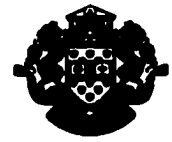

Université d'Ottawa • University of Ottawa

PERMISSION DE REPRODUIRE ET DE DISTRIBUER LA THÈSE

PERMISSION TO REPRODUCE AND DISTRIBUTE THE THESIS

NOM DE L'AUTEUR / NAME OF AUTHOR:	SHEN, Xiajun
ADRESSE POSTALE / MAILING ADDRESS:	1518-170 Lees Ave. Ottawa, ON K1S 5G5
GRADE / DEGREE:	**ANNÉE D'OBTENTION / YEAR GRANTED**
Ph.D.(Electrical Engineering)	2003
TITRE DE LA THÈSE / TITLE OF THESIS:	
MAP: A Mobile-Agent-Based Platform for Collaborative Virtual Environments	

L'auteur permet, par la présente, la consultation et le prêt de cette thèse en conformité avec les règlements établis par le bibliothécaire en chef de l'Université d'Ottawa. L'auteur autorise aussi l'Université d'Ottawa, ses successeurs et cessionnaires, à reproduire cet exemplaire par photograhie ou photocopie pour fins de prêt ou de vente au prix coûtant aux bibliothèques ou aux chercheurs qui en feront la demande.

Les droits de publication par tout autre moyen et pour vente au public demeureront la propriété de l'auteur de la thèse sous réserve des règlements de l'Université d'Ottawa en matière de publication de thèses.

The author hereby permits the consultation and the lending of this thesis pursuant to the regulations established by the Chief Librarian of the University of Ottawa. The author also authorizes the University of Ottawa, its successors and assignees, to make reproductions of this copy by photographic means or by photocopying and to lend or sell such reproductions at cost to libraries and to scholars requesting them.

The right to publish the thesis by other means and to sell it to the public is reserved to the author, subject to the regulations of the University of Ottawa governing the publication of theses.

N.B. LE MASCULIN COMPREND ÉGALEMENT LE FÉMININ

Dec. 17. 2002.

DATE

(AUTEUR) SIGNATURE (AUTHOR)

Université d'Ottawa · University of Ottawa

Université d'Ottawa · University of Ottawa

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

SHEN, Xiaojun

AUTEUR DE LA THÈSE - AUTHOR OF THESIS

Ph.D. (Electrical Engineering)

GRADE - DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT - FACULTY, SCHOOL, DEPARTMENT

TITRE DE LA THÈSE - TITLE OF THE THESIS

MAP:
A Mobile-Agent-Based Platform for Collaborative Virtual Environments

Nicolas Georganas

DIRECTEUR DE LA THÈSE - THESIS SUPERVISOR

EXAMINATEURS DE LA THÈSE - THESIS EXAMINERS

E. Petriu

A. El Saddik

D. Petriu

T. Radhakrishnan

J.-M. De Koninck, Ph.D.

LE DOYEN DE LA FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTDOCTORALES

SIGNATURE

DEAN OF THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

MAP: A Mobile-Agent-Based Platform for Collaborative Virtual Environments

By

Xiaojun Shen

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies, University of Ottawa, in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Ottawa-Carleton Institute of Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa

Dedicated to: My Parents, Wife and Son

Abstract

With the enhanced performance of modern personal computer systems, the acceptance and distribution of 3D and virtual environment (VE) applications have become more common. The first VE systems were single user systems, but now we can see that distributed VE (DVE) and CSCW (Computer Supported Collaborative Work) can open new areas of applications. Currently one of the most challenging areas of research in Virtual Reality is Collaborative Virtual Environment (CVE). It adds new dimensions to human-factors, networking, synchronization, middleware, object model acquisition and representation.

Some international standards have been developed that are very likely to make a major impact on CVE technology: the Distributed Interactive Simulation (DIS) and the High Level Architecture (HLA). Deriving from military-purpose simulations, DIS and HLA, however, have limitations on building large-scale general-purpose CVE applications. For instance, HLA ownership management only allows single participant to manipulate objects at any given time while providing meaningful real-time manipulation of a single object among multiple geographically distant users is essential for users performing collaborative tasks.

The mobile agent technology that has received a rapidly growing attention over the last few years overcomes these limitations and could be an effective solution to building large-scale general-purpose CVE applications. This research addresses the software architecture and techniques necessary to apply the philosophy of mobile agents to collaborative virtual environments. For this purpose, a mobile agent system called MAP is designed and implemented. MAP supports general-purpose CVEs, enables object dynamic introduction, joint-manipulation management and maximizes system flexibilities such as dynamic composition and task coordination that are applied for building a workload balanced interest management system.

Acknowledgments

I owe heartfelt thanks to my terrific supervisor Dr Georganas. This work would not have been possible without his perceptive suggestions, continued guidance, encouragement and support. And very special thanks to Jacynthe Georganas for her considerateness and kindness to our family.

I thank all the past and present members of the Multimedia Communication Research Lab for making my work here enjoyable: and Francois Malric for a thousand and one things he has provided with assistance and encouragement.

Life would be much more difficult without the support and companionship of good friends. There is no chance of thanking them all personally here, but it's a start: Xiaojing, Quansheng, Ryan, Lijun, ...

I thank Dr. Seok Jong Yu. Some of the ideas of the thesis came from the discussions with him.

I thank my father-in-law and mother-in-law. Without their great help, I would not be able to concentrate on my thesis writing.

Finally, I am indebted to my wife Lan Mao for her endless love, incredible intelligence, and unwavering confidence in me. I dedicate this thesis to my parents. Their love is my constant source of comfort and happiness.

Table of Contents

List of Figures

List of Tables

List of Acronyms

AEE	Agent Execution Environment
AOI	Area of Interest
AOIM	Area of Interest Manager
API	Application Programmer's Interface
CA	Commutative Algorithm
CSCW	Computer Supported Collaborative Work
CVE	Collaborative Virtual Environment
DIS	Distributed Interactive Simulation
DOI	Domain of Interest
DOR	Domain of Responsibility
DMSO	Defense Modeling and Simulation Office
DSI	Defense Simulation Internet
DVE	Distributed Virtual Environment
EC	Electronic Commerce
FED	Federation Execution Data
FOM	Federation Object Model
HLA	High Level Architecture
IE	Interest Expression
IM	Interest Manager
LRC	Local RTI Component
MA	Mobile Agent
MOM	Management Object Model
OMA	Object Mobile Agent

OMG	Object Management Group
OMT	Object Model Templates
PDU	Protocol Data Units
PCVE	Persistent Collaborative Virtual Environment
PDVE	Persistent Distributed Virtual Environment
RMI	Remote Methods Invocation
RTI	Run Time Infrastructure
RID	RTI Initialization Data
SOM	Simulation Object Model
SV	State Vector
SVE	Shared Virtual Environment
TSO	Time Stamp Order
VE	Virtual Environment
VN	Virtual Network
VR	Virtual Reality

Chapter I

1 INTRODUCTION

A Virtual Environment (VE) is a real-time simulation of a real or imaginary world where users navigate and interact with 3D objects. In a fully interactive multi-user VE, several participants connected by a network may meet and work together.

An appealing characteristic of a VE is its ability to offer intuitive models of interaction, analogous to the ways in which people communicate with each other or manipulate objects in the real world. VE applications can use 3D spatial properties to represent users, to model interaction, and to offer direct manipulation interfaces that mimic actions in the real world. They use immersive techniques that give participants the sense of being embedded in the synthetic environments. As such, Collaborative VEs (CVEs) can be seen as powerful human-computer interfaces. They provide new ways for users to communicate remotely and to access information over networks. CVEs are shared virtual spaces where geographically dispersed participants interact with each other and perform collaborative tasks.

Most existing distributed systems are based on a process-oriented approach: intelligence of collaborations is local to the concurrent processes communicating via passive messages. Mobile agent technology, on the other hand, which has received a rapidly growing attention for building distributed systems for its salient features, embeds the intelligence of collaborations in mobile entities which are capable of navigating through the underlying computational network and performing independent and/or

collaborative tasks at the nodes they visit.

A mobile-agent-based approach to distributed systems provides several functional advantages: machine-independent object description, bypassing faulty nodes, and tight encapsulation of algorithms in objects.

This research addresses the software architecture and techniques necessary to apply the philosophy of mobile agents to collaborative virtual environments. For this purpose, a mobile agent system called MAP is designed and implemented.

MAP supports general-purpose CVEs while maximizing flexibilities such as dynamic composition and task coordination that are applied for building a workload balanced interest management system. The system is implemented in the Java programming language.

This thesis discusses the features and architecture of MAP to make mobile-agent-based approaches to CVEs a reality. The thesis is further organized as follows: Chapter 2 reviews existing standards and technologies for CVEs, highlights their limitations, and clarifies the objectives of this thesis. In Chapter 3, the MAP platform is explained in detail in terms of its architecture and distributed features. Chapter 4 introduces the workload-balanced interest management over the MAP platform. A framework is proposed for interest management workload balance in either entity-based or spatial-based systems and specific algorithms are presented for spatial workload balancing. Chapter 5 describes an E-Commerce application that is implemented over HLA/RTI and MAP. A performance comparison is made between the two approaches. Chapter 6

concludes with the summary and potential future research topics.

ORIGINAL CONTRIBUTIONS OF THE THESIS:

1. To our knowledge, this is the first time that Mobile Agent concepts are introduced to the CVE domain. A Java-based mobile agent platform MAP is designed and implemented, which allows the virtual environment to be persistent by using the approach of introducing objects dynamically. An interaction mode for joint object manipulation is proposed and three means based on this mode are presented.

2.A framework is presented for dynamic workload balancing interest management based on the concepts of aura nimbus interaction model and MA objects;

3.Algorithms for exchanging workload of Interest Managers are proposed. These algorithms involve distributed decision-making, which makes them scalable; furthermore, the algorithms involve only near-neighbor communication, which makes them arbitrarily scalable; the algorithms preserve the topology of the Domain of Responsibility (DOR) decomposition which permits the same simple and efficient communication structure to be used by the work load balancing algorithms.

PUBLICATIONS RESULTING FROM THIS RESEARCH:

X. Shen, et al "vCOM: Electronic Commerce in a Collaborative Virtual World ", Electronic Commerce Research and Applications Journal (Publisher: Elsevier Science) (to appear) 2002

X. Shen, et al "*vCOM: Virtual Commerce in a Collaborative 3D World*", Proc. ACM Multimedia 2001, Ottawa, Sept. 2001

J. Oliveira, X. Shen, et al "*Collaborative Virtual Environment for Industrial Training and e-Commerce*". Proc. IEEE VRTS'2000 (Globecom'2000 Conference's Workshop on Application of Virtual Reality Technologies for Future Telecommunication Systems), San Francisco, CA, USA, 2000

X. Shen, et al , "*Agent-aided Collaborative Virtual Environments over HLA/RTI*", Proc. IEEE/ACM Third International Workshop on Distributed Interactive Simulation and Real Time Applications (D I S - R T ' 99), University of Maryland, College Park MD, Oct. 1999

J.C.Oliveira, X.Shen, et al, "*Collaborative Virtual Environments for Industrial Training and e-Commerce*", Chapter 7 in "*Virtual Reality Technologies for Future Telecommunications Systems* " (A. Pakstas and R. Komiya, editors), Willey, 2002

Chapter II

2 BACKGROUND

2.1 Collaborative Virtual Environments (CVEs)

The development of shared multi-user Distributed Virtual Environment (DVE) has become a major area of interest to the graphics community. The advancement of high-bandwidth wide area communications and the success of World Wide Web applications have fueled the expansion of DVE beyond local area networks.

Collaborative Virtual Environments are shared virtual worlds that could radically alter the way we work, learn, consume and entertain, e.g. how we play immersive video games. This is currently one of the most challenging areas of research in Virtual Reality (VR) because it adds new dimensions to human-factors, networking, synchronization, middleware, object model acquisition and representation. For example, human-factors research in VR has been traditionally focusing on the development of natural interfaces for manipulating virtual objects and traversing virtual landscapes. Collaborative manipulation, on the other hand, requires the consideration of how participants should interact with each other in a shared space, in addition to how co-manipulated objects should behave and work together. [Geor97][ShHG99]

2.1.1 Definition and use of a CVE

We can consider a virtual environment or VE-system as a means for a user to perform a given task, or as a medium in which this task can be performed (for our

purpose this makes no difference).

As a definition for a VE-system we consider the following, which lists a few necessary elements:

- A VE-system is a computer system,
- which generates a 3-dimensional virtual environment,
- with which the user can interact,
- in such a way that he/she receives real time feedback.

A Shared VE-system or SVE extends this definition with the following requirements:

- The VE-system can be used by multiple users simultaneously,
- who can be considered part of the virtual environment itself. (Hence, a user may be able to interact with other users.)

A Collaborative VE-system or CVE would then be an extension of a SVE:

- A CVE is a SVE aimed at a collaborative task.

We can think of a CVE as a virtual environment or *world*, populated by *objects* and representations of the user(s) called *avatars*. *Behavior* (sounds movement, interaction with the user or other objects) is associated with objects.

These definitions list only the necessary requirements. However, there are many aspects in which VE-systems may vary. These aspects can often be considered dimensions, on which VE-designers can make choices when designing a VE-system. To

name a few aspects:

Presence, the degree to which the user feels "part of" the virtual environment. This degree is dependent on many other aspects of the VE-system, but is probably critical to the success or failure of the task execution by the user.

Immersiveness, the degree to which the virtual environment surrounds the user. Desktop systems, Head mounted displays, CAVE [JoDe97,LeJD97] systems are some of the values along this dimension.

Interaction with reality, the degree to which the user must interact with reality while at the same time in the VE. For instance, what happens in the reality around him is of no concern to a user playing Quake [QUAKE], whereas for an Enhanced Reality task the virtual environment provides assistance for a task in reality.

Is the VE distributed or not? (Are the users at the same location?)

Does the virtual world have persistence? (Is it still there when the user logs off?) For instance, for the building task at hand in Alpha World [ALPHA] it is desirable that the world has persistence, whereas in Quake this is not (every time the game starts, you start afresh).

What are the input/output means used for the VE-system? (HMD, space mouse, data glove, speech, head tracking?) The input/output means determine which senses of the user are involved directly in the operation of the VE system.

2.1.2 Challenges of Building scalable CVEs

In order to support within a CVE collaborative work between many potential users over long distances, the architecture for scaleable distributed virtual environments is being addressed. Six important issues in implementing a CVE are listed in the sequel [Shen02]:

2.1.2.1 Consistency

The fundamental model presented by a CVE platform is a shared 3D space. Since all clients accessing or updating the data share the 3D graphics database, the issue of distributed consistency must be solved by any CVE to ensure the same view is presented to all participants. Solving consistency means satisfying user interaction predictions and providing a basis for causality in an environment. Causality implies that if event A "happens before" event B, the message for A should be delivered before the message for B. "Things" happen in the real world in a certain order (e.g., cause and effect). Events in the virtual world should happen in the same manner (Figure 1 [HLA]).

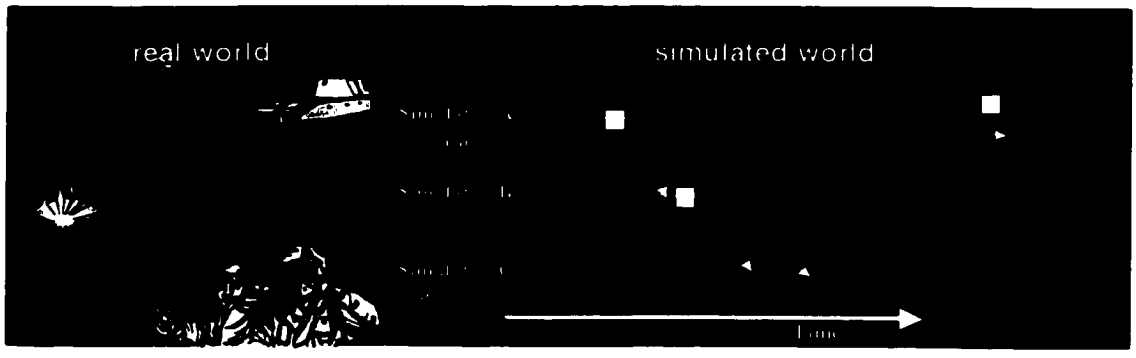

Figure 1 Causality Requirement in Distributed Virtual Environments

Since the number of participants is not fixed, and during the run time users may enter the environment after the environment has been changed from its initial state, a

CVE needs to be able to provide support for latecomers and early leavers.

2.1.2.2 Scalability

The number of possible interactions between n simultaneous users in a multi-user system is of order $O(n^2)$ at any moment. Ideally network traffic should be almost constant or grow near-linearly with the number of users. Usually, not all the data in the CVE environment would be relevant to a particular user at a given time. This suggests the idea of partitioning the environment into regions (or zones, locales, auras) [ShHG99,Gree97,Mace95] that may either be fixed or bound to moving avatars. Events and actions in remote zones need not be distributed and remote objects need not be visualized or might be visualized at a coarse-grain level. Most of the traffic is isolated within zones.

2.1.2.3 Ownership and Joint Manipulation

A multi-user CVE world is subject to conflicts. Conflicts occur when collaborating users perform opposite actions, for example, a conflict may arise if one user tries to open a window while another user is trying to close it. These conflicts must be avoided or solved. One method to determine the attributes of objects that may be modified by certain users is to assign temporary ownership to objects. Manipulation of objects may include the change of the object's coordinate system and the change in the scene graph: users may grasp objects and take them to a different world. Operations like these are essential for applications such as virtual shopping.

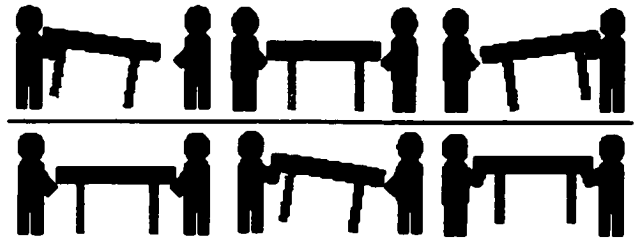

Figure 2 Cooperative vs. Collaborative

Direct (real-time) cooperation leads to completely new interaction possibilities, especially concurrent interaction of several users with one or more objects, which is lacking in most existing CVE systems. This raises serious problems for distributed systems, which so far have had to be solved for each application. Figure 2 gives a simple example [Brol95]: Two people might lift a table one after another, but they are not able to carry it individually. Both of them would have to lift the table at the same time to move it, but this example could not be replicated with existing CVE systems.

We can group architectures for Shared Virtual Environments into three major categories: static, cooperative and collaborative. In a ***static virtual environment*** each user can explore by looking and touching, but not modify the environment. Users may or may not see each other during the simulation, but cannot interact with each other. Examples are browsing geometrical or shared databases on the net. In a ***cooperative virtual environment*** users can modify the environment but may not simultaneously move or change the shape the same virtual object. This scheme can be applied for surgical or industrial training, and entertainment. In a ***collaborative virtual environment*** users can simultaneously modify the same virtual object, see and interact with each other, directly

or indirectly through a common object.

When we discuss distributed, multi-user VR systems, the terms "*cooperative*" and "*collaborative*" are used interchangeably. For us, cooperative implies joint editing of shared objects, while collaborative additionally allows truly concurrent editing.

Therefore, the change from cooperative virtual environments to collaborative virtual environments is mostly continuous. Providing meaningful real-time manipulation of a single object among multiple geographically distant users is a significant research challenge given the limitations of today's CVE technology, and solutions will inevitably require significant usage of both network and computer facilities; knowing in advance that the control of single *shared* object is in fact passed back and forth between individual users allows the system to focus its network resources on those objects that might be genuinely manipulated simultaneously.

2.1.2.4 Connectivity and Scalability

Connectivity has various dimensions including bandwidth, capacity, protocols and topologies. CVEs require that the network protocol must scale to (a large) number of users and heterogeneous network connections.

2.1.2.5 Persistence

Some of the applications that involve a large number of users need a large-scale, persistent distributed virtual environment (PDVE) system that is "never-ending" or "always on". This is either because its users require that it is always running, or because it is so large or distributed that stopping the entire simulation to make changes is just not

possible. There are a number of issues that should be addressed to support PDVE. Persistence would be the first feature that characterizes a PDVE system. It describes the extent to which the virtual environment exists after all participants have left the environment.

2.1.2.6 Dynamic Configuration

This property allows a PDVE system to dynamically configure itself without user interaction, enabling applications to take on new functionalities after their execution. CVEs should be modifiable at run-time by accepting the contributions of new objects and new behaviors.

2.2 Standards for Distributed Simulations

Several international standards are very likely to make a major impact on DVE technology: the Distributed Interactive Simulation (DIS) (IEEE Standard 1278.1) [IEEE1278, DIS93] and the High Level Architecture (HLA) (IEEE Standard 1516) [HLA]. DIS is a set of communication protocols to allow the interoperability of heterogeneous and geographically dispersed simulators. Its successor, the currently emerging HLA, defines a standard architecture for large-scale distributed simulations.

2.2.1 DIS Protocol

Development of the DIS protocol began in 1989, jointly sponsored by the United States Army Simulation, Training and Instrumentation Command (STRICOM), ARPA and the Defense Modeling and Simulation Office (DMSO). DIS was based on SIMNET [DuMa95, Frie88, John92, Pope89] and designed as a man in the loop simulation in

which participants interact in a shared environment from geographically dispersed sites. The initial objective was to develop a standard that provided guidelines for the interoperability of defense simulations. DIS provides a basis of interoperability for large-scale virtual environments using a wide variety of different hardware and software platforms. DIS has been adopted by the Institute of Electrical and Electronics Engineers (IEEE) as an International Standard (IEEE Standard 1278).

While the DIS standard adopted many aspects of the earlier SIMNET protocol including its general principles, terminology and PDU formats, it is intended to overcome the limitations of SIMNET and includes packet definitions not found in SIMNET [DuMa95]. DIS is also designed to use the Defense Simulation Internet (DSI), an ARPA project designed to allow thousands of players to link using real and simulated forces to create a Synthetic Theater of War (STOW) [DuMa95]. Another difference between SIMNET and DIS is that DIS uses the TCP/IP suite of protocols and is thus an application level protocol, allowing it to be used on any network topology that uses TCP/IP.

2.2.1.1 Protocol Data Units (PDU)

The heart of DIS is a set of protocols that are used to convey messages about entities and events, via a network, among the simulation nodes that are responsible for maintaining the status of the entities in the virtual world. Simulation and event information is conveyed by the twenty-seven PDUs defined by the IEEE 1278 DIS standard. Four of the PDUs are used for sending information about entity interaction. The other twenty-three are used for transmitting information on supporting actions, electronic

emanations and simulation control [Mace95]. DIS PDUs are independent of the network media and network protocols being used to transmit them and can be used on most current network topologies.

The PDUs used to transmit information about entity interaction are: Entity State PDU (ESPDU), Fire PDU (FPDU), and Detonation PDU (DPDU). The ESPDU is used to communicate information about an entity's current state, including its position, orientation, velocity and appearance. The ESPDU is the most commonly used PDU and, in most instances, it dominates the network traffic. The FPDU contains data on any weapon that is fired or dropped. The DPDU is sent when a munition detonates or an entity crashes. The actual structure of each PDU is very regimented and is explained in full detail in IEEE 1278.

2.2.1.2 Problems with the DIS Protocol

Using the current method, a large-scale simulation would require enormous bandwidth and place a huge computational load on host computers. It has been estimated that using DIS, a simulation with 100,000 players would require approximately 375 Mbps of network bandwidth for each computer participating in the simulation [Mace95]. Since this is not within the realm of possibility in the near future, the implementation of large-scale virtual environments will require a communications protocol with a much lower bandwidth requirement. Computational load is another problem that must be overcome. In 1994 the U.S. Army attempted to demonstrate the feasibility of large-scale simulations in the Synthetic Theater Of War - Europe (STOW-E) [CaHo94, Hook96, Hook94] exercise. The goal of the exercise was to simulate 10,000 entities over a wide area

network connecting a number of sites in the United States and Europe. However, only 1,800 entities could be represented because of computational bottlenecks in both the simulators and support equipment [Mace95].

Because DIS is a stateless system and does not utilize servers, all data must be distributed to all entities in the system. Thus when the status of one-entity changes, it must send an update, as an ESPDU, to every other entity in the simulation. As such, ESPDU's can account for up to 70% of the network traffic [Mace95]. For entities that are moving, it is necessary to send these updates as often as needed. Not doing so would corrupt the coherence of the virtual world. However, there is no such problem with entities that are stationary. An analysis of an actual combat scenario at the National Training Center showed that in ten hours of simulated combat approximately 33% of 2,191 entities did not move. As the exercise continued, over half of the vehicles stopped all movement. [MMZM95] If there were a method where each entity could have knowledge of every stationary entity without transmitting an ESPDU every five seconds there would be a significant reduction of PDU traffic. This would also remove some of the computational burden on the host computers by removing the need to read these PDU's off from the network. Mobile agents using the remote programming paradigm could be a solution to these problems.

2.2.2 High Level Architecture

This High Level Architecture (HLA) standard is a general architecture for simulation reuse and interoperability [HLA] developed by the US Department of Defense. The conceptualization of this High Level Architecture led to the development

of the Run-Time Infrastructure (RTI). This software implements an interface specification that represents one of the tangible products of the HLA. The aim of it is to develop, under the leadership of the US Defense Modeling and Simulation Office (DMSO), reuse and interoperability across large numbers of different types of simulations developed and maintained [HLA]. This HLA architecture is an IEEE standard (No.1516) and an OMG (Object Management Group) for distributed simulations and modeling.

2.2.2.1 HLA Overview

The HLA (Figure 3) standard promotes reuse of simulations and their components. It attempts to specify the general structure of the interfaces between simulations without making specific demands on the implementation of each simulation. In the HLA baseline documents, three basic concepts are defined:

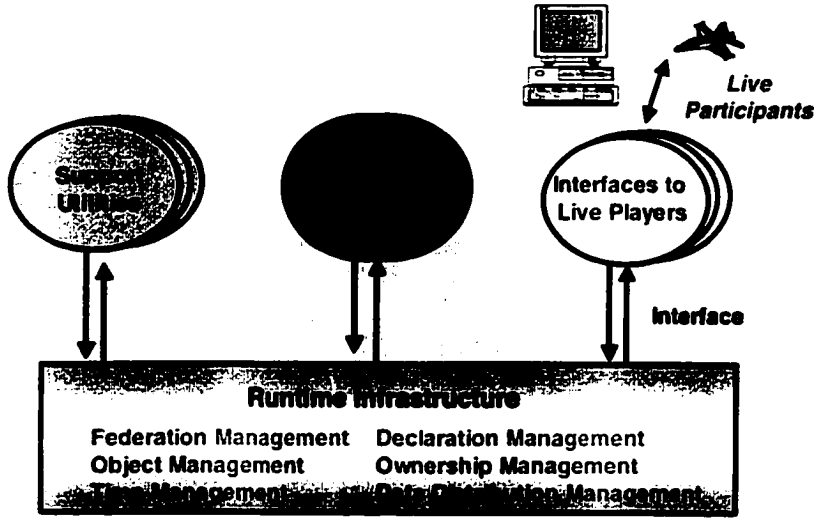

Figure 3 Functional View of HLA/RTI

1. The *Interface Specification* is a formal, functional description of the interface between the HLA application and the Run-Time Infrastructure.

2. A set of technical *rules* is defined to which an HLA participant has to comply.

3. The *Object Model Templates (OMT)* are standardized formats to define the functionality of simulation models and the interaction between models. In this way the capabilities of all simulation models are defined before the actual simulation takes place.

The Run-Time Infrastructure (RTI) software implements the interface specification and represents one of the most tangible products of the HLA. It provides services in a manner that is analogous to the way a distributed operating system provides services to applications.

2.2.2.2 HLA Concepts and Terminology

Within the HLA, *federations* are defined as a group of federates forming a community. Federations are comprised of *federates* that exchange information in the form of *objects* and *interactions*. Federates may be simulation models, data collectors, simulators, autonomous agents or passive viewers. A simulation session, in which a number of federates participate, is called a *federation execution*. Simulated entities, such as vehicles or aircraft, are referred to as *objects*. Live entities can be mapped onto objects (by using simulation surrogates), so that they can appear identical to simulated entities. All possible interactions between the federates in a federation are defined in a so-called *Federation Object Model* (FOM). The capabilities of a federate are defined in a so-called *Simulation Object Model* (SOM). The SOM is introduced to encourage reuse of

simulation models.

An *Attribute*, referred to as objects state, can be passed from one object to another. Objects interact with each other and with the environment via *interactions*, which may be viewed as unique events, such as manipulation of the object, or a collision between objects. Initially, an attribute is controlled by the federate that instantiates the object whose state the attribute is part of. However, attribute ownership may change during the execution. This mechanism allows users, for instance, to co-manipulate the same object.

In order to reduce network traffic and limit the amount of computation for each federate, the HLA provides a mechanism of *publication* and *subscription.* Upon initialization of a federation execution, each simulation registers the objects and their attributes with the RTI. It also registers which attributes and interactions it needs in order to be able to perform its task (subscription). Note that a federate can not only subscribe to attribute types, but also to (ranges of) attribute values. The aim here is to filter as much information as possible at the source. Both publications and subscriptions are dynamic and may be changed during a session. In this way, multicast communication is realized which reduces the necessary bandwidth.

The Run-Time Infrastructure supports HLA compliant simulations through a number of services. The main categories of these services are:

Federation Management: create/destroy execution (of a federation), join/resign execution (of a federate), pause/resume execution;

Declaration Management: subscription/publication;

Object Management: instantiate/delete (an object), update/reflect (an attribute), request/provide (an attribute), send/receive (an interaction);

Ownership Management: transfer ownership (of an attribute) from one object to another;

Time Management: handling of messages in different ways, depending on the requirements of their destinations, e.g., in receive order (appropriate for real-time human-in-the-loop simulators), in priority order, in causal order, or in timestamp order. These last three methods are typically suitable for non real-time event driven simulations. (e.g., analysis purposes).

Data Distribution Management: provides a flexible and extensive mechanism for further isolating publication and subscription interests – effectively extending the sophistication of the RTI's switching capability.

For details of the HLA interface specification, see [HLAa]. Documents describing HLA can be downloaded via the DMSO homepage site. [1]

2.2.2.3 HLA filtering mechanisms

A fundamental abstraction of the HLA data distribution management services is the routing space. A routing space is a multidimensional coordinate system in which

---

[1] http://www.dmso.mil

federates express interest for either receiving attributes/interactions from other federates or sending attributes/interactions to other federations. Federates also agree to maintain associations of data they own to routing spaces. The RTI uses federates' expressions of interest to establish the network connectivity needed to distribute all relevant data and minimal irrelevant data from producers to consumers.

A routing space is made of *subscription regions* and *update regions*. Subscription regions are the expression of the interest of a federate and update regions are the expression of what a federate is able to produce. The overlapping of subscription and update regions informs the RTI of the dependency between federates and therefore the communications to be established.

The RTI makes a distinction between routing of data (establishing the necessary network connectivity) and actually sending the data. The objective here is to minimize latency and maximize throughput by establishing the necessary network connectivity before it is actually needed [Hook96].

2.2.2.4 Limitations of HLA/RTI

2.2.2.4.1 Static Properties

HLA uses a *Federate Object Model* (FOM) to define the standardized formats of the functionality of simulation models and the interaction between models. A FOM describes what may be actually exchanged within a Federation. Roughly, the FOM is a negotiated subset of the intersection of the SOMs *(Simulation Object Model)* of the Federates. This negotiation takes place at the beginning of the design time of the

federation. A communication atom is either an object class attribute or an interaction. Interaction parameters are not atoms. An atom has a transportation mode ("reliable" or "best effort") and a delivery or ordering mode ("receive ordering" or "time stamp ordering").

As a result, the capabilities of all simulation models are determined before the actual simulation takes place. OMT (*Object Model Templates*) in simulation is defined as *Federation Execution Data (FED)* (Shown in Figure 4) and each federate in a federation keeps the same FED copy in its local storage. The FED is a subset of the FOM data including the object-class tree, the interactions tree and, for each communication atom, the transportation and ordering modes. This static property of the FED prevents federates from injecting undefined objects/interactions into simulation or modifying communication atom in progress. That means, to introduce new entities or modify communication atom, simulation programs have to be modified, recompiled and restarted, which obviously will cause problems for persistent application.

However, persistent CVE applications, as discussed in Section 2.1.2.5, require the dynamic property that allows a PCVE system to dynamically configure itself without user interaction, enabling applications to take on new functionalities after their execution. Thus, RTI's static properties make it not suitable for PCVE design.

```
;; user object classes here
    (class Avatar
        (attribute Name reliable timestamp)
        (attribute Message reliable timestamp)
        (attribute Scene reliable timestamp)
        (attribute Type reliable timestamp)
        (attribute Color reliable timestamp)
        (attribute Position reliable timestamp)
        (attribute Rotation reliable timestamp)
        (attribute Dimension reliable timestamp)
        (attribute Reserved1 reliable timestamp)
        (attribute Reserved2 reliable timestamp)
    )
;; user interaction classes here
    (class Communication reliable timestamp
      (parameter Message)
    )
```

Figure 4 Simplified Federation Execution Data

2.2.2.4.2 Limitation with Joint-Manipulations

In HLA, one of the attributes of a shared object in simulations is its ownership, which indicates which user (process) is allowed to access or manipulate the object. The manipulation is performed through the modification of some of the attributes of an owned object (such as position and rotation). In the RTI implementation, an attribute of a given object instance has one single owner at any given time. However, attribute ownership may change during the execution. This mechanism allows two or more users to handle the same object through the transfer of ownership, but users are not able to manipulate it simultaneously within the CVE.

In some complex situations, several users need to manipulate the same object at the same time in order to perform collaborative tasks, the example shown in figure 2, where two people would have to lift the table simultaneously. Obviously, this cannot be

replicated with the existing RTI implementation, which only allows a single owner of a shared object at any given time.

2.2.2.4.3 Limitation of the RTI Implementation

RTI provides services for distributed simulation implementation, while some services are initiated by the federate and others by the RTI, as Figure 5 depicts. Federate invoked services like "publish/subscribe/register/update", are procedures defined in the RTI ambassador. These services can be invoked like functions call to a library. RTI initiated services are callback functions to the federate.

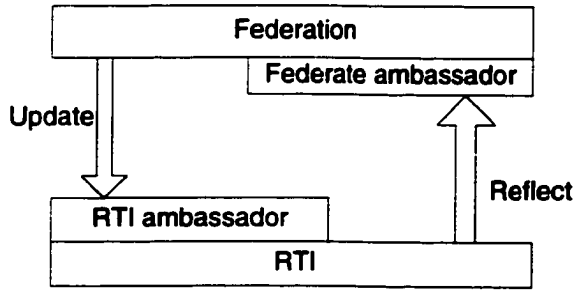

Figure 5 Federated vs. RTI Initiated Services

For current RTI v1.3 implementation [CaRi96, DaJa98, Brig98, ChSh98], RTI is composed of three main parts (Figure 6): RTI Ambassador, Federate Ambassador, and Internal RTI Managers. The solid lines represent service invocations either from the federate into the RTI or vice versa. The dashed line in the figure6 represents intra-RTI service invocations between the different RTI Internal Managers. The first two parts of the RTI are used to pass information from the RTI to the federate and vice versa. The RTI Ambassador serves as an interface for marshalling service requests made by the federate to the appropriate RTI Internal Manager. The Federate Ambassador serves as an interface

for marshalling service request responses, such as events or Time Advance Grants, from the RTI to the federate. Each federate must provide an implementation of the Federate Ambassador's services.

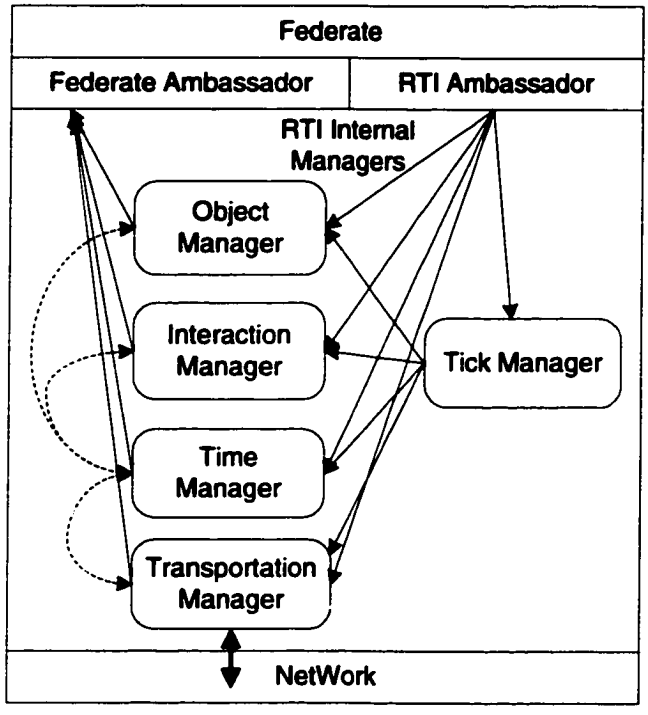

Figure 6 Implementation of RTI

The Internal RTI Managers support five categories of run-time services. The Object Manager implements the object creation, object destruction, ownership, object publication and object subscription services. The Interaction Manager implements the services that create and destroy interactions, as well as interaction publication and subscription services. The Time Manager implements the services for advancing logical time and will be discussed in detail. The Transportation Manager is used to send and receive data and supports the services provided by the other managers. The federation

management services are implemented in the RTI Ambassador.

As required by the above design pattern, the RTI is implemented to be linked in as part of the federate to form a single simulation with only one thread of control. To reduce the effort involved in integrating federates with a single threaded RTI, RTI provides a tick service in the RTI Ambassador that gives the thread of control to the RTI and it ensures that all the necessary internal RTI functions and service requests are completed.

The *tick()* method is not a part of the Federation Management functionality identified in the *High Level Architecture Interface Specification Version 1.3*. It is, however, a very important part of the RTI 1.3 release. The RTI 1.3 software is not multithreaded. The LRC (Local RTI Component) does a lot of work (e.g., exchanging information with counterparts) and needs time to do that work. The *tick()* method yields time to the RTI.

The *tick()* method exists in two forms – one taking zero arguments and another taking two arguments. The zero argument version yields time to each major activity within the LRC. A typical activity would be draining inbound event queues and providing callbacks to the federate via the *FederateAmbassador*. There is no guarantee as to the time required for a call to *tick()* to complete. The two-argument version of *tick()* also yields time to the LRC, but suggests lower and upper bounds on the time being allotted to *tick()*. Like the no argument version, the two-argument version makes no guarantees as to its overall execution time. It, too, yields time to each major activity within the LRC, iterating as time permits.

When a federate requests the tick service, that request is sent to the Tick Manager. The Tick Manager then invokes the tick service provided by each of the RTI Internal Managers to perform their necessary functions. Having the single thread of control, the Transportation Manager would deliver pending events from the network up to the Time Manager, where they are en-queued into the appropriate queue (either TSO or receive order). When the Transportation Manager's tick service completes, control is returned back to the Tick Manager who immediately gives control to the Time Manager by invoking its tick service.

Therefore, in single-threaded RTI, RTI and federate share a single thread of execution (like libraries); and the federate must explicitly pass control to the RTI, e.g., to deliver messages; the *tick()* procedure is defined for this purpose and callbacks (*Discover*, *Reflect*) are made within the tick period.

Figure7 shows an example of updating and reflecting object attribute values in single threaded RTI. Once the federate needs to update the shared object's attribute, it invokes *UpdateAttributeValues*() service, however, the updated values wouldn't be delivered until tick() is called by the federate, which releases the thread control to RTI. It is similar for the case of reflecting object attributes. The federate wouldn't have the most current updated values until it calls tick, even through these values have arrived earlier. The evaluation of RTI (see Chapter 6) shows that the single-threaded RTI is not proper for real time CVE design; its large latency hinders participants from performing collaborative tasks.

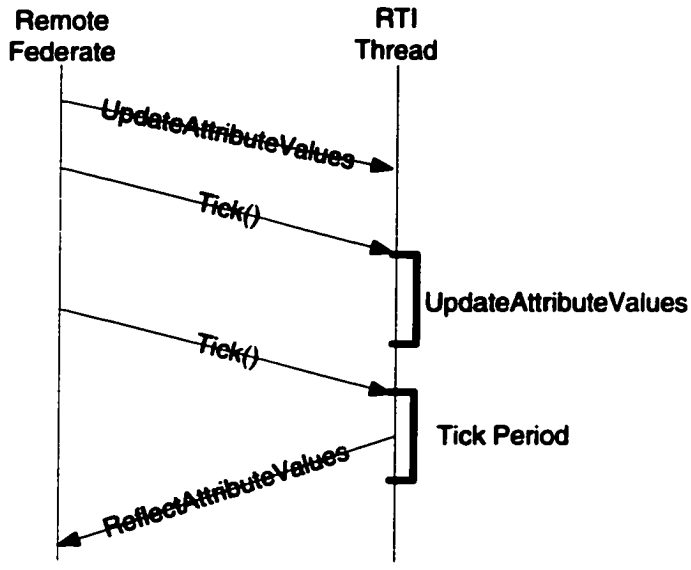

Figure 7 Update and Reflect in Single-Threaded RTI

2.3 Controlling the Visibility of Data for Scalable CVEs

The underlying assumption behind the data flow restriction is that the CVE contains a tremendous amount of information, yet an individual user only needs to know about a small piece of the total information that is available. Therefore, instead of broadcasting each packet to every participating host, we should strive to only send information to those hosts that really need to receive it. By limiting the dissemination of data, the aggregate bandwidth requirements of the CVE can often be reduced.

Data flow optimizations can be expressed generally using the ***aura-nimbus*** information model [GrBn97]. Entity data should only be available to those entities that should be capable of perceiving that information; this sphere of influence constitutes the source entity's aura. Similarly, entity data should only be delivered to those entities who are interested in that information; this sphere of influence constitutes the destination

entity's nimbus. In an ideal world, each piece of information would be processed individually and delivered only to entities whose nimbi intersect with the data's aura (Figure 8). In Figure 8, entity A can "feel" entity B since its nimbus region intersects with entity B's aura region, however the A can not "see" B. This approach, used in the MASSIVE-1 system [GrBe95], does not scale effectively because it typically requires considerable processing resources and does not allow for any network efficiencies. Most of the techniques that we will discuss next attempt to approximate the pure aura-nimbus model in a scalable manner.

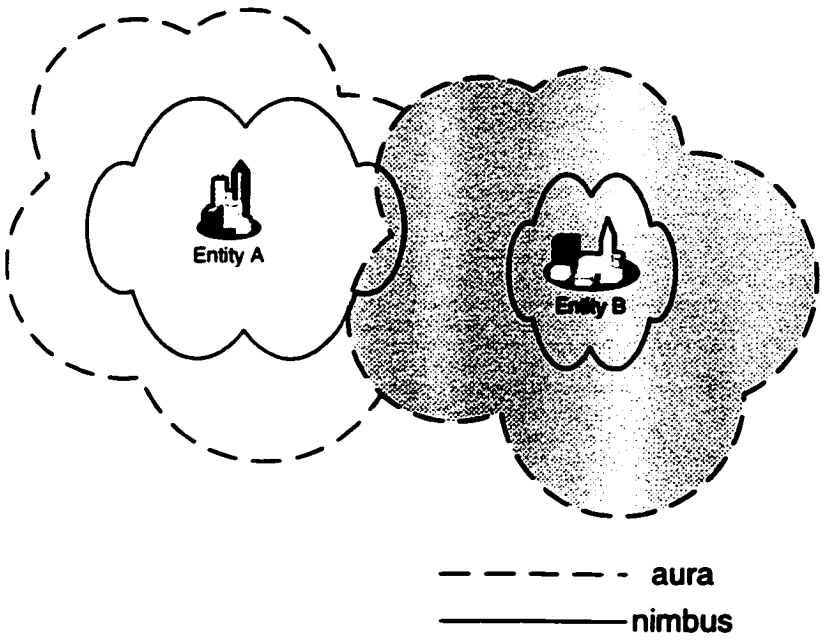

Figure 8 Aura-Nimbus Interaction Model

Data flow management can be divided into three broad categories: *area-of-interest filters, multicasting, and subscription-based aggregation.* Area-of-interest filters are explicit data filters provided by each host, allowing the CVE to perform fine-grained

data management to only deliver the information that the host needs. Multicast involves taking advantage of subscription-based network routing protocols to restrict the flow of data. Subscription-based aggregation represents a hybrid between these two approaches, grouping the available data into fine-grain "channels" to which hosts may subscribe.

2.3.1 Area-of-Interest Filtering Subscriptions

Under an area-of-interest filtering scheme, hosts transmit information to a set of subscription managers, alternatively referred to as "interest manager "(IM) or "filtering servers". These subscription managers also receive subscription descriptions from each of the participating hosts in the CVE. For each piece of transmitted data, the subscription managers determine which of the subscriptions are satisfied by that data and only disseminate the information to the corresponding subscribing hosts. In this way, the subscription managers can be thought of as "match makers" who try to link the available information with hosts who are interested in that information.

Area-of-interest filtering is simply a limited form of the pure aura-nimbus model. In a pure aura-nimbus system, each piece of data would be accompanied with a dissemination description that specifies which entities are permitted to receive the information; the area-of-interest managers would process both the dissemination descriptions and the subscription descriptions to manage data flows. The optimized approach that we consider in this Section eliminates the dissemination descriptions in order to reduce the processing and bandwidth load on the system. Aura specification is therefore eliminated, though the nimbus specification is retained.

2.3.2 Evaluating Filtering Subscriptions

Because each host can specify its own subscription request and because the subscription descriptions may be arbitrarily complex, an area-of-interest subscription can ensure that a host does not receive extraneous information that it will subsequently discard. Each host receives a customized flow of information. This model is appropriate under five circumstances:

Participating hosts have limited processor capacity and cannot afford the cost of receiving and processing unnecessary packets.

Participating hosts are connected over extremely low-bandwidth links, so network utilization must be minimized at all costs.

Multicast and/or broadcast protocols are not available, so point-to-point transmission is the only available method for exchanging information.

Client subscription patterns change rapidly so that the host cannot tolerate the delay in re-routing a network multicast group after joining or leaving a multicast group.

The CVE designer cannot determine any *a priori* groupings for CVE data and the group-per-entity approach is too resource expensive.

The biggest cost of area-of-interest filtering is the increased network bandwidth required when a significant number of hosts are interested in the same piece of information.

Because the data streams are effectively customized for each receiver, data must

generally be unicast from the subscription managers to each recipient. As a result, a piece of data may travel multiple times over the same network link on its way from the filtering server to the interested hosts. Multicasting directly served to eliminate this duplication.

Area-of-interest filters provide what can be called ***intrinsic filtering*** because the content of each packet must be inspected to determine whether it should be delivered to a particular destination host [Kath96]. The intrinsic filter is parameterized by the area-of-interest subscription provided to the filtering server. An alternative approach, ***extrinsic filtering***, filters packets based on external properties such as the network address to which the packet was transmitted. Whereas intrinsic filtering could dynamically partition data based on fine-grained entity interest, with extrinsic filtering, the CVE designer statically partitions the data based on assumptions about the subscription patterns of the participating entities.

2.3.3 Multicasting

Multicasting is a network protocol technique whereby the application sends each packet is transmitted to a ***multicast group*** by supplying a special ***multicast address*** as the destination for a UDP/IP packet. The packet is only delivered to those hosts which have subscribed to (join) the corresponding multicast group. Similarly, to stop receiving those packets, the host must unsubscribe from (leave) the multicast group. A host may be subscribed to multiple multicast groups simultaneously. Any host may transmit data to any multicast group, regardless of whether it is currently subscribed to that group.

Compared to broadcast protocols, multicasting is relatively efficient. From each

data source, the multicast group is represented as a shortest-path tree through the network with the source host at the root and the destination hosts at the leaves. The branches of the tree are network links, and the internal tree nodes are routers and gateways in the network. When a packet is transmitted, a single copy of the packet travels along each network link in the multicast tree. Whenever the packet encounters an internal node in the multicast tree, the packet is copied so that it may travel along the descendant branches.

As a result of this distribution method, the packet only travels to parts of the network that actually have interested subscribers. If no host is currently interested in the packet, then it travels no further than the source host's local LAN. In general, if only a few hosts are members of the multicast group, the packet is likely to only travel over a small subset of the network links that a broadcast packet would traverse to reach all participants in the CVE.

The key challenge for implementing multicast-based solutions is determining how to partition the available data among a set of multicast groups. Successful multicasting demands that each multicast group contain a related set of information. In other words, subscribers to a multicast group should generally be interested in receiving most, if not all, of the information that is actually transmitted along that group. If a multicast group contains completely unrelated information, then it will attract subscribers who are only interested in a subset of the transmitted information. As a result, information will be disseminated to network locations where it is not required. In the worst group and consequently subscribes to every available multicast address. In this case, the whole multicast system would degenerate into a broadcast system.

2.3.4 Group-Per-Entity Allocation

One approach to the multicast group-mapping problem is to assign a different multicast address to each entity in the CVE [AbWZ97]. Each host subscribes to the multicast groups for entities that are of local interest. Like the area-of-interest subscription filter, this approach allows a recipient's *nimbus* to be represented faithfully because each CVE host explicitly selects which entity information is of local interest; the subscription filter is executed locally based on information about which entities exist in the CVE. However, unlike the area-of-interest approach, subscriptions can only be made on a per-entity basis, rather than on a per-packet basis. Because information is transmitted on a fixed multicast group per entity, entities have no control over the aura for their information.

A form of this multicast approach was employed in the Stanford PARADISE system [SiCh96]. Each entity subscribed to information for entities located nearby in the virtual environment. Entities could also control their directional information interest. For example, a vehicle could subscribe to nearby entities that are behind it and subscribe to both nearby and distant entities located in front.

This multicast allocation technique can be extended to assign specific types of information to separate multicast groups. For example, an entity's position updates may flow on one group while its infrared data may be transmitted in another group. In this way, heat sensors can choose to only receive the infrared information without being burdened with the physical position data. Moreover, because each multicast group only has a single static sender and, consequently, a single multicast distribution tree, the group

allocation technique provides opportunities for the multicast routing algorithms to be optimized.

To be effective, this technique requires a way for hosts to learn about the entities that are located nearby in the virtual world and the multicast addresses used by those entities. To provide these functions, the system must include some sort of entity directory service that tracks the current location of entities. This service is managed by having each entity periodically transmit information over one or more designated directory multicast addresses. The directory servers listen to these periodic location announcements and provide the information to any entity that requests an updated list of entities in it vicinity. Such an entity location service is also provided by the Diamond Park CVE developed at Mitsubishi Electric Research Laboratories (MERL). Beacon[WaWA96] servers exist for each region of the virtual environment, and each has a designated multicast address for receiving information about entities in the associated region. The beacon server also shares a well-known beacon multicast group used by clients to locate the appropriate beacon server for a particular virtual world region.

This allocation of one multicast group per entity is potentially costly in large CVEs. It consumes a large number of multicast addresses. Because the pool of multicast addresses must be shared among all Internet multicast applications, a large CVE is likely to collide with one or more other applications using random multicast addresses. The large number of multicast groups creates an overhead on network routers that must process the corresponding large number of group membership join and leave requests. It introduces traffic for group discovery, as hosts need to learn the multicast addresses for

each of the entities in which they are interested. Finally, most network cards can only support a limited number of simultaneous multicast address subscriptions (in the hundreds with modern cards). When a host exceeds this limit, the card typically enters "promiscuous mode" and simply receivers all multicast packets sent on the local LAN.

To address these limitations, the CVE designer can impose an upper bound on the number of multicast addresses that are used by the CVE. Once the available addresses are exhausted, the environment recycles the addresses that are already in use. It is possible, therefore, that multiple entities may share a single address, however, that physical address can still be though of as representing multiple logical multicast groups. As more multicast groups become available, the entities can be re-assigned without disrupting the client-side code.

2.3.4.1 Group-Per-Region Allocation

Instead of assigning a multicast address to each entity, we may partition the virtual world into regions and assign each region to one or more multicast groups[ShHG99]. Each entity simply transmits its data to group(s) corresponding to regions that are adjacent to the transmitter. Each entity subscribes to groups corresponding to the regions, in which they have interest, again generally corresponding to the regions that are immediately adjacent to the entity.

The group-per-region design permits entities to have limited control over their aura by permitting information dissemination to be restricted to particular regions of the CVE. However, subscribes no longer have full control over their nimbi, because they can

only filter information according to the sender's location. The group-per-region approach manages communication overhead by forcing aura and nimbi behavior to correspond to a limited set of spatial categories.

Unlike the group-per-entity approach in which each entity transmits over the same multicast group throughout its lifetime, the group-per-region approach requires each entity to change its target group(s) as it travels throughout the virtual world. The transmitter must track the bounds of its current region and, upon entering a new region, learn the associated multicast addresses. The region boundaries and the mapping from region to group addresses are often defined statically and installed on each client. More sophisticated systems collect this information in a directory service, similar to the beacon servers described above.

Traditionally, the virtual environment is partitioned into rectangular grid regions. However, grid-based region assignments have the disadvantage of having many points at which multiple grids meet. As an entity approaches one of these corners, it must subscribe to groups corresponding to all of the regions that meet that corner. To alleviate this difficulty, the NPSNET system divides the virtual environment in hexagonal regions [MMZM95], which have the desirable property that only three regions intersect at any given point instead of four. As an entity enters the region, is transmit a Join Packet, and it sends a Leave Packet upon leaving the group. Each region has an implicit group leader corresponding to the entity that has been active in that group for the longest time. That leader maintains a list of all of the entities currently in the region and provides that list to hosts that transmit a Join Packet in the region.

Though easy to implement, a regular tessellation of the virtual environment may cause a region boundary to fall in an inconvenient location. For example, if a region boundary falls in the middle of a hallway, then all entities traveling along that hallway must subscribe to the multicast groups for both regions. Ideally, therefore, the regions should be determined based on the actual structure of the virtual environment. In the Diamond Park system [BaWA96], each region was called a Locale and could have an arbitrary geometric structure and an independent coordinate system. The Locales were "stitched" together, with a translation, rotation, and scaling matrix associated with each boundary to perform a coordinate system transformation.

Ideally, a CVE would combine the customized subscriptions of intrinsic filtering with the multicast efficiency provided by extrinsic filtering. Unfortunately, it is generally impossible to retain the full advantages of both filtering approaches within a single system. However, some recent experimental systems have attempted to include elements from both techniques. This hybrid approach is the subject of the next section.

2.3.4.2 Projection-Based Multicast Aggregation

In a projection-base data management system, information is partitioned based on multiple parameter values. Each of these groups is referred to as a projection and each projection is associated with its own multicast group address. The goal behind projections is to strike a balance between fine-grain data partitioning and multicast grouping. On one hand, projections aim to partition CVE data into fine-grain groups that allow each client to effectively customize the delivered data stream based on local interests; fine-grain grouping allows each client to receive a form of intrinsic filtering that approximates its

nimbus. In the mean time, projections aim to ensure that the partitioning is not so fine-grained that a data transmission degenerates into a simple unicast; in other words, the multicast groups should still be broad enough to include multiple source entities and multiple potential recipients, thereby guaranteeing some level of extrinsic filtering.

For example, suppose that each entity in a CVE has a type and a location. Each projection contains information about a different set of entity types and entity locations. One projection might include tanks located between (10,25) and (35, 40) while another projection includes cars located between (85,70) and (110, 85) in the CVE. In this way, the entity type therefore is effectively "projected" onto the virtual world grid.

In other words, each projection restricts the values of one or more variables in the entity's packets. Each variable must either fall into a particular range or equal a particular value.

To send a packet a host simple determines which projection's requirements are satisfied by the packet and transmits the packet to the corresponding multicast address. The restricted interest language lends itself to efficient processing at the transmitting host because each projection specification can be processed using a simple table-driven interpreter. In many systems, the set of projection filters can be specified statically, meaning that the projection analysis can be complied directly into the application. Moreover, the reliance on extrinsic multicast filtering means that there are generally fewer active projection filters than client-specific intrinsic filters.

A client host subscribes to the se of projections whose information correspond to

data of local interest. This subscription is performed by comparing each projection's interest filter with the client's intrinsic filter. If the two filters have a non-empty intersection, then the client subscribes to the projection's multicast address. As the client's interest changes, it dynamically adjusts the set of outstanding projection subscriptions.

Basic projections may be extended into two ways: aggregations and hierarchies. Instead of transmitting data directly to the projection's multicast group, source hosts can send data to an appropriate ***projection aggregation server***. Each server is responsible for managing one or more projections. These servers are responsible for collecting data for each projection and transmitting aggregated packets that contain multiple individual updates. The resulting ***projection aggregations*** consume less network bandwidth than standard projections. They are most appropriate for augmenting CVEs that use extrinsic filtering (group-per-entity or group-per-region) multicast approaches because they provide a low-bandwidth way to subscribe to a large set of entities.

Projections can also be organized into hierarchies [SiCh96]. For example, given projections that filter entities based on type and by location, higher-level projections may represent groupings of smaller projections. Consider the projections given previously: One including tanks located between (10,25) and (35,40) and a second including card located between (85,70) and (110,85) in the CVE. A "parent" projection may be defined which contains all vehicles located between (10,25) and (110,85) in the CVE. Projection hierarchies are most appropriate for enhancing CVEs that use intrinsic filtering because they allow clients to subscribe to large collections of entities without creating an added

computational burden on every transmitting host for every packet.

Visibility filtering is a rich topic of active research. As we have seen, there is a rich range of options, many of which can be tuned to the particular CVE data characteristics and execution environment.

Chapter III

3 The Proposed Mobile Agent-Based Platform (MAP) for CVEs

The term "Agent" is associated with various expectations and is used in many different contexts. Thus no unique (software) agent definition exists today. However, agents can be described by means of several attributes in order to distinguish them from "ordinary" code. Note that not all of these attributes have to be presented in a given agent implementation. The only attribute that is commonly agreed upon is *autonomy*. Thus, an agent can be described as a software component that performs a specific task autonomously on behalf of a person or organization. It therefore contains some level of intelligence, ranging from predefined rules to self-learning Artificial Intelligence.

The notion of *Mobile Agent* was established in 1994 with the release of a white paper by White [Whit94] that described a computational environment known as *Telescript*. In the environment, running programs were able to halt their execution and transport themselves from host to host in a computer network, resuming execution at the new host.

Mobile Agents are programs that are sent from a client computer and transported to a remote server for execution. A mobile agent contains both the necessary state information and the executable code for the agent to complete its mission. Mobile agents are autonomous because once they are invoked they will autonomously decide which locations they will visit and what instructions they will perform. The behavior

is either defined implicitly through the agent code or alternatively specified by an, at runtime modifiable, itinerary. This approach to computer communication is known as the Remote Programming Paradigm.

Mobile Agent systems are quite different from traditional distribution systems prevalent in industry today. These systems may be characterized as providing local interaction for communication agents, and providing facilities for mobile logic and data. It is this distinction that makes mobile agent particularly useful. In contrast to a mobile code system, for example a Java applet, only the logic is transferable. In a traditional distributed system, for example CORBA, only the data are transferred as a message. [Todd99]

3.1 Overview of Mobile Agents

The idea of a self-controlled program execution near the data source has been proposed as the next wave to replace the traditional client-server paradigm, as a better and a more flexible mode of communication [DiGa99] (Figure 9) When the client computer has work to do on a remote server, it can send the work there and supervise it locally using an agent, rather than continually sending instructions over the network. The network is called on to carry fewer messages. "The more work to be done, the more messages remote programming avoids [HyAc96]." Presently, the concepts of mobile agents have been applied in the domains of telecommunications, network management, electronic business, etc. [GhVi97, Harr95].

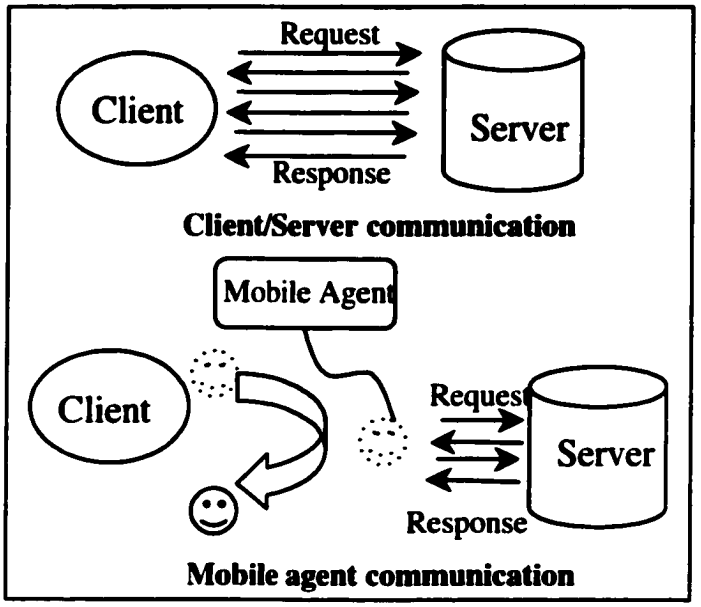

Figure 9 Client/Server vs. Mobile agent communication

The first advantage of mobile agents is performance. The network is called on to carry fewer messages. The performance advantage is dependent on the network. The lower the throughput or available, the higher the performance gain. The second advantage is customization. A mobile agent allows the sender to customize the functionality of the receiver. New procedures can be sent to the server with little efforts. This turns the network into a platform for which new applications can be developed, allowing the network's behavior to be easily modified.

3.1.1 Advantages of Using Java for Mobile Agents

Features of the Java language make it particularly appropriate for Mobile Agents technology. While Java is by no means the only language being employed by Mobile Agents, it is arguably the best choice. The reasons for this are many.

Java's main appeal for agents is its portability. Its use of bytecodes and its interpreted execution environment mean that any system with sufficient resources can host Java programs. There are even machines being built today that execute Java natively. For Mobile Agents this is a tremendous opportunity. The more platforms capable of executing the agents' code the better.

The second advantage comes from the ubiquitous nature of Java on the Internet. Because it is embedded in many Web browsers, as well as application servers, there are many platforms deployed already. Application Programming Interfaces like AWT (Advanced Windowing Toolkit), JFC (Java Foundation Classes), JDBC (Java Database Connectivity) and Java3D are leading toward even more deployment of Java. Additionally, this deployment exactly targets the sort of services that agents can best use.

From the technical viewpoint, Java provides features like code and object mobility, which enable a MA system to be easily built on a Java platform. In order to manage code mobility, Java defines the notion of *class loader*. A class loader is an object that is in charge of loading classes from any data sources, disk or network. A class loader is invoked at runtime to translate a class name into the class Java reference. Java also provides an *object serialization* feature [LADO95], which allows instances to be exchanged between different runtimes. This characteristic provides a means for translating a graph of objects to a stream of bytes, which can be sent as a message over a network. Each instance of a class that implements the *serializable* interface can be serialized. De-serialization is the reverse of serialization. A graph of an object can be reconstructed from the byte stream, which includes the serialized graph.

3.2 MAP Infrastructure

3.2.1 Network Architecture

The MAP system involves three layers as illustrated in Figure 10.

1. **Computer network layer (CN).** The lowest layer, or computer network layer may have any number of computers and their arbitrary interconnection topology. Each computer in the network has a unique network address. These addresses may be used to send messages between any two computers by using the network services (LAN or WAN). As there exists a great variety of network communication techniques in regular service, this layer may be considered to be exactly like the real world, with no need for further specification here.

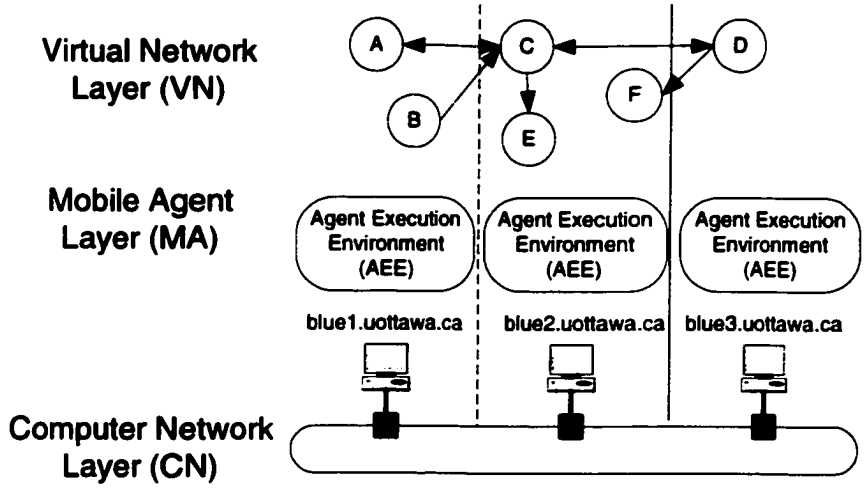

Figure 10 MAP Execution Model

2. **Agent Execution Environment (AEE).** It offers services to the mobile agents who enter it, execute and interact with other agents and local resources. Each host in the system runs an Agent Execution Environment (AEE), AEEs communicate with each

other via communication channels, such as sockets, HLA and etc.

3.**Virtual Network Layer (VN).** It reflects the structuring of information within application area. Multiple logical network nodes may be created on the same AEE node, thus running on the same physical node, and they may be interconnected by logical links into an arbitrary topology. It is the network that is used by mobile agent objects when navigating through the system. Several types of links between nodes are defined in VN such as: bi-directional, unidirectional. The link between the local nodes may be dynamically changed on the fly.

Mobile agents are written in Java programming language. They are replicated each time when MA needs to follow more than one logical link.

3.2.2 Agent Execution Environment (AEE)

The essential part of the MAP framework is the ***Agent Execution Environment***, which offers services to the mobile agents who enter it, execute and interact with other agents and local resources (Figure 11) AEE, which is designed as an RMI server, provides the execution environment where multiple agents –each with its own thread – can be executed concurrently. When an agent or its clone is created, AEE will allocate a new thread for it. Each agent will have a unique identifier. The naming service enables that the identifier is generated through combining the magic number of the AEE, a static counter number, and the local address. This means the identifier will be unique in the system.

In order to guarantee causal interaction requests delivery, a time-stamping scheme

based on a data structure – *State Vector (SV)* – is used in MAP. Each AEE in the system maintains its own *SV*, which has *N* components (*N* is the number of cooperating sites in the system. Assume these sites are identified by integers *0, ..., N-1.*) Initially, *SV[i]=0*, for all $i \ni \{0,....N-1\}$.

After executing an interaction request generated at site *i*, *SV[i] = SV[i]+1*. An interaction request is executed at the local site immediately after its generation and then propagates to remote sites with a timestamp of the current value of the local *SV*. Causality management using the state vector is discussed in section 3.4.

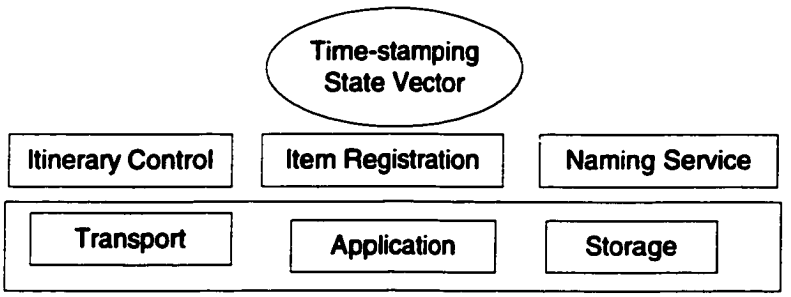

Figure 11 Agent Execution Environment

AEE interfaces with the host system through three services: *transport*, *application* and storage. The services are used to manage storage, transport agents and interface with external *applications*. The storage service lets the AEE access the local recourses such as creating an agent object from a local class file. The transport service lets the AEE access the communications network so that it can transport agents to and from other AEEs. The application service lets the current executing agent interact with other applications on the host computer.

3.2.3 Mobile Agent Object Model

In MAP, the MA objects' model is defined as a collection of Java objects, a standard Java object, the body and the queues of pending requests (Figure 12). The body is not visible from outside of the MA node, which is responsible for receiving calls to the MA node and storing these calls in the queues. Once MA receives a *SV* based request, first it retrieves local *SV* information from AEE, then it performs the causality algorithm, (which will be discussed in section 3.4), to check if the request is a causally ready request. The request will be stored in a *FIFO* queue, if it is a causally ready request, otherwise, it will be blocked and stored in the *FutureList* queue. The body has its own executing thread, which executes these requests in the *FIFO* queue.

The MA object model defines a set of abstractions and behaviors needed to leverage mobile agent technology in Internet-like open wide-area networks. The key abstractions found in this model are *proxy*, *itinerary*, and *identifier*.

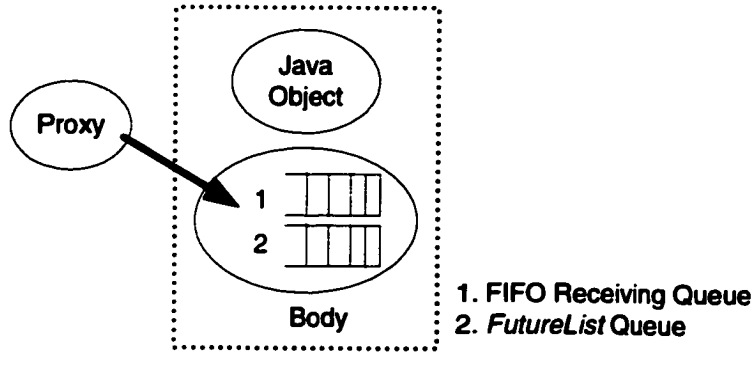

Figure 12 Mobile Agent Object Model

On the side of remote object that sends a call to a mobile agent, this MA is represented by a *proxy*. This proxy is responsible to transform calls into request objects

and perform passive objects passed as parameters. It serves as a shield for the MA that protects the MA from direct access to its public methods. The proxy also provides location transparency for the MA. That is, it can hide the real location of the MA. An *itinerary* is an MA's travel plan. It provides a convenient abstraction for non-trivial travel patterns and routing. An *identifier* is bound to each MA. This identifier is globally unique and immutable throughout the lifetime of the MA.

Behaviors supported in the MA object model include *creation*, *cloning*, *dispatching*, *retraction*, *activation*, and disposal (Figure 13).

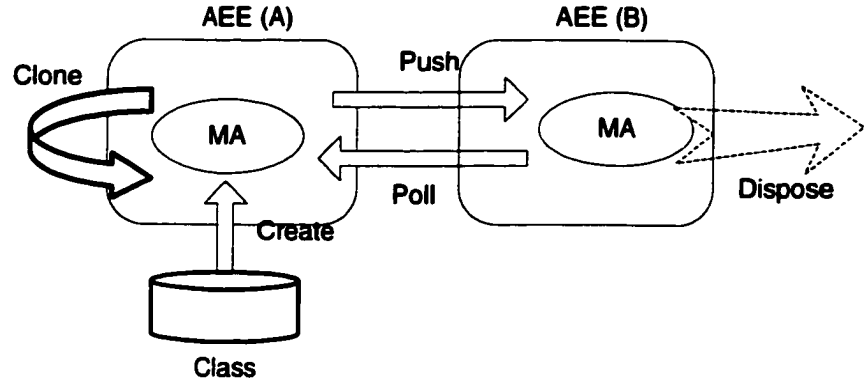

Figure 13 Behaviors in MA object model

The *creation* of an MA takes place in an AEE. The new MA is assigned an identifier, inserted into the AEE, and initialized. The MA starts executing as soon as it has been successfully initialized. The cloning of an MA produces an almost identical copy of the original MA in the same AEE. The only difference is that the cloned MA is assigned a different identifier and execution will restart. Notice that execution threads are not cloned. Dispatching an MA from one AEE to another will remove it from its current

AEE and insert it into the destination AEE, where it will restart execution. MA can be described to be "pushed" to its new AEE. The retraction of an MA will "pull" (remove) it from its current AEE and insert it into the AEE from which the retraction is requested.

Usually, the push scheme works with a predefined itinerary. When the agent is launched, the code is pushed to all stations on the itinerary. So, when the agent instance arrives at a station, the corresponding code is already there and it can start execution immediately. The pull scheme is probably the most popular one, since it is used by web browsers to download Java applets and scripts. When this scheme is used, a station downloads the corresponding agent code from a code repository after an agent instance has arrived. This scheme can be applied to mobile agents with flexible itinerary control. These agents may decide dynamically which station to visit next. The drawback is a decrease of performance. When an agent arrives at a station, its execution has to be delayed until the agent code has been downloaded.

3.2.4 Agent Control

Agent creation (injection) and migration in MAP are accomplished as follows:

3.2.4.1 Agent Creation

3.2.4.1.1 Class-based Creation

Class-based creation allows the user to specify which proxy and body to use for this MAP.

3.2.4.1.2 Object-Based Creation

Object-based creation is much of a convenience used for turning a passive object into a mobile agent.

Sample code for object-based creation is shown as follows:

```
StationaryAgent Sa,
Sa= new StationAgent();
Sa = (StationaryAgent)MAP.turnToActive(Sa, null);
```

The second parameter dedicates the location where the MA will be created. *Null* means that the MA is created on the current node.

3.2.4.2 Agent Migration

3.2.4.2.1 Migration Primitives

MAP currently provides a set of primitives for agent migration.

1. *migrateTo(String location)*

location is a string indicating the location of the remote AEE address.

2. *migrateTo(RemoteNode nodeName)*

nodeName refers to a direct reference to a *RemoteNode.*

3. *migrateTo(Object object)()*

This method is used when an agent tries to migrate to the location of another agent.

3.2.4.3 Itinerary Control

Agent migration in MAP is implemented primarily by Java's object serialization techniques. By using Java's object serialization, it is possible to capture the data and state of the agent as a sequence of bytes. The sequence of bytes can be transmitted over the network and can be de-serialized to come back to the original agent.

When an MA decides to migrate, it calls *migrateTo()* method. Then the byte stream that includes the agent data is constructed by MAP API either directly or indirectly. The serialization mechanism of Java allows the transformation of the object graph into the byte stream. The message is then transferred via the network to the destination station via RMI or socket communication. The de-serialization mechanism is used to recover the object graph at the destination station. AEE has the code information of the object either by a pull or push scheme. The class loading mechanism of Java allows incoming classes on the network to be dynamically loaded.

To summarize, an MA migration to a remote AEE consists of the following steps:

1. Serialize the agent state to byte stream.

2. Send the stream to the destination station.

3. Retrieve the stream at the destination station.

4. Deconstruct the agent at original station.

5. De-serialize the stream at the destination station.

6. Destination station loads the corresponding agent code either by a push or a pull approach.

7. Load the agent class.

8. Create a new thread for the agent's execution and restart it.

3.2.4.4 Automatic Execution

It can be useful for MA object to automatically execute a method upon arrival on a new host or departure from current host. MAP provides a mechanism to choose which method.

MAP.onArrival(String methodName, para) and

MAP.onDeparture(String methodName,para).

3.3 Agent-Based Object Creation in CVEs

MAP approach provides the functionality to dynamically load new modules into a networked simulation. However, how can this method be applied to a distributed environment? The virtual world consists of various types of entities, each of which has a visible body containing its state. In particular, it contains its geometry and physical characteristics. Different entities are characterized by different behaviors, for example, the way they move, fly or walk. Today's client-server interactions require clients' knowledge about each other's application functions and behaviors. One goal MAP aims to achieve is to relieve the client and the user of much of these burdens.

In CVE applications, whenever the client creates a new object in the shared

virtual space, it first creates a Master Agent of the entity at its own host. Then it propagates the mobile agent of the new entity -- "Object Mobile Agent (OMA)" -- to all other clients who are interested in its state information. Figure 14 depicts the OMA propagation strategy. After creating new entity, the clients send notifications out on a predefined multicast channel. All participants who intend to have the created entity send an acknowledgement reply back to the entity creator's system when first discovered. Both "pull" and "push" agent migration methods could be applied here. The creator informs other participants of the URL from which the object OMA can be downloaded and requests them to "pull" the module. Or the creator "pushes" the module to other participants' stations upon the acknowledgement.

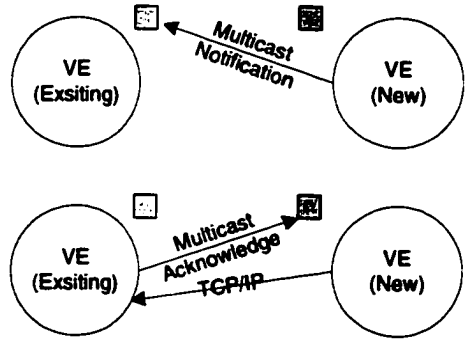

Figure 14 Propagation Strategy

The agent contains the entity's identifier and other information necessary to identify the agent, such as the current owner of the entity, the entity's current status and the implementation of the entity's attributes. The agent would travel to the AEEs on other hosts and would begin execution there. The master agent is responsible for accepting user interaction on the object, sending out updates to remote OMAs. Later, OMAs will take the responsibility of retrieving update messages about the entity from the client

communication interface and access the application reflecting the updates through accessing the AEE's application service (Figure 15).

One of the challenges of building a scalable CVE system is, for *latecomers*, how they will discover the created objects in the virtual world and retrieve the current status of the shared objects. An agent is a solution to the problem. The user who joins the virtual world will send out a query agent to get the status information about shared objects. The query agent would ideally go to the nearest host computer or the one with the fastest network connection and query the AEE about other OMAs that are located there. Through interacting with the AEE and other agents, the query agent will cause the agents representing created entities to spawn a copy and send the copy to the new client's host. The copies of the agents then travel to the host, register with the AEE and begin providing state information. This would take some amount of time but the new client would soon have an accurate view of the virtual world.

Since the OMA encapsulates the implementation of the object's attributes, it may predict the new status of the object without network propagation even though it is interacting locally with other OMAs that reside in AEE. For instance, when an avatar pushes a chair, the position of the chair is changed. In order to reflect the change, the avatar's action and the chair's new position need to be transmitted to other dispersed stations. However, since OMAs on remote stations have stored information about the chair such as its physical properties and the implementation of the attributes, the OMA of the chair can locally calculate the new status of the chair after gathering information about the avatar's action.

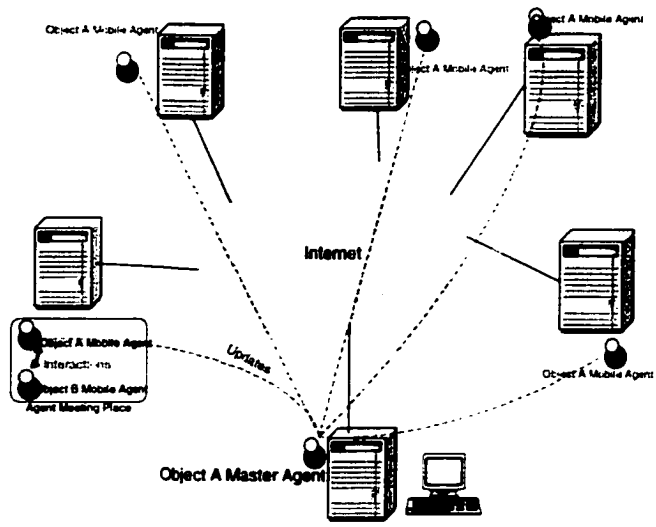

Figure 15 MA-Based Object Creation

To reduce network traffic and minimize network delays, prediction of certain events may be another possible solution. Many events, such as sudden changes in movement, can not be anticipated but others, such as interaction with an object, may be predicted through heuristics, such as collision prediction and knowledge of intent. For example, in a sports arena application, an avatar expresses an interest to interact with an entity by facing it and raising its arms. The type of interaction, for example, "hit" or "pick up" is predicted from the state of the avatar's hand. Based on this method, the effect of interaction may be predicted. Similarly, "dead reckoning" algorithms, that use extrapolation of previous received updated state information to display the new state of an object [DIS], may also be applied.

Most of current CVE systems [CaHa95, Grim91, GrBe96, MaBZ94, DiGa99] lack the ability of injecting a new object that other clients have no information about into the existing shared space. To inject a new object into an executing simulation, the

simulation has to be stopped first, then the implementation code of the object's attributes needs to be installed on all clients to enable the presentation and manipulation of the object. By applying an MA-based solution, MAP offers a flexible means for injecting new objects, regardless of the client's knowledge of the object's implementation.

MAP allows an application builder to create 3D objects and inject them into existing shared scene. The application object consists of three parts:

1. The 3D data description that represents the object in the shared scene, which could be in VRML format or created dynamically from other sources. Typically, it is provided as an URL pointer that tells the client where to download the graphics.

2. The associated scripts, if VRML is used, which accept user interaction and get back to the object.

3. The object side code that implements the object's attributes.

Whenever a new 3D object is injected into the application from one station in the collaborative environment, other stations will download the graphics from the given URL, which is encapsulated in the OMA. Then the OMA takes the responsibility of retrieving the updates of the object and reflects the updates graphically.

3.4 Causality Management

In a distributed virtual environment, the order of interaction requests (events) is often critical. IP multicast doesn't guarantee that messages will be received in the order

they are sent.

Simulations may be split into two classes in terms of causality: discrete and continuous. A discrete simulation uses the *before-after* relation as the basis for the concept of time. Thus causal ordering may be preserved by the definition of the *before-after* relation. This technique is usually applied where the causal consistency is of greater importance than real time *human-in-the-loop* interaction. A continuous simulation is more appropriate where human-in-the-loop interaction is important and causal relationships may be ignored. This technique uses the concept of periodic clocks and the clocks' behavior may be considered analogous to the natural progress of nature time. These periodic clocks are termed wall clocks and are synchronized. Clock synchronization over general wide area networks is only possible using radio clocks or knowledge of routing delay bias. These are provided by the Network Time Protocol [NTP]. It is therefore proposed that a clock server at each local area network synchronizes in accordance to NTP.

Definition: Causal ordering ($\Rightarrow$): Given two interaction requests I_a and I_b, generated at sites i and j, then $I_a \Rightarrow I_b$ *if*:

1. $i = j$, and the generation of I_a happened before the generation of I_b, or

2. $i \neq j$, and the execution of I_a at site j happened before the generation of I_b, or

3. There exists an operation I_x, such that $I_a \Rightarrow I_x$, and $I_x \Rightarrow I_b$.

MAP uses causal ordering to describe the "*before-after*" relationship between two interaction requests.

To capture the causal relationship among all interaction requests in the system, a time-stamping scheme based on a data structure – *State Vector (SV)* – is used in MAP. Let N be the number of involved sites in the system. Assume that sites are identified by integers $0, \ldots, N-1$. Each site maintains an SV with N components. Initially, $SV[i]=0$, for all $i \in \{0,\ldots.N-1\}$.

After executing an interaction request generated at site i, $SV[i] = SV[i]+1$. An interaction request is executed at the local site immediately after its generation and then propagates to remote sites with a timestamp of the current value of the local SV.

Conditions for executing remote interaction requests: Let I be an interaction request generated at site s and time-stamped by SVo. I is causally-ready for execution at site d ($d \neq s$) with a state vector SV_d only if the following conditions are satisfied:

$SV_o[s] = SV_d[s] + 1$, and

$SV_o[i] \leq SV_d[i]$, for all $i \in \{0,\ldots.N-1\}$ and $i \neq s$.

The first condition ensures that I must be the next interaction request in sequence from site s, so no interaction requests originated at site s have been missed by site d. The second condition ensures that all interaction requests originated at other sites and executed at site s before the generation of I must have been executed at site d.

Altogether these two conditions ensure that all interaction requests, which causally precede *I*, have been executed at site *d*.

It can be shown that if a remote interaction request is executed only when it satisfies the above two conditions, then all interaction requests will be executed in their causal orders, thus achieving the causality management of MAP.

3.5 Ownership Management

One of the attributes of an object in the virtual worlds is its ownership, which indicates which user (process) is allowed to access or manipulate the object. The manipulation is performed through modifying some attributes of an owned object (such as position and rotation). In the MAP implementation, a given object instance has one single owner at any given time. However, attribute ownership may change during the execution. This mechanism allows two or more users to handle the same object through the transfer of ownership within the CVE. MAP provides two means for ownership management: object-based and object attribute-based.

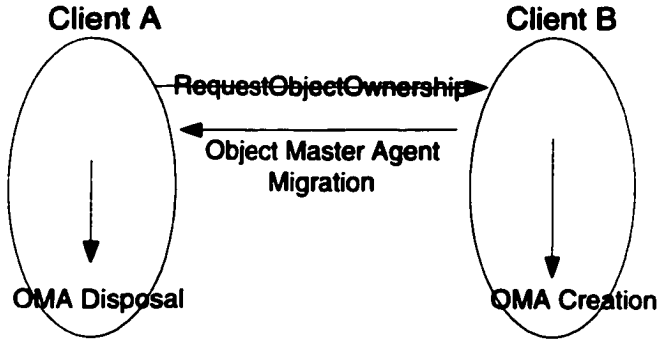

Figure 16 "Pull" Mode Ownership Interaction

The MAP provides a facility for exchanging attribute ownership among clients in a session execution using a "push" and/or a "pull" model. A client can try to give away responsibility for attributes of an object instance, or push ownership. Alternatively a client can try to acquire responsibility for one or more attributes of an object instance, or pull ownership. The push model cannot thrust ownership onto an unwitting host. Similarly, the pull model cannot usurp ownership.

The preceding diagram (Figure 16) depicts the procedure of two clients passing the ownership of a specified object with one another using the pull mode. First, client A attempts to acquire the ownership of a specified object. After B receives the requisition from client A, if client B grants the requisition, it will release the ownership by migrating the object master agent to client A' site, and create an object mobile agent on its own AEE. After an object master agent arrives at the remote station, client A secures the ownership and, simultaneously, client B releases it.

In contrast, the "push" scheme allows clients to transfer ownership without an advance request. Take a basketball game for example. After user A passes the ball to user B, A no longer has any control of the ball, and user B is certainly the new owner of the object. Therefore, to minimize the network delay of sending ownership requests, the master agent of the object will migrate to user B's host.

3.6 Joint-Manipulation Management

A major goal of collaboration is to convey one's ideas and concepts to other participants and to understand the intentions of all collaborators and to reach joint

decisions and actions. Providing meaningful real-time manipulation of a single object among multiple geographically distant users is a significant research challenge given the limitations of today's CVE technology, and solutions will inevitably require significant usage of both network and compute facilities; knowing in advance that the control of single shared object is in fact passed back and forth between individual users allows the system to focus its network resources on those objects that might be genuinely manipulated simultaneously.

To describe the approaches for joint-manipulation management clearly, an interaction model is first introduced.

3.6.1 Interaction Model

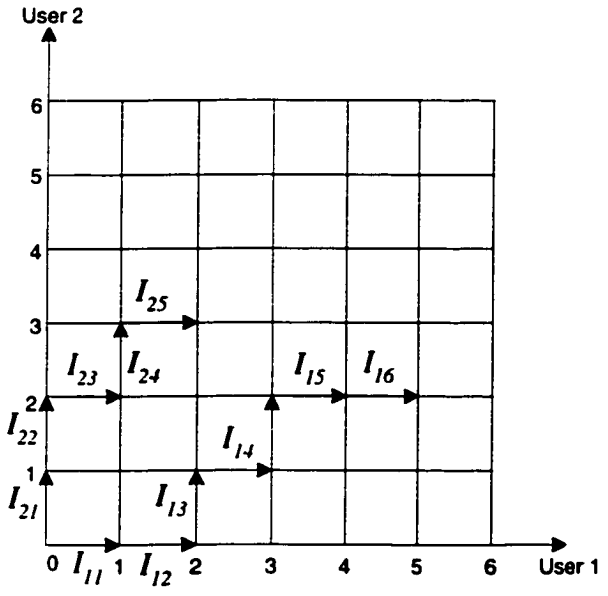

Figure 17 Interaction Model with two users manipulating a shared object

User interaction requests can be represented as grid-based diagrams. The axes

represent users, grid points representing states with a certain state vector, and arrows representing interaction requests. The above figure 17 shows a set of eleven interaction requests, with six from user1 and five from user2.

3.6.2 Constraint Based Interaction Request Resolving

In a virtual space, the state changes of a shared object are physically the summary of users' operations that are imposed on it. When a few interaction requests are combined with a high degree of freedom, the concept *constraints* proposed by Wolfgang Broll [Brol95, Brol95a] could be a good approach to realize joint manipulation. The basic idea of the constraint is that the object state is changed by a set of independent interaction requests; each collaborator only contributes one kind of interactions. This can be illustrated by the following example: an object in 3D space has six degrees of freedom: three for the position and three for the direction. One user can fix an object's position by grabbing it. The second user may now turn the object to change its direction, but he/she cannot change its position.

This example can be expressed in the above Interaction Model Diagram. User1 can only control the object's position while User2 has the access of its rotation attribute. Both of the attributes, rotation and position are independent of each other, i.e. one has no effect on the other when it is changed. One disadvantage of this approach lies in the complex nature of *constraint* resolution systems. In some complicated collaborative tasks, one interaction request on an object could be related or affected by other requests. So it becomes a challenging task to disjoint the independent interaction requests on the same object in these situations. The approach of *constraints* leads very quickly to a

situation where none of the interaction requests can be satisfied, because the underlying constraints cannot be resolved.

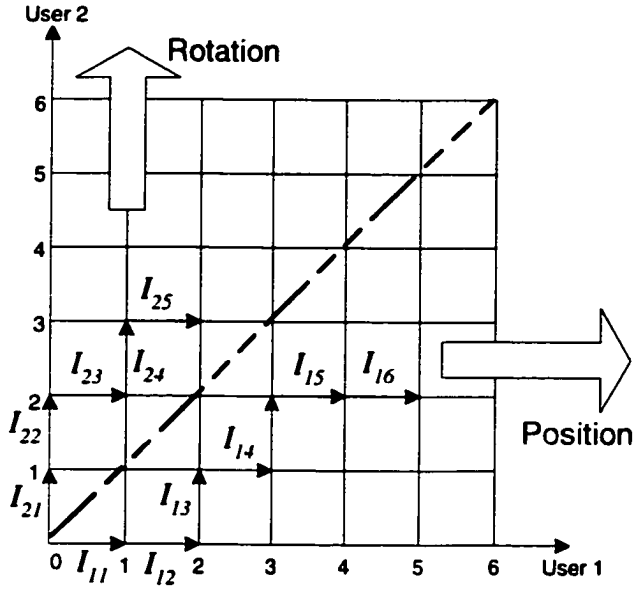

Figure 18 Interaction Model for Constraint Based Request

3.6.3 Synchronized Interaction Request Resolving

Interacting with an object is only possible if some kind of connection has been established. When multiple users intend to manipulate the object jointly, if two or more interaction connections are established at the same time from different users, interaction requests are no longer processed immediately. Interaction requests are sent to a central server, which stores these requests temporarily, processes and combines them, and sends the reflection updates back to all participants. This approach is referred to as *Synchronized Interaction Request Resolving*, since interaction requests from different users are synchronized by a central server and all participants have consistent object

status after receiving the feedback from server.

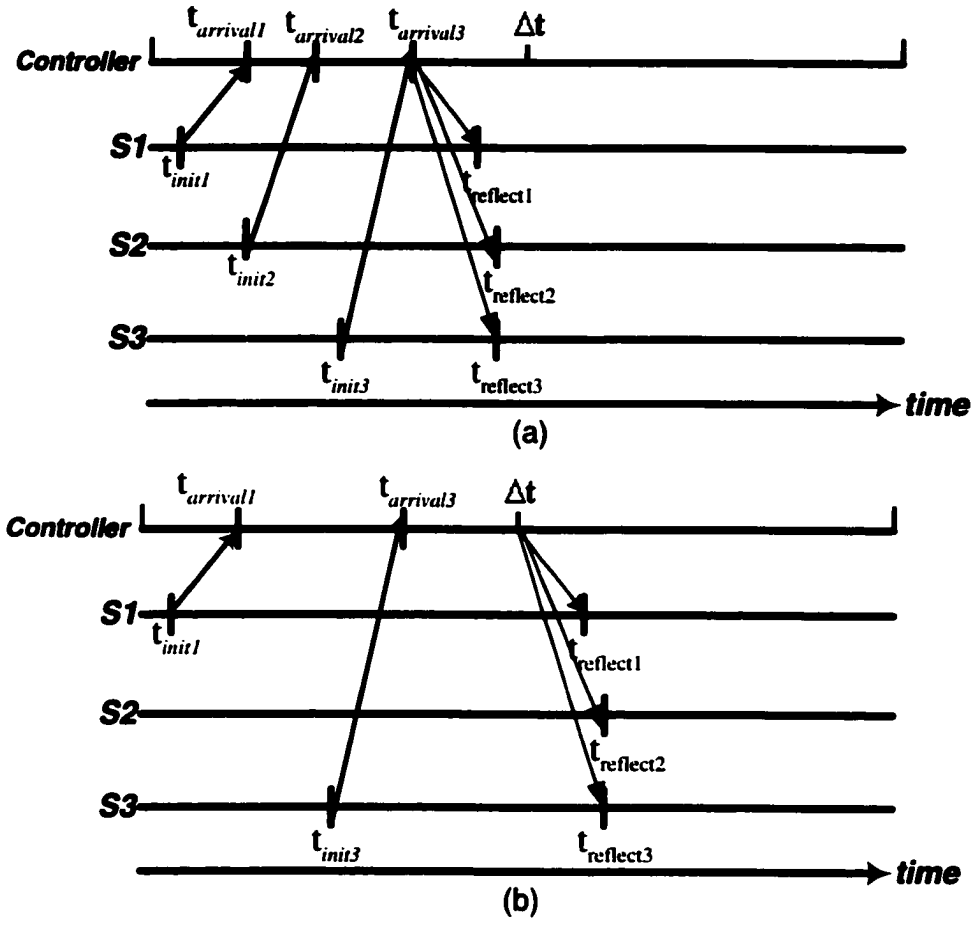

Figure 19 Synchronized Interaction Request Resolving

Figure 19 is an example of Synchronized Interaction Request Resolving with three participating sites and one controller. Interaction requests from participants are sent to the controller site where a timer is used to divide the execution time into time slots(Δt). During the given period Δt, if each participant submits one request (Case (a) in figure 19), the controller will combine then the received requests and respond to the participants. When there are one or more requests from one site during this time period

Δt, further requests will be either ignored or stored and processed after the current interaction. It is hard to imagine that every participant submits its request during the same time slot, since some people need more time than others to read and reflect on a problem. In case (b) of figure 19, participants 1 and 3 submit their requests but 2 does not.

In case (a), the interaction delay (time between interaction requests submission and feedback response) of site *i* is:

$$t_{delay(i)} = 2t_{prop} + \Delta t_{arrival} = 2t_{prop} + t_{arrival(j)} - t_{arrival(i)} \qquad (1)$$

Where t_{prop} is the network propagation delay between site *i* and controller, $\Delta t_{arrival}$ is the time period between the arrival of request from site i and the arrival of the last request (site *j*) at the controller.

In case (b), the interaction delay of site *i* is:

$$t_{delay(i)} = 2t_{prop} + \Delta t - t_{arrival(i)} \qquad (2)$$

Considering both of case (a) and (b), the interaction delay can be expressed as follows:

$$t_{delay(i)} = t_{prop} + \min(\Delta t, \max_{i=1, i\neq j}^{i=n}(t_{init(i)} - t_{init(j)}) - (t_{init(i)} - t_{init(k)})) \qquad (3)$$

Where $\min(\Delta t, \max_{i=1,i\neq j}^{i=n}(t_{init(i)} - t_{init(j)}))$ is the process delay at controller site, the time difference between the start of Δt and combined request response, and site *k* is the first site that submits request to the controller.

To get an adequate feedback of request, it is important to use as short a timer interval as possible. To avoid large interaction delays, site with a very large propagation time can be suspended from concurrent interaction. From the above derivation, the minimal Δt can be determined by:

$$\Delta t \geq \max(t_{init(i)}) + 2\max(t_{prop(i)}) \qquad (4)$$

Thus By determining the individual site propagation time delays, periodically, for all sites, this interval can be updated depending on the current interaction participants.

The synchronized request resolving approach can be expressed as in the above interaction model in Figure 20. User 1, 2 and 3 submit interaction requests I_{1l}, I_{2l}, I_{3l} to the controller respectively within the same time period Δt. The controller combines the three requests to $C(I_{1l}+ I_{2l}+ I_{3l})$ and responds to user1, 2 and 3. Obviously, all the three users have a consistent view of the shared object as the new interaction $C(I_{1l}+ I_{2l}+ I_{3l})$ has been affected.

To avoid large delays, sites with a very large signal propagation time can be suspended from concurrent interactions.

Synchronized Interaction Request Resolving approach is actually based on a client-server model, which allows multiple clients to share common object. A shared object is replicated at all client sites and server. The clients handle user interactions, while the server synchronizes (or combines) the changes made by the various clients.

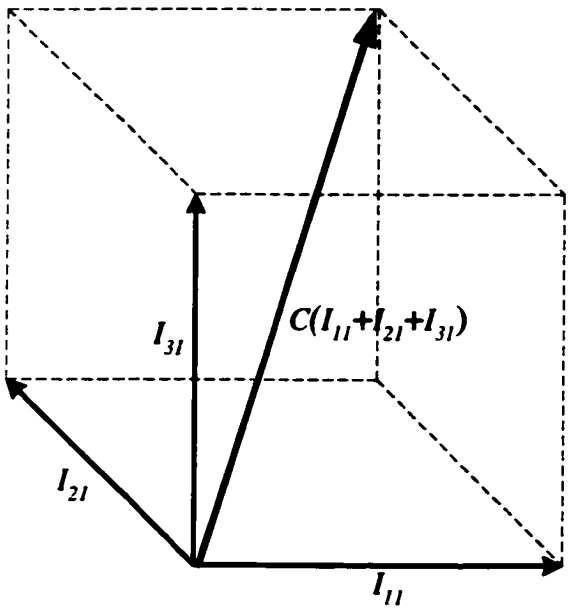

Figure 20 Interaction Model for Synchronized Request Resolving

3.6.4 Asynchronized Interaction Request Resolving

Considering the significant interaction delay of Synchronized approach, a distributed model becomes a straightforward solution for joint-manipulation. Once an interaction request is generated at one participant site, it is executed immediately at the local site and, then sent to (by using networking propagation such as multicasting) all other participants. To illustrate, consider a real-time joint-manipulation session with three sites, as shown in the time-space graph in Figure 21.

There are four interaction requests in this example: request I_{11} generated at site 1, requests I_{21} and I_{22} generated at site 2, and request I_{31} generated at site 3. Generated interaction requests are executed immediately at local sites then propagated to remote sites and executed there upon arrival. We assume that all the interaction requests follow

causal order, for example I_{22} is generated after the execution of I_{21} and I_{11}, then the request generation orders $I_{21} \Rightarrow I_{22}$ and $I_{11} \Rightarrow I_{22}$ will be the same at all sites.

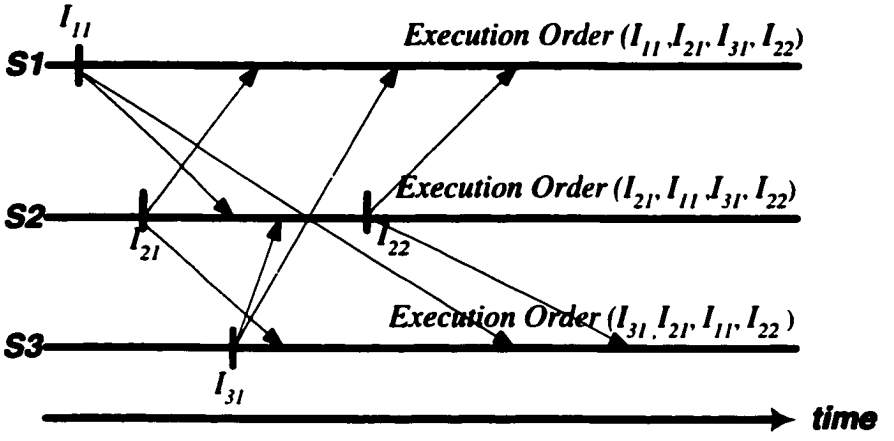

Figure 21 An Example for Asynchronized Joint Manipulation

From the example, we can see that different sites may have different interaction request execution orders:

Site 1: Execution Order (I_{11}, I_{21}, I_{31}, I_{22})

Site 2: Execution Order (I_{21}, I_{11}, I_{31}, I_{22})

Site 3: Execution Order (I_{31}, I_{21}, I_{11}, I_{22})

Following the causal ordering definition in section 3.4, there are four pairs of request that need to be delivered in causal order: $I_{21} \Rightarrow I_{22}$, $I_{11} \Rightarrow I_{22}$, $I_{21} \Rightarrow I_{31}$ and $I_{31} \Rightarrow I_{22}$, since the generation of I_{21} happens before the generation of I_{22}, the execution of I_{11} happens before I_{22}, the execution of I_{21} happens before the generation of I_{31} and the

execution of I_{31} happens before I_{22}. However what is the relationship between I_{11} and I_{21}?

Definition: Independent interaction requests:

Given any two interaction request I_a and I_b, I_a and I_b are independent, expressed as $I_a \| I_b$, *if* neither $I_a \Rightarrow I_b$, nor $I_b \Rightarrow I_a$.

Following the above definition, three pairs of independent requests are found in the example: $I_{11} \| I_{21}$, $I_{11} \| I_{31}$, and $I_{21} \| I_{31}$. Independent interaction requests may be executed in different order at different sites, which may result in the inconsistent view of the shared object to participants if these independent requests are not commutative. In order to make sure the copies of the shared object are identical at all sites at quiescence (i.e. all generated interaction requests have been at all sites), a Commutative Algorithm is proposed as follows.

Commutative Algorithm:

For any two independent interaction requests I_a and I_b, if there is another interaction request $I_a' = F(I_a, I_b)$, and $I_a O\ I_b \equiv I_b O\ I_a'$, where $\equiv$ means the two sequence of requests $I_a O\ I_b$ and $I_b O\ I_a'$ are *commutative* in the sense that when applied on the same shared object state they produce the identical output object state.

In the Commutative Algorithm (CA), an array of executed interaction requests "*Array*" is applied.

Given a causal ready interaction request I, *commutative algorithm* scans the *Array*

to make *commutative* transformation on *I* against any request in *Array* that is independent of *I*. The algorithm is described as follows:

```
CA (I, Array){
For(i=1;i<=n;i++){                    //Scan the Array
if(Array[i] ||I)
then Ic'=F(I,Array[i]);               // Commutative Transformation
}
Execute Ic';
Array[n+1] = Ic';                     //Append Ic' at the end of Array
```

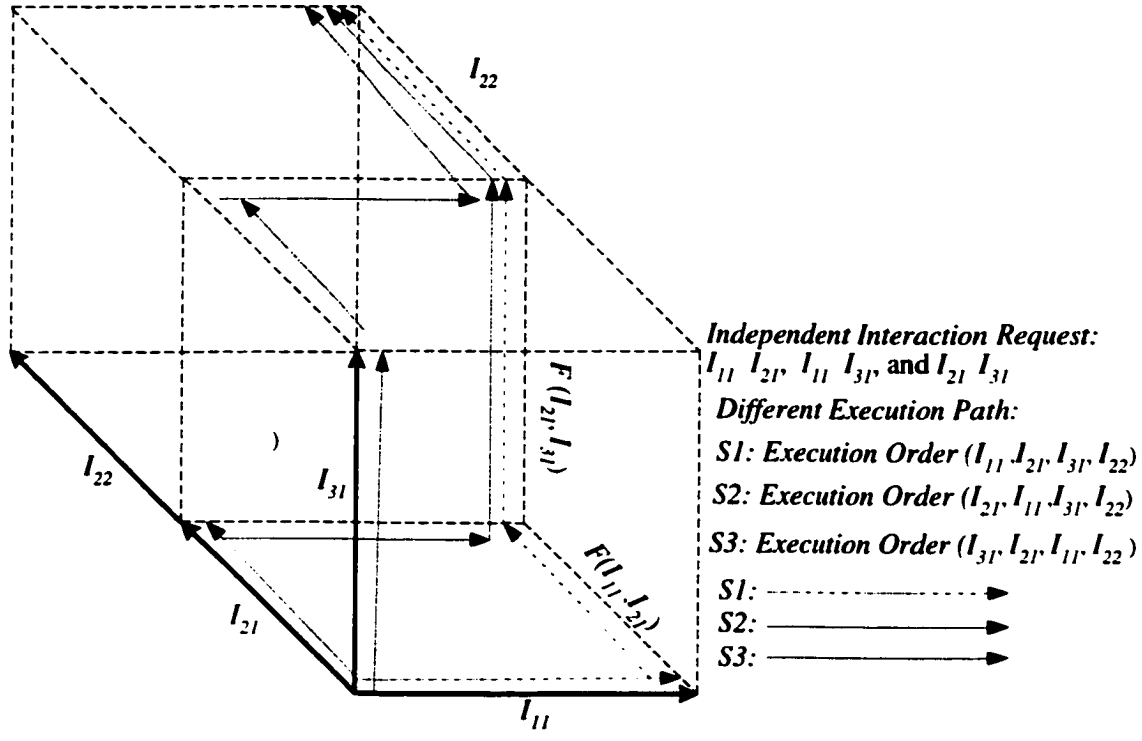

Figure 22 Interaction Model for Asynchronized Request Resolving

Figure 22 illustrates the Interaction Model for Asynchronized Request Resolving approach. At participant 1's site, the execution order of requests is I_{11} I_{21}, I_{31},

I_{22}, thus it first executes I_{11}. When I_{21} arrives, since $I_{11} \| I_{21}$, it is transformed to $F(I_{11}, I_{21})$ according to commutative algorithm then the transformed request is executed. When I_{31} arrives, it is first transformed against I_{11} since $I_{11} \| I_{31}$, to $I'=F(I_{11}, I_{21})$, then I' is transformed against I_{21} since $I_{21} \| I_{31}$ to $F(I_{21}, F(I_{11}, I_{31}))$, and executed. When I_{22} arrives, it is executed immediately since the execution of I_{31} happens before I_{22}.

3.7 Advantages of Using Mobile Agent Objects

There are three primary advantages of using Mobile Agent Objects to implement CVE applications: programming, dynamic properties and workload balancing.

1. Programming

MA object, due to its self-migrating capabilities, is a natural paradigm for CVE systems. Each entity in such a system corresponds to a separate object, which defines its behavior and its potential interactions with other entities. Hence, all programs are written from the point of view of an operating individual. This is much more natural than using communicating processes or high-level declarative programming.

2. Dynamic Properties

With most existing CVE applications, different types of entities that will be participating in a simulation must be set up *a priori*. To introduce a new entity, the simulation programs have to be modified, recompiled, and restarted. But to be more effective, the user should not only have the ability to introduce entities with known behaviors (developed at compile-time) but should also be able to create entities with new

behaviors while the simulation is in progress. One of the advantages of the MA-based paradigm is that it is inherently open-ended in that its functionalities are not bounded by the number of predefined message types to which a node is able to respond. Instead, all functionalities embodied on a MA object program are carried through the network. New entities can therefore be created and injected into the simulation space on the fly as the simulation is progressing. Similarly, existing behaviors may be changed by replacing autonomous objects on the fly.

3. Load Balancing

Self-migration offers a unique approach to load balancing based on the navigational independence of MA objects. This is applicable in situations where each node represents an area of responsibility with which application entities move by re-computing their respective positions. When the new position is outside of a node's area of responsibility, the entities automatically hop to the appropriate node. This can be exploited for load balancing by dynamically adjusting the decomposition of the logical space within entities navigate. When a node's area of responsibility decreases, some of its entities will automatically be forced to hop to neighboring nodes, thus decreasing the workload. This is the subject of the next chapter, the implementation of workload-balanced interest management system.

Chapter IV

4 Workload-Balanced Interest Management over MAP

4.1 Overview

Large-scale distributed virtual environments model the activities of thousands of entities interacting in a shared virtual environment simulated over wide area networks. Originally these systems used protocols, which dictated that all entities broadcast messages about all activities to all other entities, including remaining immobile or inactive ones, resulting in an explosion of incoming messages for all entities, most of which are of no interest. The issue of avoiding broadcast communication has been addressed mainly in the context of real time large-scale simulations where it is termed *Interest Management* [Kath96].

Using interest management filtering mechanisms, some of these systems now allow entities to express interest only in the subset of information, which is relevant to them. *Interest Management* techniques utilize filtering mechanisms based on *Interest Expressions (IEs)* to provide the processes in the simulation with only that subset of information, which is relevant to them (e.g., based on their location or other application specific attributes).

4.1.1 Interest Expressions (IEs)

An *Interest Expression* is a specification of the data that one entity needs to receive from other entities or the data that the entity wants other entities to receive from it

in order to interact with them correctly. IEs may refer to several attributes of the simulation and/or entities. For instance, a soldier needs to "listen" to the commander's "voice" which is critical to him, no matter how far the commander is.

IEs in DVEs may be arbitrarily complex. They may be geographic, referring to some attribute of an entity including its type. A tank may want to know about all ground vehicles within a four-kilometer radius around itself, but a soldier only cares about entities within a 400-meter radius.

IEs may specify a reduced resolution or frequency of data. A wide-area viewer may need data about all entities in the simulation, but not every time new data are generated. Updates every few minutes may be sufficient.

IEs can be generally expressed by the aura-nimbus interaction, which is discussed in Section 2.3(Figure 8 Aura-Nimbus Interaction Model). During the interaction, the data that the entity wants to receive are modeled by nimbus information and the data that the entity wants to propagate to other entities are modeled by aura information. Thus IEs are not expressed as the information that the entities would like to receive but the information that they want other entities to "view" from themselves.

4.1.2 Domain of Interest (DOI) & Domain of Responsibility (DOR)

The terms *Area of Interest* and *Region of Interest* have been used interchangeably to refer to both the data of interest to an entity and to the domain of parameter space for which an Interest Manager (IM) is responsible. To distinguish different perspectives of entities and IMs, Katherine Morse [Kath96] proposed two concepts: *Domain of Interest*

(*DOI*) and *Domain of Responsibility* (*DOR*).

The data of interest to an entity is its *Domain of Interest.* The domain of parameter space for which an IM is responsible is its *Domain of Responsibility.* This terminology also highlights the fact that the data of interest in both cases are a domain in the multidimensional parameter space and not just a geographic area or region, although the geographic interpretation is the most prevalent.

4.1.3 Current trends for area-of-interest management system

In order to reduce the amount of data received by an individual client in large-scale CVE applications, sophisticated area-of-interest management systems need to be developed. A common optimization in these management systems is the use of multicast groups for sending data to a selected subset of all potential receivers. However, its use to date has relied on a *priori* knowledge of a communication pattern between the application and static assignment of multicast groups to these patterns. Current trends for area-of-interest management system are toward:

- ❖ Distributed and dynamic architectures
- ❖ Flexible, general-purpose specification of interest expressions
- ❖ Optimization to improve overhead of the interest management itself, especially through the use of multicast.

4.2 Workload Balanced Interest Management over MAP

4.2.1 Data Distribution Framework in DVEs

Generally, data distribution framework in large scale DVEs can be expressed in Figure 23. This figure shows five stages in data distribution. Each stage is described in the following paragraphs.

Express Interest: Interest is expressed by each entity in the system for the data to be sent and received. Interest is specified as aura and nimbus information (regions).

Cluster: Clustering reduces the number of regions that must be manipulated by combining aura and nimbus regions together. Regions may be clustered (coalesced, combined, unified) by a number of different algorithms [Dong00, EEDQ02, Dam90, HeTa98]. This reduces communication and computation costs related to matching.

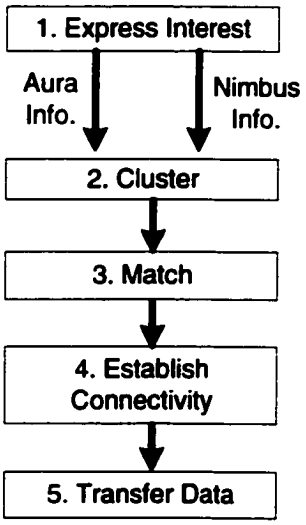

Figure 23 Data Distribution Framework

Match: Matching compares aura regions with nimbus regions to determine overlap.

Establish connectivity: Network connectivity is established based on sets of receiving clients resulting from matching aura and nimbus regions. Several methods may be applied for establishing network connectivity such as receiver-based and sender-based approaches [Hook94].

Transfer data: Entity attributes are transferred from producer to consumer using the connectivity that has been established. All relevant data, as determined from the matching of clustered aura and nimbus regions, is relayed to appropriate receivers.

This research mainly focuses on the third stage of the above described framework – matching. In large-scale DVEs, the aura and nimbus information from an entity are usually related to its current status, for example, an avatar just wants to receive the updates of the objects that are in its visible view. Therefore, in these cases, whenever the entity's status changes, its aura and nimbus information change as well, which results in the relative IM redoing the matching job. However, the computation of the matching results in heavy loads, even more when more entities are created or when these entities update their aura and nimbus areas more frequently.

A novel approach by using mobile agents in hierarchical dynamic interest management is presented in the following sections. The basic idea is that, initially, an Interest Manager (IM) is assigned responsibility for a domain in the parameter space, where the entities' aura and nimbus agents migrate and match the overlap between aura and nimbus information (regions). Aura and nimbus agents are implemented as MA objects, which navigate through the underlying network and interact with other MA objects in their proximity to perform overlap matching. As a result,

connectivity on the network is established, based on the results of the matching stage. As more entities move into that domain, the IM may become overloaded and needs to divide its domain of responsibility with another IM. Then another IM process, which resides at another node, starts and accepts the overload. "Adjacent" IMs on different nodes may also merge when they become under loaded, thereby reducing unnecessary overhead of maintaining extra IMs.

To develop a workload-balanced interest management system using MA entities, the following issues need to be solved:

- ❖ Management of DOR (Partitioning)
- ❖ Mapping of DOR onto logical nodes and physical architecture
- ❖ Creation of concurrent MA objects
- ❖ Coordination of MA objects.

4.2.2 Mapping

An MA-object-based interest management system first should somehow map the DOR, within which MA objects interact each other to a cluster of logical nodes. This is accomplished by partitioning the DOR into a hierarchical structure (shown in Figure 24). The entire domain in the parameter space is partitioned into *N* sub-domains and each of them is mapped onto a logical node in the logical network layer. Multiple logical nodes may reside on a single workstation.

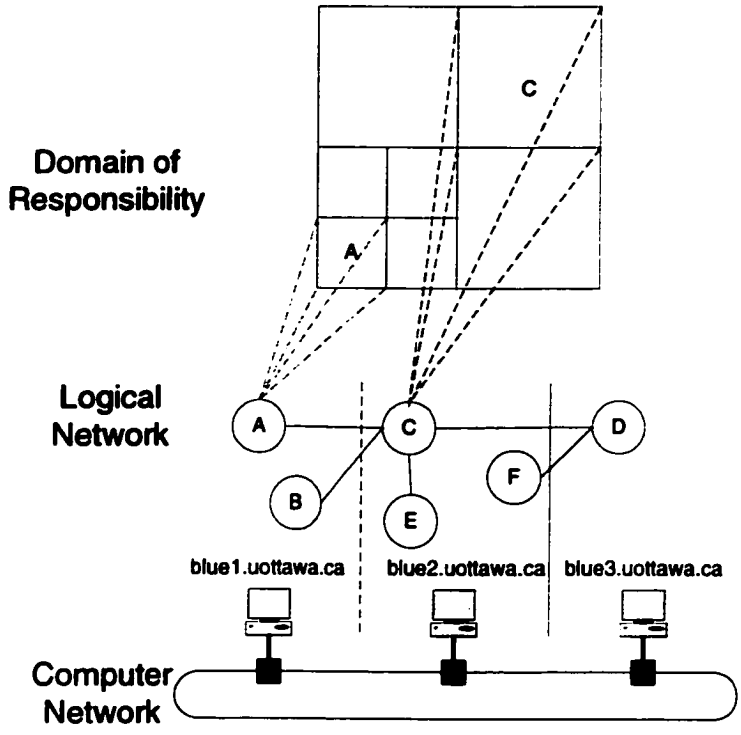

Figure 24 Mapping of Domain of Responsibility to Logical and Computer Network

DOR partitioning is accomplished by using either entity based or space based solutions. Figure 25 depicts an entity hierarchical structure of a military-purpose simulation, where the virtual space consists of objects like vehicles, aircrafts, walking entities and vessels. Each of these object types may include several child types. This hierarchical structure naturally leads to the partitioning of the virtual world as entity based shown in Figure 26.

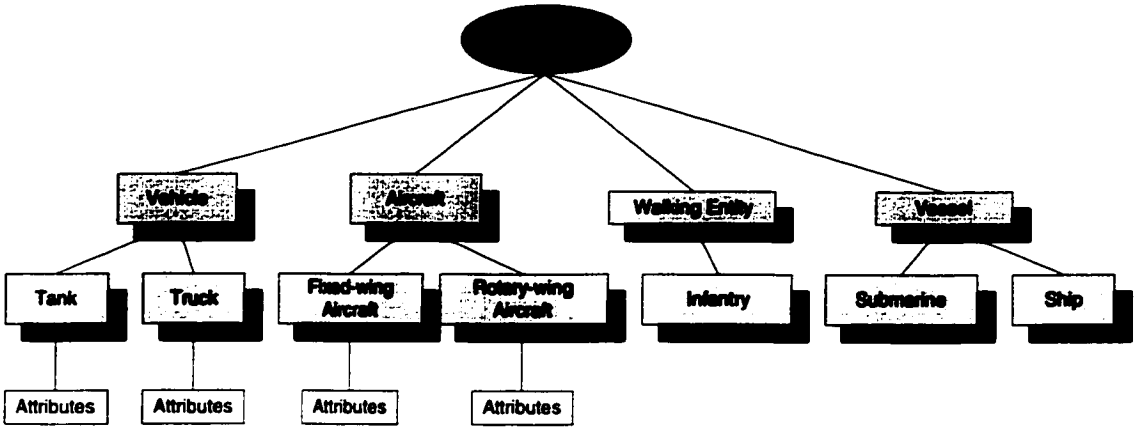

Figure 25 Entity Hierarchical Structures

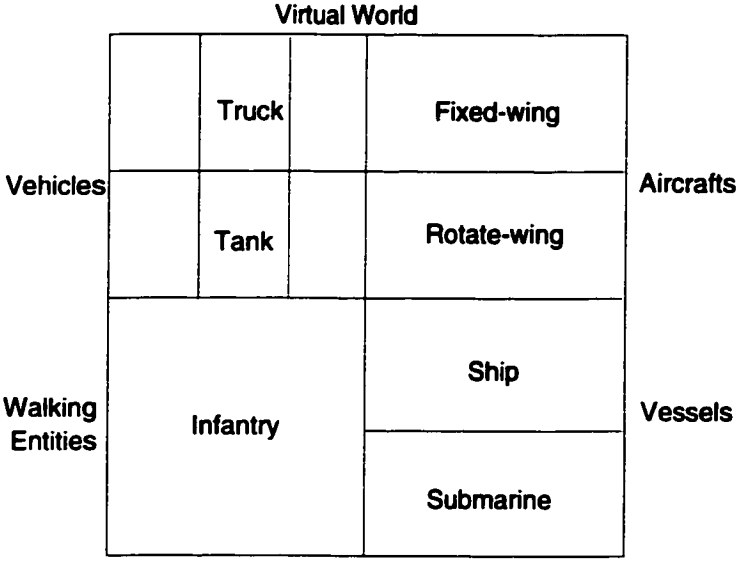

Figure 26 Entity Based DOR Partitioning

In CVEs, position-related information is used to state the users' interest expression more commonly. In that way, the entire DOR can be partitioned spatially as Figure 27 depicts. Figure 27 shows a two-dimensional DOR with two sub-DORs denoted as R1 and R2. For purpose of this example, the maximum depth of the tree is two so that

no further partitioning beyond that shown in the lower right hand quadrant is permitted. Children of DORs are denoted as 0,1,3 and 2 starting from the upper left quadrant and proceeding in a counter clockwise direction. A leaf node in this example is denoted by referencing the sequence of nodes traversed from the root to the leaf. For example, the bottom right child of the bottom right child of the root node is designed 3-3. The quad-tree data structure corresponding to the DOR is shown in the bottom part in Figure 27. The ordering of the children at a node is 0,1,3 and 2 from left to right. Multi-dimensional binary tree data structure can be very efficiently represented by a set of integer codes. The following discussion will mainly be based on spatial DOR partitioning since it is more common in interest management system.

4.2.3 MA Entity Based IE

Aura and nimbus agents are naturally implemented as MA objects, each of which has its own execution thread, identity, behavior and state. Since the agents are implemented in Java programming language, the interest expression can be arbitrarily complex. The agents can be injected into the appropriated DORs (logical nodes at the logical layer) at the time of virtual space creation or they can be injected separately at the time when the entities join the application. The work of matching interest overlaps can be accomplished either by aura agents interacting with nimbus agents or by nimbus agents checking the interest expressed by aura agents in the logical nodes. Figure 28 depicts the aura agent initiated approach. The publication information in the aura agent constitutes its "view" (the circle in Figure 28). Thus, the aura agent only needs to interact with the nimbus agent in its view and check whether there is an overlap between the interest

expressed by the aura and nimbus agents. The following discussion will be based on an aura agent initiated approach.

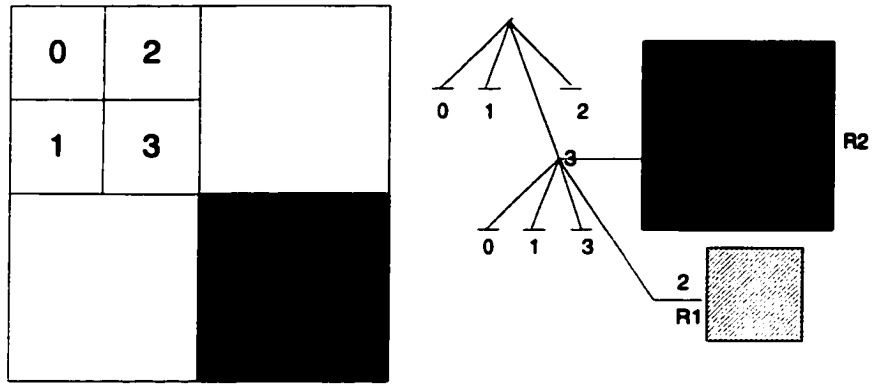

Figure 27 Spatial DOR Partitioning

Figure 28 outlines the sample of an aura agent in MAP. Each agent is designed to "live" for a certain period of time corresponding to the agents in the application. (Line 2 in Fig. 29) An entity can suspend its previous agent and inject a new agent during the simulation time. At each step, the agent determines its new position by calling the function *determineNewPosition* (Line 3 in Fig. 29). The agent may find out the position by receiving a message from its parent entity, which contains the new position. In some cases, the dead reckoning algorithm can be embedded in the agent; the algorithm calculates the new position at each step. It is the function that embodies the intelligence of agent.

After the agent adjusts it new position, it first needs to locate its neighboring nimbus agents. This is accomplished by calling the function *getNimbusAgentNeighbors* (Line 4 in Fig. 29), which returns the list of neighboring agents present in the cells within a certain distance from the agent. Each agent then registers its new position into

the appropriate cell so that the neighbors can use the information later on. This will be discussed in detail in the next section "DOR management".

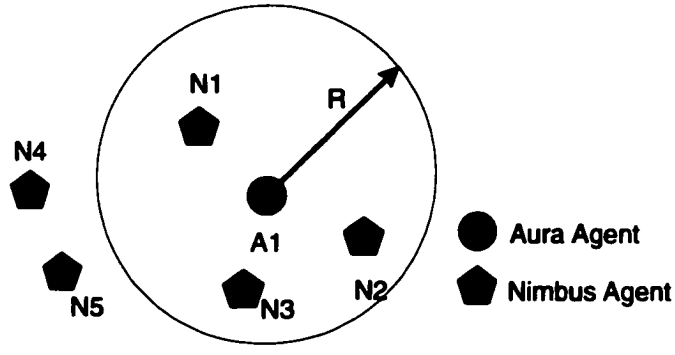

Figure 28 Aura agent initiated approach

```
1     MA_Object Aura_agent(){
2      while (live){
3     determineNewPosition();
4     getNeighborNimbusAgentList(Node logicalNode, myPosition)

4     if(agent is out of domain){
5     getIMReference();
6     moveTo(newIM)

7     MAPActive.onDeparture(String methodName,para).
8     void moveTo(IM newIM){
9     MAPActive.onArrival("registerAuraAgent")
10     }

11    Void registerAuraAgent(){
...
12    }
```

Figure 29 Aura Agent in Workload-balanced Interest Management

Each agent decides for itself which node it executes on at any point in time. These decisions are based on DOR spatial decomposition. It needs to figure out whether it is outside the local DOR, in which case it migrates itself to the appropriate logical node. Lines 4-11 in Fig. 29 describe the migration of the entity; when it belongs to a

cell that is outside of the current node's DOR, the agent migrates itself to the corresponding neighbor using the MAP static function *moveTo*(). Upon the arrival of the new IM, the agent registers itself (Line 11 in Fig.29).

4.2.4 DOR Management

Figure 30 illustrates the DOR spatial decomposition. The DOR can be divided into a number of cells, and the size of each cell is l×w (length×width). Each agent can then exist in a particular cell of the local DOR. By using this technique, an agent can find its neighboring agents more efficiently. For example, assuming that aura agent A wants to locate all its neighbors that are within a certain distance from it as shown in Figure 30, A only needs to "look" into the shaded cells instead of searching the entire local DOR of the logical node.

In MAP, each agent defines its own position in DOR by examining its neighborhood in a given radius, as shown in Figure 31. The radius is usually the interest the agent has expressed. When the entire neighborhood space is local (as is the case with agent A in figure 30), the aura agent can look for the nimbus agent that it intends to interact with within its own node by examining the appropriate local cells. If, however, the circle extends outside the local DOR, it is necessary to communicate with one or more neighboring nodes. For example, agent B on node 5 requires communications with its neighboring node 8. This is implemented by each node exchanging the contents of the boundary information with the corresponding neighbor.

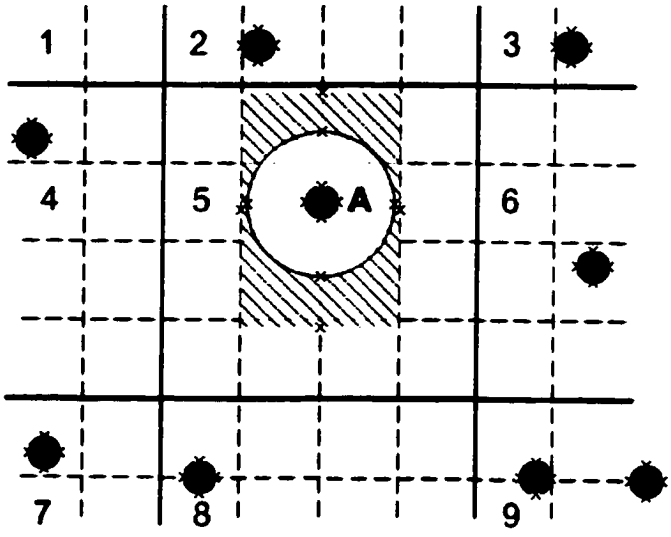

Figure 30 DOR decomposition

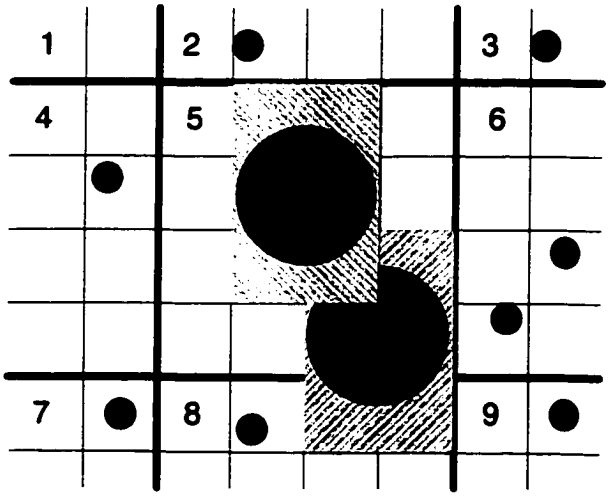

Figure 31 Interaction among logical nodes

Figure 32 shows the exchange of boundary information between node 5 and its neighbors. Node 5 needs to exchange the following information with it neighbors.

- Transfer a to node 8; receive a' from node 8
- Transfer ad to node 7; receive ad' from node 7
- Transfer d to node 4; receive d' from node 4
- Transfer cd to node 1; received c'd from node 1
- Transfer c to node 2; receive c' from node 2
- Transfer cb to node 3; receive cb' from node 3
- Transfer b to node 6; receive b' from node 6
- Transfer ab to node 9, receive ab' from node 9.

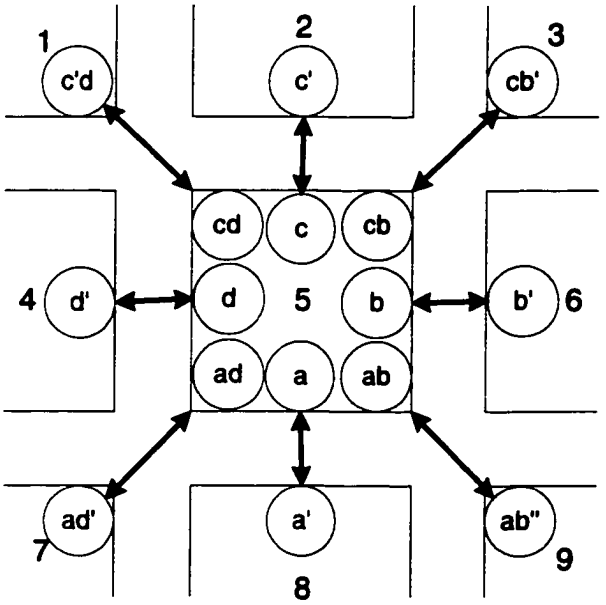

Figure 32 Exchange of boundary information

4.2.5 Synchronization

It is necessary to coordinate agents in a spatial manner to ensure the correct execution of the simulation. Since aura agents can find their neighboring nimbus agents by "looking" into particular cells of the DOR, we need to ensure that the states of the cells only change after they have been "viewed". Figure 33 illustrates the necessity of synchronization among agents. In this figure, nimbus agents A and B should not move to their new positions until aura agent E has "seen" A and before it has seen B. That means E should execute function *determineNewPosition*() before

1. Agent A executes *adjustPosition*()
2. Agent B executes function *registerAgent*()

In MAP, the synchronization among agents is accomplished by using a synchronization variable. This variable is initialized to the total number of agents at the beginning of a time step. Then, at every iteration, each agent decrements this variable by one. If the variable is not zero, it means that the agent is not the last one to synchronize, so it executes function *determineNewPosition*() to find its new position. Then the agent's execution thread goes to sleep until all the agents have completed the same procedure. The last agent to synchronize wakes up the rest of agents. All agents can then go on to adjust their new position. If an agent migrates from a remote logical node, it goes to sleep if all other entities on that node have not calculated their positions yet.

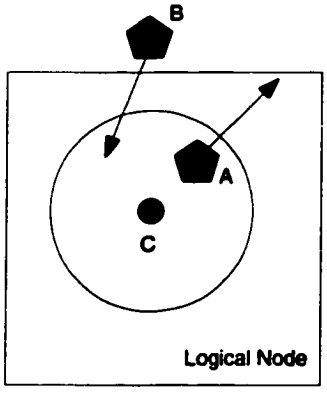

Figure 33 Agent Synchronization

4.3 Hierarchical Structure of Interest Managers

Definition: Workload: Given the logical node N_l, there are a number of p aura agents (a_1, a_2, a_3, ...,a_p) and a number of q nimbus agents (n_1, n_2, n_3, ..., n_q) residing in. Each aura agent has the weight (update/every time step) $Weight_{ai}$ and every nimbus agent has the weight $Weight_{nj}$. Then the workload of logical node N_l is defined as the times of matching between the aura and nimbus agent during the time step.

$$\text{Workload} = \sum_{i=1}^{p} Weight_{ai} \times \sum_{j=1}^{q} Weight_{nj}$$

In MAP, a single IM is assigned responsibility for a domain in the parameter space, where the entities' aura and nimbus agents migrate and match the overlap between aura and nimbus information. (Figure 34(a)). The IM performs a dynamic analysis of the pattern and frequency of agent updates. As more entities move into that domain, if the workload increases to the point that the IM becomes a bottleneck (e.g., when the

workload exceeds a predefined threshold), the IM partitions its DOR into a set of disjoint DORs and asks other stations to create one or more new IMs, to which it assigns those disjoint subsets of the DOR. (Figure 34(b)). The schemes described in section 4.2.2 can be used for DOR partitioning. The process then repeats with the newly created IM(s) monitoring the workload and generating additional IMs as required to keep the overall simulation load on the IMs within bounds (Figure 34(c)). This behavior naturally leads to a tree structure of interest managers, where the DORs are the leaves and the IMs the intermediate nodes of the tree.

In Figure 34, IM0 is overloaded, then it generates three new IMs (IM1, IM2 and IM3). Each of IMs takes the reasonability of a subset of DOR. We can say, IM0 is the parent of IM1, IM2 and IM3. Similarly, IM3 is the parent of IM4, IM5 and IM6. And also, we define that IM0, IM1, IM2 and IM3 are in the same level (their DORs are partitioned by the same parent IM0). It is the same case with IM3, IM4, IM5 and IM6. Thus, the IMs that Figure 34 depicts are in two levels: IM0, IM1, IM2 and IM3 are in level 1, and, IM3, IM4, IM5 and IM6 are in level 2. Notice that IM3 is in both of the levels. When we talk about IM3 in level 1, IM3 means the summary of itself and its children. But IM3 only means itself in level 2.

The parent IM has another task, that is to monitor its children IMs' workload. If the summary of workload of its children and itself is under the predefined value, the parent IM asks the children IMs on different nodes to merge, thereby reducing unnecessary overhead of maintaining extra IMs.

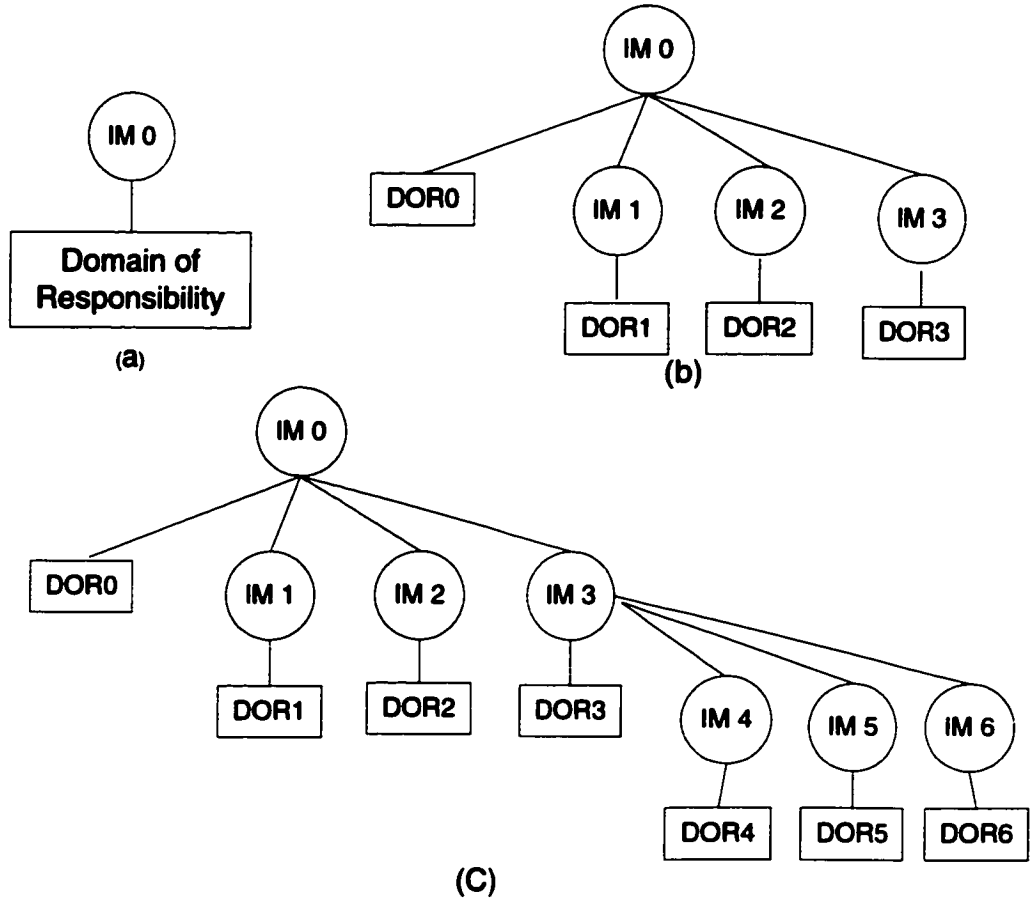

Figure 34 Tree Structures of Interest Managers

4.4 Dynamic Workload Balancing Interest Management

4.4.1 Workload balancing Scheme

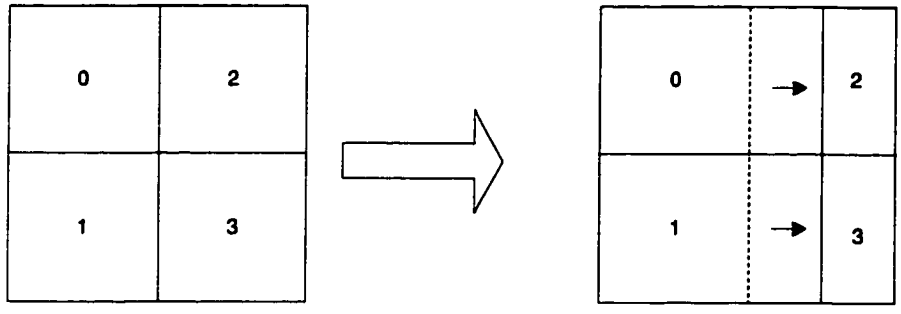

Figure 35 Workload Balancing Scheme

So far, we discussed the hierarchical structure of interest managers. However, in some cases, if the entities are not distributed uniformly in the virtual space, the workloads of IMs are significantly different, which results to having some IMs overloaded. In order to solve this problem, a workload-balancing scheme is developed in MAP. The scheme is only applied for the IMs on the same level, meaning that the workload exchange only takes place among IMs in the same level. In the IMs that Figure 34 depicts, workload exchange may happen at level 1 (IM0, IM1, IM2 or IM3) or level 2 (IM3, IM4, IM5 and IM6).

Figure 35 illustrates the workload-balancing scheme in MAP. The DOR is decomposed into rectangles based on the previous phase (IMs split); each of them is mapped onto a single logical node. Workload is then balanced by nodes, belonging to a particular horizontal or vertical strip, pulling the same local boundaries inward by a certain amount. This scheme preserves the topology of the decomposition. Thus the size and shape of the DOR regions belonging to each node may change, but each

node retains the same neighbors. This permits a simple and efficient communication structure for both the application and the workload-balancing scheme.

4.4.2 Framework of Dynamic Workload Balancing Interest Management System

A framework for dynamic load balancing interest management system using MA objects has been developed. It is based on the adjustment of each IM's DOR and it consists of eight phases, as Figure 36 depicts. The DOR is decomposed into *N* rectangles based on the IMs split-merge scheme discussed in Section 4.3. Each rectangle is mapped onto a single logical node. Each involved workstation might reside multiple nodes.

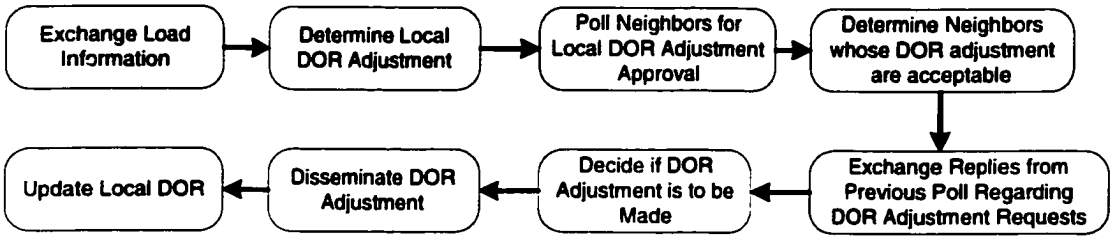

Figure 36 Framework for Load Balancing Interest management

1. Exchange workload information: This phase involves the exchange of workload information with the neighboring nodes. Each node determines its workload value, which is defined in Section 4.3. This workload is sent to its neighboring nodes. These values are then used to detect load imbalances and make DOR adjustments.

2. Determine local DOR adjustment: Based on the workload information gathered about the neighboring nodes, the imbalance factor is estimated in this phase and weighted against the load balancing overhead to determine the profitability of load balancing. If it is advantageous to try correcting the load imbalance, the required local DOR

adjustment is determined. After the neighboring nodes' workload information is received, each node randomly picks a boundary that it shares with one of its lighter node and determines how profitable it will be to move the selected boundary. Workload balancing ***profitability*** is defined by first calculating the average load of the sender and receiver nodes that share the selected boundary, $L_{average} = (L_s+L_r)/2$, where L_s and L_r is the workload of sender and receiver respectively. Next, if the sender node's workload exceeds the average load by a predefined threshold, it determines the excess load that it needs to give to its deficient neighbors,

$$Es = Ls\text{-}Lav.$$

Once the exceeded quantity, *Es*, is determined, the sender node determines how much to move the boundary inward to transfer the excess load to its neighboring deficient node and then proceeds with the third phase of the load balancing.

3. Poll neighbors for local DOR adjustment approval: If a node decides to make local space adjustments in the previous phase, it polls the neighbors whose local DOR would be affected for approval.

4. Determine neighbors whose DOR adjustment request are acceptable: Each node that has been polled by any of it neighbors, decides if the required DOR is acceptable.

In this phase, each node that has been polled by any of its neighbors, decides if the requested boundary adjustment is acceptable. Three cases can arise in this situation:

Case 1: only one of its neighbors has polled the node for the boundary adjustment;

Case 2: the node has polled one of its neighbors and it has also been polled.

Case 3: the node has been polled by more than one neighbors

In case 1, the node first sends a query to the rest of nodes in its row/column, if the node accepts the polling as well, it replies positively to the polling node. Otherwise it replies negatively (Figure 37(a)). In case 2, the node replies negatively to the polling node, as it is anticipating a positive response from the node it has been polled (Figure 37(b)). In case 3 (Figure37(c)), the node replies negatively to one of the polling nodes at a lower level, IM6 in this example. Then the node performs the same procedure for the higher-level polling, as in case 1. If the requests are from the nodes at the same level, it replies positively to the one with the greatest workload profitability.

5. Exchange replies with previous polling regarding DOR requests: Each polled node responds regarding an acceptable DOR adjustment. The replies from all polled nodes are gathered by the corresponding polling nodes.

6. Decide if a DOR adjustment is to be made. Each node decides if a DOR adjustment can be made by examining the responses received from its neighbors. It decides to make an adjustment, if responses from all the polled neighbors are confirmed.

7. Disseminate DOR adjustment: The results of the previous phases are disseminated to all the nodes that might be affected by a DOR adjustment.

8. Update local DOR: Each node updates it local DOR space to correct the load

imbalance.

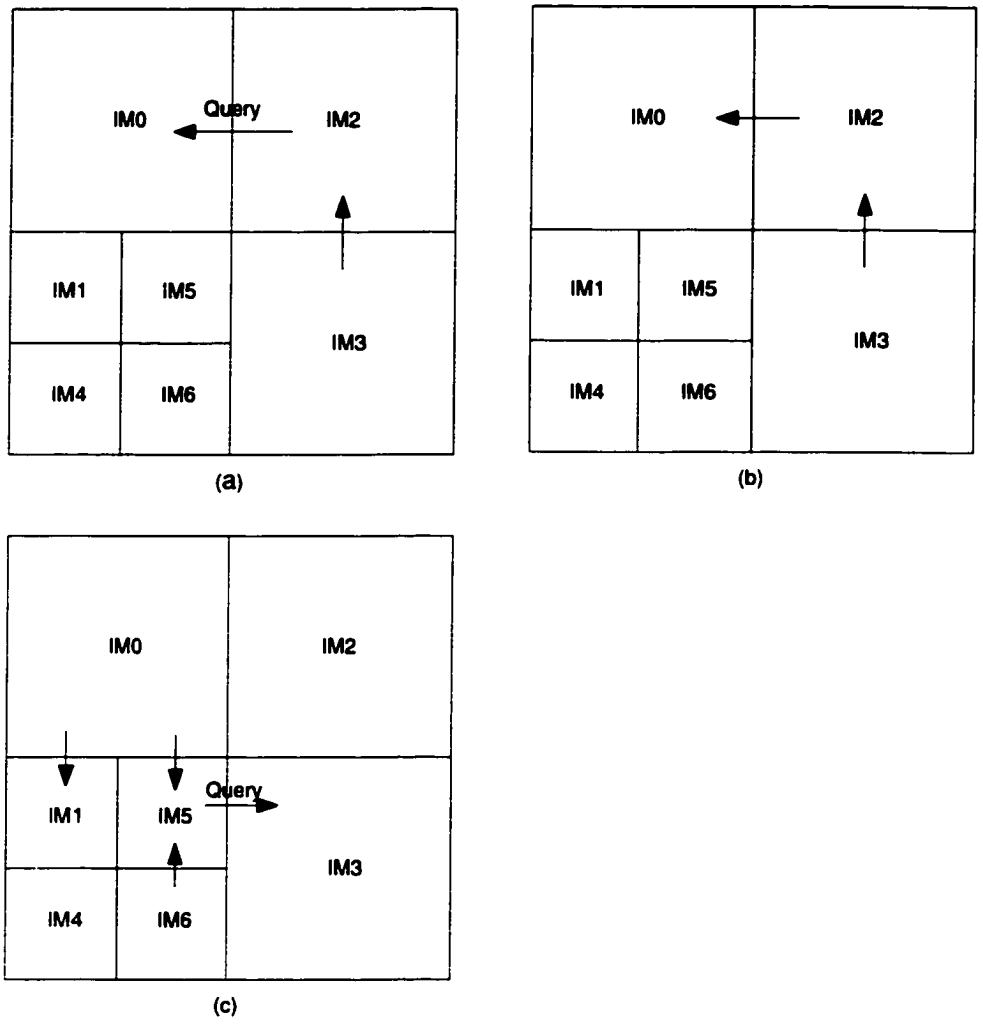

Figure 37 Phase 4 of Framework of Dynamic Interest Management

4.4.3 An Example

Figure 38 illustrates the workload-balancing scheme graphically. The DOR is divided into 7 rectangles (Figure 38(a)). All the IMs first inform their neighbors of their workload level. (Figure 38(b)). Notice that IM1 not only informs IM4 and IM5 of its own workload level, but also informs IM0 and IM1 of the summary of workload of itself and

its children (IM1, IM4, IM5 and IM6). Assuming that IM6 and the summary of IM1, IM4, IM5 and IM6 are overloaded, each of overloaded IMs selects a boundary that it shares with its under-loaded neighbor to move (Figure 38(c)). Assume that, in this example, IM6 and IM1 select their northern boundary to move inward. IM1 and IM6 ask their neighboring nodes IM0 and IM5 if it is acceptable for them to move the boundaries they share with each other (Figure 38(c)). IM1 receives only one polling request, as is the case 1 of phase 4. It first sends query IM2. Assuming that IM2 accepts the polling as well, then IM1 sends a reply positively. But IM5 has polled IM0, the request is initiated by IM1, thus it sends a reply negatively. After sending a positive reply, IM0 makes the rest of nodes sharing its row (IM2) aware of the result it has received from its under-loaded neighbor. This is accomplished by each node propagating its information westward and eastward to the rest of nodes in its row. Finally, IM0 and IM2 move their southern boundary in unison thereby correcting the imbalance and preserving the topology of the logical network. At that time IM6 is still in overload status; it does the same procedure (Figure 38(e)), and the final DOR layout is shown in Figure 38 (f).

4.5 Experimental Results

In this section, we present the experiments of the dynamic workload balancing and IM spilt-merge algorithm described in the previous sections for the interest management system. The simulation is programmed in Java, running on a Pentium 4 workstation.

4.5.1 Simulation Parameters

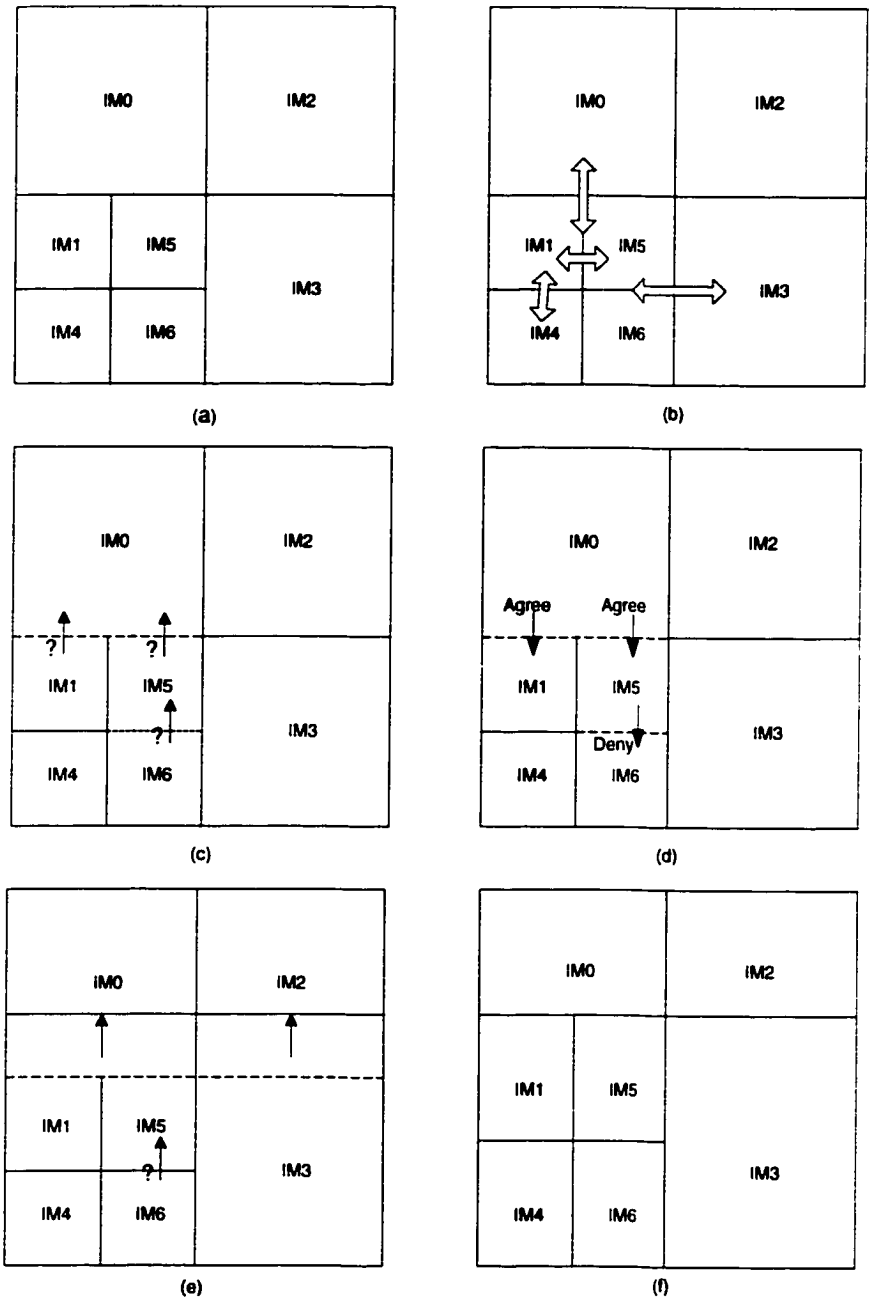

Figure 38 An Example of Workload-Balancing Scheme

The experiments simulate a large virtual world, involving a number of workstations. Each workstation runs one or more entities. Some of the involved workstations may take the responsibility as Interest Managers. The ***capability of IM*** is

defined as the ability that it can perform matching computation between the information that aura and nimbus agents carried, i.e. the number of matching it is able to perform during a time step.

In general, we use three different methods to generate the position of each entity in the virtual world. These methods are:

Uniform Distribution: Let the position of an entity be *(x,y)* and the values of *x* and *y* are uniformly distributed between *(0,Vx)* and *(0,Vy)* where *Vx* and *Vy* are the horizontal and vertical dimensions of the virtual world respectively.

Skewed Distribution: Given the size of the virtual world as *VxxVy*, we divide the number of entities in the system into four equal sized groups, namely, G_i, $i=1,2,3,4$. Let *(x, y)* be the position of the entity in group *Gi*. The value of *(x,y)* is generated in such a way that *x* is uniformly distributed between $(0, \frac{iV_x}{4})$ and *y* is uniformly distributed between $(0, \frac{iV_y}{4})$. Under this scheme, most of the entities will be positioned with the square area defined by the two coordinates [0,0] and $[\frac{V_x}{4}, \frac{V_y}{4}]$.

Clustered Distribution: Given the size of the virtual world as *VxxVy*, we generate entities around $k \geq 1$ clusters. First, we randomly generate k points *(x1,y1) ... (xk,yk)* such that *xi* and *yi* are uniformly distributed between *(0,Vx)* and *(0,Vy)* respectively. Then we divide the number of entities in the system into *k* equal-sized groups, namely, G_1, G_2, ..., *Gk*. For each entity in group G_i, we generate its position in *(x,y)* where

$$x = \begin{cases} 0 & if\ x_i + dx \times \Omega < 0 \\ V_x & if\ x_i + dx \times \Omega > Vx \\ x_i + dx \times \Omega & otherwise \end{cases}$$

$$y = \begin{cases} 0 & if\ y_i + dy \times \Omega < 0 \\ Vy & if\ y_i + dx \times \Omega > Vy \\ y_i + dy \times \Omega & otherwise \end{cases}$$

In our experiments, *dx* and *dy* are generated uniformly between (-1,1) and the parameter Ω depends on the size of the virtual world. For example, we set Ω=0.4 for the 4×4 cells sized virtual world, Ω =3.0 for the 25×25 cells sized virtual world and Ω =28.0 for the 256×256 cells sized virtual world.

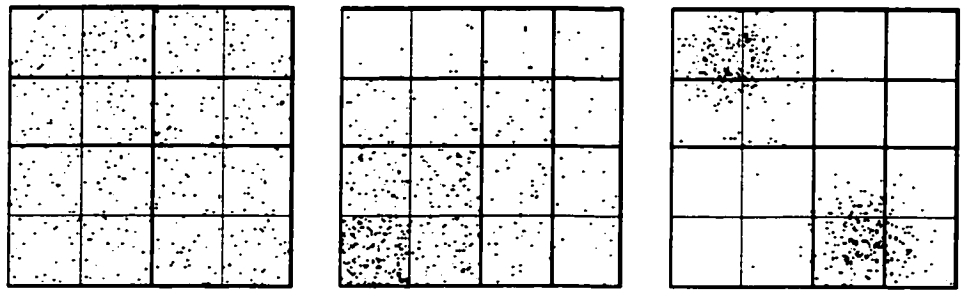

Figure 39 A 4*4 cells virtual world under (a) uniform (b) skewed (c) clustered location distribution

Figure 39 illustrates the virtual world, which is composed by 4×4 cells with the total number of entities equal to 500, under three different location distributions.

Each entity in the simulated virtual space sends its aura and nimbus agent to the corresponding IM according to its position. Aura and nimbus agents contain the interest

information denoted by A_{info} and N_{info}, respectively. The entity's ***weight*** is defined as its moving rate per time step, meaning whenever the entity moves, its aura and nimbus regions change since these regions are related to the entity's current position.

The performance is evaluated by the ***Un-satisfaction Rate*** of entities, which is defined as the following:

> **Definition: Un-satisfaction Rate**: Given the virtual space V within a number of N entities, each entity sends its IE to an IM. The communication link between entities is established based on the matching results. If the entity E only receives the data that it is interested in, E is satisfied with the link. Otherwise, if E receives some irrelevant information, then E is not satisfied.
>
> $$\textit{Un-satisfaction Rate } R = \frac{\textit{The Number of Unsatisfied Entities } M}{\textit{The Total Number of Entities } N} \%$$

example, E moves), but IM does not process it in time due to its capability limitation, or other entities' IEs change, but IM does not process them on time. Both of the reasons result in the fact that entity E receives irrelevant data.

Before giving the results of experiments, let us define the following notations:

N = Number of cells that compose the whole virtual world

P = Number of IMs in the system

n = Number of Entities in the system

E_i = Entity i where $i = 1,2,\ldots, n$

$A_{info(i)}$ = Aura information of entity i where $i = 1,2,\ldots,n$

$N_{info(i)}$ = Nimbus information of entity i where $i = 1,2,\ldots,n$

$Weight(i)$ = Update rate of entity i where $i = 1,2,\ldots,n$

WL_i = Workload in IM_i, $I = 1, \ldots, P$

T= Balance Interval

R = Un-Satisfaction Rate of the whole system

4.5.2 Workload Variation

The first experiment is to validate the necessity of workload balancing in an interest management system. In this experiment, the simulation parameters are set as the following: $N = 256$, $P = 4$, $n = 1000$, $Weight(i) = 1$, $T = 10$. Figure 40, 41 and 42 illustrate the workloads of the four IMs under uniform, clustered and skewed distribution, respectively. In these figures, we notice that uniform distribution is the ideal case, which has the most balanced workload in IMs. The average workload of each of the four IMs is 65,000 approximately, meaning that the number of entities in DOR of each IM is close. Thus, the computation load of each IM is roughly similar.

However, in the cases of clustered and skewed location distributions, the workloads of IMs are significantly different. Figure 41 shows the workload of IMs under clustered distribution. It is obvious that IM0 and IM3 have much heavier load than IM1 and IM2. From figure 41 we notice that both of IM0 and IMs have around 85% load of the entire computation work. Figure 42 depicts the workload of IMs under skewed

distribution. In skewed distribution, about 70% entities are distributed in the area in which IM3 takes the responsibility so that IM3 has 70% workload while the other three IMs only take 30% computation workload.

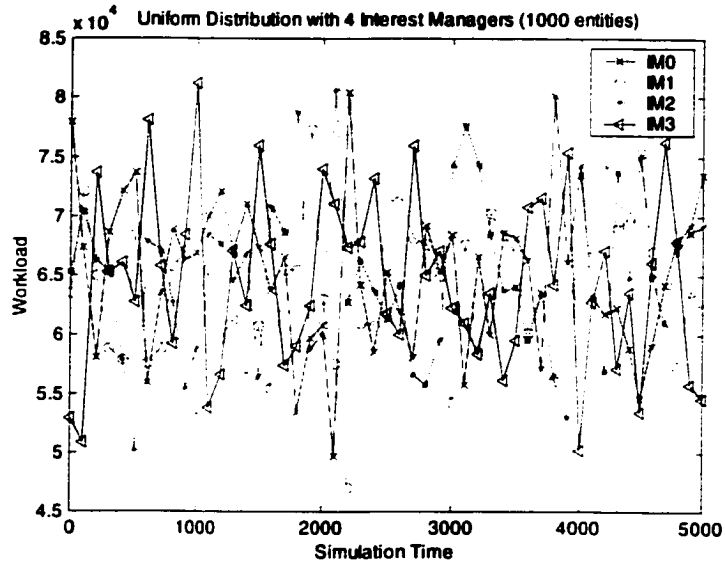

Figure 40 Workload Variation (Uniform Distribution)

4.5.3 Performance Results

For all the experiments, we calculate the un-satisfaction rate. In order to evaluate the efficiency of the workload-balancing algorithm, a 256×256 virtual space is built. There are 1000 entities in the virtual world, each of which has *Weight(i)*= 1. A_{info} and $N_{info(i)}$ are circles with radius R = 45, which means that each entity can "view" 10% of the entire virtual space. T=10.

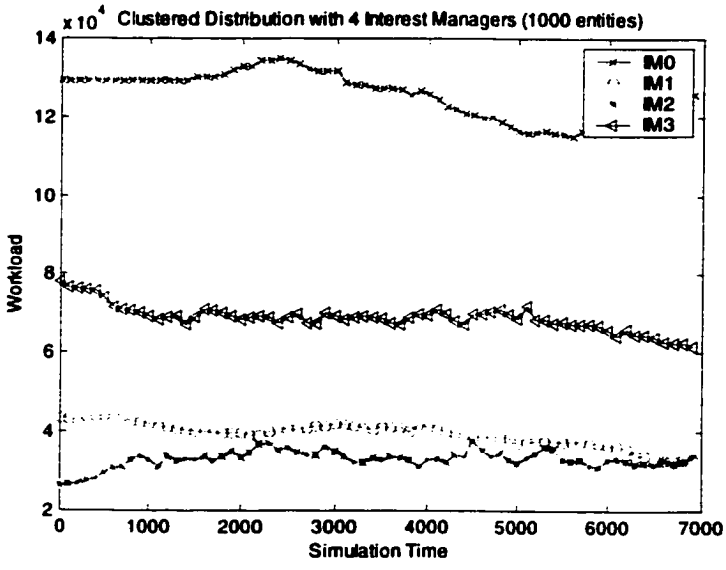

Figure 41 Workload Variation (Clustered Distribution)

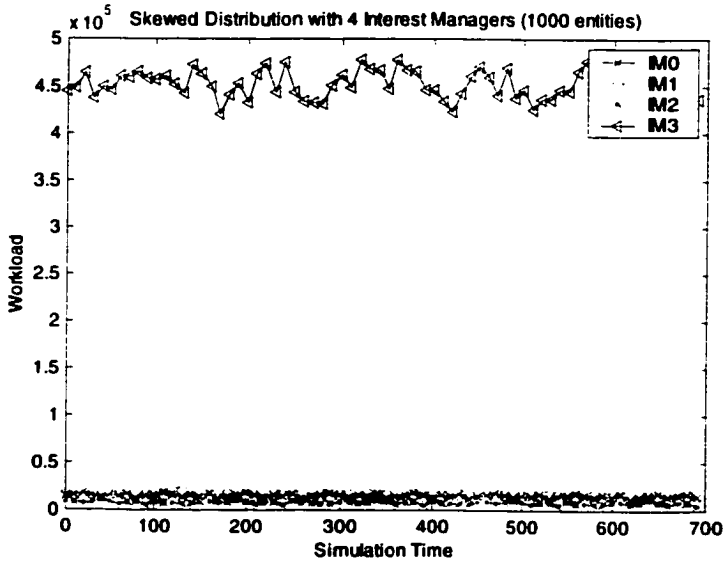

Figure 42 Workload Variation (Skewed Distribution)

Four IMs are initially built evenly, each of which takes the responsibility of 128×128 DOR area with the capability of 62500. The experiment is performed under

skewed and clustered location distributions.

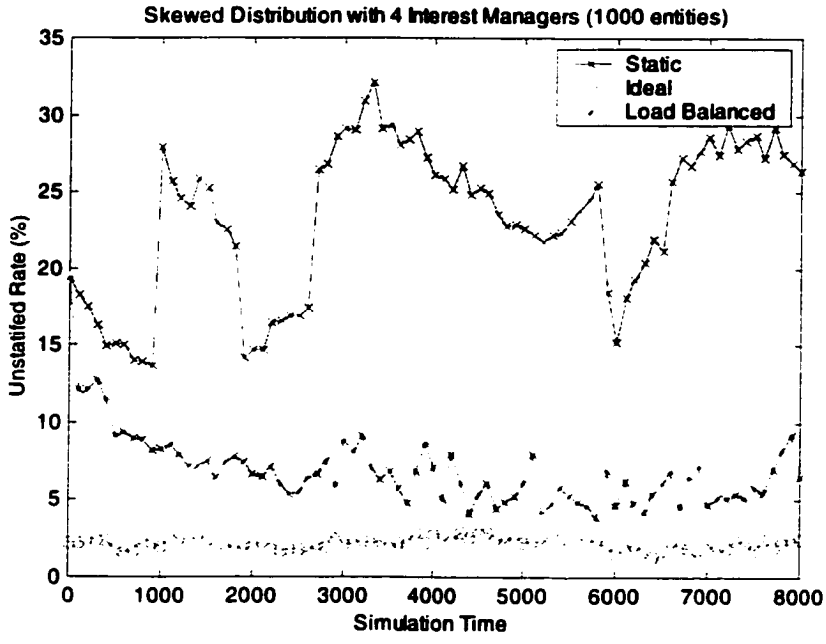

Figure 43 Skewed Distribution with 4 IMs (1000 Entities)

Figure 43 shows the workload of four IMs under skewed distribution with the workload-balancing algorithm, compared with the static case where no workload-balancing algorithm is applied for and the ideal case where all the entities are distributed uniformly at any time in the virtual space. The result shows that our workload-balancing scheme decreases the un-satisfaction rate significantly. In the static case, the average un-satisfaction rate is about 25%, with the maximum rate of 32.4% and minimum rate of 13.9%. When the workload-balancing scheme is applied for the system, the un-satisfaction rate decreases from the initial value of 18.9% to the average value of 8.9%. The goal of the workload balancing algorithm is to balance the workload among involved IMs, however, the algorithm in MAP has its limitation that it is not able to balance

workload perfectly as the uniform distribution shows, which is the reason why it still has a higher un-satisfaction rate than the ideal case.

Figure 44 shows the workload of four IMs with a clustered distribution. In a clustered distribution, entities have the tendency to be distributed in the areas of IM0 and IM3. The figure shows that the average un-satisfaction rate, in static scheme, is about 26.8%, with the maximum rate of 32.4% and the minimum rate of 13.8%. The performance is enhanced with workload-balancing scheme, in which the average un-satisfaction rate is about 8.9%, with the maximum of 18% (the initial point that the algorithm is applied for) and the minimum of 4.2%.

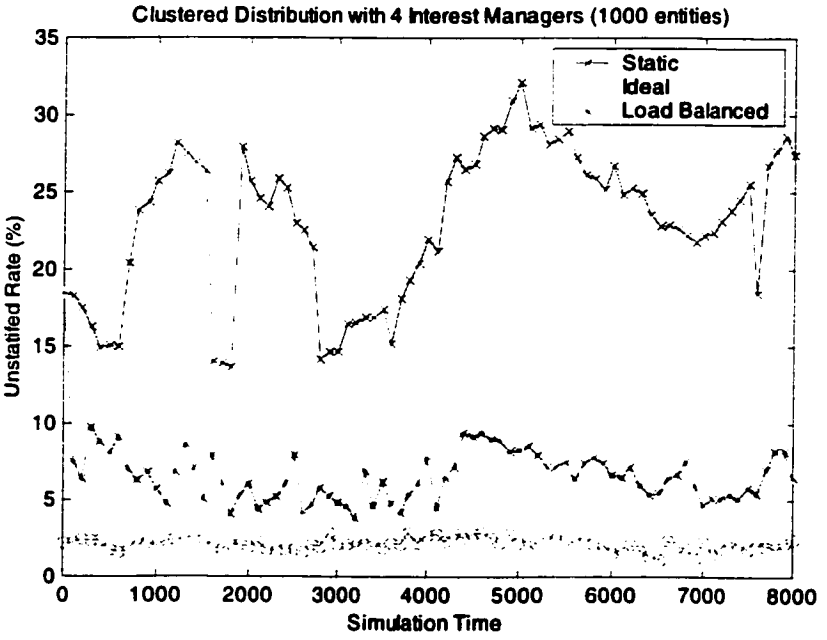

Figure 44 Clustered Distribution with 4 IMs (1000 Entities)

The purpose of the second experiment is to evaluate the efficiency of the interest manager split-merge approach. In this experiment, a 256×256 virtual space is

built. There are 30 entities in the virtual space initially and a single IM with the capability of 900 takes the responsibility of the entire DOR (Figure 46 (a)). Later on entities are injected into the virtual space by 30 periodically. Figure 45 shows the un-satisfaction rates with different number of entities in the system. The un-satisfaction rate is calculated as the average value during the first 20 simulation time steps with a different number of entities. The reason why 20 is selected as the time interval is because the un-satisfaction rate will decrease after the workload-balancing algorithm is applied at 10. That will affect showing the efficiency of the approach of the interest manager split-merge graphically. Since the distribution of these entities follows the rule of skewed distribution, they have the tendency to locate at the left-bottom of the space. When the number of entities in the system is 60, the workload of computation exceeds the capability of the involved single IM. Then, we can see from Figure 45 that the un-satisfaction rate reaches the point 4.5% (point A in Figure 45). At that point, the total workload is 60*60 = 3600, but the capability of IM is 900.

To solve the problem, the involved IM partitions the DOR into four sub-DORs, and three new IMs are generated to take the responsibility of sub-DORs. (Figure 45 (b)). When more entities are injected, the total workload exceeds the summary of four IMs, for example, when the number of entities is 150 (point B in Figure 45), the IM1 partitions its DOR into four sub-DORs again (DOR 10,11,12 and 13 in Figure 46 (c)) and three new IMs are generated to take the responsibility. The same procedure will be performed as the cases of point C in Figure 44 (240 entities, matching with Figure 45(d)) and point D in Figure 45 (390 entities, matching with Figure 45(e)).

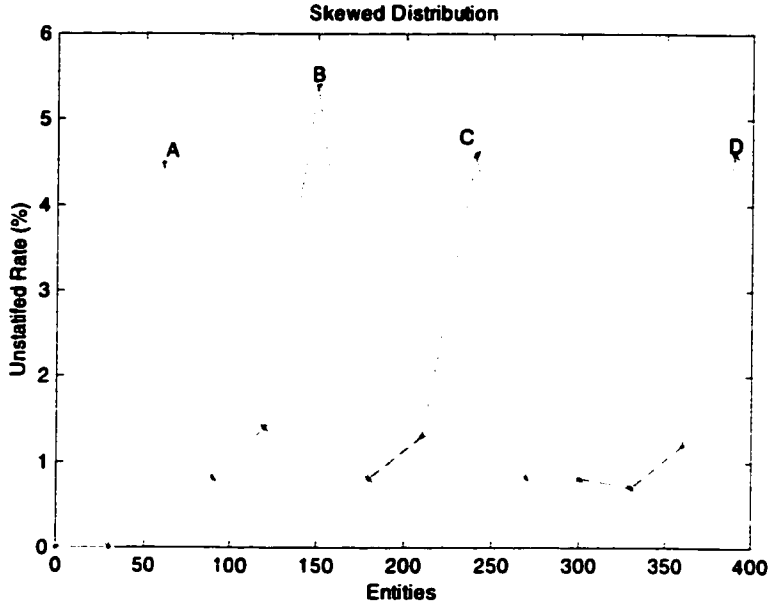

Figure 45 Interest Manager Split

The merge of IMs is a reversed procedure. The parent IM monitors its children IMs' workload. If the summary of workload of its children and itself is under the predefined value (the value of 10% greater than the summary is used in the simulation), the parent IM asks the children IMs on different nodes to merge.

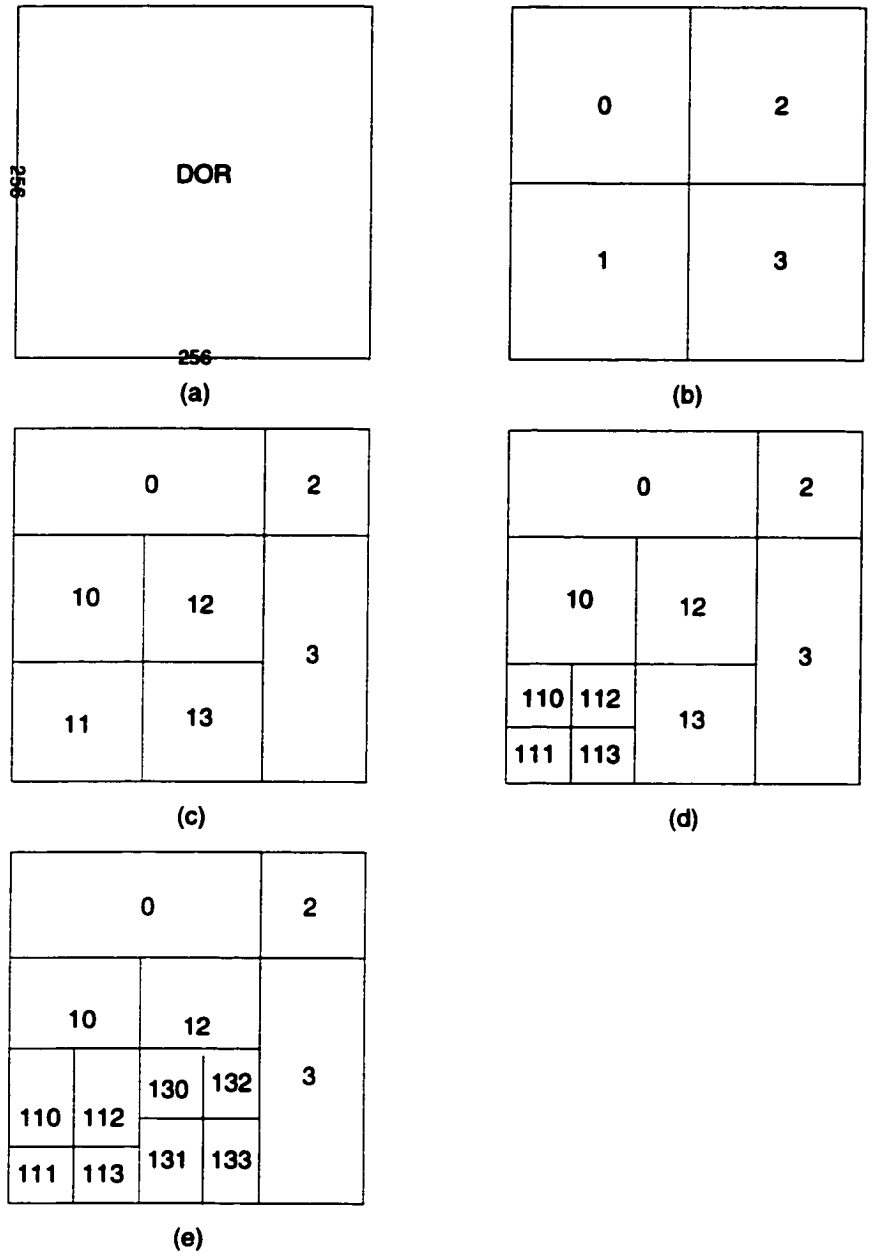

Figure 46 DOR partitioning

Chapter V

5 E-Commerce Application

5.1 Overview

It was hard to imagine ten years ago that people could handle their lives so easily by just logging onto the Internet, using computer networks, to find out all sorts of information they need, evaluating the products or services they need, comparing prices and making a purchase right away. All these things, which would have been completed in more traditional ways before the evolution of the Internet, are being conducted today through the new business tool - Electronic Commerce (EC). Electronic commerce refers to commercial activities conducted electronically, such as the buying and selling of products and services, and the transfer of funds, through public or private digital networks. The goals of e-commerce are to reduce cost, expand business, and improve customer response time and quality. The core of e-commerce, as distinguished from traditional commerce, is referred to as a "fully-digital business." A market is composed of three components: sellers and buyers (or agents), products, and processes [ChWh97]. These components are mostly digitized at the core of e-commerce. E-commerce is not simply an interactive Internet version of traditional retail business. E-commerce, as a new medium of communication, provides more useful information, expands choices, simplifying purchasing processes and lowering costs.

While the actual execution of e-commerce is so different from its real-life counterpart, what shoppers want to experience online, ironically, are characteristic of the

real-life-shopping environment. Existing electronic commerce applications only provide the users with a relatively simple, browser-based interface to access the available products. These applications often lack in the emulation of the social factor. The customers are mainly kept separated and everyone is shopping as if he were in an empty shop. Thus, customers are not provided with the same shopping experience, as they would be in an actual store or mall. Online stores until now largely resemble typical Web pages. Shopping is a social activity people enjoy doing along with friends and relatives. In particular, it is likely that shopping is an activity that is socially facilitated, meaning that when done in the company of others, people engage in it more often and enjoy it more. Marathe [Mara99] states "people don't like to shop in an empty store." To substantiate this opinion, he cites a survey, which shows that 90% of shoppers prefer to communicate while shopping. Participate.com [WaCU00] argues for shopping communities because they "increase stickiness (customer loyalty) [and] viral marketing (word of mouth), reduce cost of customer acquisition, and drive higher transaction levels." Considering the current growth in e-commerce on the Web and the desire to make shopping as easy, natural and enjoyable as possible, it is interesting to extend the way people currently shop on the Web by adding support for more collaboration between customers and salespersons or among customers. Therefore, providing a virtual reality web shopping experience is closer to the experience people have in a real shopping environment.

Considering the innovations in 3D Virtual Reality online store technology, it would not be long that customers would find online malls of the future more like their

offline counterparts. Shoppers would imitate "walking" around the virtual mall using their "avatars" and controlling their moves by pointing their mouse or using simple voice commands. Movement in the virtual mall is very much like the PC game "Doom". It offers a first-person perspective of the virtual environment. If users find items of interest in the navigation process, they can purchase them. Users navigate the virtual commerce world, visit stores and manipulate items therein, only using their browsers. In this section, a CVE technology based solution is presented. With the creation of a virtual shopping mall, simulations of the actual shopping environments and user interactions can be achieved to a great extent.

The virtual e-commerce prototype presented in this paper holds three major advantages over existing online HTML-based e-commerce applications. First, the multi-faceted dynamic virtual reality environment allows merchants to implement stronger marketing and branding initiatives than what is available on a "flat" Web page. The second benefit of the 3-D environment is a longer shopping retention period. According to the survey conducted by ActiveWorlds [ALPHA], the average user session in a 3-D shopping mall is 45 minutes -- an abundance of time to market effectively to shoppers. In comparison, the average user session at regular web site is less than five minutes. The third advantage of this new technology is the enhanced interactivity between merchants and consumers, among customers and visitors. Virtual Mall technology enables online merchants to offer features that are lacking in most of today's e-commerce stores. For example, the Virtual Mall makes it easy for store owners to provide real-time customer support, sales assistance, cross-selling, promotion and individualized care that have

traditionally been proven to improve sales.

5.2 vCOM : E-Commerce Application over HLA/RTI

5.2.1 Overview

The research work called vCOM was part of a larger project, "Enabling Technologies for Electronic Commerce", funded by the Canadian Institute of Telecommunications Research (CITR). The complete e-commerce application scenario, implemented in that project, is shown in Figure 47. The shopping process is as explained below:

- A user logs-in by using the URL of the e-commerce web-based application.
- Upon proper authentication, the user is asked to select an existing user profile or enter a new profile, specifying his/her interests.
- The system will use that profile in helping the user in finding items of interest.
- The user is then presented with the choice of searching the e-commerce world by either database search queries or by navigating virtual shopping malls.
- If the latter is chosen, a list of the available virtual shopping malls is presented and one is selected for navigation. The graphics of that shopping mall are then downloaded from the database server.
- A Quality of Service (QoS) Broker checks available network QoS and may switch connection to another database server if the network QoS there is better.
- The Virtual Mall graphics are presented on the user interface.
- The user can "navigate" through the virtual shopping mall, either by keyboard commands, mouse movements or by voice commands, through a voice recognition/synthesis system.

- A "User Interface Agent" (UIA) monitors the user's actions and modifies his searching preferences accordingly. An UIA is specialized to serve its user and is persistent over time.
- Items of interest may be found and placed in a virtual shopping cart. If, during that process, the user wants to "communicate" with a vendor or jointly examine an item of interest with others, a Collaborative Virtual Environment (CVE) is defined with the vendor's computer through HLA. Avatars are used to represent the buyer and the vendor.
- A similar CVE is defined among buyers who wish to do collaborative shopping. Items of interest to the buyer(s) can be viewed in 3D by retrieving appropriate 3D graphics from the database server.
- Finally, when a buyer is ready to buy and exit, the system brings forward a secure purchasing environment using the Secure Electronic Transactions (SET) protocol.

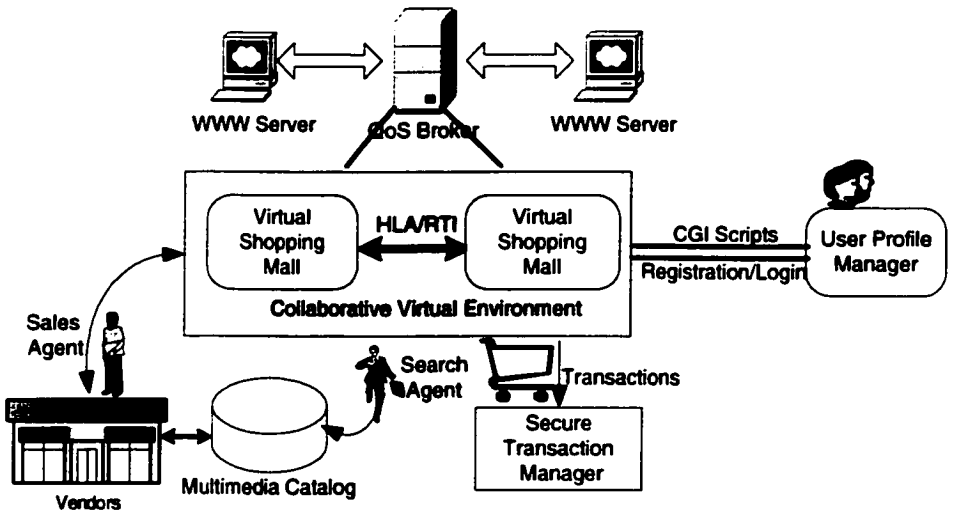

Figure 47 Overview of vCOM Prototype

A user uses its web browser, logs on to a given URL and follows proper authentication/access control. Registration is required, if it is a new user. The user needs to give his/her personal information, such as name, birthday, or credit card

number. The information will be stored in the User Profile Database. The user then may choose the browsing mode that he/she prefers – search engine or virtual mall. A search engine plays a major role in the traditional electronic shopping on the Internet. The user can check all sorts of information he/she wants, evaluate the products or services he needs, compare prices and makes a purchase right away on a "flat" web page. The corresponding application scenario was described earlier.

vCOM has a CVE user interface through which the user can retrieve the virtual shopping mall realized in VRML (Virtual Reality Modeling Language) or Java3D. He/she then can navigate the virtual world, where both the users and agents are represented on the screen by 3D animation avatars. The users can explore the cyberspace through different views, including the "eyes" of the avatar, or from a third-person view. In the third-person view, the user can view his or her avatar from "space", capturing the broad scope of the environment, or from a "wingman" position, capturing a more local and detailed context.

The Architectural components shown in Figure 47 are as follows:

Software:	Multimedia catalog
	QoS broker
	Secured Transaction Manager
	User Search Agent
	WWW server
	Virtual shopping malls (several of them)
Humans:	End users & Vendors (one per virtual store)

The Quality of Service (QoS) architecture that introduces a brokerage function between clients and servers has been defined. QoS brokers continuously monitor the performance of affiliated servers and assign them to clients according to a defined server selection policy that optimizes the capacity of the system based on server performance characteristics and user requirements.

There are a number of data management issues that need to be addressed in e-commerce applications. These include, among others, the development of virtual catalogs, intelligent query access of these catalogs, and support for the management of transactions between customers and vendors. The multimedia catalog enables users to access multiple, distributed and potentially heterogeneous catalogs in a uniform and transparent manner. The user queries the database through CGI scripts. The user search agent is responsible for generating the CGI scripts, parsing the responses from the database, which is in CGI format as well, and showing the results on a query window. The user may pick up his items of interest and add them to his/her shopping cart and negotiate with a vendor's sales agent to bargain for the best price. After that, the Secure Transaction Manager helps the user to conclude with a payment for the transaction.

5.2.2 vCOM System Architecture

The vCOM CVE is a shared 3D virtual world supported by HLA/RTI on the Internet. Figure 48 depicts the basic three-layer system architecture on each client site. The top layer is the graphics-rendering layer, where there are two kinds of browser modes from which the user can choose: a VRML-enabled web browser, i.e. Netscape, or

a Java3D-based browser.

The Java controller layer (middle layer) plays a pivotal role in the creation of CVEs. It not only controls the virtual scenes (VRML scenes) but also communicates with the RTI through the Java Native Interface (JNI). The Java controller layer also implements the services that RTI provides, such as RTI Object Declaration, Object Ownership Management and Data Distribution Management (DDM).

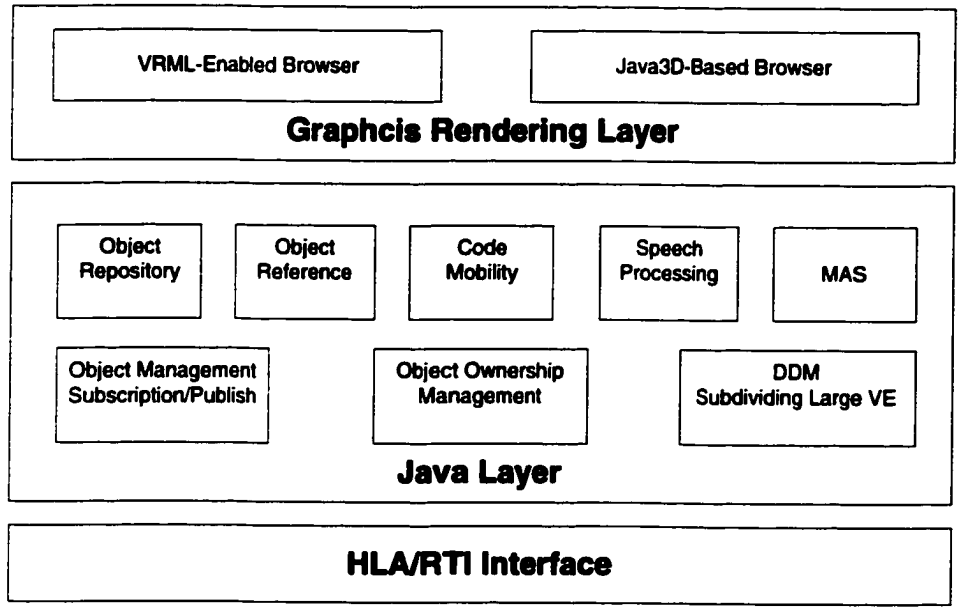

Figure 48 vCOM System Architecture

5.2.3 vCOM System Implementation

5.2.3.1 User Interface Design and Implementation

In the case of a single-user VRML world, the entire description of the world is typically contained in one VRML file. In the case of dynamically changing VRML

worlds, however, this type of organization management is not suitable, since objects are typically added, removed or their attributes changed during the lifetime of the world. Furthermore, since objects can send and/or receive events, the controlling application must be able to identify objects and access them individually. This necessitates some sort of indexing or naming scheme for objects so that events that are received can be expediently routed to the correct object in the virtual scene. For this reason, the VRML scene is considered to consist of a set of self-contained hierarchy of objects. Moreover, interaction with objects is only possible through the event interface of their top-level node. To implement this type of object encapsulation, all the active objects that might be created, removed or modified dynamically are defined by VRML prototype node.

In order to implement such a VRML-based shared virtual space, an event must be propagated to other clients, when it occurs in the VRML scene. The Java layer receives the notification of the event's occurrence by means of a callback mechanism, which is provided by an interface called External Authoring Interface (EAI). EAI defines a set of functions on the VRML browser, so that the external environment can perform to affect the virtual world. The 'callback' returns the values of the event, the (local virtual) time of its occurrence and relevant user information associated with the event. In the typical case, the Java layer sends the event immediately to other clients who are interested in the event. In some cases, however, such as the rapid succession of transformation events (e.g., translation or rotation events) caused by the movement of the object in the scene, the Java layer's logic may decide to forward the event only when some minimum changes have been exceeded, thus considerably avoiding the creation of more network

traffic.

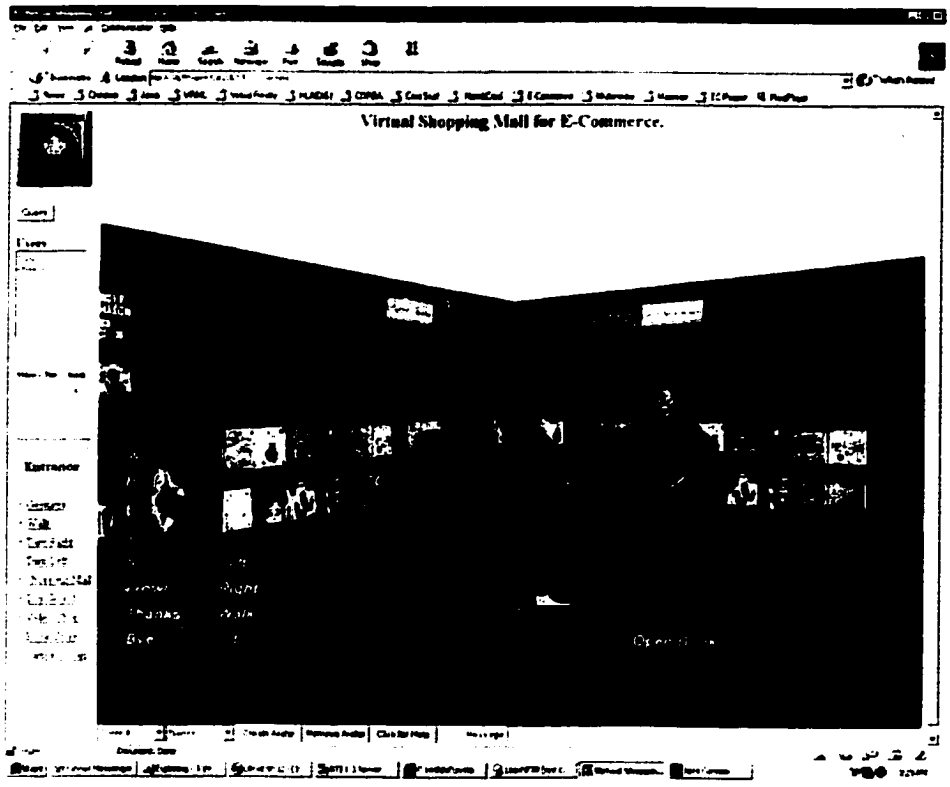

Figure 49 Web Browser Based User Interface in vCOM

5.2.3.2 Shopping Navigation

Navigation in virtual 3D spaces is not easy for the average user, using typical UI mechanisms. Controlling the mouse to maneuver to the desired direction requires a high level of coordination: in addition users could easily get lost in 3D spaces and can't find the right direction to move.

To facilitate users' navigation in multiple virtual shops (depicted in Figure 50) in vCOM prototype, the following tools are developed: the user can enter specific stores by either through navigation tools like mouse-click, 2D location map in the user interface or

via voice-enabled helper.

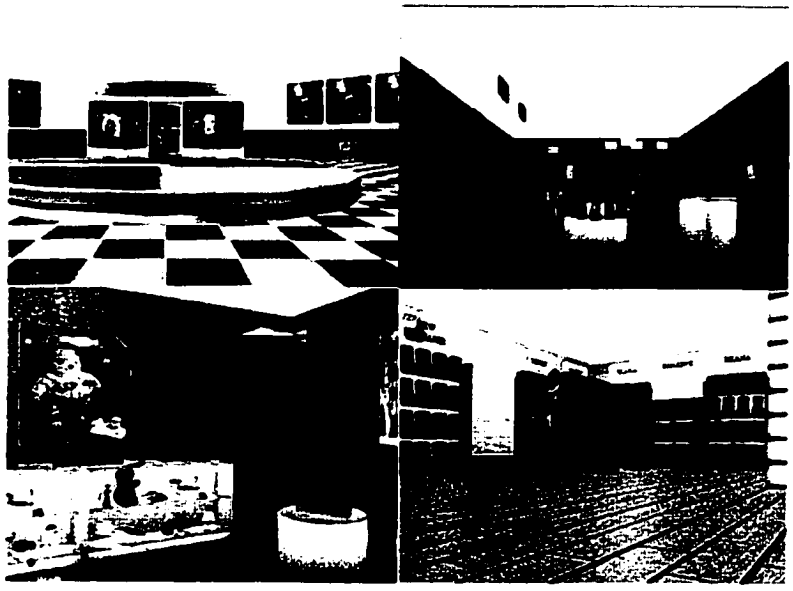

Figure 50 Multiple Virtual Scenes in vCOM

The vCOM prototype allows the user to communicate with an "intelligent assistant" (IA) using simple voice commands. This assistant interacts with the customer using voice synthesis and helps him or her use the interface to navigate efficiently in the mall. Voice-enabled agents provide users with a friendly speech-command interface, and also act as a general "help" facility for the user interface, accepting simple voice commands and giving voice responses. Speech recognition and speech synthesis provide a more natural interface for users. It frees the user's hands for other tasks and allows the user to take advantage of his natural voice communication skills. In the e-commerce application, speech recognition is put alongside with the web browser simulation to aid a customer to find items of interest in the virtual mall.

The Microsoft Speech Engine SAPI[2] is used to provide a speaker independent recognition of pre-defined commands described by a SAPI grammar. This way commands such as "go to the video room", "turn left", are recognized by the SAPI module. An example of SAPI code is shown below:

```
[<Direction>]
<Direction>=go to the "go to the" <Shop>
<Shop>=video shop "video shop"
<Shop>=toy store "toy store"
<Shop>=book store "book store"
```

In the above example, all the "go to the X" commands are programmed into the SAPI engine, where X is one of "video shop", "toy store", or "book store". These commands will invoke the appropriated actions. The speech recognition module is implemented using ActiveX components and its integration to the prototype is made via native DLL loaded by JNI. The speech recognition module works independently from the other modules, by sending recognized commands into the prototype via a predefined local socket.

5.3 E-Commerce Application over MAP

In the following sections, we will discuss the E-commerce application over MAP by using the approaches described in section 3.3. Figure 51 shows the Java3D based user interface for E-commerce application over MAP.

MAP allows an application builder to create 3D objects and inject them into existing shared scene on the fly. The application object usually consists of object 3D data

---

[2] The Microsoft Speech Engine can be downloaded from http://www.microsoft.com/IIT/Download/

and object attributes implement code. Typically, the object 3D data description contains an URL pointer that indicates to the clients where to download the graphics of the object. If VRML is used, associated scripts may be embedded in the object, which accept user interaction and get back to the object. Whenever a new 3D object is injected into the application from one station in the collaborative environment, other stations will download the graphics from the given URL, which is encapsulated in the OMA. Then the OMA takes the responsibility of retrieving updates of the object and reflects the updates graphically.

Figure 51 Java3D Based E-commerce Application over MAP

5.3.1 System Implementation

5.3.1.1 Agent Migration

Agent migration in MAP is implemented primarily by Java's object serialization techniques. Using Java's object serialization, it is possible to capture the data and state of the agent as a sequence of bytes. The sequence of bytes can be transmitted over the network and can be *deserialized* to come back to the original agent. When a MA decides to migrate, it calls the *migrateTo* method. Then the byte stream that includes the agent data is constructed by the MAP API either directly or indirectly.

The serialization mechanism of Java allows the transformation of the object graph into the byte stream. The message is then transferred via the network to the destination station via RMI or socket communication. The de-serialization mechanism is used to recover the object graph at the destination station. MAP has the code information of the object either by a pull or push scheme.

The class loading mechanism of Java allows network incoming classes to be dynamically loaded. To summarize, an MA migration to a remote AEE consists of the following steps:

1. Serialize the agent state to byte stream.
2. Send the stream to the destination station.
3. Retrieve the stream at the destination station.
4. Deconstruct the agent at original station.
5. De-serialize the stream at the destination station.

6. Destination station loads the corresponding agent code either by a push or a pull approach.

7. Load the agent class.

8. Create a new thread for the agent's execution and restart it.

Unfortunately, not all the objects in the Java language are *serializable*. When a *NotSerializable* exception is thrown, the first step to solve the problem is to identify the variable responsible, i.e. the one that is not *serializable*. In front of the declaration of this variable, put the keyword *transient*. It indicates that the value of this variable should not be sent over network. After the migration, this field will be set to null since it has not been saved. So we have to rebuild it. Then we use the automatic execution feature of MAP. Create a method, which contains the initialization of the non serializable variables. Before each migration, declare it as a method to execute on the arrival with *onArrival ("rebuilt")*.

5.3.1.2 Sharing the User Interactions

When shopping in malls, there are different kinds of activities performed by customers. Navigating through the mall to find the specific shop is the most common activity, when customers do shopping; it is abstracted in the application by avatars' movements through the virtual shopping mall. The second activity for the customers is to look for items of interest. Goods are presented using shelves. Thus, the customers might just pick the item from the shelf and examine it. This offers a first-person perspective of the item to the customer. Customers may co-operate, which is the third activity that may

take place among customers or between customer and salesperson. It is this activity that adds the social element to the virtual mall. In the implemented E-commerce prototype, the co-operation is simulated as interaction among avatars, the digital representatives of customers and sales assistants. For a social market place, the system provides awareness between users who share the same section of a shop. Users should be able to do something together, when they meet. Taking into account the real world setting, the most obvious activity would be to chat. Chats can be without any goal or they can be goal centered. The second kind of chats is especially important for communities, since it produces reasonable events. If two users currently watch "Star Wars" in the demo room at a virtual video shop, they may then be able to start a synchronous communication like text-based chatting and discuss their current interest or exchange their experiences of the movie. Besides chatting, visitors can join and do the shopping together. If they have noticed that they are looking for goods within the same category, they could talk to each other, ask the others' opinions and find the desired goods more effectively.

To support our goal of shared user interactions, we allow VRML scripts to communicate events to the scene graphs managed by UI via MAP. Our approach provides a mechanism to share user's (represented by avatar) interaction, which offers methods for non-verbal clues between collaborators, within all UIs in the 3-D world. The model is a replicated script mode with each UI downloading the same script and executing it locally. Typically these scripts would be associated with objects that are in the initial downloaded VRML file.

In Figure 52, we can see the message flow as a result of a user interaction. User A

selection (1) causes a local script to run (2). This in turn invokes the callback event in Object Master Agent, which then retrieves the event (3) and asks the MAP Transport Service to send attribute update (4). On the remote side, the MAP Transport layer retrieves the event via the network (5) and passes the event to the corresponding object OMA (6). It is the responsibility for OMA to interpret the event and reflect the updated attributed, which then causes the execution of the local script (7).

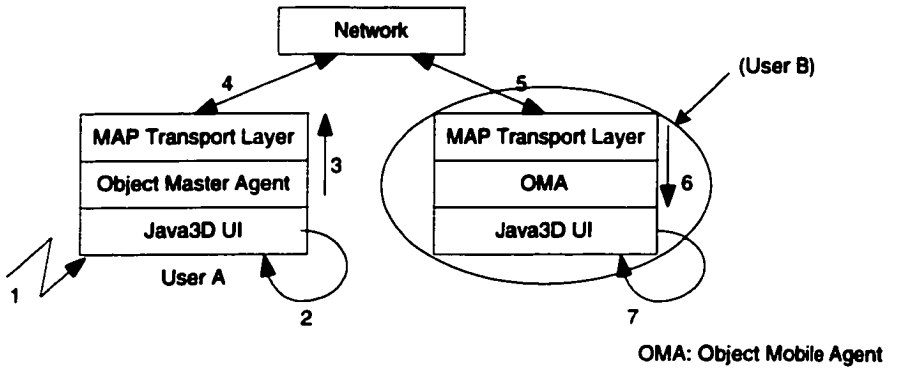

Figure 52 Sharing User Interactions

5.3.1.3 Object Manipulation

The virtual world consists of various types of entities, each of which has a visible body containing its state. In particular, it contains its geometry and physical characteristics. One entity often needs information about other entities: it is affected by them, and may need to interact with them. At first sight, it appears that there is a fundamental distinction among *active* entities (such as avatars), *passive* entities (such as the merchandises that need to be co-manipulated by avatars) and *inactive* entities (like the background objects in the virtual worlds).

A CVE should allow users to interact with objects in a natural manner such as the case of a user picking goods from a shelf. One of the attributes of an object in the virtual worlds is its ownership, which indicates which user (process) is allowed to access or manipulate the object. The manipulation is performed through modifying some of the attributes of an owned object (such as position and rotation). MAP services provide a facility to transfer attribute ownership and to keep the administration of the update responsibilities up-to-date as described in section 3.5. This mechanism allows two or more users to handle the same object through the transfer of ownership within the CVE.

5.3.2 Performance Evaluation

5.3.2.1 Elementary Evaluation of MAP platform

In order to evaluate the cost of the elementary mechanism within the platform, the agent migration function was broken into several components, which correspond to the steps presented in section 5.3.1.1. Java JDK1.2 was used for all the experiments described, completed over a 100 Mbits Ethernet.

Agent serialization: It corresponds to the first step in 5.3.1.1 section. A byte stream is constructed to serialized agent state.

Agent transfer: It consists of two phases, agent state and agent code transfer. The test is conducted by using a poll agent code transfer scheme.

Agent installation: It corresponds to steps 5, 7, and 8.

Agent migration: The agent migration time is the total time spent in all above

phases.

The execution time for these phases is given in table 1. It is noticed that agent migration of MAP costs less time than that of the Aglet system, a well-know Java based MA platform, which is more than 200ms [HaIs99]. The difference is due to the complexity of ATP (Agent Transfer Protocol), which requires several messages in order to migrate an agent.

Table 1 Basic cost of MAP

Agent Serialization	4.1 ms
Agent Transfer	47 ms
Agent installation	8.7 ms
Agent Migration	60 ms

5.3.2.2 System Performance Evaluation

In this section, we present the performance result of the E-commerce application MAP platform.

In terms of a performance model, CVE can be divided into four high level components each adding a delay to the overall system: 1) display – the "onscreen" delay created by monitors, projectors, stereo goggles, and other displaying equipments; 2) graphics – the delay caused by the graphics card/drivers and other graphical components of the system; 3) application: the lag introduced as a result of processing/computing performed by the core multi-user engine hardware and software; 4) interconnections: networking and communication delays.

Delay and jitter are the most important parameters in CVEs. Among these, delay

has been studied more extensively with actual numbers available, which indicate acceptable levels. The acceptable application + network delay is less than 200 milliseconds [Leig98], with some studies suggesting a more restrictive 100 milliseconds [Wlok95]. Jitter is also an important factor, sometimes believed to be more important than delay when it comes to coordinating collaborative tasks [Park95]. But no actual numbers for it are available, in terms of what constitutes an acceptable level of jitter.

In our evaluations, the combination of application and network delays is referred to as the *client-to-client delay* (CCD), which is defined as the average time it takes for a given update message to reach a target federate from initiating federate over the network. It does not include the rendering and graphics delay. In the E-commerce application, this delay includes all layers of Java, and physical network delays. For this CCD test, we have an "initiator" client who creates an object and performs an update on this object, such as a rotation or a translation. A second client then receives this update and extracts its information. Then, the second client performs a similar update on an object created by itself. The initiating client again performs an update on its object, and so on. This procedure is repeated for a given number of times or duration, which was about 5 minutes in our tests.

Another evaluated parameter is the *Ownership Transfer Delay* (OTD). MAP allows only the owner of an object (not the cases of joint manipulation) to manipulate it although everyone in the session will be able to "view" the manipulation. Ownership of an object must be acquired first in order for someone to manipulate the object. This transfer of ownership results in additional delay. This delay should be small enough to

maintain a natural feeling in the virtual world. For the OTD test, we have two clients obtaining and releasing the ownership of an object consecutively and in turn over a period of time. The average time to obtain ownership is then calculated. The application consists of a couple of stations running the e-commerce application where one starts as the owner of a certain object. The other client starts the loop by asking for the ownership of that object. From this point onwards the current owner grants automatically the request in the appropriated callback and asking for the ownership of the same object right away. Such loop is run for a predetermined time interval and the delay is averaged afterwards.

Object Insertion Delay (OID) measures how long it takes for an undefined object to be introduced by a client into simulation and all other clients who are interested in this object to retrieve and initialize the corresponding code stream. The procedure of the OID test can be described as follows: Client A first creates an object, then sends the OMA of the object to Client B using the "agent migration" services of MAP. After discovering the created object, Client B registers and initializes the discovered object, and then it will create its own object and sends the OMA to client A. Client A again performs the same procedure and so on.

Joint Manipulation Delay (JMD) measures the delay of joint manipulation on a given object. Both of synchronized and asynchronized modes are tested. In synchronized mode, three clients are involved in, one of them is the controller (the others are collaborators), which collects the interaction requests from collaborator clients and sends the synchronized update events to them. The collaborators send interaction requests every 20 ms to the collector. $\Delta t = 20$ ms which is less than the propagation delay before

two clients. *JMD* is measured by the time between interaction request generation and synchronized event reflection on collaborator client. In asynchronized mode test, three collaborating clients are involved in. One of them is the "initiator" client, who asks the two other collaborators to send interaction requests at a given time point (In the test, the initiator asks the collaborators to generate an interaction request in 100 ms). *JMD* is measured by the time between interaction request generation and latest request received from collaborator on initiator client. (Cost of commutative transformation computation is ignored in this test)

All tests are performed on Pentium III class PCs running Windows 2000 and connected to a 100 Mbps Ethernet. The graphics is set at 1280X1024 24bit color screen resolution with 3D accelerator cards. All machines are running typical operating system and networking background processes. All the tests have been committed on Java3D-based user interface. Table 2 below shows the remaining performance results.

Table 2 Performance Evaluation of MAP (All delays are in milliseconds)

CCD	31 ms
OTD (pull)	97 ms
OTD (push)	92 ms
OID	142ms
JMD (Synchronized)	67ms
JMD (Asynchronized)	36m

Table2 shows the performance results of CCD, OTD, OID and JMD. Since Java RMI is used for communication between clients, CCD is less than CCD in a client-server model . It should be noted that the CCD test result would not only be an indication of the

transmission speed of the underlying network, but also an indication of the end-to end delay of the entire system, including delays caused by the underlying layers such as transport layer, network layer, or LAN datalink layers, and so on.

The time OTD spends in sending an acquisition and migrating object master agent is matched by the result in Table 2. Compared with other middlewares that are applied in distributed virtual environments, such as RTI and SPLINE [Leig98], the delay is reasonably small, which makes tightly-coupled collaboration possible. This delay is measured in pull mode. In contrast, the "push" scheme allows clients to transfer ownership without an advance request. Therefore, the cost of OTD in the push scheme is less than that in the pull scheme. OTD in the push scheme is approximately equal to the object master agent's migration time.

For JMD, we notice that the asynchronized mode has a smaller delay than the synchronized mode. That is because it does not need the controller to synchronize the interaction requests, which costs more time. However, the asynchronized mode has the extra cost for computing the commutative transformation.

Chapter VI

6 Conclusions

This final chapter summarizes the work presented in this thesis and presents the comparison with HLA standard and the contributions of this work organized in terms of collaborative virtual environments. Building a scalable Collaborative Virtual Environment is a technical challenge. MAP, a mobile agent-based platform for generic CVE systems, was presented in this thesis. It supports Mobile Agent-based attribute updates, dynamic object injection, ownership transfer and joint manipulation. A framework for dynamic workload balancing interest management over MAP platform was also presented and specific algorithms based on this framework were evaluated.

6.1 Comparison with HLA Standard

6.1.1 Persistent Collaborative Virtual Environment

The e-commerce application is an example of a persistent collaborative virtual environment, where simulations of most of the actual shopping environments and user interactions can be achieved with the creation of a virtual shopping mall. Most of the existing distributed systems require clients' knowledge about each other's application functions and behaviors, thus the different types of entities that will be participating in a simulation must be set up *a priori*. To introduce new entities, simulation programs have

to be modified, recompiled and restarted, which obviously will cause problems for web applications. HLA, however, uses *Object Model Templates (OMT)* to define the standardized formats of the functionality of simulation models and the interaction between models. As a result, the capabilities of all simulation models are determined before the actual simulation takes place. OMT in simulation is defined as *Federation Execution Data* (FED) and each federate in a federation keeps the same FED copy file in its local storage. FED determines object classes at which object instances may be registered and discovered in the simulation, instance attributes that are available to be updated and reflected, as well as interactions that may be sent. This static property of the FED prevents federates from injecting an undefined objects into simulation in progress.

This limitation is solved perfectly in MAP. The MAP platform overcomes this problem by creating and injecting new entities into simulation space on the fly without making any changes to the simulation program. All functionalities are embodied in MA object programs and carried through the network. New entities can therefore be introduced anytime as the simulation progresses. Similarly, existing entities' behaviors may be changed by the replacement of MA objects on the fly. With MAP approach, it is also possible to have more than one protocol active at a given time. In particular, rather than having one protocol for N entities, there could be N protocols, one for each entity.

6.1.2 Update Latency

Table 3 shows the performance comparison with HLA/RTI standard. From this table, we notice that RTI has above 150 ms CCD and more 200 ms OTD. These significant delays limit tight-couple collaborative application to be built over RTI.

Compared with RTI, MAP has better performance on all the parameters we tested.

Table 3 Comparison with HLA/RTI (All delays are in milliseconds)

	RTI	MAP
CCD	162	31
OTD (Pull Mode)	267	97
OTD (Push Mode)	225	92
OID	362	142
JMD (Synchronized)	N/A	67
JMD (Asynchronized)	N/A	36

(RTI 1.3 v6 is used for the tests)

The main reason of the large delays of RTI is because the approach of the current RTI implementation is using a single thread. In single-threaded RTI, RTI and federate share a single thread of execution; and the federate must explicitly pass control to the RTI, e.g., to deliver messages; the *tick*() procedure is defined for this purpose and callbacks (Discover, Reflect) are made within the tick period. However, in MAP, every object has its own execution thread; the OMA can reflect the attribute updates right after receiving them.

6.1.3 Ownership Management and Joint Manipulation

The mechanisms of RTI are based on object attributes only. In cases where several attributes or even entire object have to be migrated, problems may arise. Multiple federates may react with a pull attempt on one single push attempt. Since the HLA Interface Specification does not specify how the RTI must deal with multiple pull requests for the same attribute, it is not clear which federate will be the new owner.

For implementation of joint manipulation, because the attributed-based solution of ownership management in RTI, only the Constraint Based Interaction Request Resolving approach discussed in section 3.6.2 may be applied for joint manipulation application over RTI. However, this approach lies in the complex nature of *constraint* resolution systems. In some complicated collaborative tasks, one interaction request on an object could be related or affected by other requests. So it becomes a challenging task to disjoint the independent interaction requests on the same object in these situations. The approach of *constraints* leads very quickly to a situation where none of the interaction requests can be satisfied, because the underlying constraints cannot be resolved.

Realizing the about limitations of RTI, MAP provides more flexible means for ownership management and joint manipulation. Not Only attributed-based solution, but also object-based ownership management is implemented. MAP also realizes all the three modes of joint manipulation, such as the constraint based, synchronized and asynchronized interaction request resolving.

6.2 Contributions

The main contributions of the research presented in this thesis are in two aspects: Mobile Agent-based platform for CVEs and dynamic interest management system over MAP:

MA-based platform for CVEs:

1. To our knowledge, this is the first time that Mobile Agent concepts were introduced to the CVE domain. A Java-based mobile agent architecture was designed and

implemented. An e-commerce application was also implemented over the MAP platform. This platform allows the virtual environment to be persistent by introducing objects dynamically.

2. An interaction mode for joint object manipulation was proposed and three means based on this mode were presented.

Dynamic interest management system over MAP:

1.A framework for dynamic load balancing interest management based on the concepts of aura nimbus interaction model and MA objects was presented;

2.Algorithms of exchanging workload of IMs were proposed. These algorithms involve distributed decision-making, which makes them scalable; furthermore, the algorithms involve only near-neighbor communication, which makes them arbitrarily scalable; the algorithms preserve the topology of the DOR decomposition which permits the same simple and efficient communication structure to be used by the work load balancing algorithms.

A comparison of the MAP-based e-commerce application with an HLA/RTI-based one indicated that MAP had the following advantages over HLA (discussed in section 5.4):

1. Dynamic Properties:

In HLA, Object Model Templates are used to define the standardized formats of the functionality of simulation models and the interaction between models. As a result,

the capabilities of all simulation models are determined before the actual simulation takes place. This static property prevents federates from injecting an undefined objects into a simulation in progress.

The MAP platform overcomes this problem by creating and injecting new entities into simulation space on the fly without making any changes to the simulation program. All functionalities are embodied in MA object programs and carried through the network.

2. Better performance on client-to-client latency;

The major reason why RTI has significant latency between federates is due to single threaded implementation. In MAP, every object has its own execution thread; the OMA can reflect the attribute updates right after receiving them.

3. Ownership management and joint manipulation.

MAP provides more flexible means of ownership management and joint manipulation. For example, the ownership management of HLA is only based on object attributes. Problems may arise when multiple federates react with a pull attempt on one single push attempt. MAP also provides a method of object management, which is based on migration of the entire object. It is more common to have the access of an entire object for ownership transfer in a CVE application.

6.3 Future Work

The work presented in this thesis can be further extended in the following

directions:

1. Multiple clusters of entities injection

So far, MAP only allows a single object to be injected into the system at any given time. An extension to this is be applying MAP to applications where there can be multiple clusters of entities injection. Multiple clusters of entities injection might be used in the case of introducing a new virtual scene that contains multiple active objects into an existing shared virtual space.

2. Hybrid approach for DOR partitioning

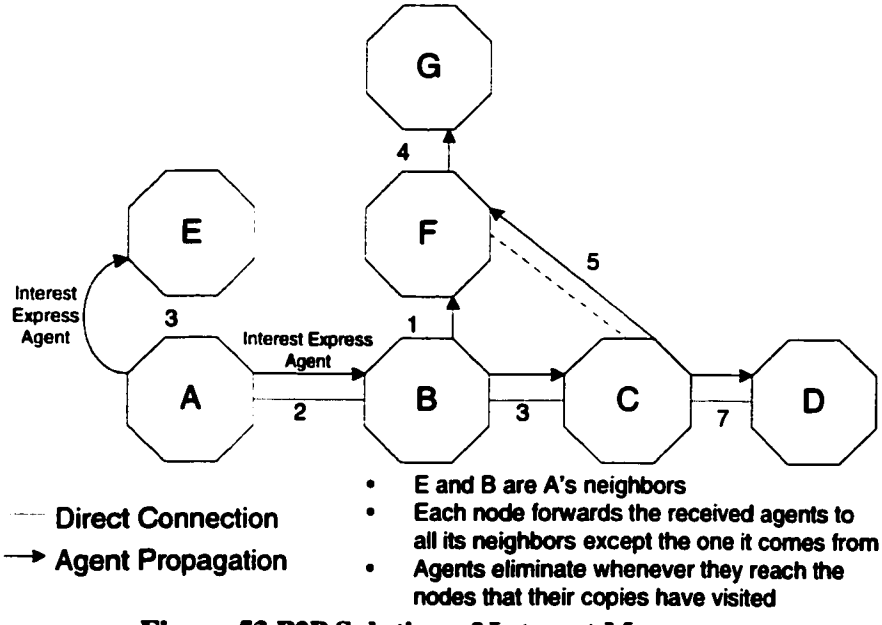

Figure 53 P2P Solution of Interest Management

Two approaches for DOR partitioning: space and entity based methods were discussed in Chapter 4. In some cases, the entity in CVE applications might express their interests based on either space or entity, for example, the user wants to receive data about

some kinds of entities that locate in a certain area. The 3-dimentional DOR can be defined in these cases. However, sophisticated algorithms of exchanging workload among IMs need to be developed.

3. Peer to peer solution of Interest Management

The approaches of dynamic interest management presented in this dissertation still rely on a central workstation(s), which takes the responsibility of interest management. The extension could be a completely distributed structure based on peer-to-peer solution.

```
procedure buildTopoMap(NeighborList Nodes, Graph map)
for (all elements n in nodes)
    connect to n
    if (connection established) then
     send InterestExpressionAgent
     for (all IEA messages received)
      if
         (response node n2 != n) then
         add edge between n and n2 to map
      if (n2 is not in hosts) then
        add n2 to the end of hosts
      endif
      endif
    endfor
  endif
endfor
end procedure
```

Figure 54 Topology Discovery

Figure 53 shows a logical network of peer-to-peer solution. The basic idea is that, instead of sending an Interest Express Agent to a central IM, each involved client propagates its agent to all its neighbors in the logical network. After arriving on a remote client, the agent checks the interest of the node that it resides on, and

establishes the connection based on the matching result. Figure 54 depicts the algorithm of topology discovery.

7 References

[AbWZ98] H. Abrams, K.Wastsen, and M. Zyda "*There Tied Interest Management for Large-Scale Virtual Environments*", Proceedings of VRST98, November 1998, Taipai

[AnVJ99] J.Anderson, R. Vincent, M. Jack "*Usability Assessment of Collaborative Shared-Space Telepresence Shopping Services*", ECMAST 1999 4th European Conference, Madrid, Spain, May 1999

[BaWA96] J.W. Barrus, R.C. Waters, and D.B. Anderson. "*Locales and Beacons: Efficient and Precise Support for Large Multi-User Virtual Environments.*" Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS). IEEE Neural Networks Council. Santa Clara, California, 204-213, March 1996.

[Brig98] K. Briggs "*A Required RTI Gateway Standard as a Solution to RTI Interoperability,*" 98S-SIW-188, 1998 Spring Simulation Interoperability Workshop, March 9-13, 1998.

[Brol95a] W. Broll "*Interaction and Behavior Support for Multi-User Virtual Environments*". In Proceedings of the ACM SIVE'95 - First Workshop on Simulation and Interaction in Virtual Environments. Iowa City, University of Iowa/ACM (July 1995), 246-264.

[Brol95b] W. Broll "*Interacting in Distributed Collaborative Virtual Environments*". In Proceedings of the IEEE Virtual Reality Annual International Symposium - VRAIS'95. Los Alamitos, Ca., (March 1995), IEEE Computer Society Press, 148-155.

[CaHa96] C. Carlsson and O. Hagsand, "*DIVE: a platform for multi-user virtual environment*", Computer & Graphics Vol 17, N. 6, pages 663-669.

[CaHo94] O. Calvin, J. Van Hook, "*AGENTS: An Architectural Construct to Support Distributed Simulation*," 94-11-142, Eleventh Workshop on Standards for the Interoperability of Distributed Simulations, Sept 26-30, 1994.

[CaRi96] O.Calvin, W. Richard, "*An Introduction to the High Level Architecture (HLA) Run-Time Infrastructure (RTI)*," 96-14-103, Fourteenth Workshop on Standards for the Interoperability of Distributed Simulations, March 11-15, 1996.

[CDGB95] Chess, David, Grosof, Benjamin, Harrison, Colin, Levine, David, Parris, Colin, and Tsudik, Gene, "*Itinerant Agents for Mobile Computing*", IBM T.J. Watson Research Center, Yorktown Heights, New York, 1995.

[ChSh98] B Christina , and D. Shen: "*Implementing Ownership Management Services With a Bridge Federate*," 98S-SIW-197, 1998 Spring Simulation Interoperability Workshop, March 9-13, 1998.

[ChWh97] S. Choi and A. Whinston "*Economics of Electronic Commerce*", Indianapolis: Macmillan Technical Publishing. 1997

[DaJa98] J. Daniel J. and O. James , "*Data Distribution Management in RTI 1.3*," 98S-SIW-206, 1998 Spring Simulation Interoperability Workshop, March 9-13, 1998.

[Dam90] F. Van Dam , "*Computer Graphics: Principle and Practice*", Addison-Wesley,

1990

[DiGa99] C. Diot and L.Gautier, "A Distributed Architecture for Multiplayer Interactive Application on the Internet", IEEE Network, July/August 1999, pages 6-15

[Dong00] S. van Dongen *"A cluster algorithm for graphs Cluster"*, 2000, INS-R0010, ISSN 1386-3681

[DuMa95] N.Durlach, and S. Mavor, *"Virtual Reality: Scientific and Technological Challenges"*, National Academy Press, Washington, D.C. 1995.

[EEDQ02] Enright A.J Enright., S. Van Dongen, C.A. Ouzounis *"An efficient algorithm for large-scale detection of protein families"* Nucleic Acids Research 30(7):1575 - 1584 (2002).

[Frid88] Friedman et al *"SIMNET Ethernet Performance"* Technical Report. BBN Communications Corporation, Cambridge, Massachusetts, January 1988.

[Geor97] N. D. Georganas, *"Advanced Distributed Simulation and Collaborative Virtual Environments"*, CRC (Industry Canada) Report, Dec.1997, 135 pages.

[GhVi97] C. Ghezzi, C and G. Vinga, *"Mobile Code paradigms and Technologies: A Case Study"*, Proceedings of the First international Workshop on Mobile Agents, MA'97

[GrBe95] C. Greenhalgh and S.Benford *"Virtual Reality Teleconferencing: Implementation and Experience."* Proceedings of the Third Europenn Conference on Computer Supported Cooperative Work (ECSCW'95). Stockholm, Sweden, September

1995

[GrBe97] C. Greenhalgh and S.Benford. "*Boundaries, Awareness, and Interaction in Collaborative Virtual Environments.* "Proceedings of the Sixth IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE). IEEE Computer Society, Cambridge, Massachusetts, 193-198, June 1997

[GrBn97a] C. Greenhalgh and S. Benford, "*MASSIVE: a virtual-reality system for teleconferencing*", ACM Transactions on Computer-Human Interfaces (TOCHI), ACM Press, pages 239-261.

[Gree97] C. Greenhalgh, "*Large Scale Collaborative Virtual Environments*", Ph.D. Thesis, Computer Science Department, University of Nottingham, October 1997.

[Grim91] C. Grimsdale, "*dVS - Distributed Virtual Environment System*", Proceedings Computer Graphics '91, London, UK.

[HaCD95] G. Harrison, G. Colin , M. David , "*Mobile Agents: Are They a Good Idea?*" IBM Research Report, IBM T.J. Watson Research Center, New York, 1995.

[HaIs99] D. Hagimont and L. Ismail "*A performance evaluation of the mobile agent paradigm*", Proceedings of the 1999 ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications November 1 - 5, 1999, Denver, CO USA

[Harr95] G. Harrison, "*Smart Network and Intelligent Agents*", IBM Research Report,

IBM T.J. Watson Research Center, New York, 1995.

[HeTa98] M. H. Hettler, A. N. Tahvildar-Zadeh, "Dynamical Cluster Algorithm for Highly Correlated Electron Systems", available at: http://www.aps.org/BAPSMAR98/abs/S4890003.html

[Hook94] J. Van Hook et al "*An Approach to DIS Scalability*," 94 11-141, Eleventh Workshop on Standards for the Interoperability of Distributed Simulations, September 26-30, 1994.

[Hook96] D. Van Hook et al., "*Approaches to RTI Implementation of HLA Data Distribution Management Services*", Proceedings of the 15th DIS Workshop, 1996.

[Hook96a] J. Van Hook et al "*Performance of STOW RITN Application Control Techniques*," 96-14-157,

[Hook96b] J. Van Hook "*RITN IM and IM History*". Personal Communication, January 1996.

[Hyac96] S. Hyacinth , "*Software Agents: An Overview*", Knowledge Engineering Review, Vol. 11, No 3, pp.1-40, Sept 1996.

[JoDe97] J., Johnson, A., DeFanti, , "*Issues in the Design of a Flexible Distributed Architecture for Supporting Persistence and Interoperability in Collaborative Virtual Environments*" Proceedings of Supercomputing '97 San Jose, California, Nov 15-21, 1997.

[John92] L. John et al, "*Integrating SIMNET with NPSNET Using a Mix of Silicon Graphics and Sun Workstations*" Naval Postgraduate School, Monterey, California, March 1992.

[John95] H. John, "*Talk to My Agent: Software Agents in Virtual Reality*", Computer-Mediated Communication Magazine, Volume 2, Number 2, February 1, 1995 / Page 3

[Kath96] M. Katherine, "*Interest Management in Large Scale Distributed Simulations*" UCI Technical Report #96-27.

[KaJe97] M. Katherine, S. Jeffrey, "*Data Distribution Management in the HLA: Multidimensional Regions and Physically Correct Filtering*" Proceedings of the 1997 Spring Simulation Interoperability Workshop, 1997.

[KiCh96] S. K.Singhal and D.R.Cheriton. "*Using Projection Aggregation to Support Scalability in Distributed Simulation.*" Proceeing of the 16th International Conference on Distributed Computing System (ICDCS). IEEE Computer Society. Hong Kong, 196-206, May 1996

[LADO95] Lingnau, Ansel and Drobnik, Oswald, "*An infrastructure for Mobile Agents: Requirements and Architecture*", Proceedings of the 13th DIS Workshop, Orlando, 1995

[Leig98] J.Leigh et al, "*Preliminary STAR TAP Tele-Immersion Experiments between Chicago and Singapore*", High Performance Computing Asia Conference & Exhibition, Singapore, 1998.

[LeJD97] Leigh, J., Johnson, A., DeFanti, T., "CAVERN: A Distributed Architecture for

Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments" Virtual Reality: Research, Development and Applications, Vol 2.2, December 1997, pp. 217-237.

[MaBz94] M.J. Macedonia, M, P.T. Barham and S. Zeswitz, "*NPSNET: A network software architecture for large scale virtual environments*", MIT Presence, 3(4), 1994.

[Mace95] M. Macedonia, "*A Network Software Architecture for Large-Scale Virtual Environments*", Ph.D. Thesis, Computer Science Department, Naval Postgraduate School, Monterey, California, June 1995.

[Mara99] J. Marathe "*Creating Community Online*", Durlacher Research Ltd, 1999, available at http://unpan1.un.org/intradoc/groups/public/documents/apcity/unpan003006.pdf

[MRZJ95] M .Macedonia, Michael R., Zyda, Michael J., Pratt, David R., Barham, Paul T., "*Exploiting Reality with Multicast Groups*", IEEE Computer Graphics and Applications, September 1995.

[OlSG99] J.C.Oliveira, S. Shirmohammadi and N.D.Georganas, "*Distributed Virtual Environment Standards: A Performance Evaluation*", Proc. IEEE/ACM Third International Workshop on Distributed Interactive Simulation and Real Time Applications (D I S - R T ' 99), Greenbelt MD, Oct.1999

[Park97] K.S.Park, "*Effects of Network Characteristics and Information Sharing on Human Performance in COVE*", Master's thesis, Electronic Visualization Laboratory,

University of Illinois at Chicago,

[Pope89] A. Pope, BBN Report No. 7102, "*The SIMNET Network and Protocols*" BBN Systems and Technologies, Cambridge, Massachusetts, July, 1989.

[Roeh95] B.Roehl "*Distributed Virtual Reality -- An Overview*", http://www.ece.uwaterloo.ca/~broehl/distrib.html

[Shen02] X. Shen, et al "vCOM: Electronic Commerce in a Collaborative Virtual World ", Electronic Commerce Research and Applications Journal (Publisher: Elsevier Science) (to appear) 2002

[ShHG99] X. Shen, R. Hage and N.D.Georganas, "*Agent-aided Collaborative Virtual Environments over HLA/RTI*", Proc. IEEE/ACM Third International Workshop on Distributed Interactive Simulation and Real Time Applications (D I S - R T ' 99), University of Maryland, College Park MD, Oct. 1999

[ThCa95] W. Thomas and R.Callahan. "A Large-Scale Complex Virtual Environment for Team Training". IEEE Computer, 28(7), pages 49-56, July 1995.

[Todd99] P. Todd, "*Mobile Agents: Are They Useful for Establishing a Virtual Presence in Space?* ", available at http://luckyspc.lboro.ac.uk/Docs/Papers/aaai-ss99.pdf

[WaCU00] A. Warms, J. Cothrel, T. Underberg "*Return on Community: Proving the Value of Online Communities in Business*", Participate.com, April 12, 2000.

[Whit94] J.E. White, "*Telescript technology: the foundation for the electronic*

marketplace", White Paper, General Magic, Inc. 1994

[Whit95] J.E. White, "*Telescript technology: Mobile Agents*", General Magic White Paper, Mountain View California, 1995

[Wlok95] M.M.Wloka, "*Lag in Multiprocessor VR*", Presence: Teleoperators and Virtual Environments (MIT Press), Vol. 4, No. 1, spring 1995.

[AGLET] IBM Japan –Aglets Workbench: http://www.trl.ibm.co.jp/aglets/.

[ALPHA] Alpha World: http://www.activeworld.com

[DIS93] DIS Vision Document (1993) at *ftp.sc.ist.ucf.edu/SISO/dis/library/vision.doc*

[GM] General Magic – Telescript/Odyssey: http://www.genmagic.com/agents/odyssey.html.

[GRAS]IKV++ Gmbh – Grasshopper: http://www.ikv.de/products/grasshopper/.

[HLA] HLA Homepage, URL at http://hla.dmso.mil/hla

[HLAa] Draft Standard For Modeling and Simulation (M&S) High Level Architecture (HLA) -Federate Interface Specification, DRAFT 1, 20 April 1998

[IEEE1278] IEEE Standard 1278.1 (series): IEEE Standards Department: Simulation Interoperability page at *http://standards.ieee.org/catalog/simint.html*

[NTP] Network Time Protocol (NTP), http://www.eecis.udel.edu/~ntp/

[OS] Object Space – Voyager : http://www.objectspace.com/voyager/.

[QUAKE] Quake, http://www.quake.com