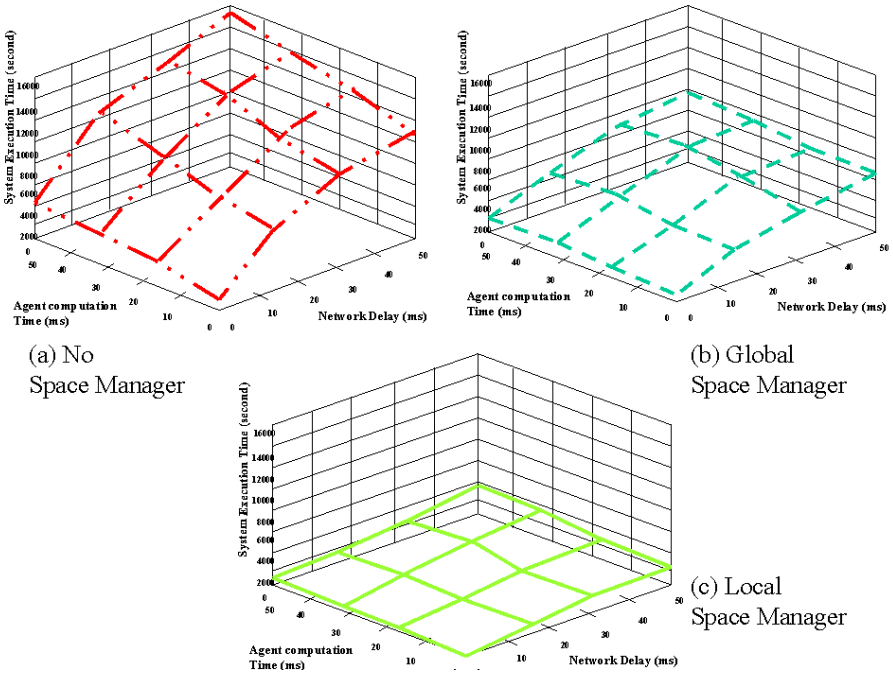


Figure 7.13 Influence of Network Delay and Computation Load

8 PROJECTILE/MISSILE APPLICATION

8.1 Projectile/Missile Application Overview

In this dissertation, a real application (projectile/missile) working in a real-world environment is used to evaluate the performance of the DEVS/GDDM environment. This application uses the geocentric-equatorial coordinate system [54, 55]. The projectile is a ballistic flight and accounts for gravitational effects, drag, and the motion of the rotation of the earth relative to it. A missile is assigned a projectile, and it follows its projectile until it hits its projectile. In modeling the projectile/missile application, there are two main models: projectile and missile. The projectile model is the model of a sphere of uniform density following a ballistic trajectory. This model begins at an initial position with an initial velocity, moves, and stops until it meets a missile. The missile model is the model of the same sphere of the projectile, and it begins at a certain initial position and a certain initial velocity, which are different from those of the projectile model. The missile model follows the projectile model assigned to it. When the missile model is close to the assigned projectile model within a certain distance, it stops and we consider the missile hits its projectile.

8.2 Projectile/Missile Modeling

The projectile model includes three sub-models: acceleration model, velocity model, and position model. The acceleration model uses parameters (e.g. gravity, atmosphere velocity, atmosphere density, etc.) to generate the acceleration values. The earth model calculates these parameters using the position values of the projectile model. For the real implementation of the velocity and the position models, the integrator model is developed. The integrator model has two types: the DTSS integrator and the DEVS predictive integrator. The velocity model receives the acceleration input from the acceleration model, and the position model receives the velocity input from the velocity model. Finally, the position model generates three dimensional position values of the projectile and sends them to both the earth model and the missile model.

The missile model includes two sub-models: the velocity generator model and the position model. The velocity generator model receives the position update message from the projectile model, and it generates the velocity values and sends them to the position model. The position model (an integrator model) receives the velocity values and generates the three dimensional missile position values. As the simulation time is advanced, the position of the missile model gradually becomes closer to the position of the projectile model.

To realize the projectile/missile application in the DEVS/GDDM environment proposed in this dissertation, we made four systems to perform the case study with the projectile/missile application: The first system is a basic system, which is not applied by

the interest-based quantization scheme of the DEVS/GDDM environment; The second system uses the non-predictive interest-based quantization scheme. The third system employs the predictive interest-based quantization scheme; and the multiplexing method is included in the fourth system.

In the basic system, there are two federates; the projectile federate and the missile federate. The projectile federate includes the projectile models, and the missile federate includes the missile models. In the basic system, the DTSS integrators are used for the velocity and the position models in the projectile model, as well as for the position model in the missile model. The position model of the projectile model in the projectile federate sends the three dimensional position double values (x, y, z) to the missile federate in a fixed step time due to the fact that the DTSS integrators are used. The results from the basic system are the standard results for evaluating the performance of the DEVS/GDDM environment. The system with the non-predictive interest-based quantization scheme is supported by the non-predictive interest-based quantization method of the DEVS/GDDM environment, and the high level modeling of the system in the DEVS model layer is the same as that of the basic system. Figure 8.1 and Figure 8.2 illustrate the component diagram of the projectile and of missile models in the basic system, as well as the second system using the non-predictive interest-based quantization scheme.

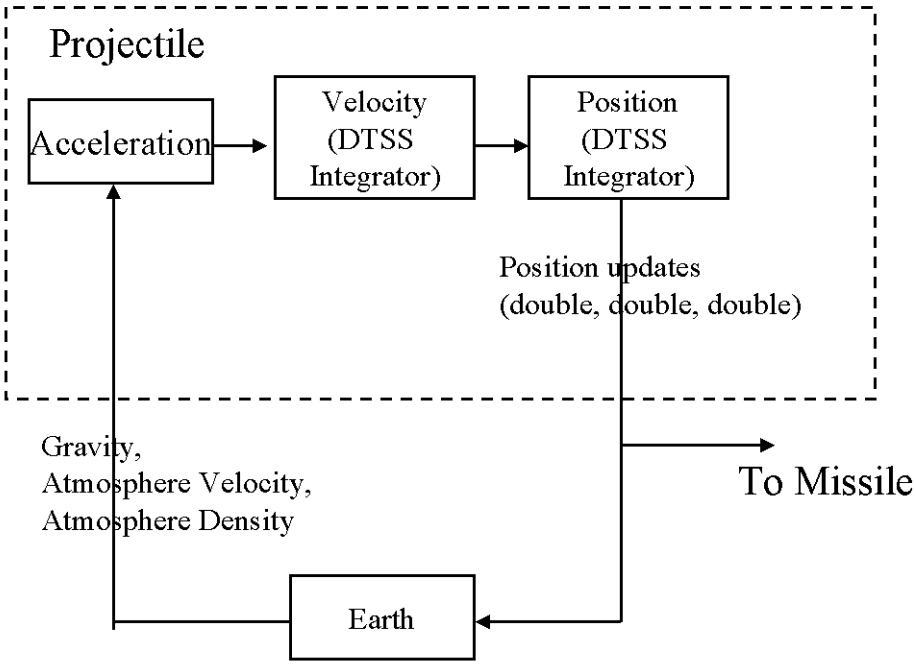


Figure 8.1 Component diagram of the projectile model in the basic system and the second system using the non-predictive interest-based quantization scheme

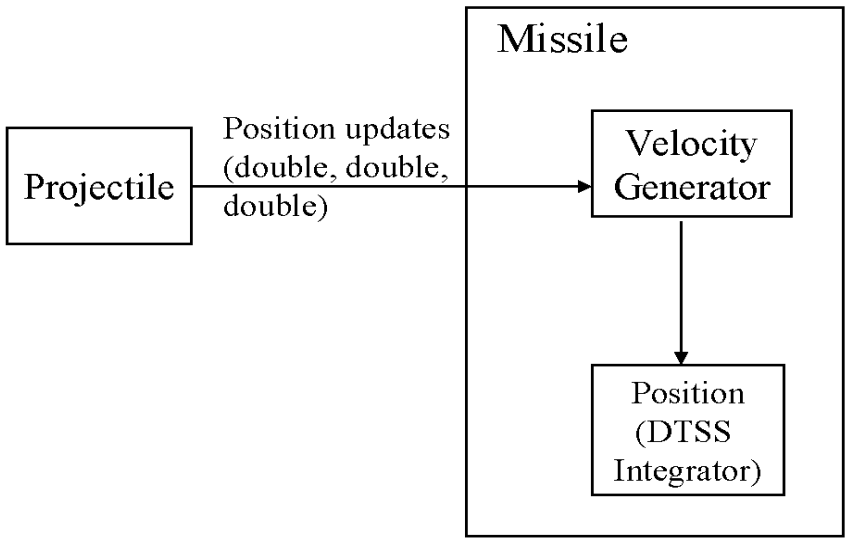


Figure 8.2 Component diagram of the missile model in the basic system and the second system using the non-predictive interest-based quantization scheme

8.2.1 Predictive Interest-based Quantization scheme

To reduce the tremendous data bits communicated between the projectile model and the missile model with only reasonable error, we made the third system which utilized the predictive interest-based quantization scheme of the DEVS/GDDM environment. In order to apply to this predictive interest-based quantization scheme, the DEVS predictive integrators are used for the velocity and position models in the projectile and the missile models. The third system includes the projectile and the missile federates which are the same as the basic system. The position model in the projectile

model sends the three dimensional position integer values, such as (-1, 0, 1), to the velocity generator model in the missile model. To generate the velocity of the missile model, the velocity generator model needs the current position values of the projectile model; thus, it calculates the current position values of the projectile model by multiplying the input integer values and the current quantum size and by adding the multiplied result to the old position values.

To avoid error of the projectile position values in the missile model, the current quantum size in the missile model should be the same as the current quantum size in the projectile model. The current quantum size is decided by the space manager in the DEVS/GDDM environment, and it is distributed to both the projectile model and the missile model. In the third system, by sending the integer values (not the double values) from the projectile federate to the missile federate, the inter-federate communication data bits are tremendously reduced. To save more of the data bits, the three dimensional position integer values, such as (-1, 0, 1), are not transferred directly, and only five ($5 > \log_2 3^3$) data bits representing the three dimensional integer values are sent. Therefore, the encoder, which changes the three dimensional integer values to the five data bits, is needed and supported by the DEVS/GDDM environment.

In order to change the received five data bits to the exact three dimensional integer values in the missile federate, the decoder is supported by the DEVS/GDDM environment. Also, the position model of the projectile model sends the five data bits to the space manager in the DEVS/GDDM environment. The space manager needs the decoder to decode the five data bits to the exact three dimensional integer values and

generates the position values of the projectile model as the velocity generator in the missile model does. Figure 8.3 and Figure 8.4 illustrates the component diagram of the projectile and missile models in the third system using the predictive interest-based quantization scheme of the DEVS/GDDM environment.

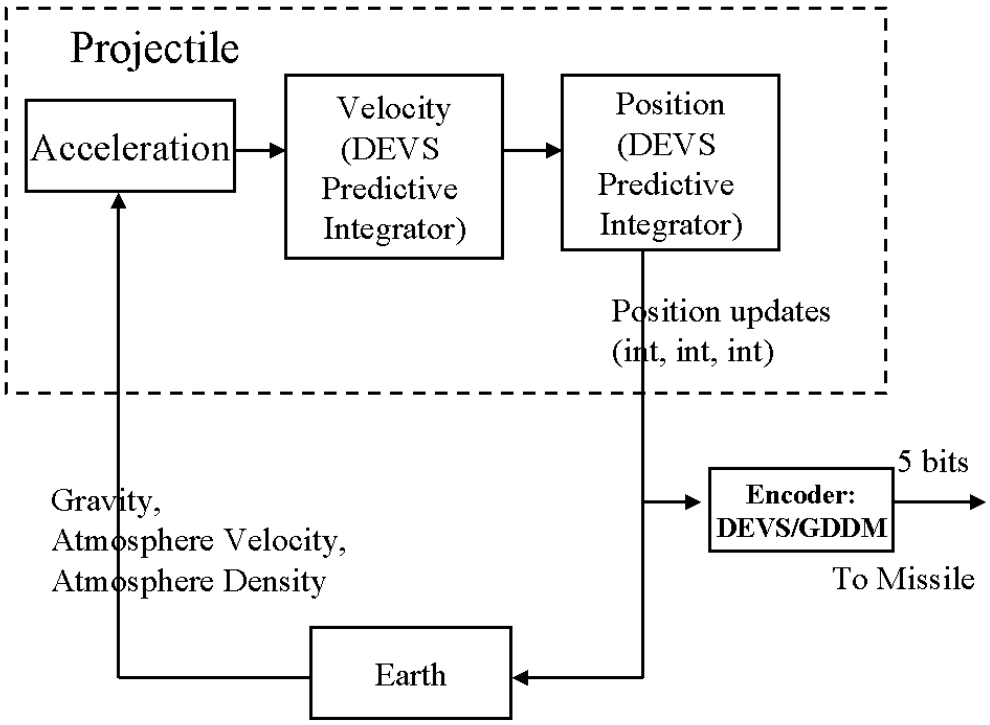


Figure 8.3 Component diagram of the projectile model in the third system using the predictive interest-based quantization scheme of the DEVS/GDDM environment

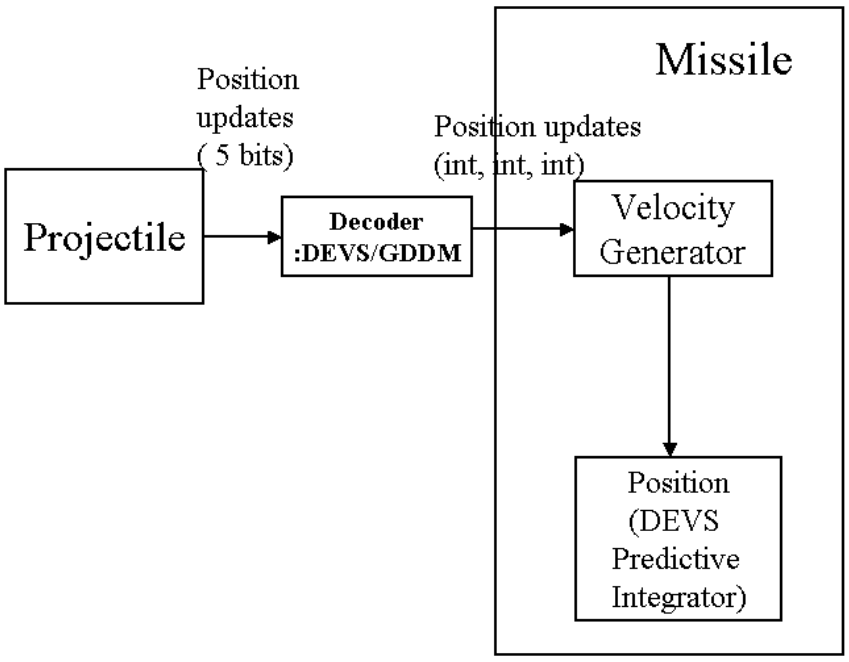


Figure 8.4 Component diagram of the missile model in the third system using the predictive interest-based quantization scheme of the DEVS/GDDM environment

8.2.1.1 The methods to reduce the error in the systems using the DEVS predictive integrators

In the third system using the predictive interest-based quantization scheme, error occurs due to using the DEVS predictive integrators (not DTSS integrators) in the projectile/missile application. To reduce this error, three methods are applied to this system. The first method is to use a smoother model, which can reduce the error from the multi-dimensional output values of the DEVS predictive integrator with each different

unfixed time advance. The velocity model and the position model outputs with variable time advance in the projectile model since these models are developed with DEVS predictive integrators. Also, because the positions of the projectile and the missile are three dimensional, the position values of the DEVS predictive integrator are outputted with each variable time advance for each dimension. The earth model generates the gravity, atmosphere density, etc., with all three dimensional position values at a same time. The position values, outputted with each different variable time advance, include the old values and current values at a given event time. The use of the old values of the projectile position in the earth model causes the error.

In this third system, the smoother model receives, keeps, and updates the three dimensional position values outputted from the DEVS predictive integrator with each different variable time advance for each dimension until the fixed time step of the DTSS integrator is advanced. When the fixed time step advance of the DTSS integrator comes, the smoother model outputs the three dimensional position values to the earth model. By using the smoother model, the error caused by the use of the old values of the projectile position can be reduced. Figure 8.5 illustrates the component diagram of the projectile model in the third system using the smoother model.

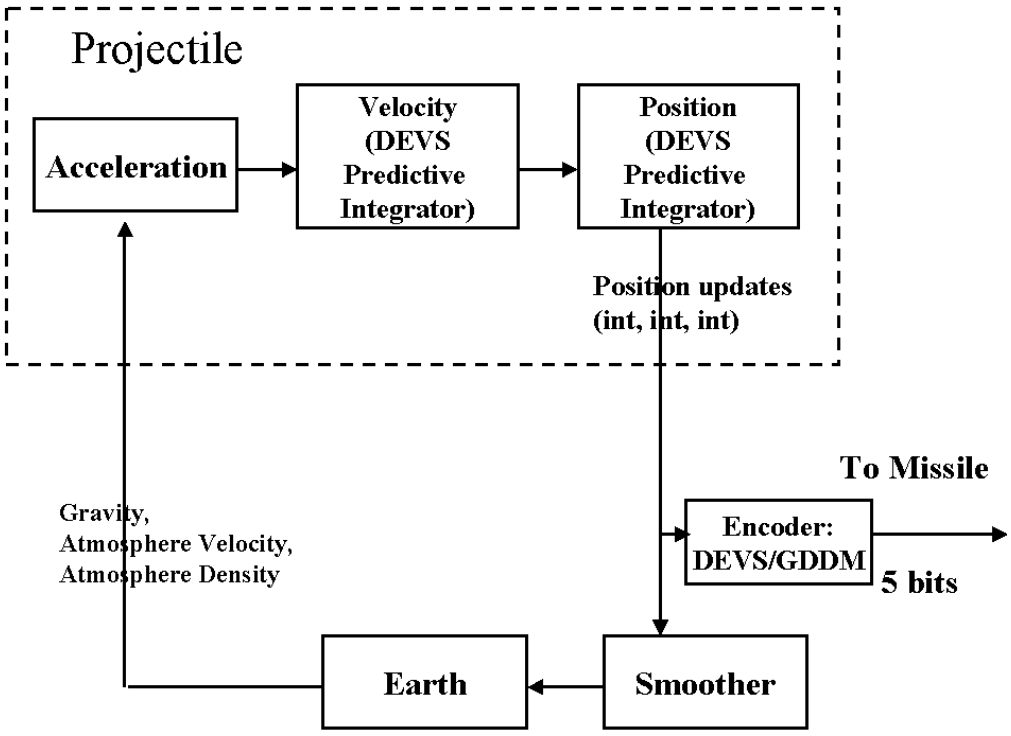


Figure 8.5 Component diagram of the projectile in the third system using the smoother model.

The second method for reducing error, which occurs due to using the DEVS predictive integrator, is to use the standard quantum size which decides the message filtering rate in the DEVS predictive integrator. In the predictive quantization theory, the standard quantum size provides the same accuracy as that caused by the time step (h) of a DTSS integrator. Figure 8.6 illustrates the relationship of standard quantum size and the time step (h) of a DTSS integrator.

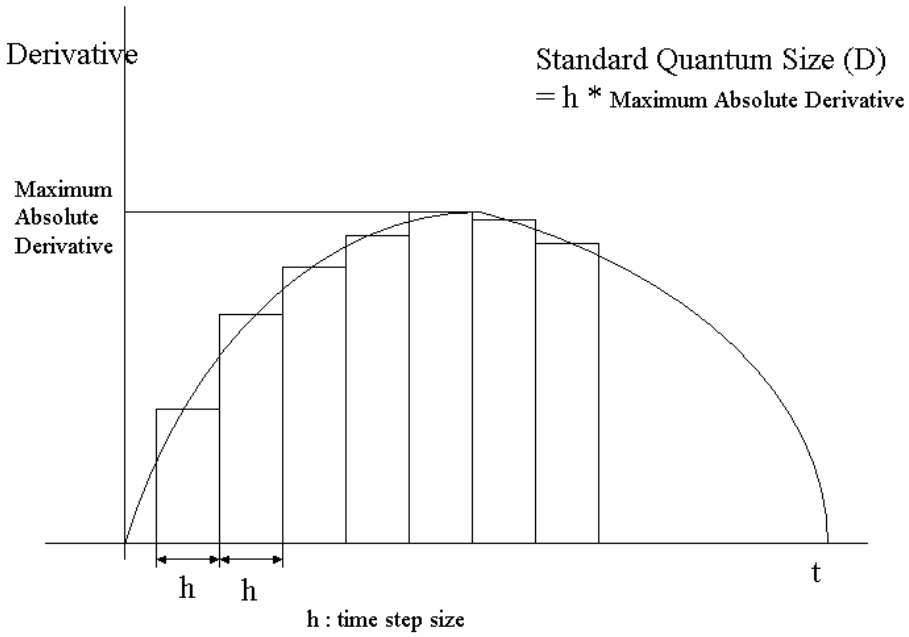


Figure 8.6 Standard Quantum Size (D) and Time Step (h) of a DTSS integrator.

The standard quantum size is calculated by multiplying the step time (h) and the maximum absolute derivative of the DTSS integrator, which is the maximum input of the DTSS integrator. In the basic system, the time step (h) of the DTSS integrator is fixed. The DTSS integrators, working in the basic system, have three dimensional maximum absolute derivatives. Therefore, we have three dimensional standard quantum sizes. When this method for using the standard quantum size is applied, the space manager in the DEVS/GDDM environment decides the multiple factor for multiplying to the standard quantum size rather than deciding the actual quantum size. Thus, the quantum decision table includes the multiple factors related to the distances between the projectile model and the missile model. In initializing time, the standard three dimensional quantum

sizes are given to the DEVS predictive integrators. Each DEVS predictive integrator in the projectile model generates its own three dimensional quantum sizes by multiplying the multiple factor and the standard three dimensional quantum sizes. Simultaneously, the velocity generator model in the missile model gets the standard three dimensional quantum sizes of the projectile model's position model in the initializing time, and the velocity generator model calculates its three dimensional quantum sizes by multiplying the multiple factor from the space manager and the standard three dimensional quantum sizes in run time.

Table 8.1 and Table 8.2 show the maximum absolute derivatives and the standard quantum sizes in the velocity and position models when the step time (h) of DTSS integrators of the velocity and position models is 0.01. The maximum absolute derivatives and the standard quantum sizes include three dimensions and have their values which indicate x, y, and z directions.

Table 8.1 Maximum absolute derivatives and the standard quantum sizes in velocity model (h = 0.001)

	X direction	Y direction	Z direction
Maximum absolute derivative	4.878143177E-5	0.030262613307	9.799137940
Standard Quantum Size (D = h * Maximum absolute derivative)	4.87E-8	0.0000302626133	0.0097991379

Table 8.2 Maximum absolute derivatives and the standard quantum sizes in position model (h = 0.001)

	X direction	Y direction	Z direction
Maximum absolute derivative	1.698550814E-4	10.0	98.77168099258
Standard Quantum Size (D = h * Maximum absolute derivative)	1.69E-7	0.01	0.0987

The third method to reduce the error in the third system is to remove the error incurred when the DEVS predictive integrator receives a new quantum size from the space manager in the DEVS/GDDM environment and generates the new output values (not the pre-scheduled output values). To avoid this error with the new quantum size, the position model of the projectile model sends the actual position double-precision values (not encoded small data bits) to the velocity generator model in the missile model and to the space manager in the DEVS/GDDM environment. When the velocity generator model in the missile model receives the actual position values and the new quantum size from space manager, it updates the current position values of the projectile model and stores the new quantum sizes for subsequent use. Also, the space manager in the DEVS/GDDM environment receives the actual position values and updates its representation of position values of the projectile model.

8.2.2 Mutiplexing Interest-based Quanitzation Scheme

As the number of projectile model and the missile model pairs in federates increases, the number of messages communicated among federates increases also significantly. To perform the multiplexing interest-based quantization scheme for reducing message traffic of the increased pairs, two components (e.g. sender multiplexer and receiver de-multiplexer components) are used in DEVS/GDDM environment.

The sender multiplexer gathers the messages outputted from the sender agents within a time granule into a large message, which is sent to the receiver de-multiplexer in

the other receiver federate. The receiver de-multiplexer separates the large multiplexed message to the small-unmultiplexed messages and distributes the small messages to the proper receiver agents. As the number of sender and receiver pairs increases, through this multiplexing interest-based quantization scheme, tremendous communication bits can be saved. Moreover, by using this multiplexing interest-based quantization scheme, many HLA interactions are reduced to only one HLA Interaction. To exchange the message between sender and receiver pair in two different federates, one HLA interaction is needed. As the number of sender and receiver pairs increases, the number of the HLA interactions for the increased pairs also increases if the multiplexing interest-based quantization scheme is not used; therefore, the number of the increased HLA interactions causes memory and the computation overhead in HLA/RTI communication. By reducing the number of HLA interactions, the multiplexing interest-based quantization scheme is more effective in a large-scale distributed simulation.

In order to analyze the performance of the multiplexing interest-based quantization scheme, we investigated the ratio of the number of bits needed for the multiplexing predictive quantization to the number of bits needed for the non-multiplexing, non-predictive quantization with the same number of components. The analysis is given in Table 8.3 where we consider all six combinations of quantization (non-predictive and predictive) and multiplexing (fixed and variable). The fixed and variable multiplexing schemes were discussed in chapter 5.

Table 8.3 Network bandwidth requirement for quantization and multiplexing schemes

(S_{OH} : the number of overhead bits for a packet (160 bits); S_D: the non-quantized data bit size (64*3 bits for double precision real numbers for three dimensions); S_Q: the quantized and encoded data bit size (5 bits for three dimensions ($\log_2 3^3 < 5 = S_Q$)); S_L: the encoded data bit size for sender ID (10 bits for 1000 N_{pair} ($\log_2 1000 < 10 = S_Q$)); N_{pair} : the number of pair components (1000), a: the ratio of active components).

Scheme		# bits required for N_{pair}	Ratio to Non-predictive quantization for large N_{pair}	Ratio for N_{pair} =1000 S_{OH}=160 bits S_D=64*3bits S_Q= 5 bits S_L= 10 bits
Non-predictive quantization		aN_{pair} (S_{OH} + S_D)	1	1
Predictive quantization (non-multiplexed)		aN_{pair} (S_{OH} + S_Q)	(S_{OH} + S_Q) / (S_{OH} + S_D)	0.46
Fixed Multiplexing	Multiplexing non-predictive quantization	(S_{OH}+N_{pair}(S_D+1))	(S_D + 1) / a(S_{OH}+S_D)	0.54/a
	Multiplexing predictive quantization	(S_{OH}+N_{pair}(S_Q+1))	(S_Q+1) / a(S_{OH}+S_D)	0.017/a
Variable Multiplexing	Multiplexing non-predictive quantization	(S_{OH}+aN_{pair}(S_D+S_L))	(S_D + S_L) / (S_{OH} + S_D)	0.57
	Multiplexing predictive quantization	(S_{OH}+aN_{pair}(S_Q+S_L))	(S_Q + S_L) / (S_{OH} + S_D)	0.034

The predictive quantization without multiplexing performs 46 (%) reduction in network load relative to non-predictive quantization. In the multiplexing non-predictive quantization scheme, the reduction (approx. 54(%)/a) is performed by combining the actual double value outputs into one message. Greater advantage is obtained from the

multiplexing predictive quantization, which combines the encoded data bit size (5 bits for three dimensional data of message and 10 bits for sender ID) per component into one message. When the fixed multiplexing predictive quantization scheme is used, in order to reduce the bit sending ratio below 10 (%), at least 17 (%) active components are required. For variable multiplexing predictive quantization, the reduction ratio is 3.4 %.

Table 8.4 Network bandwidth requirement for fixed and variable multiplexing schemes with varying *a* (*a*: the ratio of active components)

Scheme		Ratio for N_{pair} =1000 S_{OH}=160 bits S_D=64*3bits S_Q= 5 bits S_L= 10 bits	a = 0.6	a = 0.5	a = 0.4	a = 0.1
Fixed Multiplexing	Multiplexing non-predictive quantization	0.54/a	> 1.0	> 1.0	> 1.0	> 1.0
	Multiplexing predictive quantization	0.017/a	0.028	0.034	0.0425	0.17
Variable Multiplexing	Multiplexing non-predictive quantization	0.57	0.57	0.57	0.57	0.57
	Multiplexing predictive quantization	0.034	0.034	0.034	0.034	0.034

In order to compare both fixed and variable multiplexing predictive quantization schemes with varying *a*, Table 8.4 is extended from Table 8.3. Table 8.4 shows the effectiveness of both fixed and variable multiplexing with different *a* value in a specified

case of Table 8.3. If a is greater than a_c (in **Figure 5.13**), the fixed multiplexing predictive quanitzation is more effective. In this case, a_c is 0.5 ($? \frac{1.7 * D(? 3)}{\log_2 1000}$) and determines the effectiveness of the fixed and variable multiplexing predictive quantization schemes.

8.3 Experimentation and Results

8.3.1 The Effect of Predictive Interest-based Quanitzation Scheme

To evaluate the performance of the predictive filtering approach supported by the DEVS/GDDM environment, we developed the projectile/missile system using the predictive interest-based quantization scheme. The DEVS predictive integrators were used in the system and we performed the predictive interest-based quantization scheme by changing the quantum size related to the distance between the missile and its assigned projectile. In the basic system, the DTSS integrators are used and the interest-based quantization scheme is not used, so that the basic system is considered the standard system in which no error occurs.

To evaluate the performance of the predictive interest-based quantization scheme, we developed two federates: projectile and missile. The projectile in the projectile federate sends the position update message, which includes the encoded bits (5 bits for three dimensions) and HLA packet overhead (160 bits), to the missile. The missile in the

missile federate sends the same size data bits as the projectile's position update message data bits to the space manager as Figure 8.7 illustrates.

This experiment compares the total passed data bits and the error incurred between the basic system and the system using the predictive interest-based quantization scheme. The error is defined as the difference between the projectile positions in these two systems. The total passed data bits indicate the data bits that a missile receives from the projectile and that the space manager receives from the missile as Figure 8.7 shows. The overhead data bits sent to the space manage are needed to perform for a quantum decision operation of the predictive interest-based quantization scheme. Thus, these overhead bits were included in the total passed data bits. Figure 8.7 illustrates the total passed data bits using the predictive interest-based quantization scheme.

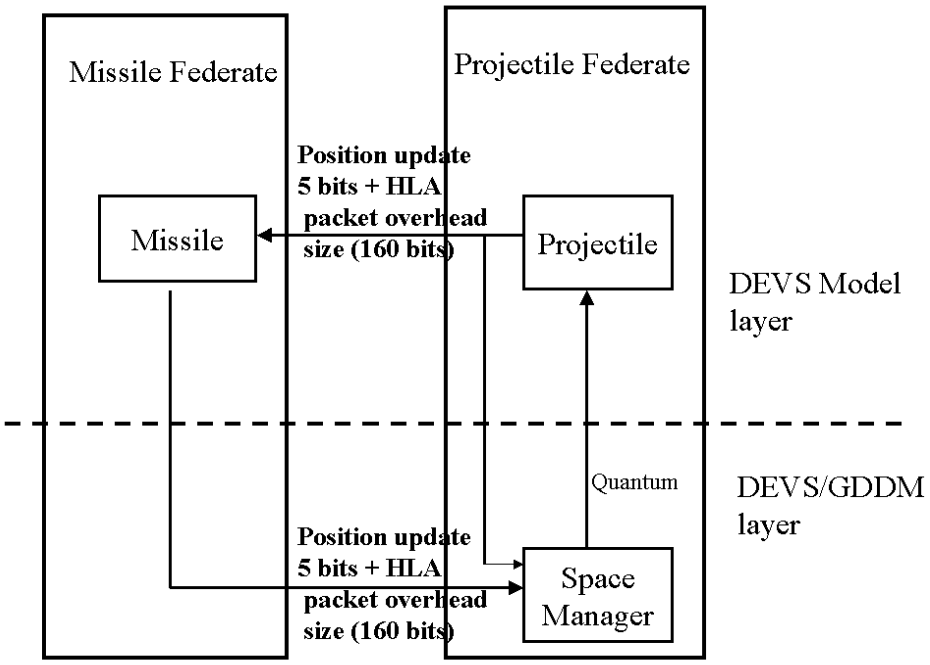


Figure 8.7 Data bits passing including the overhead data bits in the system applied by the predictive interest-based quantization scheme.

Table 8.5 shows the error trajectory of the system, which uses the predictive interest-based quantization scheme, in varying range of the multiplying factors of the standard quantum sizes. As the simulation time increases, the error decreases because the multiplying factor of the standard quantum sizes decreases. As Table 8.5 shows, while the multiplying factor varies from 40 to 1, the error decreases significantly.

Table 8.5 Error (%) Trajectory for varying range of multiplying factors of the standard quantum sizes

Simulation Time Range of Multiplying factors	1.0	3.0	5.0	7.0	9.0
10 ~ 1.0	0.16	0.08	0.04	0.05	0.02
20 ~ 1.0	0.36	0.05	0.11	0.07	0.10
40 ~ 1.0	1.87	0.40	0.12	0.13	0.10

Table 8.6 illustrates ratio trajectory of passed data bits in resulting from different ranges of multiplying factors of the standard quantum sizes. The ratio of passed data bits is calculated by:

Ratio of passed data bits =

passed data bits when using the predictive interest-based quantization scheme /

passed data bits when no quantization is used

As the simulation time increases, the ratio of passed data bits increases because the multiplying factor of the standard quantum sizes decreases. In other words, as the multiplying factors of the standard quantum sizes increases, the passed data bits decrease significantly when using predictive interest-based quantization scheme.

Table 8.6 Ratio trajectory of passed data bits for varying range of multiplying factors of the standard quantum sizes (predictive interest-based quantization vs. No quantization

Simulation Time Range of multiplying factor	1.0	3.0	5.0	7.0	9.0
10 ~ 1.0	0.054	0.056	0.060	0.066	0.082
20 ~ 1.0	0.033	0.034	0.036	0.039	0.048
40 ~ 1.0	0.017	0.017	0.018	0.019	0.023

8.3.2 The Effect of the Multiplexing Interest-based Quantization Scheme

To evaluate the performance of the multiplexing interest-based quantization scheme, we made the non-multiplexing and the multiplexing systems with this projectile/missile application. These systems include two federates: projectile and missile. Each federate is assigned to a different computer and the experimental computers are connected in a LAN environment.

Figure 8.8 illustrates the non-multiplexing system. The system includes a certain number of projectile and missile pairs in the projectile and the missile federates. Each projectile model sends its position update message, which includes the encoded five data bits (for three dimensions) and HLA packet message overhead (160 bits), to its assigned missile. Also, each missile model sends its position update message (same size data bits as projectile's position message data bits) to the space manager in the projectile federate.

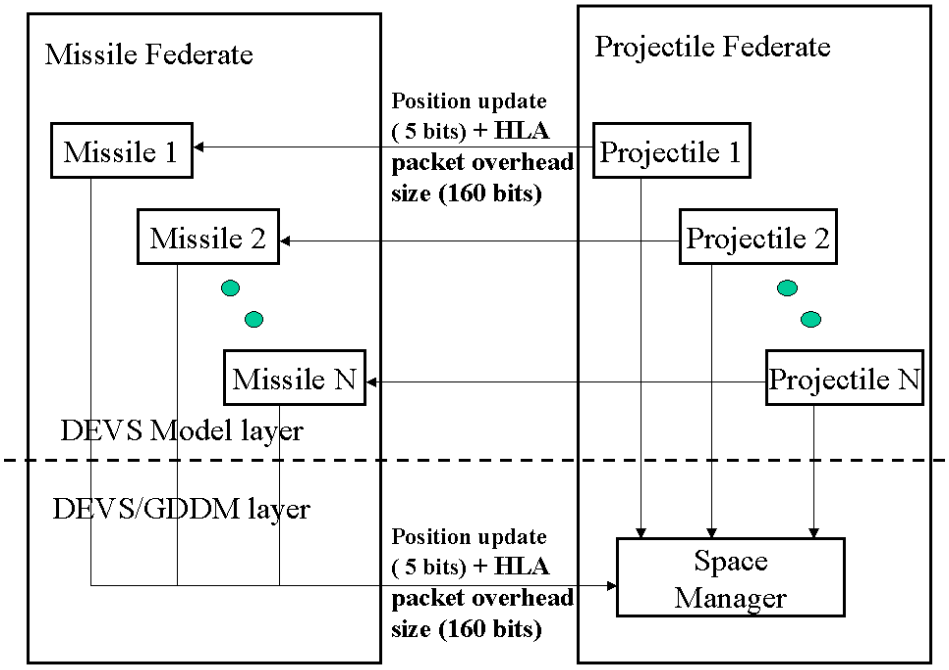


Figure 8.8 Non-multiplexing system in the projectile/missile application

Figure 8.9 illustrates the component diagram of the multiplexing system. The projectile federate includes the multi-projectile component, which contains a certain number of the projectiles. The sender multiplexer in the projectile federate gathers the messages (including the encoded five data bits) from the projectiles at the same time, makes a large multiplexed message, and sends the multiplexed message to the receiver de-multiplexer in the missile federate. The receiver de-multiplexer in the missile federate separates the multiplexed message into small, unmultiplexed messages and distributes the small messages to the proper missiles. The sender multiplexer in the missile federate gathers the messages from the missiles at the same time and sends the multiplexed

message to the receiver de-multiplexer in the projectile federate. The space manager in DEVS/GDDM layer of the projectile federate side receives the missile position update messages from the receiver de-multiplexer and directly receives the projectile position update messages from the projectiles.

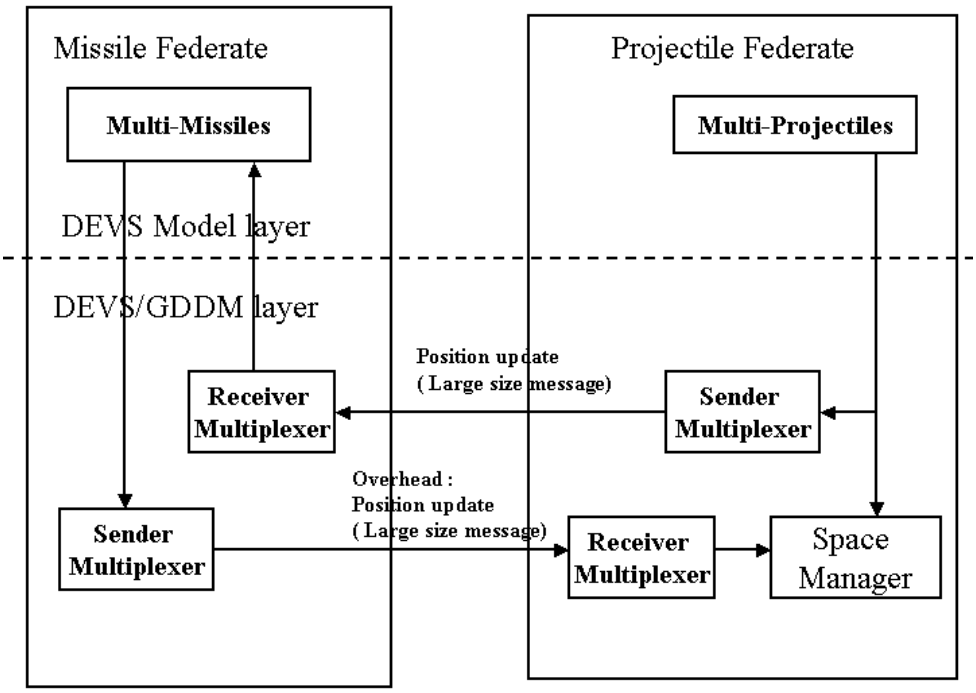


Figure 8.9 Multiplexing system in the projectile/missile application

In order to evaluate the performance of the multiplexing interest-based quantization scheme in the real projectile/missile application, we extracted the results, which are based on the analysis of the ratio of the message size needed for the

multiplexing predictive quantization to the number of bits needed for the non-multiplexing quantization at the Table 8.3 in section 8.2.

In this experimentation of the multiplexing interest-based quantization scheme, the boundary crossings within a certain time granule are considered simultaneous. As the time granule increases, the error occurred from the time granule increases. We investigated the error trajectory with varying time granule, where the multiplying factor varies from 10 to 1.

Table 8.7 Error (%) Trajectory with Varying Time Granule (Range of multiplying factors (10 ~ 1.0))

Simulation Time Time Granule	1.0	3.0	5.0	7.0	9.0
0.01	3.79	4.38	5.87	4.20	4.55
0.001	0.16	0.27	0.64	0.11	0.13
0.0005	0.16	0.08	0.08	0.07	0.13
0.0001	0.16	0.02	0.04	0.05	0.10

As Table 8.7 shows, when the time granule is 0.01, the error is large. While the time granule is below than 0.001, the error is below 1.0 (%) and is reasonably small acceptable error.

As we discussed in chapter 5, the ratio (a) of active components separates the effectiveness of the fixed and variable multiplexing schemes. The ratio (a) of active components is dependent on the time granule and the multiplying factor of the standard quantum size. We investigated the trajectory of the ratio (a) of active components with

varying time granule. As Table 8.8 and Table 8.9 show, as the time granule increases, the ratio (*a*) also increases. As the multiplying factor of the standard quantum size decreases, the ratio (*a*) also increases.

Table 8.8 Trajectory of the ratio (*a*) of active components (Time Granule: 0.001)

Simulation Time Range of multiplying factors	1.0	3.0	5.0	7.0	9.0
10 ~ 1.0	0.191	0.208	0.249	0.317	0.409
20 ~ 1.0	0.133	0.142	0.158	0.201	0.316
40 ~ 1.0	0.124	0.122	0.125	0.142	0.269

Table 8.9 Trajectory of the ratio (*a*) of active components (Time Granule: 0.0001)

Simulation Time Range of multiplying factors	1.0	3.0	5.0	7.0	9.0
10 ~ 1.0	0.116	0.116	0.117	0.117	0.120
20 ~ 1.0	0.116	0.114	0.116	0.117	0.117
40 ~ 1.0	0.115	0.113	0.116	0.117	0.117

To see the effectiveness of both fixed and variable multiplexing schemes in varying the ratio (*a*) of active components, we calculated the network bandwidth requirement (using Table 8.3) of both fixed and variable multiplexing schemes using the trajectories of the ratio (*a*) of active components in Table 8.8 and Table 8.9. Table 8.10

and Table 8.11 show the trajectory of ratio of passed data bits between the fixed and variable multiplexing schemes in a varying time granule. The ratio of passed data bits is calculated by:

$$\text{Ratio of passed data bits} = \frac{\text{passed data bits when using variable multiplexing}}{\text{passed data bits when using fixed multiplexing}}$$

As we discussed in chapter 5, when the component pair number is 80, the ratio (a_c) of active components needed to separate the effectiveness between fixed and variable multiplexing schemes is 0.78. In the projectile/missile application, since the maximum value of the ratio (a) of active components is 0.409, the variable multiplexing scheme requires less network bandwidth than that of the fixed multiplexing scheme. As the multiplying factor increases and the time granule decreases, both the ratio (a) of active components and the network bandwidth requirement of the variable multiplexing scheme decrease; therefore, the ratio of the passed data bits between variable and fixed multiplexing schemes decreases in Table 8.10 and Table 8.11. The decreased ratio of the passed data bits indicates that the variable scheme is more effective than the fixed multiplexing in Table 8.10 and Table 8.11.

Table 8.10 Trajectory of ratio of passed data bits (variable/fixed multiplexing) (Time Granule: 0.001, Component pairs: 80)

Simulation Time Range of multiplying factors	1.0	3.0	5.0	7.0	9.0
10 ~ 1.0	0.536	0.561	0.623	0.725	0.862
20 ~ 1.0	0.449	0.463	0.487	0.552	0.724
40 ~ 1.0	0.430	0.430	0.436	0.461	0.652

Table 8.11 Trajectory of ratio of passed data bits (variable/fixed multiplexing) (Time Granule: 0.0001, Component pairs: 80)

Simulation Time Range of multiplying factors	1.0	3.0	5.0	7.0	9.0
10 ~ 1.0	0.424	0.424	0.425	0.426	0.431
20 ~ 1.0	0.423	0.419	0.423	0.423	0.426
40 ~ 1.0	0.422	0.419	0.422	0.423	0.425

Here, we can decide an optimal time granule. As we see in Table 8.7, while the time granule is below than 0.001, the error is below 1.0 (%). We consider that an error below 1.0 (%) satisfies a reasonable error tolerance. To investigate the variation of the network bandwidth requirement in a varying time granule, we provide the trajectory of the ratio of the passed data bits between two time granules (0.001 vs 0.0001) in Table 8.12 and Table 8.13. All ratios of the passed data bits between two time granules (0.001

vs 0.0001) are less than 1.0, and less network bandwidth is required when the time granule is 0.001 than is required when the time granule is 0.0001. As the multiplying factor of the standard quantum size decreases, the ratio of the passed data bits between two time granules (0.001 vs 0.0001) decreases and the difference of the two network bandwidth requirements increases. When the fixed multiplexing scheme is used, the difference of two network bandwidth requirements (between two time granules (0.001 vs 0.0001)) increases more than when the variable multiplexing scheme is used. Therefore, when the time granule is 0.001, we can save the network bandwidth requirement with a reasonable error tolerance using both fixed and variable multiplexing schemes.

Table 8.12 Trajectory of ratio of passed data bits in fixed multiplexing (Time granule: 0.001 vs 0.0001)

Simulation Time Range of multiplying factors	1.0	3.0	5.0	7.0	9.0
10 ~ 1.0	0.623	0.571	0.501	0.432	0.379
20 ~ 1.0	0.923	0.803	0.743	0.647	0.551
40 ~ 1.0	0.951	0.911	0.915	0.842	0.706

Table 8.13 Trajectory of ratio of passed data bits in variable multiplexing (Time granule: 0.001 vs 0.0001)

Simulation Time Range of multiplying factors	1.0	3.0	5.0	7.0	9.0
10 ~ 1.0	0.789	0.754	0.731	0.669	0.639
20 ~ 1.0	0.978	0.890	0.856	0.800	0.744
40 ~ 1.0	0.982	0.950	0.959	0.917	0.837

In order to evaluate the actual system execution performance of the multiplexing interest-based quantization scheme, we compared the passed data bits and the system execution time of a non-predictive quantization system, a predictive quantization system and a multiplexing predictive quantization system.

As Figure 8.10 shows, the multiplexing predictive quantization system greatly reduces the passed data bits compared to the non-predictive and predictive quantization systems with varying the multiplying factors. As the range of the multiplying factors increases, the passed data bits decrease in all systems (e.g. the non-predictive and predictive quantization systems and the multiplexing predictive quantization system). The multiplexing predictive quantization system greatly reduces the passed data bits more than both the non-predictive and predictive quantization systems. Compared to the non-predictive and predictive quantization systems, the predictive quantization system shows more reduction of passed data bits due to the theoretical advantages of predictive

quantization (i.e., both the number of messages and their size can be reduced over non-predictive quanitzation).

Figure 8.11 illustrates how much the multiplexing predictive quantization system saves the system execution time by comparing non-predictive and predictive quantization systems. The non-predictive quantization system is a discrete time system and includes a lot of local computations based on the system's use of DTSS integrators. Meanwhile, predictive quantization and multiplexing predictive quantization systems are discrete event systems which tremendously reduce those local computations since the systems use the DEVS integrators.

The saving of system execution time between the non-predictive quantization system and the predictive quantization system demonstrates the effect of the reduction of network bandwidth and of the big local computation that the DTSS integrators cause. In order to get the result of system execution time, we experimented with Windows NT machines connected via a 10 Base T Ethernet network which has less network delay than that of WAN. In experimenting in WAN, we expect lowering the system execution time by reducing of network bandwidth. The reduction of the system execution time between the predictive quantization system and the multiplexing predictive quantization system demonstrates the effect of the reduction of network bandwidth in compensating the local computation overhead for multiplexing.

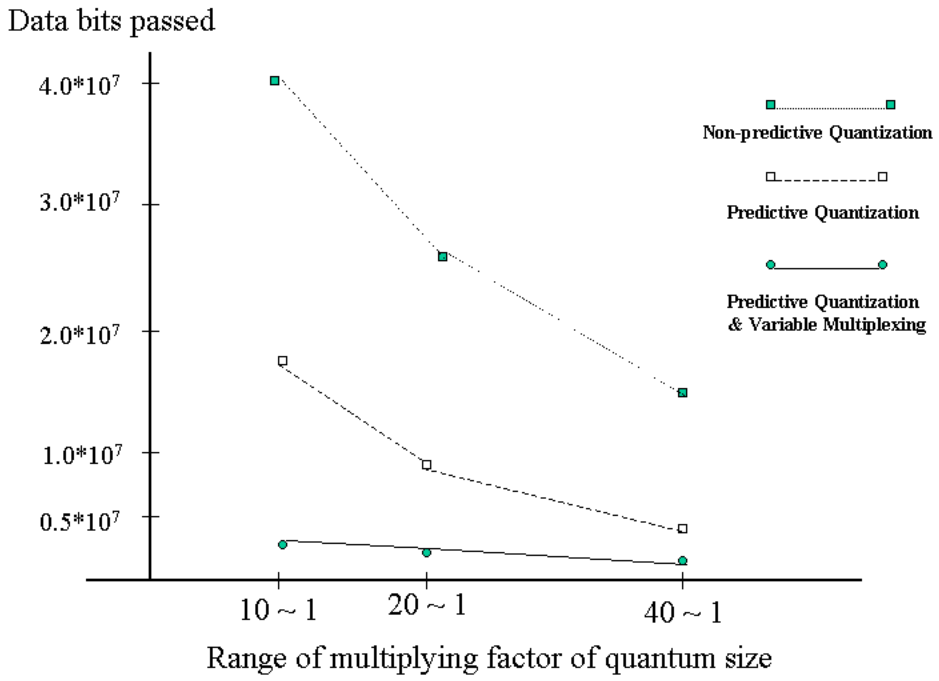


Figure 8.10 Passed data bits for varying multiplying factors in a non-predictive quantization system, a predictive quantization system, and a multiplexing predictive quantization system (Component pairs: 40)

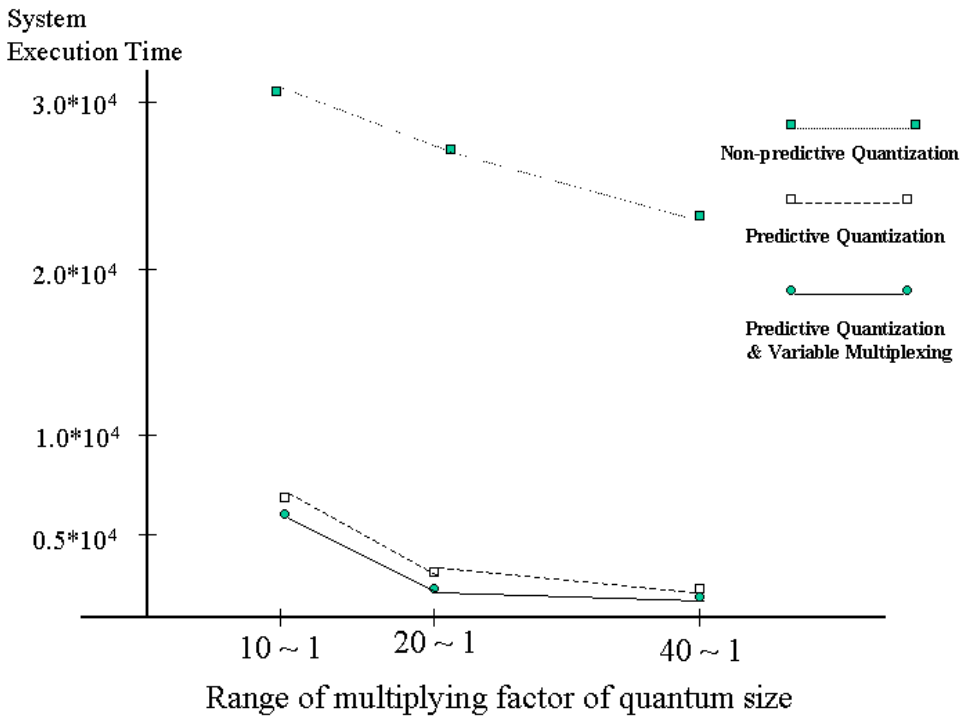


Figure 8.11 System execution time for varying multiplying factors in a non-predictive quantization system, a predictive quantization system, and a multiplexing predictive quantization system (Component pairs: 40)

Figure 8.12 shows the passed data bits of three systems (a non-predictive quantization system, a predictive quantization system, and a multiplexing predictive quantization system) for varying numbers of component pairs. As the number of component pairs increases, the passed data bits of the non-predictive and predictive quantization systems increase significantly, and the multiplexing predictive quantization system tremendously reduces the passed data bits. Compared to two non-multiplexing systems (non-predictive quantization system and predictive quantization system), the

predictive quantization system shows more reduction of the passed data bits than that of the non-predictive quanitzation system.

Figure 8.13 illustrates the variation of the system execution time of the three systems in varying the number of component pairs. In the predictive quantization and the multiplexing predictive quantization systems, as the number of component pairs increases, the system execution time increases slowly and proportionally to the passed data bits. However, the system execution time in the non-predictive quantization system increases in an exponential manner due to saturation of network transmission.

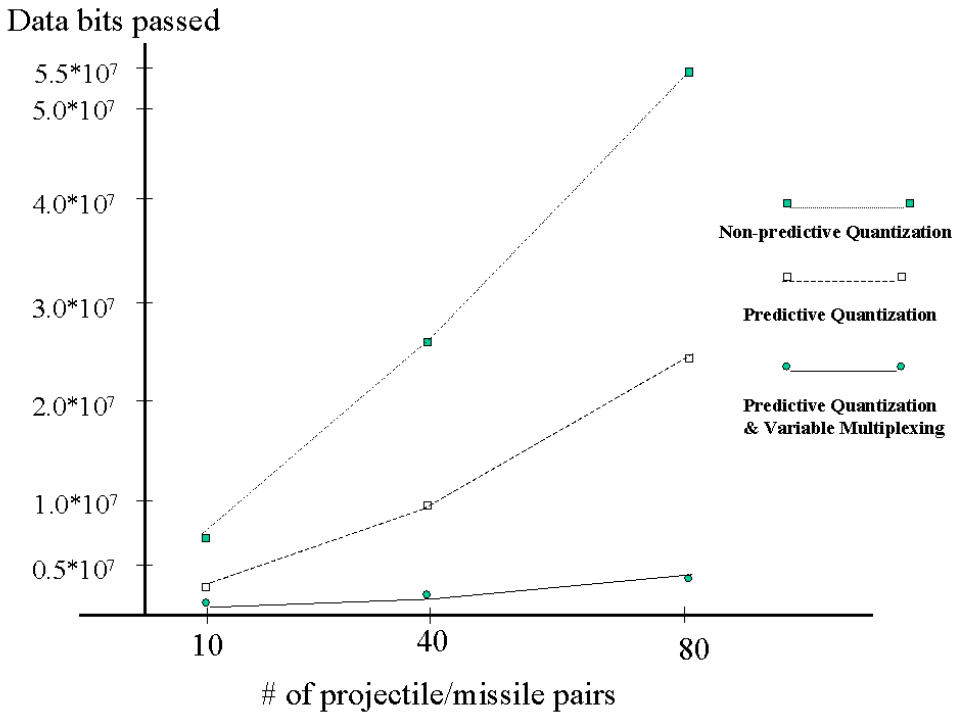


Figure 8.12 Passed data bits for varying numbers of component pairs in a non-predictive quantization system, a predictive quantization system, and a multiplexing predictive quantization system

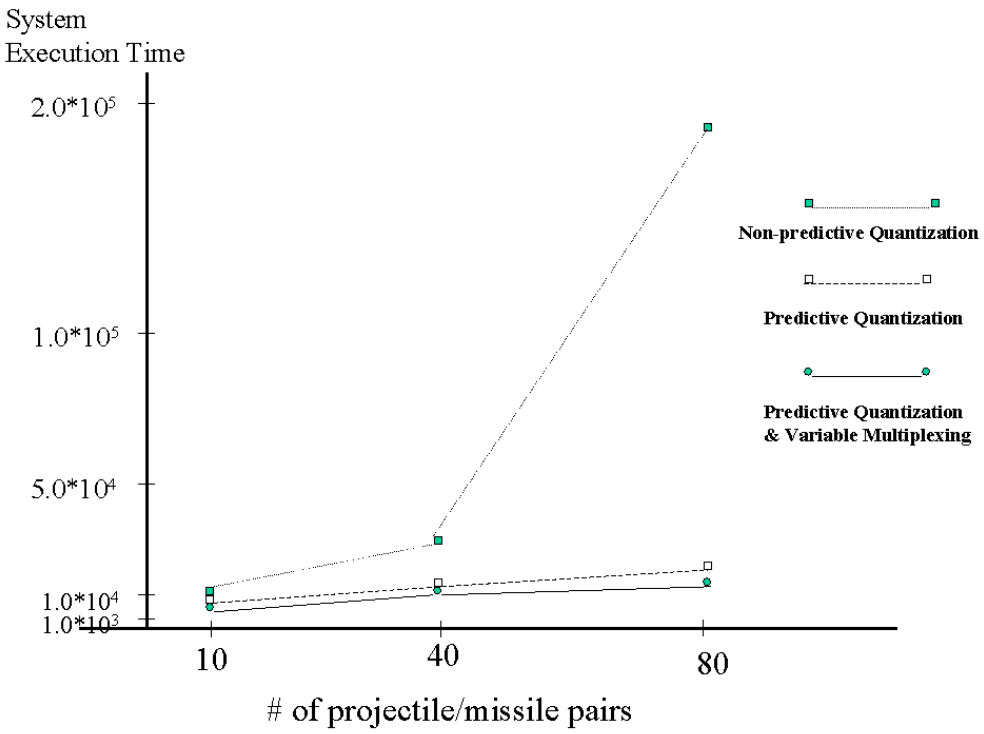


Figure 8.13 System execution time for varying numbers of component pairs in a non-predictive quantization system, a predictive quantization system and a multiplexing predictive quantization system

The results of the passed data bits and the system execution time in a non-predictive quantization system, a predictive quantization system, and a multiplexing predictive quantization system show that the multiplexing predictive quantization scheme is very effective in saving the inter-federate data and actual system execution time in a distributed simulation.

9 CONCLUSION

9.1 Contribution

9.1.1 Interest-based Quantization Scheme

Distributed simulation supports many practical application domains, such as process control and manufacturing, military command and control, transportation management, and so on, that require reliable communication linkage among multiple, geographically distributed systems. With a large number of communicating entities in such distributed systems, however, execution time of a distributed simulation sharply increases due to message exchanges increasing quadratically with the number of communicating entities. Both network data load and delay among communicating entities determine how large-scale distributed systems can be modeled and simulated in a reasonable execution time. Under limited communication resources, reducing message traffic among communicating entities is an approach to increase the scalable execution of large-scale distributed simulations.

We investigated message traffic reduction schemes, such as quantization and interest management, DDM of HLA, that have been proposed for reliable distributed simulation within reasonable execution time. Each message traffic reduction scheme requires understanding of the semantic and dynamic characteristics of the application to

tune their parameters for effective filtering with acceptable error. The interest-based quantization scheme that we proposed in this dissertation, was established by combining the quantization scheme and the interest management scheme. This scheme allows for the stratification of the degree of interest for a communication-specified attribute and thereby controls the update exchange frequency of the attribute based on the time-varying distance between communicating entities in distributed simulations. The distance in any suitable space, which is not just physical space, controls the size of the quantum governing communication of a specified attribute. In contrast, the HLA DDM works the only all-or-none interest scheme underlying the HLA routing space. In this sense, the interest-based quantization scheme can be viewed as a generalization of the all-or-none interest scheme of HLA DDM.

To support the scalability improvement of the interest-based quantization scheme, we presented two approaches: the global space manager approach and the local space manager approach. With these two approaches, the workload of agents is efficiently balanced and concurrently processed in distributed processors. Therefore, the systems that employed these two approaches demonstrated a greater performance in terms of saving system communication and computation time than the system that did not use any space manager operation. When the two approaches were compared, the local space manager approach was shown to reduce system communication and computation more than the global space manager approach. This is because, unlike the global space manager approach, the local space manager approach reduces the amount of communication overhead and solves the problem of computation bottleneck. The

analytical and empirical results from the pursuer/evader example in chapter 3 and chapter 7 demonstrated the effectiveness of the interest-based quantization scheme in reducing both message traffic and overall simulation execution time. Those results include the inevitable presence of communication and computation overheads for monitoring the communication condition for a specified attribute among communicating entities and for filtering the messages among communicating entities.

As a means to discover more efficient approaches of the interest-based quantization, we investigated the interest-based predictive quantization and the interest-based multiplexing predictive quantization approaches, applied those approaches to the projectile/missile application with realistic three dimensional dynamics, and analyzed the network bandwidth requirement for those approaches. In the interest-based predictive quantization approach, the predictive quantization's advantage for reducing both the number of messages and their size is added to the interest-based quantization. As a result, the approach greatly reduced network bandwidth within a reasonably small error.

For simulation with a large number of projectile/missile pairs, we applied the multiplexing approach to the interest-based predictive quantization. In order to compensate for the disadvantage of the fixed multiplexing approach at low active components, the variable multiplexing approach was discussed in this dissertation, and the effectiveness for reduction of the network bandwidth requirement of both fixed and variable multiplexing approaches was analyzed in varying message dimensions and simulated component pairs. In experimenting with the projectile/missile application, we investigated the variation of a (ratio of active component) and the effectiveness of both

fixed and variable multiplexing in varying quantum sizes and time granules. This research on effectiveness of both fixed and variable multiplexing is an improvement over the previous multiplexing approach, in which only fixed multiplexing is discussed with a fixed time granule and a fixed *a* (ratio of active component: *a*=1) [46, 57]. The analytical and experimental results from projectile/missile application in chapter 8 showed that the multiplexing predictive quantization scheme was very effective in saving the inter-federate data transmission and actual system execution time in a distributed simulation.

9.1.2 DEVS/GDDM Environment

Both the quantization scheme and the interest management scheme are very effective message traffic reduction schemes, especially in a large-scale distributed simulation. The DEVS/GDDM environment, provided in this dissertation, uses the interest-based quantization scheme to take advantage of both schemes, so that the DEVS/GDDM environment gives greater promise for simulation performance. Also, since the DEVS/GDDM environment compensates for the disadvantages of DDM of HLA mentioned earlier, the DEVS/GDDM environment points in a good direction for modifying DDM of HLA as a means to obtain further message traffic reduction.

The DEVS/GDDM environment supports a variety of message traffic reduction methods (non-predictive interest-based quantization, predictive interest-based quantization, and multiplexing interest-based quantization). Thus, a simulation designer

can choose from the message reduction schemes provided by DEVS/GDDM environment according to the need of each different application.

The DEVS/GDDM environment is an HLA-compliant modeling and simulation environment. While the HLA-Interface layer supports the interoperation at the simulation level, the DEVS/GDDM layer supports the modeling level features inherited from DEVS, which has a generic dynamic system formalism with a well defined concept of modularity and coupling of components. The high level modeling paradigm based on the DEVS formalism reduces the level of complexity for a model designer to construct models in a hierarchical modular fashion and improves the maintenance, reusability, and modifiability of models. The DEVS/GDDM environment is supported by the four libraries; container, DEVS, DEVS/HLA-Interface, and HLA-Interface. Through the container library, an object can be stored, retrieved, and organized. The DEVS library provides methods for the DEVS formalism. The DEVS/HLA-Interface layer supports interface methods between the DEVS and the HLA-Interface layers. The HLA-Interface layer supplies the simulation-friendly methods, which encapsulates all the complex details of HLA connectivity, so that the DEVS/GDDM environment provides the ease and effectiveness for modeling to a model developer. In general, to work the HLA/RTI based distributed simulation, the model developer has to know and use the HLA/RTI functions. In DEVS/GDDM environment, a model developer does not have to know the HLA/RTI functions and only develops the DEVS models.

As we have discussed, the DEVS/GDDM modeling and simulation environment employs:

- Sound system theories based on the DEVS formalism
- Highly efficient message traffic reduction scheme called the interest based quantization scheme
- Reliable distributed simulation with reasonably small error
- Flexibility for a simulation designer to make a choice out of methods supported by the interest-based quantization scheme according to each specific application
- Ease and effectiveness for modeling from the hierarchical and modular object-oriented technology and high level modeling paradigm
- Friendly user interface in which a user develops only DEVS models
- Encapsulation mechanism to hide all the complex details of HLA connectivity from a simulation designer in the HLA-Interface layer
- Highly reliable interoperation facility among federates using HLA/RTI

9.2 Future Work

9.2.1 Extension of DEVS/GDDM environment for a Non-Paired Application

In future research, the multiplexing predictive quantization scheme of the DEVS/GDDM environment can be extended for a non-paired application, in which an agent broadcasts its message to all other agents in the same federate or in other federates. In this section, we will discuss a prototype of an extended multiplexing predictive

quantization scheme in a non-paired application by extending the message's data format and the space manager.

9.2.1.1 Extension of message's data format

In a non-paired application, multiple quantum sizes are assigned to the multiple pairs between a sender and multiple receivers. The sender outputs each message which has each data value quantized with a particular quantum size. Therefore, in order to perform an extended multiplexing predictive quantization, a message's data format should be extended to include a Quantum ID to represent quantum size. For example, if the message includes three dimensional position values, the message's data format is:

(ID, QID, x, y, z)

where ID is an agent identification number and QID is a quantum size identification number.

QID presents what quantum size the data value is quantized with. Using QID, the extended multiplexing predictive quantization scheme routes the message (which has a certain QID) to the exact receiver specified by the quantum sizes of sender/receiver pairs. Figure 9.1 illustrates the message passing between two federates (federate1 and federate2) in a non-paired application using the extended message's data format (ID, QID,

x, y, z). The message passing is performed by the sender multiplexer and the receiver de-multiplexer.

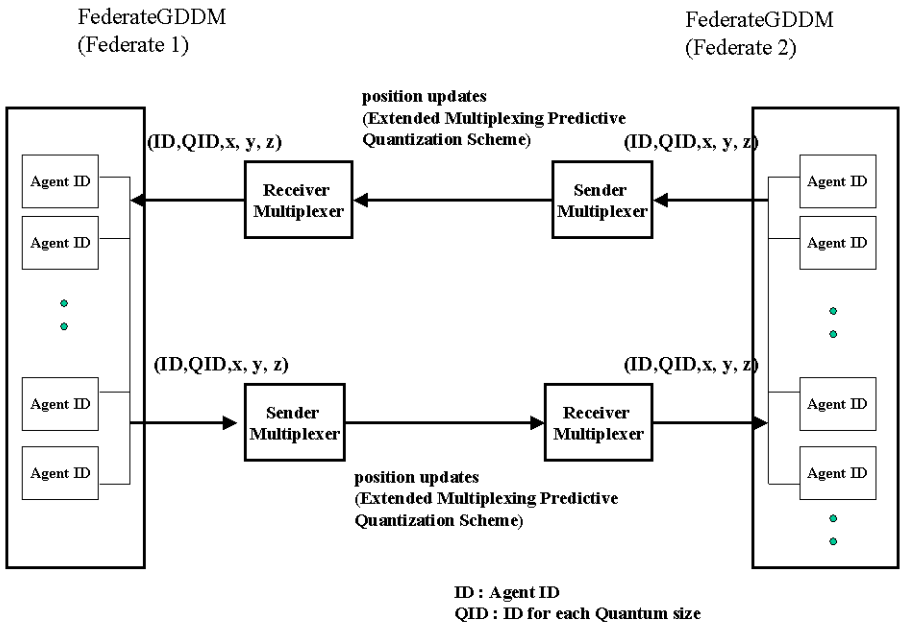


Figure 9.1 Message passing between two federates in a Non-Paired Application

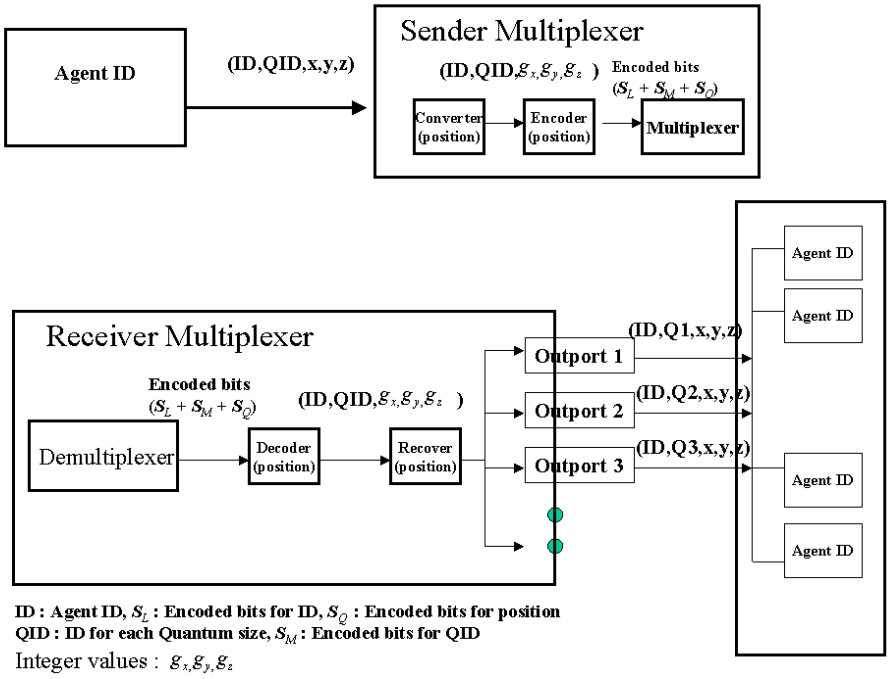


Figure 9.2 Detailed Description of message passing in a Non-Paired Application

Figure 9.2 illustrates a detailed description of the message passing in a non-paired application. An agent sends its messages including double precision values (ID, QID, x, y, z) quantized with different quantum sizes to the sender multiplexer. The sender multiplexer has three sub-components (converter, encoder, and multiplexer) to pass a multiplexed message to the receiver federate. The converter changes the double precision values (ID, QID, x, y, z) to integer values (ID, QID, g_x,g_y,g_z), where ??1,0,1?? g_x,g_y,g_z , and the encoder changes the integer position values to the encoded five bits ($S_Q = 5 > \log_2 3^3$; for three dimensions). The encoder also changes the agent ID to properly encoded bits (S_L), and changes the Quantum ID to properly encoded

bits (S_M). For example, if the number of quantum sizes is five, three bits ($3 > \log_2 5$) are needed to represent a Quantum ID. The multiplexer receives the encoded bits (S_L, S_M, and S_Q), creates a large, multiplexed message, and sends it to the receiver federate. The receiver de-multiplexer has three sub-components (de-multiplexer, decoder and recover) to recover the original double precision values (ID, QID, x, y, z) from the multiplexed message. The de-multiplexer separates from the multiplexed message to each encoded bits (S_L, S_M, and S_Q). The decoder changes the encoded bits to the integer values (ID, QID, g_x, g_y, g_z), and the recover component converts these integer values to the original double precision values (ID, QID, x, y, z). The receiver de-multiplexer outputs the messages, including double precision values (ID, QID, x, y, z), through different output ports related to each Quantum ID.

9.2.1.2 Extended Space Manager

To perform the multiplexing predictive quantization scheme in a non-paired application, the role of the space manager needs to be extended. The space manger must make the exact connections between the output ports (related to each Quantum ID) of the receiver de-multiplexer and the input ports of agents in receiver federate. As Figure 9.2 illustrates, an agent outputs each message which contains each data value quantized with each quantum size and each message includes the Quantum ID for presenting its quantum size. Each output port of the receiver de-multiplexer is assigned to each Quantum ID and a message (which has a certain Quantum ID) should be outputted through the exact

output port assigned to the Quantum ID of the message. The space manager has to know the quantum sizes pertaining between a sender agent and multiple receiver agents and control the connections between each output port (assigned to each quantum size) of the receiver de-multiplexer and the input ports of multiple receiver agents.

Two approaches related to scalability of space manager operation and their comparison with results were discussed in chapter 3 and chapter 7. These approaches are based on a centralized global space manager and distributed schemes based on local space managers. For a non-paired application, we prefer the distributed approach assigning each local space manager in a different federate due to the advantages over the centralized approach with the global space manager. The distributed approach reduces the computation load because each local space manager controls the connections only for those agents within its own federate and the connection computation (to know quantum sizes) is divided up and those pieces are assigned to local space managers for concurrent processing.

9.2.2 Extension of DEVS/GDDM environment for real-time distributed simulation

This DEVS/GDDM environment and the interest-based quantization scheme can be applied to a real-time distributed simulation. This DEVS/GDDM environment can help to overcome the time constraint requirements in real-time distributed simulation involving humans and/or hardware in the loop. Also, the theoretical and empirical results we obtained for global and local versions of the space manager will be tested for

scalability in real-time distributed simulation. The point of future work is how to extend the DEVS/GDDM environment to a real-time distributed simulation and to real-time execution infrastructures.

APPENDIX A. SMOOTHER MODEL

The smoother model is developed to connect a DEVS component and a DTSS component. The behavior of the smoother model is based on the coupling of DEVS to DTSS, which indicates a coupling of an output of a DEVS component to an input of a DTSS component. In this coupling, we define the output event value of the DEVS component occurred prior to the state transition of the DTSS component as the input event at the next state transition of the DTSS component.

In this dissertation, a DEVS component generates three dimensional output events with each variable time advance. A DTSS component receives the three dimensional input events from the DEVS component and operates in the same time. The smoother model is a connector between the DEVS component and DTSS component and it plays a role to reduce error caused by different state time advances between DEVS and DTSS.

A Parallel DEVS representation [27] for the smoother model follows. This representation is provided using the "Parallel DEVS with Ports" formalism.

$DEVS_{smoother}$ = < X, Y, S, $?_{int}$, $?_{ext}$, $?_{conf}$, ?, ta >, where

InPorts = {"in1", "in2", "in3"}
OutPorts = {"out"}
X = {(in_i , x_i) | i = 1,2,3, x_i ? R}
Y = { ("out", (x1,x2,x3))
S = { "active", "passive" x Y x R_0^+ }
?
$?_{ext}$(("active",(x1,x2,x3), ?), *e*,("in_i",x_i')) = ("active",($\overline{x1, x2, x3}$),? -e),? ?
?

?

?

where ($\overline{x1, x2, x3}$) = (x1',x2,x3) if i =1
= (x1,x2',x3) if i =2
= (x1,x2,x3') if i =3?

?

?

? $_{int}$("passive",(x1,x2,x3), ?) = ("passive",(x1,x2,x3), ?)
? $_{int}$("active",(x1,x2,x3), ?) = ("active",(x1,x2,x3), ? *t*)

where ? *t* = time step
?
? $_{conf}$(s,ta(s),x) = ? $_{ext}$(? $_{int}$(s),0,x)
?
? ("active",(x1,x2,x3), ?) = ("out", (x1,x2,x3))

ta("active",(x1,x2,x3), ?) = ?
ta("passive",(x1,x2,x3), ?) = ?

Figure Appendix A.1 illustrates the discrete time segment trajectory given by the parallel DEVS representation for the smoother model.

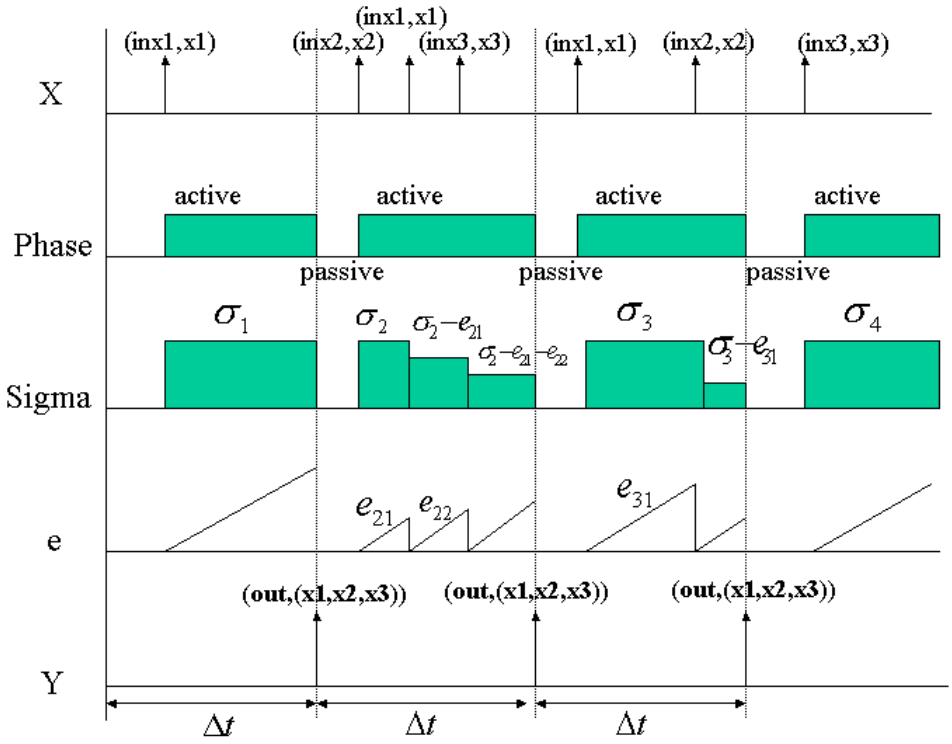


Figure Appendix A.1 Discrete Event Time Segments for Smoother model

APPENDIX B. PROJECTILE and EARTH MODELS

B.1 Projectile Model

The projectile model is a (relatively) simple model of a sphere of uniform density following a ballistic trajectory. It begins at some initial position with an initial velocity and falls until it hits the surface of the Earth, at which point it stops.

B1.1 Formal Description

Inputs

$\mathbf{a}_g(\mathbf{x})$ – Acceleration due to gravity at our position
$\mathbf{v}_e(\mathbf{x})$ – Inertial velocity of the atmosphere at our position
$p(\mathbf{x})$ – Atmospheric density at our current position

State variables and outputs

x – Position
v – Velocity
a – Acceleration

Parameters

C – Coefficient of drag
m – mass
A – cross sectional area

Behavior specification

$\mathbf{x}' = \mathbf{v}$
$\mathbf{v}' = \mathbf{a}$
$\mathbf{r} = \mathbf{v} - \mathbf{v}_a(\mathbf{x})$
$\mathbf{a} = -0.5\ C\ A\ p(\mathbf{x})\ |\mathbf{r}|\ \mathbf{r} / m + \mathbf{a}_g(\mathbf{x})$

B.1.2 Program code for Projectile model

Projectile Model

```
public class Projectile {

/// Position (m)
private          VectorD x;
/// Integrator for variable x
private          trap_integ xinteg;
/// Velocity (m/s)
 private                  VectorD v;
/// Integrator for velocity
private          trap_integ vinteg;
/// Acceleration (m/s^2)
private          VectorD a;

/// Velocity of the atmosphere (m/s)
private          VectorD va;
/// Acceleration due to gravity (m/s^2)
private          VectorD ag;
/// Density of the atmosphere (kg/m^3)
private          double p;

/// Mass of the projectile (kilograms)
private          double m;
/// Drag coefficient (dimensionless)
private          double C;
/// Area of cross section in direction of flight (m^2)
private          double A;

/**
Create a projectile with the parameters m (mass in kilgrams), C, (drag coefficient),
A (cross sectional area in meters)
and initial conditions x0 (initial position in meters) and v0 (initial velocity in meters/s)
*/

public Projectile (double m, double C, double A, VectorD x0,  VectorD v0) {
                         this.m = m;
                         this.C = C;
                         this.A = A;
                         this.x = x0;
                         this.v = v0;
```

```
                        p = 0.0;
     a = new VectorD(0.001,0,0);
    va = new VectorD(0.,0,0);
    ag = new VectorD(0,0,-9.80665);
    xinteg = new trap_integ(x0);
    vinteg = new trap_integ(v0);
}

// Compute a single state change with time advance dt.
// Convention is internal transition, then external transition.
/// Compute the next state using timestep dt (i.e. state at time t + dt)

public void delta (double dt) {
// Compute the position and velocity after dt time units based on acceleration
// up to current time

x.plus( xinteg.integ (v, dt));
v.plus( vinteg.integ (a, dt));

// Compute acceleration for next state transition

VectorD r = new VectorD(v);
 r.minus(va);
//a = (-0.5 * C * A * p * r.mag () * r / m) + ag;
 r.times(-0.5 * C * A * p * r.mag ());
 r.divideBy(m);
 r.plus(ag);
 a = r;
}

/// Set the atmospheric density (apply the p(x) input).  Units are kg/m^3
public          void setDensity (double p) { this.p = p; }
/// Set the acceleration due to gravity (apply the ag(x) input).  Units are m/s^2
public          void setAcclGrav ( VectorD ag) { this.ag = ag; }
/// Set the atmospheric velocity (apply the va(x) input).  Units are m/s
public  void setAtmoTangVel ( VectorD va) { this.va = va; }
/// Get the position of the projectile.  Units are meters
public           VectorD getPos ()  { return x; }
/// Get the velocity of the projectile.  Units are m/s
public          VectorD getVel ()  { return v; }
/// Get the acceleration of the projectile.  Units are m/s^2
public   VectorD getAccl ()  { return a; }

}
```

```
public class trap_integ{
protected VectorD xlast;

 /// Create an integrator whose intial state is (0,0,0)
public trap_integ(){ }

/// Create and set the initial state
public trap_integ (VectorD x0) { this.xlast = x0; }

/// Set the integrators initial state

public void init (VectorD x0) { this.xlast = x0; }

// Single step trapezoidal integration

private double trap (double x, double y, double dt) {
return (dt * (Math.min (x, y) + Math.abs (x - y) / 2.0));
}

public VectorD integ (VectorD x, double dt){
VectorD results = new VectorD();
results.x = trap (xlast.x, x.x, dt);
results.y = trap (xlast.y, x.y, dt);
results.z = trap (xlast.z, x.z, dt);
xlast = x;
return results;

}
}
```

Acceleration Model

```
public class instantAccelFn extends Atomic{

/// Velocity (m/s)
private VectorD velocity;
/// Velocity of the atmosphere (m/s)
private VectorD AtmoTangVel;
/// Acceleration due to gravity (m/s^2)
private VectorD accelGravity;
/// Density of the atmosphere (kg/m^3)
private double density;
/// Mass of the projectile (kilograms)
```

```
private double m;
/// Drag coefficient (dimensionless)
private double C;
/// Area of cross section in direction of flight (m^2)
private double A;
// stop projectile when hits ground
private boolean hitGround = false;

public instantAccelFn(String name, double mm, double CC, double AA){

super(name);
inports.add("velocity");
inports.add("Gravity");
inports.add("AtmoDensity");
inports.add("AtmoTangVel");
inports.add("hitGround");

m = mm;
C = CC;
A = AA;
initialize();
}


public void initialize() {
accelGravity = new VectorD(0,0,0);
AtmoTangVel = new VectorD(0,0,0);
super.initialize();
}

public void  deltext(double e, message x) {

Continue(e);
for (int i=0; i< x.get_length();i++)
   if (message_on_port(x,"velocity",i)) {
      entity ent = x.get_val_on_port("velocity",i);
      velocity  = (VectorD)ent;
      sigma = 0;
   }
   else if (message_on_port(x,"Gravity",i)) {
      entity ent = x.get_val_on_port("Gravity",i);
      accelGravity  = (VectorD)ent;
      sigma = 0;
   }
```

```
   else if (message_on_port(x,"AtmoDensity",i)) {
      entity ent = x.get_val_on_port("AtmoDensity",i);
      density  = ((doubleEnt)ent).getv();
      sigma = 0;

   }
   else if (message_on_port(x,"AtmoTangVel",i)) {
      entity ent = x.get_val_on_port("AtmoTangVel",i);
      AtmoTangVel  = (VectorD)ent;
      sigma = 0;

   }
   else if (message_on_port(x,"hitGround",i)) {
      hitGround = true;
      sigma = 0;
   }

}

public void  deltint() {

phase = "hitGround: " + hitGround;
sigma = INFINITY;

}

public VectorD acceleration() { /// Acceleration (m/s^2)

if (hitGround) return new VectorD(0,0,0);
VectorD r = new VectorD(velocity);
r.minus(AtmoTangVel);
//a = (-0.5 * C * A * p * r.mag () * r / m) + ag;
r.times(-0.5 * C * A * density * r.mag ());
r.divideBy(m);
r.plus(accelGravity);
return r;

}

public message out( ) {
message m = new message();
m.add(make_content("out", acceleration()));
return m;
}
```

```
}
```

DEVS Predictive Integrator for Velocity and Position Models

```
public class DEVSGenInt extends Atomic{

protected double old_inp,inp,quantum,position,initialPosition;
protected int lowerBound, nextLowerBound, input_nextLowerBound;
protected boolean positiveRestriction = false;


public DEVSGenInt(String  name, double Quantum, double Position){
super(name);

inports.add("in");
inports.add("setQuantum");
inports.add("stop");

quantum = Quantum;
initialPosition = Position;
initialize();
}

public void initialize(){

     inp = 1;
     position = initialPosition;
     super.initialize();
     lowerBound = (int)Math.floor(position/quantum);
     nextLowerBound = lowerBound;
     hold_in("doReset",0.01);
}

public int signOf(double x){
    if (x == 0) return 0;
    else if (x > 0) return 1;
    else return -1;
}

public void setInp(double buf){
    inp = buf;
}
```

```
public void timeAdvance(double diff){
   sigma = Math.abs(diff/inp);
}

public void update(double e){

   position = position + e *inp;

   if ((inp > 0 && position > nextLowerBound*quantum)||
          (inp < 0 && position < nextLowerBound*quantum))
          {
            position = nextLowerBound*quantum;
//   System.out.println(get_name() + " INPUT VIOLATION");
          }
}

public void computeIntNextPosition(){

     lowerBound = nextLowerBound;
     nextLowerBound = lowerBound + signOf(inp);
     timeAdvance(signOf(inp)*quantum);

     if (inp ==  0)
        System.out.println(get_name()+ "ERROR: input can't be zero");

}

public void computeExtNextPosition(){

     if (inp ==  0)
         sigma = Double.POSITIVE_INFINITY;
     else{
       if (inp  < 0){
          if (nextLowerBound > lowerBound) {
             nextLowerBound = lowerBound;
          }
          if (nextLowerBound < lowerBound) {
             nextLowerBound = lowerBound-1;
          }

       }

       else { //if (inp > 0)
```

```
          if (nextLowerBound > lowerBound) {
              nextLowerBound = lowerBound+1;
          }
          if (nextLowerBound < lowerBound) {
              nextLowerBound = lowerBound;
          }

        }

  phase = "" + position;

timeAdvance(nextLowerBound*quantum - position);

      }
}

public void deltcon(double e,message x)
{
 deltint();
 deltext(0,x);
}

public void  deltext(double e,message   x)
{
  Continue(e);
 for (int i=0; i< x.get_length();i++)
  if (message_on_port(x,"in",i)){
       entity ent = x.get_val_on_port("in",i);
       phase = "" + position;
       doubleEnt f = (doubleEnt)ent;
       setInp(f.getv() );
       update(e);
       computeExtNextPosition();


    }


 for (int i=0; i< x.get_length();i++)
  if (message_on_port(x,"setQuantum",i)){
       entity ent = x.get_val_on_port("setQuantum",i);
       quantum = ((doubleEnt)ent).getv();

       lowerBound = (int)Math.floor(position/quantum);
```

```
      nextLowerBound = lowerBound;

      lowerBound = nextLowerBound;
      nextLowerBound = lowerBound + signOf(inp);

      update(e);
      computeExtNextPosition();

  }
  else if (message_on_port(x,"stop",i))
     passivate();

}

public void  deltint( )
{
position = nextLowerBound*quantum;
phase = "" + position;
computeIntNextPosition();
}

public message    out( )
{
int N_Quantum = nextLowerBound - lowerBound;

message    m = new message();

if (N_Quantum != 0) {
m.add(make_content("out",  new doubleEnt(nextLowerBound*quantum)));
}

return m;
}

}
```

B.2 Earth Model

The earth model consists of three sub-models; an atmospheric model, a gravity model, and a motion model. The atmospheric model uses lookup tables based on the

1976 atmospheric modeling standard. The gravity model is a simple force model based on a spherical Earth. The entire system rotates (i.e. objects attached to the earth and the atmosphere have a rotational velocity about (0,0,0)).

B.2.1 Formal Description

Inputs

x – A point at which to compute values for the system outputs.

Outputs

p(**x**) – Atmospheric density at position **x**
$\mathbf{v}_e(\mathbf{x})$ – Rotational velocity of the earth at a position **x**
$\mathbf{v}_a(\mathbf{x})$ – Rotation velocity of the atmosphere at a position **x**
$\mathbf{a}_g(\mathbf{x})$ – Acceleration due to gravity at position **x**

Parameters

R_e – mean equatorial radius
w – rotational velocity
m – gravitational parameter

Behavior specification

$\mathbf{a}_g(\mathbf{x}) = -m / |\mathbf{x}|^3$

$\mathbf{v} = \mathbf{w} \times \mathbf{x}$

where **w** is a vector (0, 0, w) and **x** is a position in the plane of the equator. That is, **x** = (x, y, 0). In this simple model, we let $\mathbf{v}_e(\mathbf{x}) = \mathbf{v}_a(\mathbf{x})$. That is, the inertial velocity of the atmosphere at a position **x** is equal to the inertial velocity of a point attached to the earth at position **x**.

B.2.2 Program code for Earth model

```
public class Earth {

public static final double mu = 3.986012E14;
public static final double w = 7.292115856E-5;
public static final double R = 6378145;

/// Create the Earth ()
public  Earth () { }

/// Compute the atmospheric density (kg/m^3) at position x (m)
public static double getAtmoDensity (VectorD x) {
double rho = atmo76.computeRho (x.mag() - R);
return rho;
}

//// Compute the acceleration induced by gravity (m/s^2) at position x (m)
public static VectorD getGravity ( VectorD x) {

// (-mu * x / pow (x.mag (), 3));
VectorD X = new VectorD(x);
X.times(-mu);
X.divideBy(Math.pow (x.mag (), 3));
return X;
}

/// Compute the tangential velocity (m/s) at position x
public static VectorD getEarthTangVel (VectorD x) {
VectorD X= new VectorD(x.x, x.y, 0.0);
VectorD W = new VectorD(0.0, 0.0, w);
return W.cross(X);
}

/// Compute the tangential velocity (m/s) of the atmosphere at position x (m)
public static VectorD getAtmoTangVel ( VectorD x){
return getEarthTangVel (x);
}
}

/**
This class computes properties of the 1976 U.S. Standard Atmosphere
*/
```

```
public class atmo76 {

// sea-level mean molecular weight of air (kg/mol)
public static final double m0 = 28.9644;
// acceleration due to gravity (m/s^2)
public static final double g0 = 9.80665;
// radius of the Earth (m)
public static final double r0 = 6356766.0;
// gas constant (N m K / kmol)
public static final double rstar = 8314.32;

// This array of constants was not documented in the original listing
public static final double pb [] =
   { 101325.0, 22632.06468902076, 5474.889006665066,
     868.0187719024579, 110.9063215028894, 66.93888345713616,
     3.956421275130599, 0.3733836885098447 };

// This array of constants was not documented in the original listing
public static final double tmb [] =
   { 288.15, 216.65, 216.65, 228.65, 270.65, 270.65, 214.65, 186.946 };

// This array of constants was not documented in the original listing
public static final double altb [] =
   { 0.0, 11000.0, 20000.0, 32000.0, 47000.0, 51000.0, 71000.0,
     84852.0, 91000.0, 110000.0, 120000.0, 1000000.0 };

// This array of constants was not documented in the original listing
public static final double lmb [ ] =
   { -0.0065, 0.0, 0.001, 0.0028, 0.0, -0.0028, -0.002 };

/**
Compute the atmospheric density at z meters above the Earth's surface.
i.e. If (0,0,0) is the center of the Earth in your coordinate system,
the altitude above the Earth's surface is (z - r0) meters.
Returns density (kg/m^3).
*/
public static double computeRho (double alt) {

// atmospheric density
double rho = 0.0;

// geopotential altitude above sea level (m)
double h = r0 * alt / (r0 + alt);
```

```
int ib = 0;

while ((h > altb[ib+1]) && (ib < 8)) ib++;

// temporary values
double delalt = 0.0;
// molecular-scale temperature (K)
double tm = 0.0;
// atmospheric pressure
double p = 0.0;

if (h <= altb[7]) {
   delalt = Math.max (-5.0, h - altb[ib]);
   tm = tmb[ib] + lmb[ib]*delalt;
   if (lmb[ib] == 0.0) p = pb[ib] * Math.exp(-g0*m0*delalt/rstar/tmb[ib]);
   else p = pb[ib] * Math.pow(tmb[ib]/(tmb[ib] + lmb[ib]*delalt),g0*m0/rstar/lmb[ib]);
   rho = p*m0/rstar/tm;
   }
else if (alt <= altb[8]) {
   rho = 2.860e-6;
   }
else if (alt <= altb[9]) {
   rho = 9.708e-8;
   }
else if (alt <= altb[10]) {
   rho = 2.222e-8;
   }
else if (alt <= altb[11]) {
   rho = 3.561e-15;
   }
else if (alt > altb[11]) {
   rho = 0.0;
   }

return rho;

} // end of function computeRho

} // end of class atmo76
```

REFERENCES

[1] Bernard P. Zeigler, "Theory of Modeling and Simulation," New York: John Wiley, 1976.

[2] Jeffrey Bradshaw, Ed., Software Agents, AAAI Press, Menlo Park, CA, 1997.

[3] N. R. Jennings and M. Wooldridge, "Applications of intelligent agents," in Agent Technology: foundations, Application, Markets, N. R. Jennings and M. Wooldridge, Eds., pp. 3-28. Springer-Verlag, 1998.

[4] John Anderson, "A generic distributed simulation system for intelligent agent design and evaluation," in Proceedings of the Tenth Conference on AI, Simulation and Planning, AIS-2000, Hessam S. Sarjoughian, Francois E. Cellier, Michael M. Marefat, and Jerzy W. Rozenblit, Eds. March 2000, pp. 36-44, Society for computer Simulation International.

[5] Paul R. Cohen, Michael L. Greenberg, David M. Hart, and Adele E. Howe, "Trial by fire: Understanding the design requirements for agents in complex environment,: Al Magazine, vol. 10, no. 3, pp. 32-48, Fall 1989.

[6] Martha E. Pollack and Marc Ringuette, "Introducing the tile-world: experimentally evaluating agent architectures," in Proceedings of the Ninth National Conference on Artificial Intelligence, 1990, pp. 183-189.

[7] Thomas A. Montgomery and Edmund H. Durfee, "Using MICE to study intelligent dynamic coordination," in Proceedings of the Second International Conference on Tools for Artificial Intelligence. 1990, pp. 438-444, IEEE

[8] S. M. Atkin, D. L. Westbrook, P. R. Cohen, and G. D. Jorstad., "AFS and HAC: Domain general agent simulation and control.," in Software Tools for Developing Agents: Papers from the 1998 Workshop. 1998, number Technical Report WS-98-10, pp. 89-96, AAAI Press

[9] Defense, D.o., *High Level Architecture Interface Specification, Version 1.0*, Defense Modeling and Simulation Organization, 1996, http://msis.dmso.mil

[10] Defense, D.o., *Draft Standard For Modeling and Simulation (M&S) High Level Architecture(HLA) - Federate Interface Specification, Draft 1*, . 1998

[11] "CORBA Overview" http://ww.infosys.tuwien.ac.at/Research/Corba/OMG/arch2.htm#446864

[12] OMG, "Comparing ActiveX and CORBA/IIOP," http://www.omg.org/news/activex.html

[13] Ecoscope, "Is DCOM Truly The Object Of Middleware's Desire?," http://techweb.cmp.com/nc/813/813r12.html

[14] Zeigler, B.P., *DEVS Theory of Quantization*, . 1998, DARPA Contract N6133997K-0007: ECE Dept., UA, Tucson, AZ.

[15] Zeigler, B.P. and J.S. Lee. *Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simulation Environment*. in *Enabling Technology for Simulation Science(II), SPIE AeoroSense 98*. 1998. Orlando, FL.

[16] Software Technology, "Middleware" http://www.sei.cmu.edu/technology/str/descriptions/middleware_body.html

[17] Vogel, "WWW and Java Threat or Challenge to CORBA?" http://www.dstc.edu.cu/AU/staff/andres-vogel/papers/mws96/paper.html

[18] "Choosing between CORBA and DCOM," http://www.cerfnet.com/~mpcline/Corba-FAQ/corba-and-dcom.html

[19] Magic, "DCOM and CORBA," http://www.magic-sw.be/wite4.htm

[20] T. Brando, "Comparing DCE and CORBA," http://www.mitre.org/research/domis/reports/DCEvCORBA.html

[21] Bernard P. Zeigler, "Multifacetted Modelling and Discrete Event Simulation," London: Academic Press, 1984.

[22] Bernard P. Zeigler, "Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems," San Diego, CA: Academic press, 1990.

[23] Yoonkeon Moon, "High Performance Simulation Based Optimization Environment: Modeling Spatially Distributed Large Scale Ecosystems," Ph.D. Dissertation, The University of Arizona, Tucson, Arizona, 1996.

[24] Bernard P. Zeigler, Y. Moon, D. Kim, and J.G. Kim, "C++ DEVS: A High Performance Modeling and Simulation Environment," 29th Hawaii International Conference on System Sciences, Jan. 1996.

[25] Bernard. P. Zeigler, "The Support for Hierarchical Modular Component-based Model Construction in DEVS/HLA," in SIW. 1999. Orlando, FL.

[26] Bernard. P. Zeigler and D. Kim. *Design of High Level Modelling / High Performance Simulation Environments.* in *10th Workshop on Parallel and Distributed Simulation*. 1996. Philadelphia.

[27] Bernard. P. Zeigler, H. Praehofer, and T.G. Kim, *Theory of Modeling and Simulation*. 2 ed. 1999, New York, NY: Academic Press

[28] Lin, C. 1994a. Study on the network load in distributed interactive simulation. In Proceeding of the AIAA on Flight Simulation Technologies.

[29] Lin, C. 1994b. The performance assessment of the dead-reckoning algorithm in DIS. In Proceedings of the 10th DIS Workshop on Standards for the Interoperability of Distributed Simulation, March.

[30] BASSIOUNI, M., CHIU, M.,AND GARNSEY, M. 1993. "Real-time data filtering in the gateways of wide area aimulation networks," In 15th Interservice / Industry Training Systems Conference (I/ITSC), Dec., 891-900.

[31] BASSIOUNI, M., WILLIAMS, H., AND LOPER, M. 1991. "Intelligent filtering algorithms for real-time networked simulators," In Proceedings of IEEE Conference on Systems, Man and Cybernetics, 309-314.

[32] Katherine L. Morse, "Interest management in large scale distributed simulations," Tech. Rep. 96-127, Department of Information and Computer Science, University of California, Irvine, 1996.

[33] Katherine L. Morse, Lubomir Bic, Michael Dillencourt, and Kevin Tsai, "Multicast grouping for dynamic data distribution management," in Proceeding of the 31st Society and Computer Simulation Conference (SCSC"99), 1999.

[34] J. Saville, "Interest Management: Dynamic group multicasting using mobile java policies," in Proceedings of the 1997 Fall Simulation Interoperability Workshop, 1997, number 97F-SIW-020.

[35] A. Berrached, M. Beheshti, O. Sirisaengtaksin, and A. Dekorvin, "Alternative approaches to multicast group allocation in HLA data distribution," in Proceeings of the 1998 Spring Simulation Interoperability Workshop 1998.

[36] *High Level Architecture Run-Time Infrastructure Programmer's Guide 1.3 Version 3*, 1998 DMSO

[37] Nico Kuijpers, et al. *Applying Data Distribution Management and Ownership Management Services of the HLA Interface Specification*. in *SIW*. 1999. Orlando, FL

[38] Boukerche and A. Roy "A Dynamic Grid-Based Multicast Algorithm for Data Distribution Management" 4th IEEE Distributed Simulation and Real Time Application, 2000.

[39] Gary Tan et. al. "A Hybrid Approach to Data Distribution Management", 4th IEEE Distributed Simulation and Real Time Application, August 2000.

[40] Zeigler, B.P., *et al. Bandwidth Utilization/Fidelity Tradeoffs in Predictive Filtering*. in *SIW*. 1999. Orlando, FL

[41] Zeigler, B.P., *DEVS Theory of Quantization*, . 1998, DARPA Contract N6133997K-0007: ECE Dept., UA, Tucson, AZ.

[42] Zeigler, B.P. and J.S. Lee. *Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simulation Environment*. in *Enabling Technology for Simulation Science(II), SPIE AeoroSense 98*. 1998. Orlando, FL

[43] Bernard. P. Zeigler, Hyup Cho, J.S. Lee, Y.K. Cho and Hessam Sarjoughian, et al. *Predictive Contract Methodology and Federation Performance*. in *SIW*. 1999. Orlando, FL.

[44] Hall, S.B. and B.P. Zeigler. *Joint Measure: Distributed Simulation Issues In a Mission Effectiveness Analytic Simulator*. in *SIW*. 1999. Orlando, FL.

[45] Averill M. Law, and W. David Kelton. *Simulation Modeling and Analysis*, 1982. McGraw-Hill, Inc.

[46] Bernard P. Zeigler, Hyup J. Cho, Jeong G. Kim, Hessam Sarjoughian, and Jong S. Lee, "Quantization-based filtering in distributed simulation : experiments and analysis" in Journal of Parallel and Distributed Computing, March 2001.

[47] Bassiouni, M.A., *et al., Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation.* ACM Trans. on Model. and Comp. Sim. (TOMACS), 1997. **7**(3): p. 293-331

[48] Bassiouni, M.A., *et al., Relevance Filtering for distributed Interactive Simulation.* Journal of Computer Systems Science and Engineering, Volume 13, 1998.

[49] Logan, B., and Theodoropoulos, G. *Dynamic Interest Management in the Distributed simulation of Agent-based systems*, AI, Simulation & Planning In High Autonomy Systems, Tucson, AZ, 2000.

[50] Bernard. P. Zeigler, George Ball, Hyup Cho, and J.S. Lee, "Implementation of the DEVS Formalism over the HLA/RTI: Problems and Solutions," Simulation Interoperation Workshop(SIW), June. 1999. Orlando, FL.

[51] Bernard. P. Zeigler, George Ball, Hyup Cho, J.S. Lee, and Hessam Sarjoughian, "*The DEVS/HLA Distributed Simulation Environment And Its Support for Predictive Filtering*," DARPA Contract N6133997K-0007: ECE Dept., UA, Tucson, AZ. 1998.

[52] G. Wainer, and B.P. Zeigler. "Experimental Results of Timed Cell-DEVS Quantization." AI and Simulation, AIS 2000, Tucson, AZ.

[53] Bernard. P. Zeigler, H. Sarjoughian, and H. Praehofer, "Theory of Quantized Systems: DEVS Simulation of Perceiving Agents." J. Sys. & Cyber, Vol. 16, No. 1, 2000.

[54] Roger R. Bate, Donald D. Mueller, Jerry E. White, *Fundamentals of Astrodynamics*, Dover Publications, New York, 1971

[55] Erwin Kreyszig, *Advanced Engineering Mathematics*: *Seventh Edition*, John Wiley& Sons Inc, New York, 1993.

[56] Ernesto Kofman, Sergio Junco, "Quantized-State Systems, a DEVS Approach for Continuous System Simulation", Transactions of SCS, 2001

[57] Hyup J. Cho, "Discrete Event System Homomorphism: Design and Implementation of Quantization-Based Distributed Simulation Environment," Dissertation, University of Arizona, May 1999.

[58] S.B. Hall, S. M. Venkatesan, D.B. Wood, H. S. Sarjoughian, B.P. Zeigler, "Object Oriented HLA Interface Design for Military Simulations."

[59] Bernard P. Zeigler, *OBJECTS & SYSTEMS: Principled Design with Implementation in C++ and Java*, 1997 Springer-Verlag New York Inc.

[60] Doohwan Kim and Bernard P. Zeigler, "Efficient Implementation of Parallel Container Classes for High Performance Simulation."

[61] Y. K. Cho, B.P. Zeigler, H.J. Cho, H.S. Sarjoughian, S. Sen "*Design Considerations for Distributed Real-Time DEVS,"* AI and Simulation, AIS 2000, Tucson, AZ.

[62] Daryl R. Hild, "Discrete Event System Specification (DEVS) / Distributed Object Computing (DOC) Modeling and Simulation," Dissertation, University of Arizona, March 2000.