Mobile Agent Based Framework For Large-Scale Collaborative Virtual Environment

Zhang Liang

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2007

Acknowledgements

First of all, I would express my gratitude to Nanyang Technological University and the Information Communication Institute of Singapore for providing me the opportunity to carry on my research work.

I would like to give my sincerely and deepest thanks to my supervisor, Dr. Lin Qingping, for his patient guidance, careful mentoring, and warm supports during the course of my PhD candidature. His broad horizon, clear thinking and great enthusiasm make me learn more. Without whom I could not have fulfilled my task of research, for his guidance and the patience given to my research work and in reviewing my thesis.

Especially, I would thank my parents, my sister, my brother-in-law, and my wife for their dedication, encouragement, and support for me in my work throughout the period.

Last but not least, I would thank all the other students and the technicians in ICIS for their kind helps in my research work.

Summary

Collaborative Virtual Environment (CVE) provides an online shared virtual world for the geographically dispersed people to interact with each other. However, the scalability of the existing CVE systems is limited due to the constraints in processing power and network bandwidth of each participating host. The research for this thesis is motivated to improve the scalability of CVE within a heterogeneous computing and networking environment from the data communication perspective. It is achieved through a Mobile Agent based framework for large-scale CVE (MACVE).

In MACVE, the large-scale CVE (LCVE) software system is decomposed into a group of mobile agents. Each mobile agent is responsible for an independent task to provide services to exchange or process the data flows in LCVE. The mobile agents do not bond with any fixed hosts and they can migrate or clone dynamically at any suitable participating host which includes traditional server hosts and qualified user hosts. By agent migration, LCVE tasks will not converge at a single host; by remote agent cloning, a single task will be shared by multiple hosts. The mutual independence of services and hosts provide large freedoms for LCVE system to utilize the system resource efficiently and avoid the potential bottleneck. The ability for qualified user hosts to take over the mobile agents makes the system's scalability less confined by the number of traditional servers. Thus, the system data flows and workloads can be pervasively distributed. Moreover, MACVE provides adaptive consistency control to dynamically decide the consistency model and data communication architecture for different object state data depending on run-time activities and consistency requirements in each particular part of the LCVE, which will further enhance LCVE system scalability.

This thesis contributes a mobile agent framework for large scale CVE (MACVE); a specially designed mobile agent system for MACVE which includes Mobile Agent Environment, MACVE APIs, and MACVE Protocol; a prototype CVE system with eleven basic types of agents and a web-based user application. Experiments with the prototype system have demonstrated the feasibility of MACVE. It also has shown the

advantages of the proposed MACVE for improving the scalability of the CVE applications.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the widespread popularity of the World Wide Web and the miscellaneous online software applications, Internet now is changing from a repository of information to an interactive digital social world. Collaborative Virtual Environment (CVE) just functions as a new type of interface which attempts to model the Internet into a real social world using the power of network communication and Virtual Reality (VR). This chapter introduces the background of CVE and presents the research motivation, objectives, major contributions and the organization of this thesis.

1.1 Collaborative Virtual Environment

CVE is a virtual world shared by participants across a computer network. Participants are provided with graphical embodiments called avatars that convey their identity, presence, location, and activities to others. They are able to use these avatars to interact with the contents of the world and to communicate with one another using different media including text, audio, video, and graphical gestures in real-time[1]. Each user accesses the virtual environment via a user interface, which may be a space ball, a Head-Mounted Display (HMD), data gloves, or simply a computer console with a mouse, joystick, keyboard, etc. These environments aim to provide users with a sense of realism by incorporating realistic 3D graphics and stereo sound to create an immersive experience[2, 3].

CVE can be seen as the result of a convergence of research interests within the virtual reality(VR) and computer-supported cooperative work(CSCW) communities[1]. Within VR, the CVE represent a natural extension of current commercial single-user VR

technology to support multiple participants. This extension allows VR to support for a larger range of applications. CSCW is a relatively recent discipline that research on how collaborative activities and their coordination can be supported by means of computer science[4]. The real CSCW systems are often referred as groupware[5]. Within the CSCW community, CVE represents a technology that may support some aspects of social interaction not readily accommodated by technologies such as audio and videoconferencing and shared desktop applications.

CVE exhibits a number of new features which include: a shared sense of space, a shared sense of presence, a shared sense of time, natural ways of communications, and natural ways of interaction [2] [3]:

- A shared sense of space

  In a CVE, all participants are presented with the illusion of being located in the same place, such as in the same room, building, or terrain. That shared place represents a common context within which the interactions take place. The shared space presents the same characteristics to all participants.

- A shared sense of presence

  When entering the shared place, each participant is represented as an avatar, which includes a graphical representation, body structure model (e.g., the presence of arms, legs, heads, joints), motion model (e.g., the range of motion that joints may take), physical model (e.g., height, weight), and other characteristics. Upon entering the shared VE, each participant can see other avatars that are located in the shared space, and those users can see the new participant's avatar. Similarly, when a participant leaves the VE, other participants should see his avatar’s departing. Not all avatars need to be human-controlled. They may be synthetic entities controlled by event-driven simulations or even by rule-based inference engines.

- A shared sense of time

  Participants in the CVE should be able to see each other's behavior as it occurs. In

other words, the CVE system should enable real-time interaction to occur. The shared sense of time requires a consistent world among the distributed users.

- Natural ways of communication

Different kinds of mediums should be supported by a CVE system to provide users with natural ways of communication: graphics, gesture, text, voice, and video. These kinds of communications add a necessary sense of realism to a CVE for the users.

- Natural ways of interaction

The true power of the CVE system derives from users' ability to interact with each other, and with the VE itself. The interactions need to be modeled realistically. For example, users should be able to pick up, move, and manipulate items that exist in the environment, and be able to pass items to other participants.

In summary, CVE systems provide users with the ability to interact with each other within a common context, even though those users may be physically located around the world. CVE are most appropriate for applications that require the creation of tele-presence, the illusion that other users are visible from remote locations. In these applications, a sense of realism is needed in order to achieve effective collaboration which can otherwise only be possible by face-to-face contact.

CVE has a broad spectrum of applications. It originated from military war simulation which create a synthetic environment (or virtual battle field) for military commanders and troops training. It has no danger of life and also can save the cost and achieve more scalable and distributed simulation[6]. With its successful military application, CVE caught more attentions from academia and industries and began to apply to teleconferencing[7], medical or industrial team training[8], collaborative design and engineering[9], collaborative scientific visualization[10], and social activity simulation. And now CVE applications extend to virtual shopping malls and showrooms, virtual libraries[11], distant-learning[12] and entertainment-like Internet-based multi-user VR games[13], which can be reached by everyone with the power of Internet. In future, CVE tends to become an indispensable tool to facilitate people to communicate and interact in

their daily life.

1.2 Motivation

The current research and commercial development of CVE still requires addressing a variety of technical challenges such as human interface design, graphics realism, etc. One of these challenges is to build Large scale CVE (LCVE) in a heterogamous computing and networking environment. LCVE provides a spatially large, content-rich, and evolving-enabled virtual environment that can support a large number of simultaneous participants geographically dispersed over a wide area network, such as Internet. At present, researches on LCVE are mainly focus on two aspects: the end-system virtual reality performance and the data communication in the system[14]. This thesis has been motivated by the challenge of improving the LCVE scalability from the data communication perspective.

In a LCVE, large spatial extent and a large number of virtual entities enrich the VE content and bring more realism to participants. A large number of simultaneous participants make CVE more popular and anyone, at the edge of network, can communicate and interact with each other freely and easily without any technical restriction. Therefore, scalability is one of the key research issues for CVE that expects to be resolved.

From the data communication point of view, the fundamental requirement of a CVE system is to convey the most needed data with an appropriate level of detail to its relevant receivers so that participants can maintain a consistent view of the CVE and engage in a real-time conversation[2]. Thus, data flow exchanging and data flow processing are the basic tasks of a CVE system. In a CVE system, there are five main streams of data flows needed to be exchanged and processed:

- VE scene data flow which conveys the VE description data to newly arriving users. VE scene data is the VE description files including 3D geometry files, 2D image files, audio files, video files and behavior script files to construct the scene, virtual entities and avatars;

- VE shared state data flow which conveys the snapshot of current VE state to newly arriving users;
- Object state data flow which conveys virtual entity and avatar's dynamic activities in terms of their state changes and events to the relevant receivers;
- VE consistency control data flow which conveys the control messages among a group of relevant receivers to ensure the required level of consistency;
- VE persistency data flow which conveys persistent information i.e. changed VE state, to non-volatile storage place to archive, such as database systems.

Exchange of these data flows requires high network bandwidth, for example, to deliver the VE scene data to all participants. Processing these data flows, for example consistency control among a group of users, requires large computational resource.

The scalability issue of CVE is challenging because the limited and heterogeneous network bandwidth and computer-processing power of each participating host cannot cope with the unpredictable, large volume of data flows which need to be exchanged and processed timely to guarantee the responsiveness and consistency in the LCVE. When exchanging or processing the data flows within CVE, if more of them converge at a host and exceed that host's network bandwidth or computing capability, the data will not be transferred or processed timely, and the consistency or real-time interaction in the CVE will be penalized. That host becomes system bottleneck. Thus, limitations on CVE scalability mainly arise from the bottlenecks of the network bandwidth and computer-processing power.

To exchange and process the data flows, three types of data communication architecture are adopted in the existing CVE systems. In the initial CVE systems, client-server architecture was used because this is a conceptually and practically simplest approach. In client-server architecture, each participant's application is called a client and it communicates with a common server program. Server program run at a host with high networking and computing capacity and perform system basic tasks, such as data flow

filtering for each client. A client sends its messages to others via a server and it also receives other participant messages from the same server. In nature, a server functions as a proxy for clients to reduce their network traffic or processor's workload. Because all clients' interactive messages are sent through a server, CVE consistency is easy to achieve. Moreover, as the server can perform the special data flow filtering for the clients that have low networking or computational capacity, heterogeneous participant is easily supported. Finally, because the server can preserve the changes happening in the CVE, persistency is also easy to support in client server architecture. However, the problem with this architecture is that the server will soon become the bottleneck when the number of user connections increases. The latency of the system also prolongs because each message must send through the server.

To improve CVE system scalability, peer-to-peer architecture is adopted. In such architecture, each individual participating host is called a peer. There is no server and each peer sends its information directly to other peers. All the basic tasks of the CVE system, such as the data flow filtering, consistency control, are performed by every peer. A CVE system with peer-to-peer architecture can scale up by an effective data flow filtering mechanism over IP-multicast. However, this kind of architecture requires each peer to possess large bandwidth and computational resource. So peer-to-peer architecture does not have a good support of heterogeneous participants. In addition, since the consistency control is performed in a distributed manner, the consistency of the CVE is not easy to achieve. Consequently, most of CVE systems with a peer-to-peer architecture only have approximate consistency control. Finally, there is no central site in the system, the VE Scene data is replicated at each peer side before the VE session begins. Dynamic VE scene data retrieving and VE persistency maintenances are not supported. Therefore, peer-to-peer architecture is mainly adopted by CVE systems that have dedicated networks and computers, such as military war training or academic experimental systems.

Multi-server architecture is proposed to take advantages of both client-server and peer-to-peer architectures. It merges client-server and peer-to-peer architectures by using multiple servers instead of a single server. Each client sends and receives all updates via one or more servers. The servers themselves communicate using peer-to-peer architecture.

When a client sends an update message to its server, that server forwards the update to the relevant servers that provide services to clients which would be interested in the information. The other servers, in turn, forward the update to their interested clients. Compared with pure client-server architecture which centralizes the exchanging and processing of the data flow, the multi-server architecture can support more simultaneous participants. As the data flow may not be balanced among the servers due to the dynamic user activities in the virtual world, it could results in partial capability wastage at certain servers whereas other servers may be overloaded. Load balancing between servers is proposed to average workload of the system. So the scalability of the system can be improved by adding more servers which share the overall workloads equally. However, in practice, multi-server architecture may still have problems. With the increasing of the participant number, the server-to-serve communication and the computational complexity for the real-time VE state synchronization will exceed the pace of server expanding[15]. In addition, the simultaneous participant number is unpredictable and may increase in a bursting manner. In order to cope with the potential increase or the temporary burst of the simultaneous participants joining a LCVE, more servers have to be reserved, while much capacity of those reserved servers will be wasted in most of the time. The utilization of computing resources of the servers is not efficient. Therefore, multi-server architecture face the problem of either the number of servers is insufficient during peak-load or there is an over-dimensioning of the server deployed[16].

From the above discussion, we find client-server, peer-to-peer and multi-server architecture all have problems to efficiently control the data flows in LCVE. An adaptive architecture is needed to enable more efficient data flows exchanging and processing so as (1) to fully distribute data communication to avoid potential bottleneck; (2) and, at the same time, to adaptively control consistency of a persistent CVE.

1.3 Objectives

The main objectives of this research are as follows:

- To undertake a critical review on the existing technologies for improving the scalability of CVE systems;

- To propose a framework for building CVE systems with adaptive communication architecture to improve the CVE system scalability;
- To evaluate the performance of the proposed framework in a VRML (Virtual Reality Modeling Language) based CVE prototype system;
- To make recommendations on further research studies for optimizing the proposed framework.

1.4 Approach

In this thesis, a Mobile Agent based frame for large-scale Collaborative Virtual Environment (MACVE) is proposed to build LCVE systems. MACVE provides a kind of adaptive communication architecture, which is different from the traditional client-server architecture, peer-to-peer architecture and multi-server architecture.

In MACVE, the LCVE software system is decomposed into a group of mobile agents. Each mobile agent is responsible for an independent task to provide services to exchange or process the data flows within CVE. The mobile agents do not bond with any fixed hosts and they can migrate or clone dynamically at any suitable participating host, which includes traditional server hosts and qualified user hosts. By agent migration, multiple tasks will not converge at a single host; by remote agent cloning, the workload of a single task will be shared by multiple hosts. The mutual independence of services and hosts provide large freedoms for CVE system to utilize the system resource efficiently. The ability for qualified user hosts to take over the mobile agents makes the system's scalability less confined by the number of traditional servers. Thus, the system data flows and workloads can be pervasively distributed. Moreover, the data communication in different parts of a LCVE (e.g. region or cell or different activities in a cell) may adopt client-server, peer-to-peer, or multi-server architecture depending on run-time activities and consistency requirement in each particular part of the LCVE, which will further enhance LCVE system scalability.

A mobile agent system has been specially designed for MACVE to allow effective distribution of LCVE tasks. The asynchronous agent migration mechanism in the mobile

agent system ensures the continuity of the CVE activity during agent migration. Agent failure recovery mechanism is also proposed in MACVE to increase the system robustness.

1.5 Major Contributions of the Thesis

This thesis claims the following contributions to the state of the art:

(1) A Mobile Agent based framework for large scale CVE (MACVE) is proposed to improve the scalability of CVE systems, which provides the following unique advantages:

- The mutual independence of LCVE services and the participating hosts provide large freedoms to utilize the system resource efficiently.
- The ability for qualified user hosts to take over system mobile agents enables the system to utilize the extra computing resources of user hosts.
- The autonomy to decide the message communication architecture for different parts of a LCVE (e.g. region or cell or different activities in a cell) ensures the required consistency level and improves the system scalability.

(2) A mobile agent system, including Mobile Agent Environment (MAE), MACVE APIs, and MACVE Protocol, has been specially designed for MACVE to allow effective distribution of LCVE tasks. Based on our mobile agent system, a prototype CVE system is implemented, which includes eleven basic types of agent and a web-based user application.

(3) Experiments within the prototype system have demonstrated the feasibility of MACVE. It has also shown the advantages of the proposed MACVE to improve the scalability of the CVE applications.

1.6 Organization of the Thesis

The thesis is organized as follows:

- Chapter 2 presents a survey of related concepts, existing approaches and standards used in CVE systems to improve the scalability, and gives a critical assessment of the related works.
- Chapter 3 describes the details of the proposed mobile agent-based architecture for large scale CVE. The scalability of the framework is theoretically analyzed.
- Chapter 4 presents the details of the implementation of the mobile agent system, the prototype system of MACVE and the web-based user application.
- Chapter 5 describes our experimentation with the prototype system. System performance data has been collected and analyzed.
- Chapter 6 concludes with a summary of contributions and suggestions for future research.

Chapter 2

Literature Review

This chapter presents a review of related works on the scalability of CVE. It begins with a review of general concepts and terminology of CVE to serve as background knowledge of this research work. Then we classify the communication architecture of the existing CVE systems into three categories and based on the communication architecture, we review the existing approaches for improving the scalability of CVE. Well-known CVE systems are also reviewed with critical assessment. Finally, we summarize the need for further research and review mobile agent technology for potential application.

2.1 General Concepts and Terminology

Before discussing the scalability issues of CVE systems, some general concepts and terminology coupled with the system scalability are introduced.

2.1.1 CVE Consistency

Consistency is a fundamental goal of CVE, which is to provide participants with the illusion that they are all seeing the same virtual world and they can have a truly interactive experience in real time[17, 18]. In [19, 20], absolute consistency of CVE is defined as that at any point in time, all participants see the same information in a shared virtual environment at the same time through network connections. Absolute consistency in distributed application is impossible to attain[18], because of the existence of network latency. In some networks, such as LAN, which typically has very small delay, the inconsistencies between different participants are short lived and pass unnoticed to the participants[17]. However, in the wide area networks, such as Internet, the inconsistencies adversely affect the participants' real-time interaction[21-23].

CVE consistency is different from the traditional database consistency[17], because CVE must deal with the problem of "human in the loop", i.e. that participants provide real-time inputs to the system and they expect to receive the real-time responses from the system[24, 25]. The presence of human users requires system timely response. Low responsiveness makes the system feel unnatural and causes frustration[26, 27]. On the other hand, the participants may also be able to cope with levels of inconsistency and imprecision. CVE consistency issue refers to a number of aspects[18]:

- Synchronization

Synchronization is to maintain temporal relations of events so that the time of each event relative to other events across the CVE is the same for all participants[18]. As a continuous simulation system, CVE state is changed as a result of the passage of time and the occurrence of events. The synchronization of the events has a vital effect on the consistency of VE state among all participants[28]. To guarantee high degree of consistency among participants, the event initiator must wait until everybody has received the event before it may proceed with its event computations to update the state. The delay includes transmission time and network delay of the original event, waiting for acknowledgements that all remote hosts have received the event and possible retransmission in case the event was lost by the network. As the event initiator must wait for all other participant to receive the event to ensure the exactly same VE state, it can only generate VE state updates at a limited rate. In other words, there is a conflict between system responsiveness and consistency.

- Causal ordering

Events sent over the network maybe lost or arrive in a different order to which they were sent. A lack of event ordering may cause complete confusion. Total ordering of all events produces much overhead and it also reduce responsiveness of a CVE. So event ordering must be balanced with responsiveness. To reduce the overhead of the total event ordering, Lamport developed an optimization event ordering algorithm called causal ordering[29]. Causal ordering firstly define the event causal relationship. The definition of causal relationship is based on the subjective application knowledge which allows the

application to define the "before-after" relation. For example, a fire event may cause a detonation event. The fire event must happen before the detonation event [20]. If the two events are presented to participants in a reverse order, it will violate the realism. Causal ordering only ensure the proper order of the events in a causal relationship. It removes the need to order events that could not have been related. Causal ordering improves the CVE responsiveness and sacrificed the absolute consistency.

- Concurrency control

Concurrency control is another essential aspect of consistency management that deals with the prevention of concurrent conflicting updates. When a group of people try to move a given object in conflicting directions at the same time, the events for requesting moving the object will have the same timestamp. Without concurrency control, it is difficult to determine outcome because the events with the same timestamp can have different orderings during the event synchronization at different participants. The lack of ordering of these concurrent events can cause inconsistency and lead to frustrating or unrecoverable divergence. The common mechanisms for concurrency control is transferable object ownership, where a user can only affect an object once the ownership has been transferred to him across network. Many CVE systems that support real concurrent editing, do employ locking or ownership algorithms, adversely affected by network latency, resulting in reductions in responsiveness [30, 31].

To sum up, in reality, a CVE system cannot achieve both absolute consistency and high responsiveness at the same time, and there must be a tradeoff between these two attributes[32]. Especially in a LCVE, large number of concurrent users and virtual entities makes event synchronization and concurrency control more complex. CVE that demands absolute data consistency, causality and reliable communications while supporting real-time interaction are not likely to scale well[2]. In other words, a LCVE system can only achieve real-time performance if the view of the virtual world is not absolute consistent throughout all the users that participate in it[17, 33]. This conflict is called consistency-responsiveness tradeoff. So consistency control does not mean to ensure all events to be synchronized and all details of VE state to be consistent; whereas

it means to adopt different mechanisms to manage the events and updates of the corresponding VE state according to various consistency and responsiveness requirement. In this thesis, the consistency and responsiveness requirement for a certain type of events and the updates of the corresponding VE states is defined as a certain level of consistency.

2.1.2 CVE Persistency

CVE Persistency is the ability for the virtual environment to survive and to evolve even after all participants have left the environment[34, 35]. As introduced in Section 1.1, a CVE is a set of virtual entities perceived and manipulated by a group of geographically dispersed participants, who are embodied as avatars. Participants can join, leave and re-join in the CVE at will. When joining the CVE, they can change the CVE state through interacting with virtual entities or adding/deleting these entities. Persistency management is responsible for recording these changes even after all of the users leave.

There are two basic forms of persistency: state persistency; and evolutionary persistency. State persistency maintains an object in a static state after the participant who adds this object into the CVE has left. For example a participant built a house in the CVE and then he left. When he entered the CVE again, the state of the house was exact the same as he left (If there is no further changed to house by other participants). Evolutionary persistency supports ongoing behaviors of an entity once its creator has left[36, 37]. Hence when participants re-enter the CVE, the state of the world may have changed. For example a participant cultivated flowers in the CVE and then he left. When he entered the CVE after sometime, the flowers would grow up.

A successful persistent CVE also need to be extensible. Most virtual environments are statically extensible; that is, the environment implementation can be modified during a halt in execution, and clients must then be accordingly modified before rejoining the environment. An CVE cannot be considered truly persistent unless it both (1) maintains state regardless of client participation, and (2) has no planned interruptions in execution[38].

Supporting persistency is straightforward when the underlying CVE infrastructure is

client-server architecture or multi-server architecture (defined in section 2.2). Server can host master pieces of entities which are responsible for the state and evolutionary persistency of each entity. However, in peer-to-peer architecture (defined in section 2.2) master pieces of entities are held at the participating host which created the entities. The master entities need to move to other participating hosts when the creator participant leaves. And when all participating nodes leave, there is no one to preserve the CVE state. Therefore, to efficiently maintain CVE persistency in peer-to-peer architecture is still an unsolved problem.

2.1.3 Virtual Reality Modeling Language

Virtual Reality Modeling Language (VRML) is a standard file format for representing 3D interactive vector graphics[39]. It can be used to build LCVE on Web environment. The 3D scene described by a VRML file is known as virtual world and can be distributed over the World Wide Web (WWW) and presented in special VRML browsers, most of which are plug-ins for the web browsers. A reference to a VRML file can be embedded in a Hypertext Markup Language (HTML) page and hyperlinks to other media such as text, sounds, movies, and images can also embedded in VRML files. Thus, VRML is a way of extending WWW to 3D.

VRML originated in 1994 and the first official VRML 1.0 specification was released in 1995. The VRML 1.0 was a good start as an Internet-based 3D graphics format, but it was a static scene description language, which can not support user interactions. In 1996, VRML 2.0 was released which supports dynamic, interactive 3D scenes[40]. In 1997, VRML 2.0 was accepted by the International Standard Organization (ISO) as ISO/IEC 14772 standard, known as VRML97. The following is the main features of VRML 97:

- Hierarchical scene graph

  In VRML97, the basic element is the *node* which describes 3D shape and geometry, appearance properties to be applied to the shape's geometry, light sources, viewpoint, and so on. Each node is typed and has a set of *fields* that parameterize the node. The 3D scene consists of a set of nodes arranged in a hierarchical fashion. The hierarchy

is built up by using a parent-child relationship in which a parent may have any number of children, some of whom may, in turn, be parents themselves. *Scene graph* refer to the entire ordered collection of these scene hierarchy.

- Event routing mechanism

VRML97 provides an event routing mechanism to support dynamic and interactive 3D scene. Some nodes can generate events in response to environmental changes or user interaction and other nodes can receive events to change the state of the node, generate additional events, or change the structure of the scene graph. These nodes may be connected together by *ROUTE* to achieve real-time animations including entity behaviors, user-entity interaction and inter-entity coordination.

- Prototyping mechanism

Prototyping provides a way to extend the build-in node types. It defines new type of nodes, known as prototype node, to parameterize the scene graph. Once defined, prototyped node types may be instantiated in the scene graph exactly like the built-in node types.

- External Authoring Interface

External Authoring Interface (EAI) is an interface specification that allows an external program to manipulate the VRML scene graph. The implementation of EAI in Java is a set of classes with methods that can be called to control the VRML world to support dynamic scene changes.

However, VRML does not, at present, address any of the issues to do with networking these worlds to enable multiple participants to interact with each other or distribute the workload associated with the simulation.

2.1.4 Existing Standard and Framework for CVE

In order to facilitate the development of CVE, some standards have been proposed. We review three key standards in the following sections.

2.1.4.1 Distribution Interactive Simulation Standard

Distributed Interactive Simulation (DIS) standard is a suite of protocols, approved as IEEE 1278-1993 series standards, for interconnecting large numbers of heterogeneous simulators across local and wide area networks to create a consistent, time and space coherent synthetic environment with respect to human perception and behaviors[41].

DIS design principles

DIS standard emergence stems from the success of the SIMNET. SIMNET (SIMulator NETworking) was a synthetic virtual environment (or virtual battlefield) for military commanders and troops to practice their skills. It is sponsored by Defense Advanced Research Projects Agency (DARPA), in partnership with the US Army, and was developed and implemented between 1983 and 1990. SIMNET was the first successful implementation of large-scale, real-time, man-in-the-loop simulator networking for team training and mission rehearsal in military operations. It can link together hundreds or thousands of simulators (representing tanks, infantry fighting vehicles, helicopters, fixed-wing aircraft, etc) to create a consistent virtual world, in which all participants experience a coherent, logic sequence of events. In order to extend SIMNET's underlying principles to widespread use with assurance of interoperability between heterogeneous simulation components, DIS standard was proposed. DIS inherits the key design principle and architecture decisions of SIMNET and then progressively refines them. These principles include[42, 43]:

- Object/Event architecture

DIS environments model the world as a collection of objects, which interact with each other through a series of events. All dynamic objects in the virtual environment inform all other objects of their status and actions through the transmission of standard formatted packets, called Protocol Data Units (PDUs).

- Distributed autonomous simulation nodes

The database of the DIS environment is fully replicated at each node. All events are broadcast on the simulation network, and are available to all objects that may be

interested in them. In DIS environments, there is no central control process. Each node is responsible for receiving event reports from other nodes, and calculating the effects of this event on the object(s) they are simulating.

- Dead reckoning algorithms

Dead reckoning algorithm provides a way to reduce network traffics by controlling the frequency of object state update. The idea is to transmit state update packets less frequently and the missing information is computed with an approximation technique. Based on previously received information, the node predicts movement of a particular object.

Dead reckoning follows the players and ghosts paradigm. In this paradigm, each object is controlled on its own host workstation by a software object called a Player. On every other workstation in the network, a version of the Player is dynamically modeled as an object called a Ghost. The Ghost objects on each workstation update their own position each time through the simulation loop, using a dead-reckoning algorithm. The most common dead reckoning protocols use derivative polynomials to predict an entity's future position. In the case of positions, their natural interpretations are velocity, acceleration, and jerk. In the case of first-order derivative polynomials, the velocity of an object is transmitted in addition to the position. Velocity improves prediction accuracy noticeablely. To improve accuracy further, the acceleration is added to the transmitted terms. This second-order polynomial prediction is the most popular technique, which is used in DIS protocol[44]. The Player tracks both its actual position and the predicted position calculated with dead-reckoning. An updated Entity State Message is sent out on the network when the two postures differ by a predetermined error threshold, or when a fixed amount or time has passed since the last update (nominally 5 seconds). When the updated posture (location and orientation) and velocity vectors are received by the Ghost object, the Ghost's is corrected to the updated values, and resumes dead-reckoning from this new posture.

Dead reckoning can introduce also other side effects. By using prediction, Dead reckoning algorithm sacrifice consistency of the VE at each host[2]. In addition, It

reduces network traffic by minimizing updates at the cost of extra computation by the host system[45].

DIS Protocol Data Unit

The core of the DIS standard is the PDUs which describe the dynamic activities and events happening in the virtual environment. The DIS standard defines a set of 27 PDUs for events in military combat simulation. In practice, only four of these PDUs are for node interaction in the virtual environment (Entity State, Fire, Detonation, and Collision). The most significant PDU is the Entity State PDU (ESPDU), which is used to define the position and orientation as well as velocities and accelerations (both linear and angular) of entities in the simulation.

DIS Scalability

The broadcast scheme for the PDU transmission that is adopted in the early DIS systems can be a problem for DIS scalability, because most of the PDUs received by a participant simulation are irrelevant, for example, the sending participants are out of its view. This result in an increase of receiver's processing load in order to filter out his irrelevant data, as well as an increase of the network traffic needed to deliver this unneeded data. Thus, the notion of *interest management* over multicast has been proposed to filter PDUs at network layer. NPSNET was the first DIS system to use multicast protocol, which is discussed in Subsection 2.4.1 in detail.

The object/event architecture, the distributed autonomous simulation node, the dead reckoning algorithm and the interest management over multicast make the DIS incredibly successful as far as scalability is concerned. However, as the DIS is designed for the military war simulation, most PDUs are not suitable for general purpose CVE systems. The fully replicated VE scene data limits the virtual world to have evolving and rich content. As there is no center control process, DIS does not provide persistency.

2.1.4.2 High Level Architecture

As the next generation of the DIS standard, High Level Architecture (HLA) was proposed in 1995 by the US Defense Modeling and Simulation Office (DMSO) to

provide a structure which will facilitate interoperability amongst heterogeneous simulation and promote reuse of simulations and their components, ultimately reducing the cost and time required to create a synthetic environment for a new purpose. As such, the HLA is an integrated approach that has been developed to provide a common architecture for simulation[46, 47].

Two basic concepts have been proposed in the HLA: federate and federation. Federate is a software application participating in a federation, which may be simulation model, data collector, simulator, autonomous agents or passive viewer. Federation is a named set of federate applications that are used as a whole to achieve some specific objective. Figure 2-1 shows the functional view of an HLA federation. All federates incorporate specified capabilities to communicate through the Runtime Infrastructure (RTI), which is an implementation of a distributed operating system for the federation. The HLA runtime interface specification provides a standard way for federates to communicate with each other by interacting with the RTI.

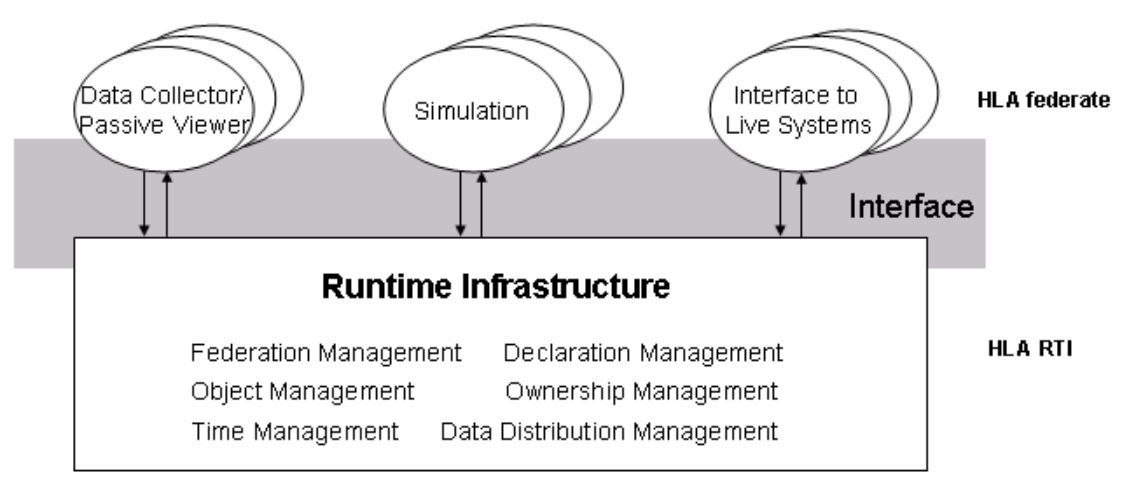

Figure 2-1 Functional view of an HLA federation [46]

The HLA consists of three components: the interface specification, the object model template and the HLA rules.

HLA interface specification

The HLA interface specification describes the runtime services provided to federates by the RTI, and to RTI by federates. It defines the way to access these services, both functionally and in programming APIs. These services are classified into six categories, namely [48]:

- Federation Management (FM): offers basic functions required to create/delete federation executions; join/resign federation executions, as well as pause/resume federation executions.
- Declaration Management (DM): supports efficient management of data exchange by data publication/subscription.
- Object Management (OM): provides services to create/delete object instances; update/reflect attributes of objects, request/provide attributes of objects, etc.
- Ownership Management (OWM): supports dynamic transfer of ownership of object attributes during an execution.
- Time Management (TM): supports synchronization of runtime simulation data exchange.
- Data Distribution Management (DDM): supports efficient routing of data among federates.

HLA Object Model Template

The Object Model Template (OMT) is a standardized format to define the functionality of a federation and the interaction between each federate[49]. HLA use the OMT to describe the essential sharable data used by the simulation or federation. The OMT are intended to be the means for open information sharing across the community to facilitate reuse of simulation.

The HLA specifies two types of object models: the HLA Federation Object Model (FOM) and the HLA Simulation Object Model (SOM). The HLA FOM describes the set of objects, attributes and interactions which are shared across a federation. The HLA SOM

describes the simulation (federate) in terms of the types of objects, attributes and interactions it can offer to future federations. Both FOM and SOM are documented using the HLA OMT.

HLA rules

The HLA rules summarize the key principles for defining the HLA. There are ten basic rules. Five related to the federation and five related to the federate. The detailed rules can be found in [47].

HLA Scalability

HLA is more scalable than DIS, because HLA provide a fine-grained filter to reduce the necessary network bandwidth through use of some RTI mechanisms. RTI DM provides a mechanism of publication and subscription. Data source federates publish interactions or attributes it represent to a federation and data consumer federates only subscribe to certain types of interactions or attributes it needed. This mechanism filters the irrelevant types of interaction and attributes data between federates. RTI DDM defines routing spaces for federations and subscription regions and update regions for federates. The overlapping of subscription and update regions in the routing space enable the RTI to establish the communication between federates. This mechanism filters the irrelevant interaction and attributes of certain federates. Both DM and DDM can use multicast as the underlying communication protocol. Based on DIS experience with multicast, HLA proposed standard ways to implement multicast schemes.

2.1.4.3 Virtual Reality Transfer Protocol

Virtual reality transfer protocol (VRTP)[50] is an application level protocol to provide powerful network capabilities for large-scale collaborative VRML world.

As introduced in Subsection 2.1.3, the emerging of VRML makes it possible to building 3D virtual world on WWW. However, the underlying network support provided by the hypertext transfer protocol (HTTP) is insufficient to make the virtual world to support large number of concurrent users' interactions. Additional capabilities for many-to-many peer-to-peer communications plus network monitoring need to be combined with the

client-server capabilities of HTTP. To accomplish this task, VRTP is designed to support VRML worlds in the same manner as http was designed to support interlinked HTML pages[50].

In LCVE, users and virtual entities have both static content data and dynamic streams data, which can be delivered through client-server mode or peer-to-peer mode. Entity interactions among LCVE may occur in multiple ways, which also can be sent by client-server mode or peer-to-peer mode. Thus the participants in LCVE simultaneously need to act as clients, server and peers. VRTP provides such flexibility by the following four components[50, 51]:

(1) Client components

Client component provide the capability to act as a client to request 3D world data from other application. The proposed solution for client component is to provide efficient hooks to commercial Web browser application program interfaces (APIs) so that client component can integrate with the browser and other plug-in programs.

(2) Server component

Server component provide the capability to act as a server to share its 3D world data to others. The proposed solution for server component bases on existing http server applications. Server component added into http server via Common Gateway Interface (CGI).

(3) Peer-to-peer component

Peer-to-peer component provides the capability to act as a peer to send and receive multicast interaction messages or real-time streams of multimedia data. Multicast interface and DIS are included in peer-to-peer component.

(4) Monitoring component

Monitoring component provides the capability to monitor, diagnose and correct the network problems.

As VRTP needs multicast enabled backbone services which provide dedicated multicast, dedicated bandwidth and dedicated latency, it does not used by most of the commercial CVE systems as it intended to be.

2.2 Communication Architecture

The communication architecture of CVE describes how data exchange among participants to construct the distributed virtual world. Essentially, three basic architectures are exploited by existing CVE systems[52, 53].

2.2.1 Client-Server Architecture

As shown in Figure 2-2, in client-server architecture, participant's host is called a client. Each client communicates only with a common server. Clients send their messages to others via a server and they receive other client' messages from the same server. A server is usually a computer with high capability which actually functions as a proxy for clients to forward messages. Clients-server architecture is conceptually simple and adopted by most small-scale CVE systems[54].

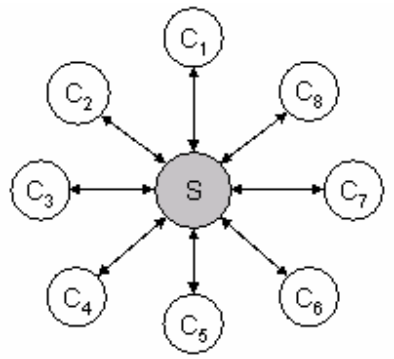

Figure 2-2 Client-server architecture

The client-server architecture is easy to support heterogeneous client hosts because the server can tailor its communication to match the network and machine capabilities of each client. The server can reduce message traffics at individual client by not sending messages to those clients if the packet is not needed. It can also convert the bursting rates of packets generated by an individual client into smoother packet rates to cater for the clients with slow network connection speed. For example, SPLINE supports dial-up users

through specialized servers that connect them to the core system over a compressed and optimized application protocol.

Moreover, the client-server architecture has other good features. As all messages pass through the server, it is easy to support consistency control. As the server can record the changes occurred in the VE, it is easy to support persistency management. Server can also compress multiple packets into a single message which eliminates redundant and unnecessary message flow. Finally, Client-server architectures are also preferred if accounting and security problem is concerned. Many commercial CVE systems use the client server model, not only for technical but also for administrative and financial reasons[1].

However, the client-server architecture has scalability problems. Because all traffics must pass through the central server, as the number of participant increases, the server rapidly becomes the bottleneck component that impedes further growth in the number of VE participants. Experience and research have shown that even if million–dollar servers are used, the limit on the number of clients happens much faster than if peer-to-peer, fully distributed architecture has been designed first[55, 56]. Another problem of client-server architecture is that since each message must be via the server, it brings additional delay for participant's communication. This delay reduces the responsiveness of the CVE. Finally, single server may also become the single point of failure of the whole system. Therefore, client-server architecture can not be used to build LCVE systems.

2.2.2 Peer-to-Peer Architecture

In this architecture, each individual participating host is called a peer. Peer sends information directly to other peers to maintain a consistent CVE at each participant side. The VE scene data is replicated at each peer in advance. As there is no intermediate process for message exchanging between peers, the CVE system with peer-to-peer architecture has high responsiveness. As shown in Figure 2-3, the underlying communication protocol for peer-to-peer architecture can be unicast, broadcast, or multicast[57, 58].

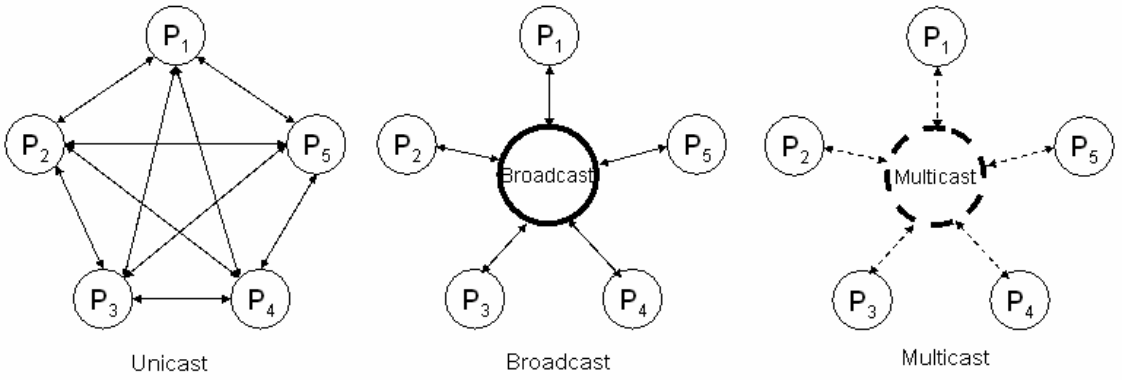

Figure 2-3 Peer-to-peer architecture

When the underlying protocol is unicast, messages are exchanged by point-to-point communication protocol, e.g. TCP/IP, UDP/IP. If there are *N* peers in the CVE, each peer has to send the same message to *(N-1)* peer repeatedly to guarantee the consistent VE state at each peer. This has lead to the burst of the network traffic and the burden on single peer's processor[58]. There will be *N*(N-1)/2* duplex connections in the network which is bandwidth intensive for the network. Therefore, it does not scale to large number of simultaneous users before the network saturated. Reality Built For Two[59], VEOS[60], and MR Toolkit[61] adopt this architecture.

When the underlying protocol is broadcast, each peer needs only send its messages once; every peer in the same network can receive the message. It reduces the bandwidth requirement for each peer. However, broadcast is less than an ideal solution. It does not transit multiple networks well (if at all). In switched environments, broadcast will flood an entire broadcast domain with traffic. IP broadcast requires that all hosts examine a packet even if the information is not intended for that host. Hosts not participating in the CVE still need process broadcast packets and then throw them away, which wastes CPU cycles. For these reasons, using broadcast in LCVE systems is only workable on dedicated networks, such as those that might exist at a LAN or at a single military training site[62]. This architecture is adopted in SIMMET[42] and VERN[63].

When the underlying protocol is multicast, similar as broadcast, each peer needs only send a message once; all other peers in the same multicast group can receive the message. The difference is that multicast extends broadcast ability to wide area networks, for

example the Internet. Multicast is bandwidth efficient and it provides one-to-many and many-to-many delivery services for applications. When an application subscribes to a multicast group, it can exchange messages with all the application subscribed to the same multicast group. Unlike broadcast, multicast do not consume more processor cycles. The multiplexing and de-multiplexing of data belonging to different multicast group is done at the network layer, so it does not consume processor cycles. NPSNET[56] is the first CVE system to adopt IP-multicast to achieve the scalability in terms of simultaneous user number. Multicast is also used exclusively in DIVE[33], MASSIVE-2[64], SPLINE[65], SmallTool[66] and SCORE[67]. Therefore, peer-to-peer architecture with multicast can be used to build LCVE systems.

However, there are some shortcomings for peer-to-peer architecture. Since there is no central control process and every peer is equal in the system, the consistency control is done by peer itself. So it is difficult to maintain CVE consistency in peer-to-peer architecture. Actually, most of the CVE systems with peer-to-peer architecture only provide approximate consistency control. In addition, since after all peers log off, there is no control process to maintain the CVE state, persistency management is also difficult to maintain in the system with peer-to-peer architecture. Finally, since all data flows exchanging and processing is done by each peer, peer-to-peer architecture do not have a good support for heterogeneous participants and it requires each peer to have high networking and computational capability.

2.2.3 Multi-Server Architecture

As shown in Figure 2-4, in multi-server architecture, multiple servers work together to support the clients. Multi-server architecture is also called hybrid architecture, which merges client-server and peer-to-peer architectures.

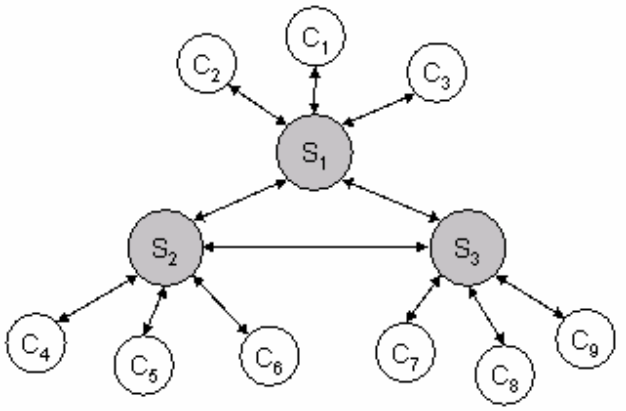

Figure 2-4 Multi-server architecture

Multi-server architecture attempts to take advantage of both architectures. Each client sends and receives all updates via one or more servers. The servers themselves communicate using peer-to-peer architecture. When a client sends an update message to its server, that server forwards the update to other peer servers which provide service to clients that would be interested in the message. The other peer servers, in turn, forward the update to their interested clients. Multi-server architecture can be seen as the extension of client server architecture which uses multiple servers instead of a single server.

One of the methods to extend pure client-server architecture to multi-server architecture is to use multiple servers to preserve a same VE. These servers synchronize the real-time state of the VE so that clients can connect to any of them to join the VE. We define this method as server mirroring. To replicate component across a distributed system is a good way to improve the system scalability[68]. The replicated servers not only increase the availability, but also help to distribute the system overload, which leads to better performance[69]. In the server to server communication, besides exchanging the object state data generated by their connected clients, the servers periodically exchange control messages containing composite information about interests of their respective clients. Having multiple servers effectively reduces the computational workload on each server because the clients are effectively divided among them. However, the introduction of multiple servers does incur some overhead. First, because updates need to be exchanged through multiple servers, they encounter a greater latency than through a single-server

system. Second, the output bandwidth required by the servers is greater than the resources required in a single server system, because except exchange all the client update messages, the servers need exchange control messages to inform other servers the area of interest of their clients[2]. The early CVE system, BrickNet [70], adopted this architecture.

Another method to extend pure client-server architecture to multi-server architecture is to geographically partition the VE into multiple parts (e.g. region or cell) and use multiple servers to preserve these parts of the VE respectively. With this method, each server is only responsible for clients located within a particular part of the VE, and each client communicates with different servers when it moves through different part of VE. We defined this method as server splitting. In order to support the continuous VE, the servers need to exchange information to make the border area visible to the client in the neighboring VE part. Partitioning the VE can eliminate the information exchange among the servers[2]. However, there are two difficulties in this method. Firstly, it is difficult to find a general algorithm to partition the VE. Secondly, it is difficult to synchronize among different servers in order to maintain the VE coherence[15]. NetEffect[71], SpaceFusion[72] and CyberWalk[73] adopt this architecture. In fact, most multi-server systems adopt both server mirroring and server splitting, such as RING[74], extensive version of Community Place[75], NATIVE[76], MMORPGs[15, 77], etc.

In summary, more concurrent users (participants) can be supported in a multi-server architecture by adding more servers. However, servers mean that eventually there is some finite limit to the number of participants[2]. As the number of concurrent users increases, inter-server communication and the computational complexity for the real-time VE state synchronization will exceed the pace of server expanding[15]. In addition, the utilization of computing resources of the servers is not efficient, because the multi-server architecture faces the problem of either the number of servers is insufficient during peak-load or there is an over-dimensioning of the server deployed[78]. The number of concurrent participants is unpredictable and may increase in a bursting manner. In order to cope with the potential increase or the temporary burst of the simultaneous participants

joining, more servers have to be reserved, while much capacity of those reserved servers will be wasted in most of the time.

2.3 Existing Approaches for improving the CVE Scalability

Based on the communication architecture, there are numerous research works attempting to address the scalability issue with various methods or algorithms, which can be categorized into three broad categories: data flow transforming, data flow filtering, and data flow redistribution.

2.3.1 Data flow transforming

Data flow transforming approach aims to reduce the required network bandwidth by changing the data format to transmit fewer bits in underlying network to improve the data transmission efficiency. There are two specific data flow transforming methods: packet compression and packet aggregation.

Data compression is the process of encoding information using fewer bits than a more obvious representation would use, through use of specific encoding schemes. Packet compression seeks to use the data compression method to reduce the size of the packets transmitted within a CVE. The sender is responsible for shrinking data for transmission, while receiver is responsible for restoring the actual data.

Packet aggregation seeks to reduce the number of packets that are actually transmitted by merging information from multiple packets into a single packet to share a single package header. This packet merging saves network bandwidth by reducing the number of network packet headers that are sent over the network. For example, every UDP/IP packet includes a header of 28 bytes, and every TCP/IP packet includes a header of 40 bytes[2].

However, the effect of data flow transforming to reduce the data flow is limited. At the same time, extra computing resource is needed to compress/uncompress or aggregate/ separate the dataflow. Thus, this method only has limited contributions to improve the scalability of CVE systems.

2.3.2 Data flow filtering

Data flow filtering approach aims to reduce application layer data exchanges within the CVE as much as possible without harming the shared context and interactive performance. Data flow filtering is also called interest management[79]. The underlying assumption behind data flow filtering is that a LCVE contains a tremendous amount of information, yet an individual user only needs to know about a subset of the total information because of the scope of their interest and perception. In the past ten years, various filtering models have been proposed to control the data exchanged scope and data flow quality to decrease the unnecessary bandwidth and processor consumption.

2.3.2.1 Controlling data flow propagating scope

Real-world observation finds that at a given moment, each individual only has a limited visibility and "sphere of interaction". In other words, a user's interest is localized[79]. Therefore, instead of broadcasting each packet to every participating host, the LCVE system should strive to only send the information to those hosts that really need to receive it. By limiting the dissemination of data, a CVE system confines the scope of a user's awareness and the range of interactions so that the consumption of network bandwidth and computing resources decreases. Various models have been proposed to determine the data propagation scope and we summarize them into four broad categories: distance-based filter model, zone based filter model, subscription-based filter model, and behavior-based filter model.

In distance based filter model, the distance between two interactive objects is used to decide the data propagation scope. In order to contend with the limited network bandwidth, only the objects within a limited distance threshold can exchange data with each other. In RING, when an object change its state, update messages are sent only to participating hosts with entities that can potentially perceive the change, i.e., the ones that can see it. The aura model proposed by DIVE and MASSIVE-1 is also a distance based filter model. An interactive object's aura defines its overall region or scope of interest in a medium. The aura is attached to the user and moves with the user's navigation steps. The collision of two entities' auras enables them to make connection and find out about

each other. Community place[75], VELVET[80] and other system inherit and refine this aura model. The Aura model includes other concepts which can also control the data flow fidelity. We will discuss it as a key feature of MASSIVE in Subsection 2.4.4 in details.

In zone-based filter model, VE is spatially divided into multiple connected zones. A participant only propagates his messages to others who locate in the same zone or neighboring zones. The zones divide participants into groups and multicast can be used for the group communication. Locale notion in SPLINE [81] is a mechanism to filter the data outside the neighboring chunks. Dynamic partitioning the virtual environment into spatial areas and the association of these areas with multicast groups in SCORE [82] provide the method to dynamically ameliorate the filtering scope to decrease the superfluous traffic.

In subscription based filter model, each participant submits its interest description to a subscription manager. For each participant's message, the subscription manager determines which participants are interested in this message and only disseminate the message to the corresponding participants. Different from distance-based model and zone-based model, subscription based filter model can define logic communication groups, which has no relationship with the object spatial information. Subscription based filter model can also base on multicast. For example in NPSNET, functional class limits the military control messages only within a subset of entities[83].

Behavior-based filter model[84] incorporates a two-tiered architecture, which includes the high-level role behavior-based interaction management and the low-level message routing. In the high level, the interaction management is achieved by enabling the natural interactions based on the collaborative behavior definitions provided by CVE application developers. Thus, it extends the developer's management ability of the collaborations in the CVE application. In the low level, the message routing controls the propagation scope of the interaction messages according to the run-time status of the interactive entities, and hence reduces the network bandwidth consumption. SmartCU3D[85] adopts this model to filter the unnecessary messages for the clients.

2.3.2.2 Controlling data flow fidelity

Controlling data flow fidelity is a technique to tune the exchanged data flows to match more closely with the fidelity requirements of the participants at the receiving hosts[2]. As different participant may have different focus and perceptual scope, a single data flow is inappropriate to serve all receivers. For example, if the two participants are too far away from each other, it is unnecessary to send one participant's detailed facial expression messages to the other.

LCVE system can reduce the amount and timeliness of information that must be presented to that participant. First, information about entities in the VE can be presented at multiple levels of detail[3]. Only the users who are located near the entity need to receive the high-detail information, while others can be adequately serviced with less detail. Second, if particular information does not directly concern the local user, then that information need not be provided in an up-to-date manner[2, 86]. In particular, the user can be presented with a slightly out-of-date representation of distant entities, thereby reducing the real-time delivery requirements of the network without noticeably affecting the realism of the net-VE.

However, although data flow filtering reduces bandwidth requirements for data communication in CVE, data processing for filtering algorithm generate huge computational complexity, which may cause bottleneck of computing power. Therefore, dataflow filtering alone can not make the CVE scalable either.

2.3.3 Data flow redistribution

Data flow redistribution approach aims to efficiently disseminate information in the CVE by optimizing the route of data flows on the fly, which can avoid data flows converging at a single host to produce bottleneck.

As discussed in section 2.3.3, the basic client-server architecture can extend to multi-server architecture to improve the system scalability. In multi-server architecture, according to the server that the client is connected to, the client data can have different route to transmit to its destinations. Because of the dynamic user activities in the virtual

world, data flow may not be distributed reasonably, and thus the workload is unbalanced among the servers. Load balancing among servers is a way to dynamically change the route of the data flow. It can be achieve by dynamically adjusting the size of the VE content or number of participants managed by each server. In CittaTorn[87], CyberWalk[73], a similar idea of dynamic data flow balancing was proposed. The servers' loads are equalized by dynamically adjusting the size of the space managed by each server.

Data flows redistribution approach improves CVE scalability from system architecture perspective. More adaptive architecture for CVE system is needed to be researched[2].

2.4 Review of Well-known CVE systems

To examine how the existing approaches and standards/frameworks for improving scalability are used in the previous CVE systems, we reviewed several well-known CVE systems.

2.4.1 NPSNET

NPSNET (Naval Postgraduate School Networked Vehicle Simulator) [45, 56] is a 3D networked virtual environment developed by the NPSNET Research Group of the Naval Postgraduate School in Monterey, California, U.S. It is designed to support large-scale military training and simulation exercises. NPSNET is capable of simulating articulated humans, stealth objects, sea-going vessels, ground vehicles, and air vehicles. It also allows adding new entity models such as symbols, terrain, landmarks, as well as user-defined models to the simulation.

Among all the versions of NPSNET, NPSNET IV is most successful one and it also the first CVE system to adopt the Multicast Backbone (MBone)[88] of the Internet. NPSNET also complied with the Distributed Interactive Simulation (DIS) protocol to interoperate with other simulation systems. Besides dead-reckoning algorithm inherited from DIS, Area of Interest Manager (AOIM) is the key features NPSNET IV to allow system scalability.

As shown in Figure 2-5, AOIM partitions the VE into a set of spatial, temporal, and functional classes and associate these classes with multicast groups.

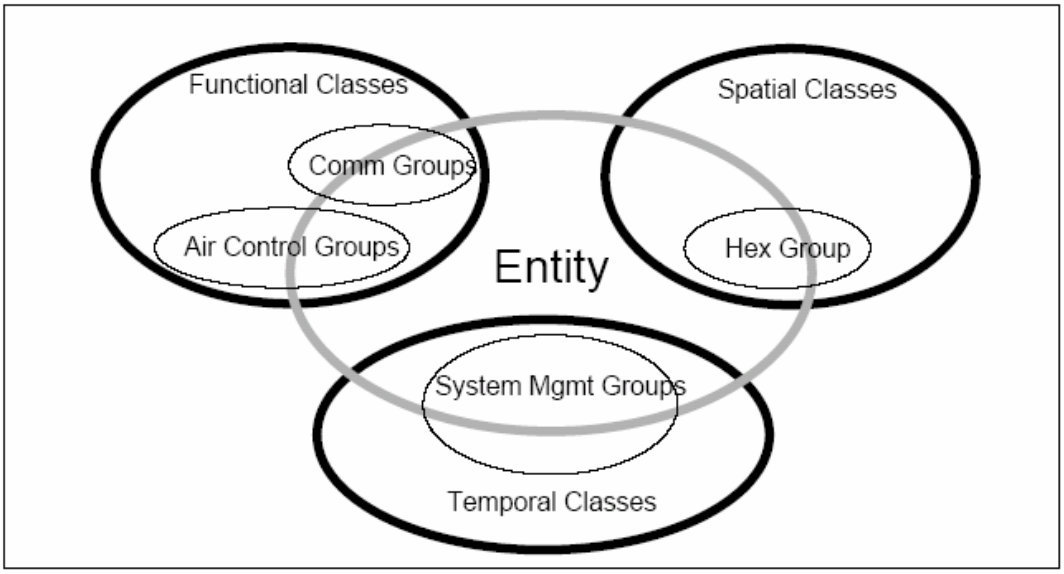

Figure 2-5 Relationship between entity and multicast groups in NPSNET[83]

Participants only need subscribe multicast groups of its interested classes to reduce the computational load on individual hosts, and to minimize communication on the network. For example, in NPSNET, the VE is spatial divided into hexagons called cells. Each cell associates with a multicast group. Participant only sends/receives state information to/from multicast groups corresponding to the local cell it is located in and the surrounding cells. NPSNET IV declared it can support more than 1,000 simultaneous participants.

However, AOIM works well when entities are distributed evenly within the VE, but fails if all entities are clumped within the same cell[86]. In addition, NPSNET was designed for military war simulation with dedicated network and hosts. To support heterogeneous internet based users, it needs to take into account of user hosts having very different computing power and network connection speed/stability. Moreover, although NPSNET is very suitable for the specific application area (military simulations), it does not fit very well to general purpose CVE, where the type of objects and their states usually will not be known in advance[66].

2.4.2 Spline

The Scalable Platform for Large Interactive Networked Environments (Spline)[65] is a software platform suitable for implementing multi-user interactive environments developed by the Mitsubishi Electric Research Laboratory. Spline aims to make the multi-user Virtual Environment large in spatial extent, large in number of objects, and large in numbers of users interacting with the environment. It has been used to develop Diamond Park VE[65] with bicycling and social interaction as the main topic.

The Spline platform provides a convenient architecture for implementing multi-user interactive environment based on a shared world model. Spline's world model is an object-oriented database that contains a snapshot of what the virtual world is like at the current moment. Spline applications do not communicate directly with each other, but rather only with the world model. This allows applications to be written without thinking about how communication is achieved. The Spline communication model is shown in Figure 2-6.

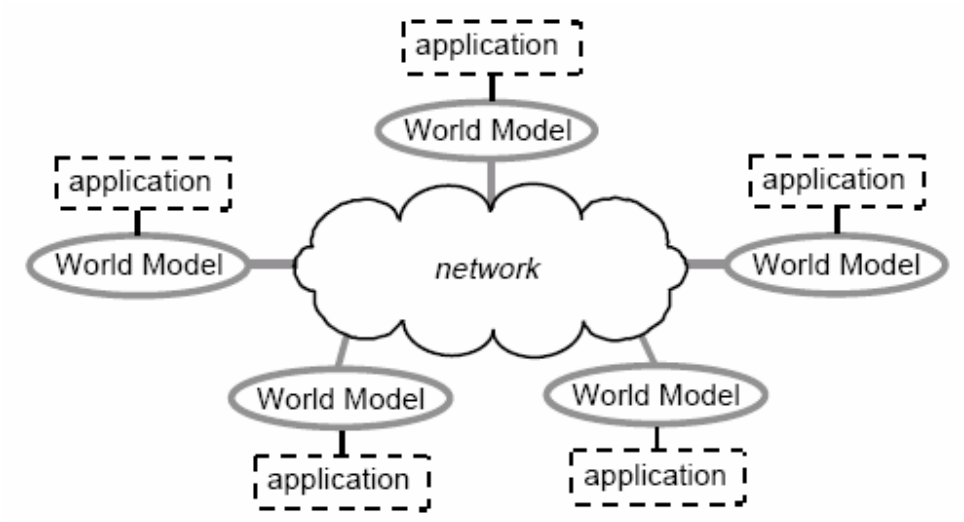

Figure 2-6 Spline's communication model[65]

A central feature of Spline is that it is designed to be scalable to a large number of users (e.g., thousands) interacting in real time. Two key aspects of Spline support the scalability:

(1) Spline provides only approximate equality of local world model copies. Although different users may observe the VE at slightly difference, approximate equality of local world model copies allows multi-user real-time interactions.

(2) Spline divides the world model into chunks called locales. The concept of locale is based on the idea that even in a very large virtual world most of what a single user can observe at a given moment is nevertheless local in nature. Users in a locale communicate only to the small group of users and entities that are located in the same locale and its immediate neighboring locales, rather than to all the users of the world model. Each locale can be any size or shape and is associated with a separate set of multicast addresses, which guarantees the efficiency of the communication.

The primary communication in Spline is peer-to-peer. Centralized processes are also used to provide services such as session management, naming service -- Beacons[65] etc. However, an object exists only as long as the application that owns it runs, so to maintain the Spline world persistency overtime is not supported by Spline itself.

2.4.3 DIVE

The Distributed Interactive Virtual Environment (DIVE) is a software architecture for implementing multi-user interactive virtual environments, developed at the Swedish Institute of Computer Science (SISC). DIVE aims to build multi-user environments that are spatially large, possibly infinite, contain many detailed objects and visualizations, and allow many simultaneous participants to interact with each other, objects, and processes present in the environment.

DIVE architecture consists of a set of applications processes running on hosts with a local or wide area network. An application or process is an active program to construct the VE. On each host, there is a shared, distributed world database called world abstraction. DIVE applications operate solely on the world abstraction and do not communicate directly with each other. This technique allows a clean separation between application and network interfaces[33]. The shared database is partially replicated at each application

process. Modifications to the database are applied locally first, then distributed to all connected peers through network message and applied to the local copy at each receiving peer. DIVE provides low-latency local user interaction while it tolerates world copies that temporarily differ slightly.

DIVE focuses on peer-to-peer multicast communication. A DIVE world is a hierarchical database of what is called entities. DIVE world itself is the topmost entity. The world database modifications are sent using a scaleable reliable multicast (SRM) protocol which minimizes the network load. DIVE uses multicast communication at two different levels of the world hierarchy. The topmost entity of the hierarchy (DIVE world) is associated with a multicast group that will be used by default for all data communication within the world. Any entity of the hierarchy can be associated with a different multicast group. When a modification message concerning an entity is to be sent, the algorithm will climb the hierarchy, starting from the entity. As soon as a multicast group is found, it will be used as a communication medium. If no group is found during the ascension, DIVE will use the default world group associated with the topmost entity. In order to allow multicast unaware processes to join regular multicast sessions, DIVE have proxy servers to act as a message replicator for all connected DIVE applications.

To reduce network traffic overhead, DIVE provides a mechanism for dividing the world into sub-hierarchies which may be associated with a multicast communication channel, called a lightweight group. Processes that are not interested in these sub-hierarchies can simply ignore this branch of the entity tree and cut down network traffic. Lightweight group provides the method to exchange updates only among a group of entities, which can be used to implement interest management algorithm. DIVE also implements dead-reckoning mechanisms to reduce the number of messages generated when objects move.

The current DIVE version handles persistency by running a neutral process, called PERSISTENT, that keeps its own active copy of the whole world database and periodically saves its state to a file, from which the world database can be restored if the system needs to be restarted.

However, the lightweight group in DIVE is predefined. When large number of entities subscribe to a same lightweight group at run time, scalability is not easy to achieve.

2.4.4 MASSIVE

MASSIVE[89] (The Model, Architecture and System for Spatial Interaction in Virtual Environments) was developed as an experimental CVE system by the Computer Science Department of the University of Nottingham, which aims to be a large-scale multi-user Virtual Environment with rich facilities to support user interaction and cooperation.

In MASSIVE-1, the researchers proposed a spatial model for interactive messages filtering, which received widely acceptance in the later CVE research. The key concepts in the spatial model includes: --- aura, focus, nimbus, and awareness.

- Aura: Each object in a virtual world has an aura for each medium in which it can interact. This aura defines the extent to which interaction with other objects is possible.
- Focus: Focus describes the observer’s allocation of attention.
- Nimbus: Nimbus describes the observed object’s manifestation or observability.
- Awareness: One object's awareness of another object defines the relevance of another object in a given medium. Since awareness of objects needs to be mutual, levels of awareness are negotiated between objects, in terms of focus and nimbus.

Aura, focus, and nimbus are medium-specific, and may have arbitrary values and extents. The combination of objects' auras, i.e., the observer object's focus, and the observed object's nimbus restrict the communication between objects.

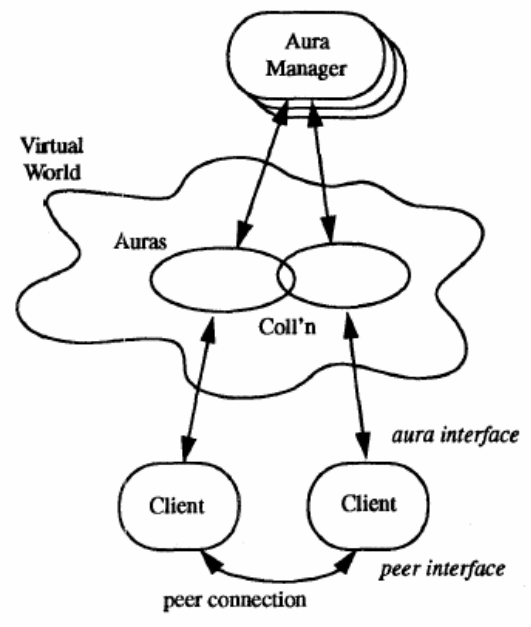

Figure 2-7 Aura manager in MASSIVE[89]

As shown in Figure 2-7, a spatial trader serves as an aura manager to check for aura collisions. If aura manager detect a collision, it will notifies any object concerned to set up a peer-to-peer connection and find out about each other. When the peer-to-peer connection is enabled, the communication is managed according to mutual levels of awareness which are negotiated through the use of focus and nimbus. The calculation of mutual awareness levels is the responsibility of the peer objects. The aura model decreases the unnecessary communication traffic. Another pioneering CVE system, VLNET (Virtual Life Network)[90], also exploits distributed communication architecture with the aura model to support more users.

However, the point-to-point connections are only appropriate for single request-reply pair. Multicast and broadcast are not supported, which makes the scalability questionable. Another possible bottleneck in LCVE is the spatial trader which checks for aura collision. MASSIVE-1 only supports up to about 10 users across Internet, tested in experimental international trials across Europe.

MASSIVE-2[91] extends the notion of adapters in MASSIVE-1 and proposes a notion of third-party objects. It also changes the communication infrastructure to multicast instead

of point-point connection. A third party object is an independent object which affects the awareness between other objects, through so called “adaptation" (increasing and decreasing) of existing awareness, and “secondary sourcing" for providing an overview or summary of the activities for a group of objects. For example a crowd or a room is a third-party object.

Each third-party object manages one or more multicast groups. Observers receive information from potentially many groups depending upon the location and the extent of their auras. Third-party objects decrease the network traffic and improve the system scalability.

However, MASSIVE-2 supports only up to 20 mutually aware users (with audio) on the workstations. This limitation was due to available CPU throughput, rather than bandwidth. MASSIVE-2 was also hard to extend it to support access over lower-bandwidth networks (e.g. dial-up modem connections)[92].

2.4.5 Butterfly Grid

Butterfly Grid [93] is the first scalable, resilient grid based platform for running massively multiplayer online games (MMOGs). It is provided by Butterfly.net Inc. Butterfly use IBM-e-business infrastructure technology and the Globus Toolkits 2.2[94, 95] to build a self-managing, fully meshed server farm to host MMOGs.

Butterfly Grid consists of a cluster of IBM eServer xSeries servers. There servers are divided into four categories based on their role within the grid: game servers, gateway servers, daemon controllers and database servers. The overall architecture of Butterfly is shown in Figure 2-8.

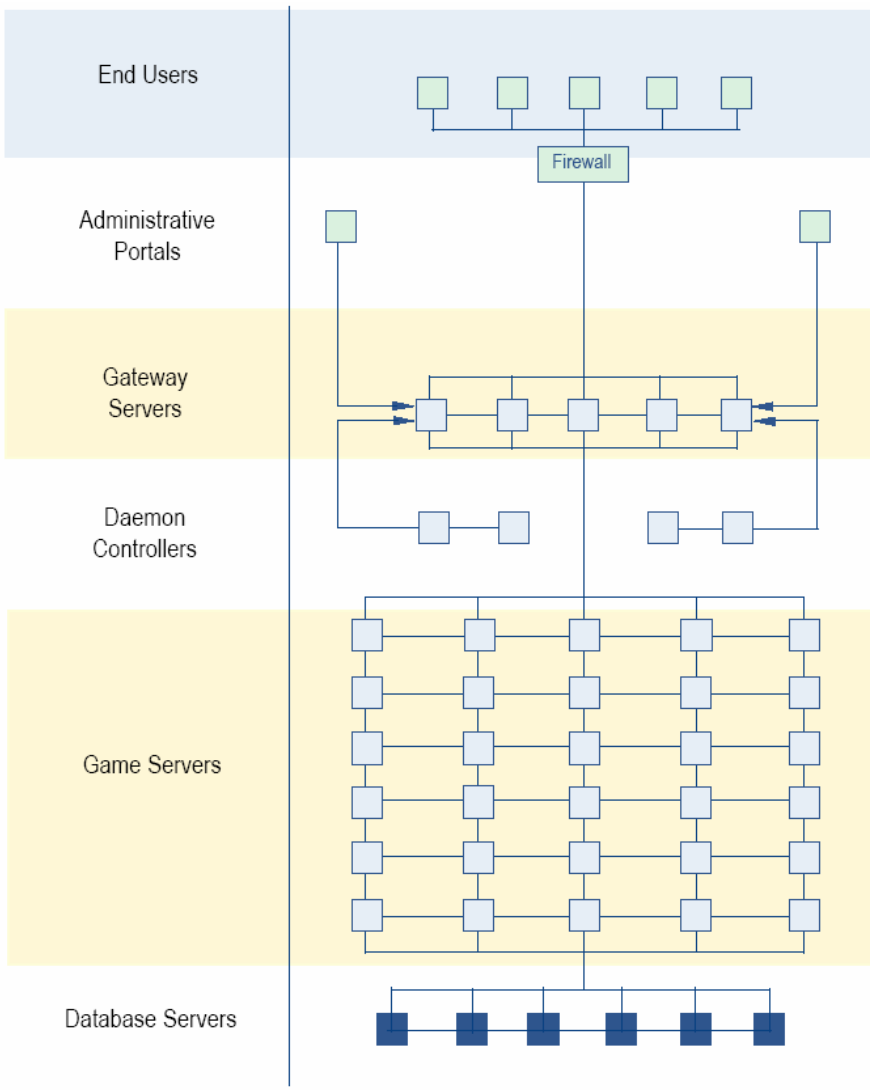

Figure 2-8 Butterfly Grid architecture [93]

Game servers are responsible for running games within the Grid. Butterfly Grid divides the world into a series of mutually exclusive sectors known as “locales,” each of which is assigned to a specific game server. When a game server becomes overloaded, that game server sends a controlling message to the gateway servers, which are ultimately responsible for redirecting players to new game servers.

Gateway servers are responsible for linking players to game servers. Gateway servers also perform protocol translations and route player connection to game servers. Daemon controllers are AI servers that control game elements not directly controlled by players'

actions. Daemon controllers interact with the Grid's gateway servers. Database servers are responsible for storing all the information (physics, geometry, game rules, etc.) needed to maintain persistence in the game world.

In the Butterfly Grid architecture, each server is connected to all others over high-speed fiber-optic lines, which is called fully meshed. In the course of a game, each server will communicate (or multicast) to all the other servers in the grid in real-time. Under this "peer-to-peer" networking approach, players are transparently routed to the optimal server in the Grid such that server resources are allocated to the most popular games. This combination of a meshed topology and intelligent routing capability is central to the Butterfly Grid's resiliency, scalability and performance.

Although Butterfly Grid scales with the number of game players, the game server has to be over-provisioned to handle peak loads[78]. As a multi-server architecture, Butterfly Grid is still a cost-expensive solution for LCVE.

2.5 The Needs for Further Research

This review reveals that the scalability is still one of the key issues in the research of CVE. Scalability issue has strong correlation with other CVE issues, such as consistency, persistency, responsiveness, extensibility etc. To achieve scalability for CVE system, there are a number of fundamental issues that need to be addressed. These include: how to build a LCVE efficiently in a heterogeneous environment; how to efficiently maintain a persistent LCVE with adaptive consistency control; how to support functional extensibility in a LCVE at runtime; and how to avoid the potential bottleneck emerging at any host in LCVE.

Thus, further research is needed to find the answers to these questions. In this research study, a new framework will be proposed to address the following main issues:

- To avoid the potential bottleneck emerging at any host of a LCVE system in a heterogeneous environment ;

- To efficiently maintain a persistent LCVE in a heterogeneous environment with adaptive consistency control;

To build our proposed framework, we review a new technology – mobile agent in the next section.

2.6 Mobile Agent

Mobile agent[96] is a new technology that has emerged with the intense research of intelligent agent and the arrival of the Internet and web technologies. As a new paradigm, mobile agent is becoming increasingly popular for network-centric programming.

2.6.1 What is Mobile Agent

Mobile agent is an autonomous software entity that can migrate with its states from one machine to another in a heterogeneous network. It does not bond with the system on which it begins execution and it is free to travel among the hosts in the network[97]. Created in one execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution. The term "state" typically means the attribute values of the agent that help it determine what to do when it resumes execution at its destination[98]. Different from the remote evaluation and code on demand paradigms, mobile agents are active in that they may choose to *migrate* between computers at any time during their execution. Different from the traditional term of process migration[99, 100] that decision of which process to move to which node at what time is made by the underlying system, mobile agent migration is totally under control of the agent program itself.

Mobility is one of the key properties of mobile agents. In order to be mobile, the mobile agents need an infrastructure in the network that handles the execution and transportation of the agents [101]. The infrastructure can also be called agent environment, agent server, agency or place, etc. In this thesis, we call it agent environment. Agent environment is responsible for executing agent code, transferring agent code with its state, helping agent communication, accessing host resources, etc[102].

With regard to the inner mechanism, there are two kinds of mobility: strong mobility and weak mobility. In strong mobility, the underlying system captures the agent's data state (i.e., the values of the internal variables) and the execution state (i.e., the stack and the program counter) and transfers it together with the code to a new location where the state of the agent is restored. Strong mobility is transparent to the programmer, but it does have a high cost for the system. On the other hand, in week mobility, only agent data state is transferred with the code to the new location. And further more, the size of the transferred state information can be limited even more by letting the programmer select the variables making up the agent state. Therefore weak mobility is light-weighted and provide more flexibility to developers[103, 104].

A number of mobile agent systems have been designed and implemented in academic institutions and commercial firms, which include Aglet from IBM[105], Agent Tcl from Dartmouth College[106], Agents for Remote Access(ARA) from the University of Kaiserslautern[107], Concordia from Horizon Systems Laboratory and Mitsubishi Company[108], Mole from the Institute for Parallel and Distributed Computer Systems (IPVP)[109], TACOMA from Cornell University[110], etc.

2.6.2 Mobile Agent's Possible Contribution and Challenge for LCVE System

Because of the mobility, mobile agent has many applications[101], such as distribute information retrieval, remote control, software deployment, distributed computing, programmable networks, etc.

Distributed computing is one of the most attractive applications of mobile agent. It provides an innovative concept to create distributed systems. Computing is no longer localized to a single central computer; instead processing takes place in the distributed environment of the network [111-113]. Mobile agent can move to the host with enough resources to execute. This helps to avoid potential bottleneck[113]. For example, in [16], dynamic set of servers are implemented using mobile agents and can deploy new replicas when the demands rises, or migrate towards the location of the majority of the clients. Therefore, the mobile agent paradigm has the potential to be used for LCVE systems to model the system tasks, i.e. exchanging and processing the data flows. As the intrinsic

heterogeneity, mobile agents can migrate to both traditional server hosts and qualified user hosts to execute. By applying the mobile agent paradigm, the system tasks of LCVE have the capabilities to be dynamic distributed to both server hosts and qualified user hosts so that the overall system workloads are shared by more hosts, which will improve the system scalability. Moreover, mobile agent paradigm also provides a simple way to deploy system tasks, which provides flexibility and extensibility for LCVE software deployment.

In order to apply mobile agent paradigm to build LCVE systems, the detailed agent migration process of the existing agent platforms needs to be investigated. Basically during the agent migration, the agent firstly stops its working and serialize it code and state information. Next, the serialized data is sent from the origin agent environment to destination environment. Finally, the destination agent environment de-serializes the received data and recreates the agent[114]. During agent migration, the service provided by the agent will be suspended until the agent is recreated. The overhead of agent migration is represented by the time it takes an agent to move its code and state from the origin agent environment to the destination agent environment. In [115], the agent migration overhead is measured. The minimal agent migration time for an aglet with size of 1475 bytes on the Internet is 148 ms. With the increases of the agent size, the migration time grows. When agent's size reaches 500 kbytes, the migration time is near 6 seconds. In [116], similar results for migration performance are presented for agents implemented in Concordia and Voyager platforms.

As we plan to model the tasks of data flow exchanging and processing as mobile agents, we require the agent migration overhead not to impact the real-time performance of the LCVE. From the above reviews, we find the existing mobile agent platform can not be used to construct LCVE systems, because the overhead of agent migration is not neglectable, which will frustrate the real-time interactions in LCVE. Therefore, we will implement a new agent migration model to reduce the impact of agent migration on the LCVE performance. Detailed mechanism for the new agent migration model and the comparisons of new agent migration model with the existing ones are presented in section 4.2.

2.7 Summary

In this chapter, we review the general concept and terminology, the communication architecture, the existing approaches for CVE scalability, and several well-known CVE systems. We analyze the advantages and disadvantages of the existing communication architecture and the existing approaches for CVE scalability. Our review reveals that there is a need for an adaptive architecture to build LCVE systems in heterogeneous environments with adaptive consistency control. We review the mobile agent technology and believe the mobile agent paradigm can be used to construct a scalable architecture for CVE. In the following chapter, we will present our proposed mobile agent based framework for LCVE to achieve this goal.

Chapter 3

MACVE

In this chapter, we present a Mobile Agent based framework for large-scale CVE systems (MACVE) to avoid potential bottleneck by pervasive distribution of the data communication and to adaptively control the consistency of a LCVE. MACVE is divided into three layers and each layer is composed of multiple collaborative mobile agents to provide different kind of services. The mobile agents are deployed to both traditional servers and qualified user hosts. The agent mobility algorithm enables the agents to make decision to migrate or clone to reduce the workload of its hosting node before it becomes a potential bottleneck. The adaptive consistency control dynamically decides consistency model and message communication architecture for different parts of a LCVE to ensure the required consistency and improve system scalability. Agent failure recovery mechanism has been incorporated in MACVE to improve the system stability and robustness. Finally, the scalability of MACVE has been discussed.

3.1 Key Ideas behind MACVE

Our review in Chapter 2 reveals that both peer-to-peer architecture and multi-server architecture has limitations to efficiently control the data flows in LCVE. In order to pervasively distribute the data flows to avoid the potential bottleneck and to adaptively control the consistency of a LCVE as well, we proposed a Mobile Agent based framework for large-scale CVE systems (MACVE).

Before introducing our proposed architecture, we classify the participating hosts in a LCVE system, as illustrated in Figure 3-1. A typical LCVE system consists of a large number of computer hosts connected through networks. Each host is called a

participating node. Based on the nature of each participating node, we classify them into two main categories: User Nodes and Service Provider Nodes. User Nodes refer to any user host participating in the LCVE. Service Provider Nodes refer to the nodes belonging to the LCVE system owner.

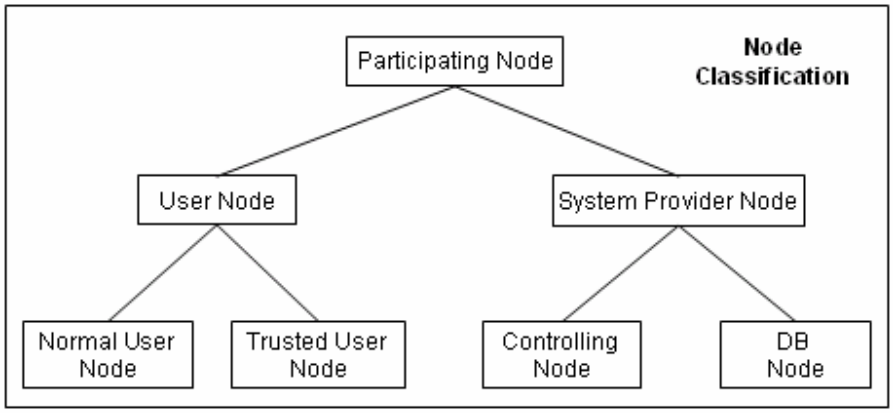

Figure 3-1 Node classification

User Nodes are further classified into *Normal User Nodes* and *Trusted User Nodes* based on their functional roles.

- Normal User Node is a host that only allows a user to navigate through the LCVE and interact with virtual entities or other users in the LCVE.

- Trusted User Node is a host that not only functions as a Normal User Node, but also has spare capacity in terms of computing power, memory and network bandwidth to host mobile agents for the whole system. It should at least meet the minimum capability and security requirements set by the LCVE system.

Service Provider Nodes are further classified into *Controlling Nodes* and *DB Nodes* based on their functional roles.

- Controlling Node is a host provided by the system owner, which perform system tasks to manage a LCVE system and maintain the multi-user collaborative interactions in a consistent, persistent, and evolving LCVE.

- DB Node is a host owned by the system owner, which provides the data storage support for the LCVE system.

Since Controlling Nodes and Trusted User Nodes are used to share the system workloads, we define both of them as System Computing Nodes (SC Nodes). And system computing resources are defined as the computing and network capability of all the joining SC Nodes in the system.

After the classification of participating hosts, we begin to introduce the key ideas behind our framework. To effectively decentralize the system workloads for data flows control, the system tasks of a LCVE should be independent and fine-grained. MACVE models a LCVE software system as a group of collaborative agents, as illustrated in Figure 3-2. Each agent is a lightweight software component which performs an independent task to provide a certain service for controlling the system data flows, such as CVE consistency control service, VE scene data delivery service, etc. Agents collaborate with each other to maintain a LCVE.

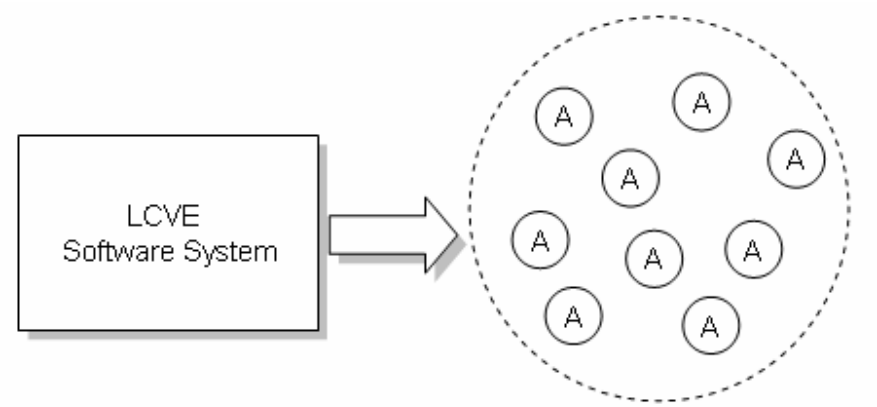

Figure 3-2 Decompose LCVE software system into agents

To further enhance CVE system scalability, MACVE allows all agents to be mobile without bonding with any fixed host. As the system scales up, agents can automatically migrate or clone to any qualified participating host which includes Controlling Nodes and Trusted User Nodes to dynamically distribute the workload and ensure the high quality of CVE performance. The mutual independence of services and hosts provide large flexibility to utilize the computational and network resources of the system with high

efficiency. The ability for Trusted User Nodes to take over mobile agents enables the system to utilize the extra computing resources of user hosts. As an example shown in Figure 3-3, Agent A1 and A2 ran at a busy Controlling Node. To avoid potential bottleneck emerging at the busy Controlling Node, A1 automatically migrates to another less loaded Controlling Node and A2 automatically clones new instances (A2.1 and A2.2) respectively at two Trusted User Nodes. Consequently, the overall workloads of the whole system are shared by more nodes and the system scalability will be improved.

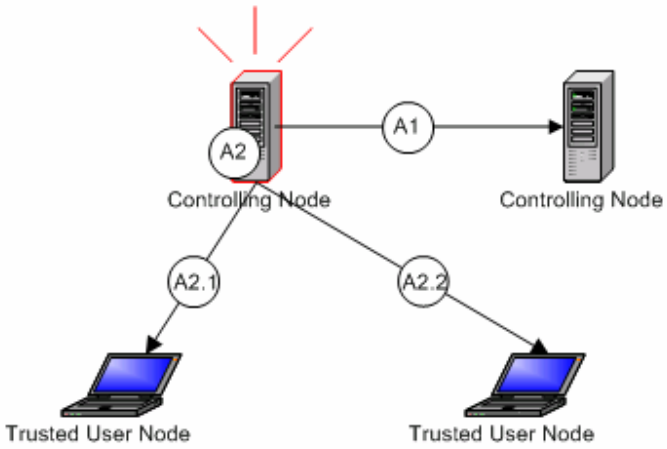

Figure 3-3 Agent mobility in MACVE

As mentioned above, the mobility of agents in MACVE has two forms: agent migration and agent remote cloning. We call them agent mobile actions.

- Agent migration is a mobile action to transfer an agent with its run-time state from one host to another, without degrading the real-time performance of the CVE application. Agents migrate from overloaded nodes to less loaded ones so that they will have sufficient computing and network bandwidth resources to execute their LCVE tasks. Agent can also migrate to the node proximate to user nodes to provide better services. In MACVE, Agent migration provides a method to distribute the workload of system tasks to multiple hosts.

- Agent remote cloning is a mobile action to create a new instance of an agent with its run-time state at another host. After the remote cloning, the cloned agent provides the same service corporately with the original one. Agent remote cloning

provides the mirroring service at a remote host, which can enhance the throughput of the service. In MACVE, Agent remote cloning provides a method to distribute the workload of a same task to multiple hosts.

To adaptively control the consistency for a LCVE, MACVE provides the mechanism to dynamically decide the consistency models and message communication architecture for different parts of a LCVE (e.g. region or cell).

Besides the dynamic workload distribution, the mobility of agent can also be used for system administration, new services deployment, etc. We only focus on agent mobility for CVE system scalability in this thesis.

3.2 System Architecture

MACVE is divided into three layers: Resource Layer for System Resource Management, Content Layer for VE Content Management and Gateway Layer for VE Directory Management, which is illustrated in Figure 3-4. Each layer is composed of multiple collaborative mobile agents to achieve the management. Resource layer manages the distribution of agents, SC Nodes and the storage space. Content layer maintains the real-time VE activities. Gateway layer link the users to the agents at content layer. The System Resource Management is subdivided into Agent Resource Management, Computing Resource Management, and Database Resource Management, which will be discussed in Section 3.3.

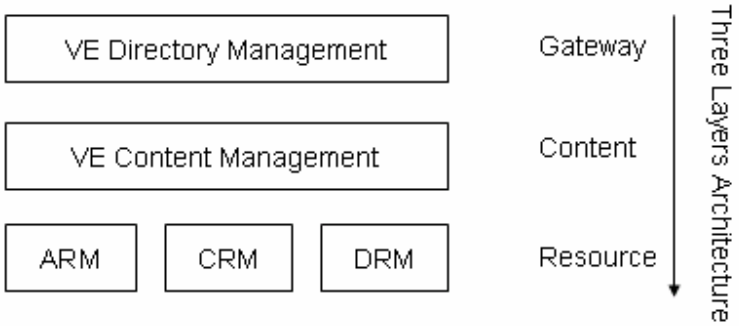

Figure 3-4 System architecture

3.3 System Resource Management

The bottom layer of MACVE is resource layer which manages the distribution of system resources. In MACVE, we define mobile agents, SC Nodes and the data storage space as system agent resource, system computing resource and system database resource respectively. Accordingly, resource layer is further subdivided into three parts: Agent Resource Management (ARM), Computing Resource Management (CRM), and Database Resource Management (DRM). ARM, CRM and DRM make resource layer independent of different CVE application and scenario. It provides resource management services for the high layers and hides the complexity of the resource distribution.

3.3.1 Agent Resource Management

In MACVE, there are eleven basic types of agents running in the system. Each type of agent corresponds to one LCVE service, as show in Table 3-1. These services do not bond with any fixed host and can be dynamically distributed by agent migration or remote cloning. We define the agents as a kind of resource. ARM manages all these agents effectively. ARM is realized by ARM Agent.

Table 3-1 Basic types of agent in MACVE

Agent	Location in MACVE	Service
ARM Agent	Resource layer - ARM	Manage agent resource
Node Agent	Resource layer – CRM	Monitor performance of the node and the agents running on the node
Group Manager Agent	Resource layer – CRM	Manage the nodes and agent actions in its group
CRM Agent	Resource layer – CRM	Manage the node groups
DRM Agent	Resource layer – DRM	Manage all DB resources
DB Agent	Resource layer – DRM	Manage the DB Node
Region Agent	Content layer	Manage its cells
Cell Agent	Content layer	Manage cell scene data delivery

Consistency Agent	Content layer	Manage cell consistency control
Persistency Agent	Content layer	Manage cell persistent VE state
Gateway Agent	Gateway layer	Manage user account and direct user to CVE

3.3.1.1 ARM Agent

ARM Agent provides the service to manage the agent resource. It is composed of three main components: an Agent Code Repository, an Agent Monitor Center, and a System Administration Interface.

- Agent Code Repository

In MACVE, different types of agents have different agent codes. The same type of agents may have different implementations which correspond to different versions of agent code. Before distribution, the code for each type of agents is registered with ARM Agent. ARM Agent stores and manages these codes and distributes the agent code to SC Node on demand. It guarantees the integrity and consistency of the deployed agent codes in the whole system. The network location of an ARM Agent is also called an agent code base. As ARM Agent can have multiple clones, there may be multiple agent code bases in a MACVE system.

- Agent Monitor Center

Once MACVE system is launched, the agents automatically migrate or clone among SC Nodes. The ARM Agent functions as an agent monitor center to record the agent distribution and agent mobile activities. It provides an agent location directory for all the running agents. When a new agent begins to run, it registers its network locations and running states with ARM Agent. When agents perform some mobile actions, e.g. migration or remote cloning, ARM Agent records the logs of these agent activities.

- System Administration Interface

To allow manual intervention to adjust the agents' distribution and agent execution at the system level at run-time, ARM Agent functions as a System Administration Interface. It can send commands at run-time to launch agents, to request agents to migrate or remote clone, to update agents to new versions, and to kill running agents, etc.

The ARM Agent supports migration and remote cloning. For the security reasons, it can only migrate or clone at Controlling Nodes. Since the network traffics at ARM Agent are only agent running state, agent commands and agent code data, which are small in size and do not belong to frequent traffic compared with the VE scene related data, there is little chance for an ARM Agent to become a bottleneck. Even though the remote chance happens, ARM Agent can migrate to the node with more resources or clone itself at other nodes to redistribute its workloads before it becomes a potential bottleneck.

3.3.2 Computing Resource Management

CRM is designed for effectively managing all the participating hosts to share the system workloads on demand. In MACVE, the system tasks are shared not only by the Controlling Nodes as most conventional LCVE systems do, but also by Trusted User Nodes. According to the resources that the Trusted User Node is willing and available to contribute to the system, it may receive agents from other SC Nodes to share the system workloads.

As Trusted User Nodes log-in and log-off the system at will, CRM provides the mechanism to avoid agent service lost in case that Trusted User Nodes log off or crash accidentally. When a Trusted User Node logs in, it will register with the system and be ready to receive agents if it has spare resources. When logging off, it will transfer all the agents running on it to other suitable SC Nodes. The detailed algorithm for the agent migration will be discussed in Section 3.7. Compared with Controlling Node, Trusted User Node has relatively less stability. A Trusted User Node may crash before it successfully transfers all the agents running on it. Therefore, MACVE has an agent failure recovery mechanism to restart the lost agent from the state just before it crashes. Agent failure mechanism will be discussed in Section 3.9.

As there may be thousands of SC Nodes in a LCVE system, in order to improve the efficiency of data communication between the SC Nodes during agent migration or agent remote cloning, the nodes are grouped according to their network proximity, as shown in Figure 3-5. Each group has a group manager to gather nodes resource information for reasonable workloads distribution in the group.

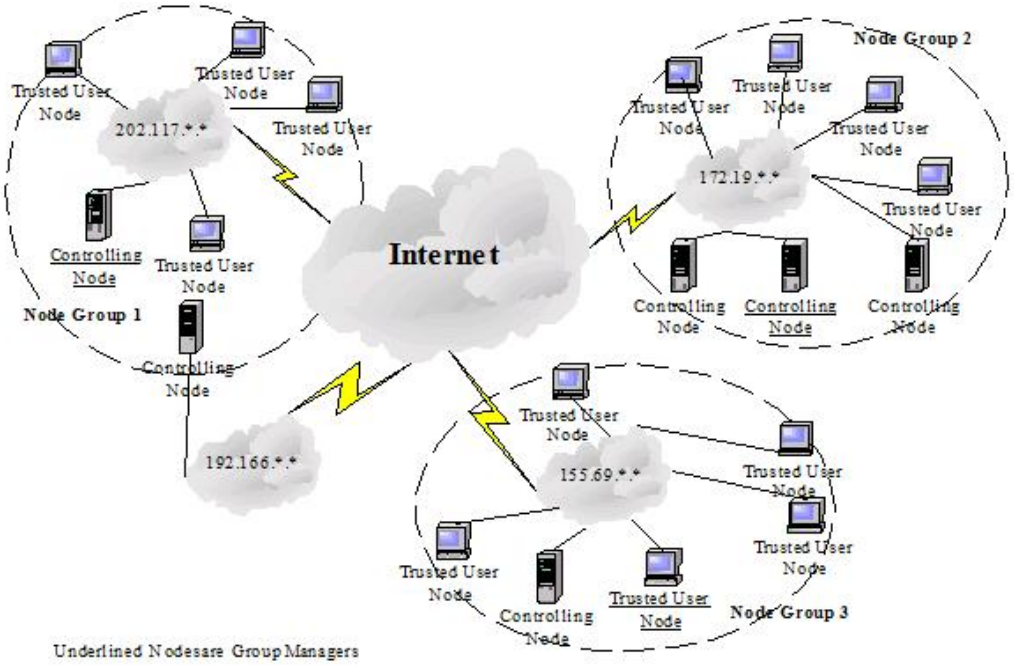

Figure 3-5 Grouping the SC Nodes

CRM is achieved through Node Agents, Group Manager Agents and a CRM Agent in a hierarchy structure, as illustrated as Figure 3-6.

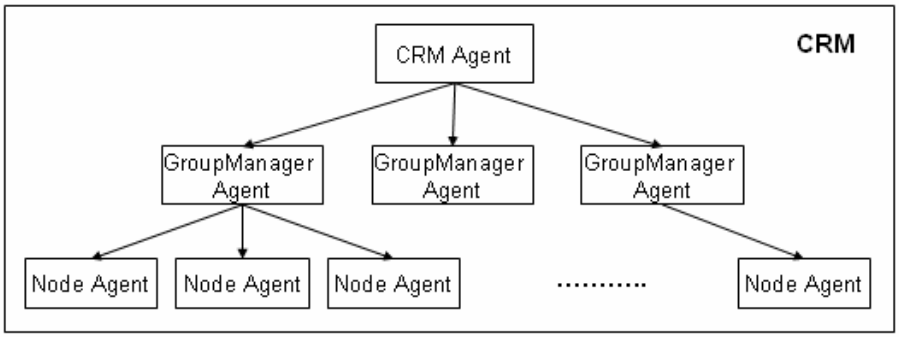

Figure 3-6 Relationship of agents for CRM

3.3.2.1 Node Agent

Node Agent runs stationary on each SC Node. It provides the workload monitoring and ideal node discovery services for agent mobility decision. Node Agent monitors the workloads of its hosting node and the other agents running on that node. It collaborates with Group Manager Agent and CRM Agent to discover ideal SC Node for agent migration or agent remote cloning.

The workload in each SC Node can be measured simply by one metric, such as CPU utilization, network throughput, etc., or comprehensively by a combination of multiple metrics. We call these metrics load index. The load index of the combined workloads metrics of a node can be represented by a vector.

$$W = (c, m, t_i, t_o)$$

where

W.c -- the average percentage of CPU utilization of the node
W.m -- the average percentage of memory in use
$W.t_i$ -- the average incoming network traffic rate
$W.t_o$ -- the average outgoing network traffic rate

Because of the heterogeneity of SC Nodes, a CPU weighting factor for each SC Node is used to make each CPU power comparable, which eliminates the variations of CPU speeds, operating systems, and hardware organizations of the SC Nodes.

For each SC Node, there are two thresholds for its workload: export threshold T_e and import threshold T_i. Thresholds are expressed in units of the load index. For load index of the combined workload metrics, the threshold is represented as:

$$T = (c, m, t_i, t_o)$$

Where

T.c -- the threshold for CPU utilization rate
T.m -- the threshold for memory usage rate

$T.t_i$ -- the threshold for incoming network traffic rate
$T.t_o$ -- the threshold for outgoing network traffic rate

In order to make the above workload W and threshold T comparable, we define the semantic for their comparison.

$$W.c > T.c \text{ OR } W.m > T.m \text{ OR } W.t_i > T.t_i \text{ OR } W.t_o > T.t_o <=> W > T \quad (3\text{-}1)$$

As described in Equation 3-1, if any element of W in a SC Node is greater than the corresponding element of threshold T, we get W > T.

$$W.c < T.c \text{ AND } W.m < T.m \text{ AND } W.t_i < T.t_i \text{ AND } W.t_o < W.t_o <=> W < T \quad (3\text{-}2)$$

As described in Equation 3-2, if all the element of W in a SC Node is less than the corresponding element of threshold T, we get W < T.

When $W > T_e$ (i.e. workload of the node exceeds the its export threshold), Node Agent negotiate with the other agents on that node to decide whether there is a need to share its workloads by transferring or cloning some agents to other nodes to alleviate this node's workloads. On the other hand, when $W < T_i$ (i.e. workload of the node drops below its import threshold), the SC Node can be selected as a suitable node to host migrating agents or to create cloned agents. Each SC Node sets its own T_e and T_i based on its resource capability. The detailed agent mobility algorithm is discussed in Section 3.7.

3.3.2.2 Group Manager Agent

Group Manager Agent runs at a SC Node which is designated as a group manager by the CRM Agent. It collaborates with Node Agents in its group to search for the suitable nodes for its member nodes.

Group Manager Agent plays a mediator role. Node Agents in its group advertise their nodes' capabilities (CPU speeds, operating system information, physical RAM, network interface card's speed etc.) and periodically (every 1 minute in MACVE) report its real-time workload information to the Group Manager Agent. When an agent on a node needs to search for another node for migration/remote cloning, its Node Agent will send a message with its resource requirement to the Group Manager Agent. The Group Manager

Agent will provide the locations of candidates of suitable nodes to the requesting Node Agent. If the Group Manager Agent cannot find a suitable node within its own group, the Group Manager Agent will negotiate with CRM Agent to search for suitable candidate nodes in other groups. The detailed explanation of agent collaboration is discussed in section 3.7

3.3.2.3 CRM Agent

CRM Agent provides the services for SC Node registration, authentication, dynamic node group creation, and the snapshot of the overall available computing resources under its charge.

If a user node is willing to be a Trusted User Node, it needs to register with CRM Agent. CRM Agent evaluates the resource capacities of the node. If the node is satisfied with the minimal Trusted User Node requirement, CRM Agent will allocate ID and grant a security key to that node. When a SC Node logs on the system, CRM Agent dynamically allocate it to a node group according to its IP address. In each node group, there is a SC Node designated by CRM Agent as a group manager to manager its member nodes. According to the scale of existing node groups, CRM Agent can dynamically create new node group with a new Group Manager Agent to avoid degrading group management performance. The dynamic creation of Group Manager Agent guarantees the dimension of system computing resource to scale up autonomously.

CRM Agent also provides a map of all computing resources of the system, which helps the system administration. The information includes the registered SC Nodes parameter, the distribution of running node groups and running SC Nodes, etc.

3.3.3 Database Resource Management

To support CVE persistency, storage and database systems are needed to save and update VE scene data, VE state data, user account information, and system administration information, etc. For a LCVE system, the storage and database systems needs to be distributed to multiple DB Nodes for easier expansion, monitoring, backup and recovery of data resources. As the system evolves, new DB Nodes can be added into the MACVE

system at any time. DRM is achieved by agents in a two-level's hierarchy structure. The agents include a group of DB Agents and a DRM Agent. The relationship of these agents is illustrated in Figure 3-7. DB Agent runs at each DB Node. DRM Agent runs at one of the Controlling Nodes. All these agents work together to provide a uniform interface for agents at content layer and gateway layer to access the system database resource.

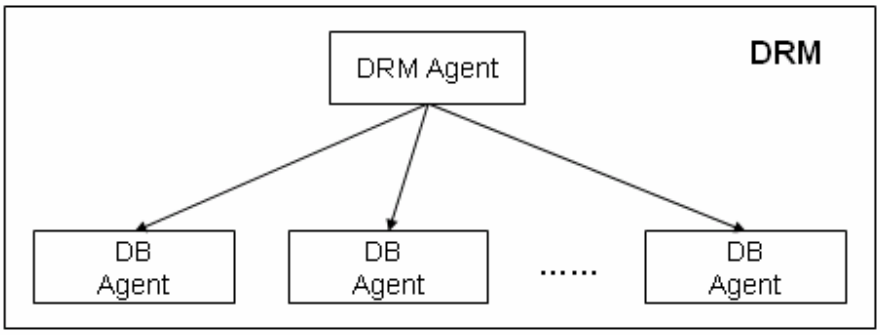

Figure 3-7 Relationship of agents for DRM

3.3.3.1 DRM Agent

DRM Agent manages the distribution of all DB Agents. It provides a directory service for the persistent VE data and the system administration data. Agents in content layer and gateway layer look up the network location of the DB Agent that they need to connect to from the DRM Agent. DRM Agent also monitors the performance of DB Agents. It provides the information of all DB nodes' storage capacity, fetching capacity, and I/O workload, etc.

3.3.3.2 DB Agent

In MACVE, agents in content layer and gateway layer need the persistency service to access the storage and database system. To distinguish them with DB Agent, we call these agents Task Agents (TA). DB Agents sit between task agents and the storage and database system. It hides the storage and database system from the task agents by providing a uniform interface for these task agents to access the system database resources. As shown in Figure 3-8, each task agent only needs to connect to one DB Agent at a time. TA sends its database request/update messages to the DB Agent. The

task agents do not need to know where the data in DB messages will be stored. The connected DB Agent will forward the DB messages if the messages need to be processed by another DB Agent. After processing the message, DB Agent maps the TA's request to the DB operation commands and applies to the database system. DB Agents minimize TA's dependency on storage and database systems so that they are less coupled to each other. Therefore Database location and the database realization can be changed on the fly. It also provides flexibility for the database maintenance, such as system upgrade.

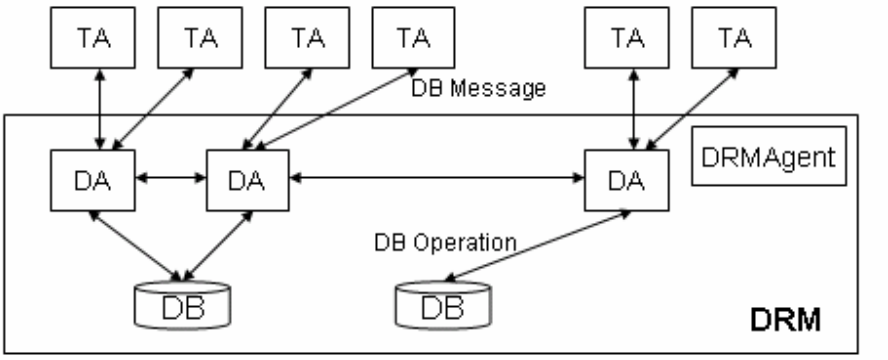

Figure 3-8 Relationship of task agent and DB Agent

3.4 VE Content Management

Content layer is on top of the resource layer which provides the VE content services to participants. It is application and scenario dependent. In MACVE, the VE is a hierarchical database and spatially divided into several manageable continuous regions. The criterion for partitioning VE into regions is based on VE scenario variety. Regions may have different themes or belong to different organizations. Thus each region may have its own management policy. Region supports dynamic partitioning and is further divided into cells. The criterion for partitioning region into cells is based on the VE geometry features and VE workload distribution.

A cell includes human-controlled or computer simulated actor called avatar and static, interactive or computer simulated object called virtual entity. Each cell is a basic unit for VE scene data delivery, user group communication, maintenance of VE consistency and

persistency. Each cell is associated with a separate set of multicast channels for different interest classes to ensure the efficiency of the communication.

The management of the VE corresponds to the Region-Cell hierarchy, which is achieved by Region Agents, Cell Agents, Consistency Agents and Persistency Agents, as illustrated in Figure 3-9. The relationship of these agents structures as a tree, which makes agent failure recovery possible (discussed in Section 3.9).

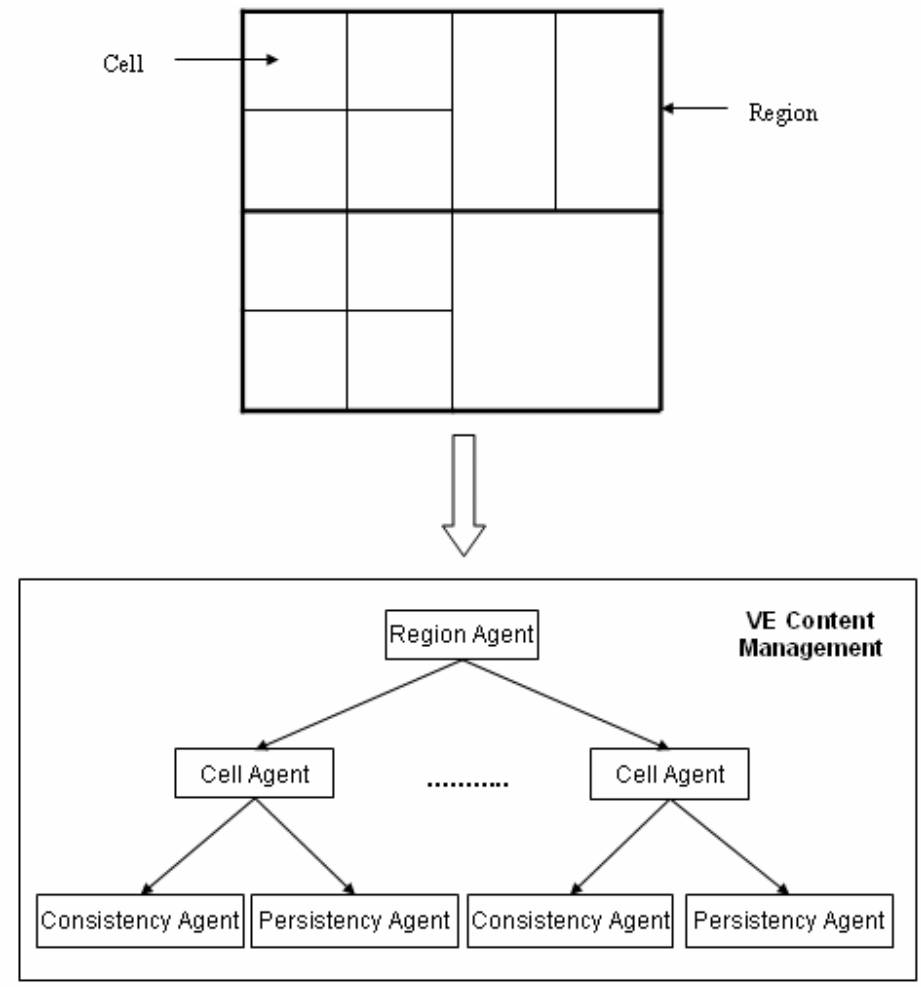

Figure 3-9 VE management

In VE Content Management layer, Region Agents, Cell Agents, Consistency Agents and Persistency Agents are the basic components in the framework. Agents for other content services can also be added. The extended agents may include: Message Aggregation

Agent for merging multiple object state messages into one message to improve the object state transmission efficiency; Robot Agent for simulating autonomous computer generated avatar or entity, etc. As MACVE is an open architecture, any new type of agent can be integrated to the system in the future.

3.4.1 Region Agent

Region Agents provide the service to manage its Cell Agents and make the VE Content Management hierarchical. Region Agent itself does not maintain VE scene data, VE state data and object state data. The VE content is managed by Cell Agents, Consistency Agents and Persistency Agent. Each cell has at least a Cell Agent, a Consistency Agent and a Persistency Agent to provide the basic CVE content services. The three agents do not need to run at the same SC Node.

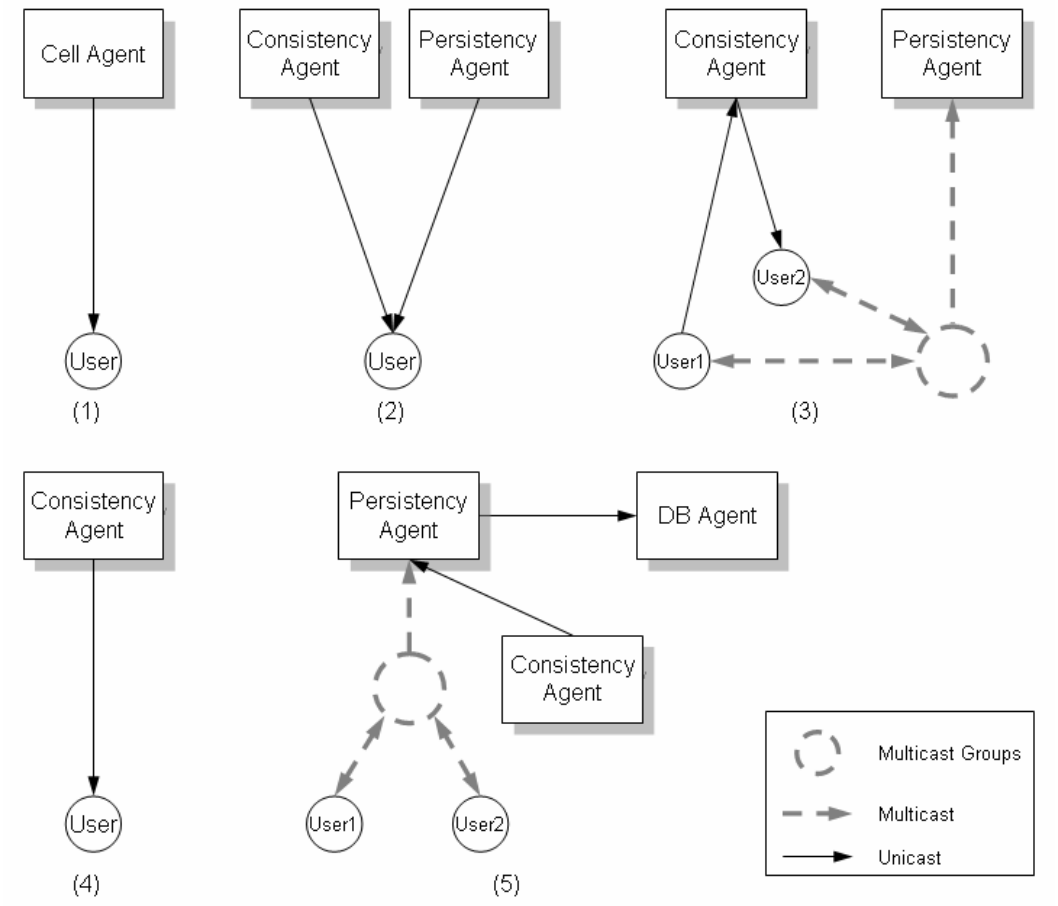

Figure 3-10 Communication architecture for VE content data

As shown in Figure 3-10, Communication architectures for VE content related data are depicted based on the following five main streams of data flows:

(1) VE scene data flow

Users get their most needed VE scene data from Cell Agent

(2) VE shared state data flow

Users get the part of cell state with high consistency requirement from Consistency Agent and get the rest of cell state with low consistency requirement from Persistency Agent;

(3) Object state data flow

Users disseminate their interactive messages by unicast or multicast according to different consistency control requirement for the data;

(4) VE consistency controlling data flow

Consistency Agent sends control message to relevant users to ensure adaptive consistency control.

(5) VE persistency data flow

Persistency Agent receives the object state data from consistency agent with strict consistency processing or directly from users by multicast groups. It preserves the runtime VE state and periodically sends the changes to the corresponding DB Agent to ensure a persistent CVE.

The detail services provided by Cell Agent, Consistency Agent and Persistency Agent are discussed in the following sections.

3.4.2 Cell Agent

Cell Agent provides the VE scene data delivery service for cells. It only sends the most needed scene data to the relevant users. The delivery process is required to be timely to ensure the real-time performance of CVE.

As VE scene data includes 3D geometry files, 2D image files, audio files, video files and behavior script files, etc, which is large in size, it often takes long time for the users to download them. The delay of VE scene data transmission may destroy CVE immersion and make the CVE interaction useless. The data delivery also results in significant bandwidth consumption. The traffic at the server is often in a bursting manner. To sum up, there are two issues for the VE scene data delivery:

(1) How to ensure the scene data to be delivered timely to the user sides when it is needed;

(2) How to ensure the scene data delivery not to become a bottleneck or a single failure point.

To solve the two problems, Cell Agent consists of two major components: Scene Data Cache Manager and Scene Data Delivery Manager, as shown in Figure 3-11.

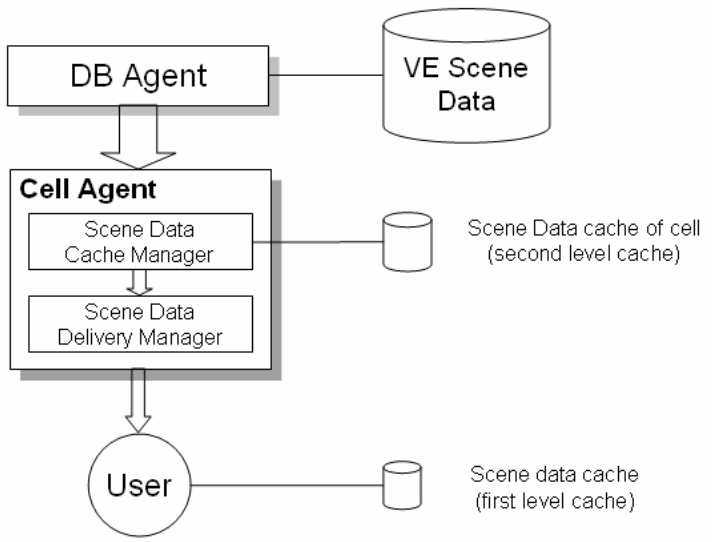

Figure 3-11 Cell Agent architecture

- Scene Data Cache Manager

Cache Manager maintains a copy of the scene data for a cell. As persistent data, all of the VE scene data is initially saved at DB Nodes. When a Cell Agent is launched, cache manager loads the scene data of a cell from the corresponding DB Agent and saves them locally. We call this copy of scene data second level cache ($Cache_2$) to differentiate the scene data cache at user side. As Cell Agent can clone itself at other suitable SC Nodes to provide the mirroring scene data delivery service, there can be multiple $Cache_2$ distributed over networks, which provide the advantages to avoid the potential bottleneck and the single point of failure immerging at a single node.

- Scene Data Delivery Manager

Delivery Manager delivers the most needed scene data to each relevant user timely. There exist some algorithms for efficient scene data delivery, which include demand-driven transmission[117], predictive pre-fetching or multi-resolution caching[118], etc..

The ideas behind these algorithms include using AOI (Area Of Interest) and LOD (Level of Detail) models to only deliver the most needed data to users; to predict user future behaviors to deliver the possible needed data in advance; caching scene data at user local storage to avoid future reloading from the network. Delivery Manager is designed to allow incorporating of the above algorithms. We call the cache data at user side the first level cache ($Cache_1$).

Cell Agent supports migration and remote cloning. It can migrate/clone at any suitable SC Nodes, especially Trusted User Nodes which have enough bandwidth and already cached scene data of that cell. The mobility of Cell Agent brings the following benefits:

(1) By migrating, Cell Agent can work at the SC Node with sufficient resources which can provide better service and avoid the bottleneck emerging at the original SC Node.

(2) By remote cloning, multiple Cell Agents work together to provide the scene data delivery service for a cell, which enhances the delivery service throughput and avoid the potential bottleneck emerging at a single point.

(3) By migrating/cloning at Trusted User Nodes which already cached scene data of the cell, Cell Agent can directly use $Cache_1$ as $Cache_2$, which improve the data cache efficiency.

(4) By migrating/cloning at Trusted User Nodes, the scene data service distributed more pervasively, which improve the delivery service from the network topology perspective.

(5) By migrating/cloning at Trusted User Nodes, newly arriving user can download the scene data at nearby Trusted User Nodes instead of Controlling Nodes, which conserves the bandwidth at Controlling Nodes for other services and saves the system owner's cost for hosting the LCVE infrastructure. Moreover, even if the Controlling Node which hosts the Cell Agent fails or exits for maintenance, the scene delivery service can still be available.

The detailed mechanism for agent migration and agent remote cloning is discussed in Section 3.7.

3.4.3 Consistency Agent

Consistency Agent provides the service to maintain the required level of consistency in the cell. Its main tasks include (1) preserving part of the runtime cell state with high consistency requirement, (2) delivering the part of cell state to newly arriving participants, (3) maintaining different level of consistency according to the semantic of the object state data, and (4) routing the object state data flows. These tasks are taken care by the three components of the Consistency Agent: Consistency Controller, Cell State Manager, and Intelligent Message Router, as illustrated in Figure 3-12.

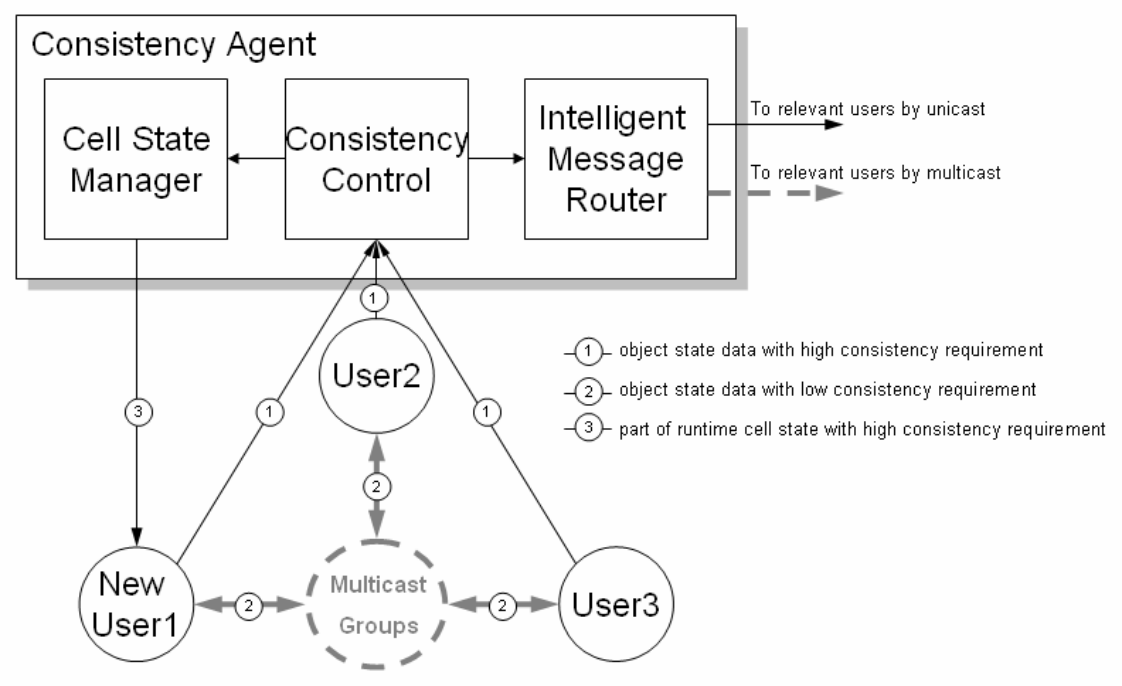

Figure 3-12 Consistency Agent architecture

- Consistency Controller

Consistency Controller is the component to process the object state data and to maintain different levels of consistency according to the semantic of the data. The object state data includes the events happened in MACVE. Due to the ever changing network traffic condition and the possibility of packet loss in wide area networks, users may receive the same event at different time point or receive multiple related events from different senders out of order. This causes the inconsistent VE state among different users and more severely confuses or frustrates the interaction results. Consistency Controller is designed to allow incorporating of different consistency algorithms, such as causal ordering[29, 119], concurrency control[31], etc., to tradeoff consistent VE state and responsiveness of the interaction in CVE. To effectively support the dynamic and various consistency requirements in different parts (e.g. region or cell or different activities in a cell) of a LCVE, consistency controller provide an adaptive consistency control mechanism to dynamically decides the consistency model and network architecture for the object state data with different

consistency requirements. The detailed mechanism for adaptive consistency control is introduced in Section 3.8

After the consistency processing, the object state data is sent to the Cell State Manager to maintain the updated object state and to the Intelligent Message Router for further propagation.

- Cell State Manager

Cell State Manager is responsible for maintaining runtime VE state of the cell with strict consistency requirement. When a new user navigates through the cell, Cell State Manager delivers the part of cell state with high consistency requirements to newly arriving users. Cell State Manager also handles the check-in/check-out service. When a user crosses the boundary of two cells, the Consistency Agent that the user enters creates a new instance of this user and the Consistency Agent that the user leaves removes the record of this user. Consistency Agent and User Communication Manager (introduced in Subsection 3.6.1) of the user application work together to hide the complex of this Consistency Agent shifting from the VE Synthesizer (introduced in Subsection 3.6.2) of user application so that the crossing boundary activity is transparent to user interaction experience.

- Intelligent Message Router

Intelligent Message Router controls the user interaction message propagation policy. As the number of concurrent users and entities in a CVE grows, its complexity increases exponentially. Intelligent Message Router functions as an object state data flow filter to control the data flow's propagation. The data flow filtering algorithms can be incorporated into intelligent message router to best express the scope of user's awareness and the range of interaction. By Intelligent Message Router, object state data is routed to different multicast groups or send directly to users by unicast.

Consistency Agent supports migration and remote cloning. Being responsible for processing the object state data from users, Consistency Agents are computational intensive. In order to make use of idle processor cycles of other SC Nodes, it can

migrate/clone at any suitable SC Nodes, especially Trusted User Nodes which have enough bandwidth and computing capabilities. The mobility of Consistency Agent brings the following benefits:

(1) By migrating, Consistency Agent can work at the SC Node with sufficient resources which can provide better service and avoid the potential bottleneck emerging at the original SC Node.

(2) By remote cloning, multiple Consistency Agents work together to provide consistency services. This can support more concurrent users in a cell to perform real time interaction and avoid the potential bottleneck emerging at single point.

(3) By migrating/cloning at Trusted User Nodes, Consistency Agent can run at ideal Trusted User Nodes instead of Controlling Nodes, which conserves bandwidth and computing capacities at Controlling Nodes and saves the system owner's cost for hosting the LCVE infrastructure. Moreover, even if the Controlling Node which hosts the Consistency Agent fails or shutdowns for maintenance, the consistency controlling service can still be available.

The detailed mechanism for agent migration and agent remote cloning is discussed in Section 3.7.

3.4.4 Persistency Agent

Persistency Agent provides the service to record the VE state changes happened in the cell and to deliver part of the cell state with low consistency requirement to the newly arriving participants. As shown in Figure 3-14, Persistency Agent consists of two major components: Cell State Manager, Cell State Recorder.

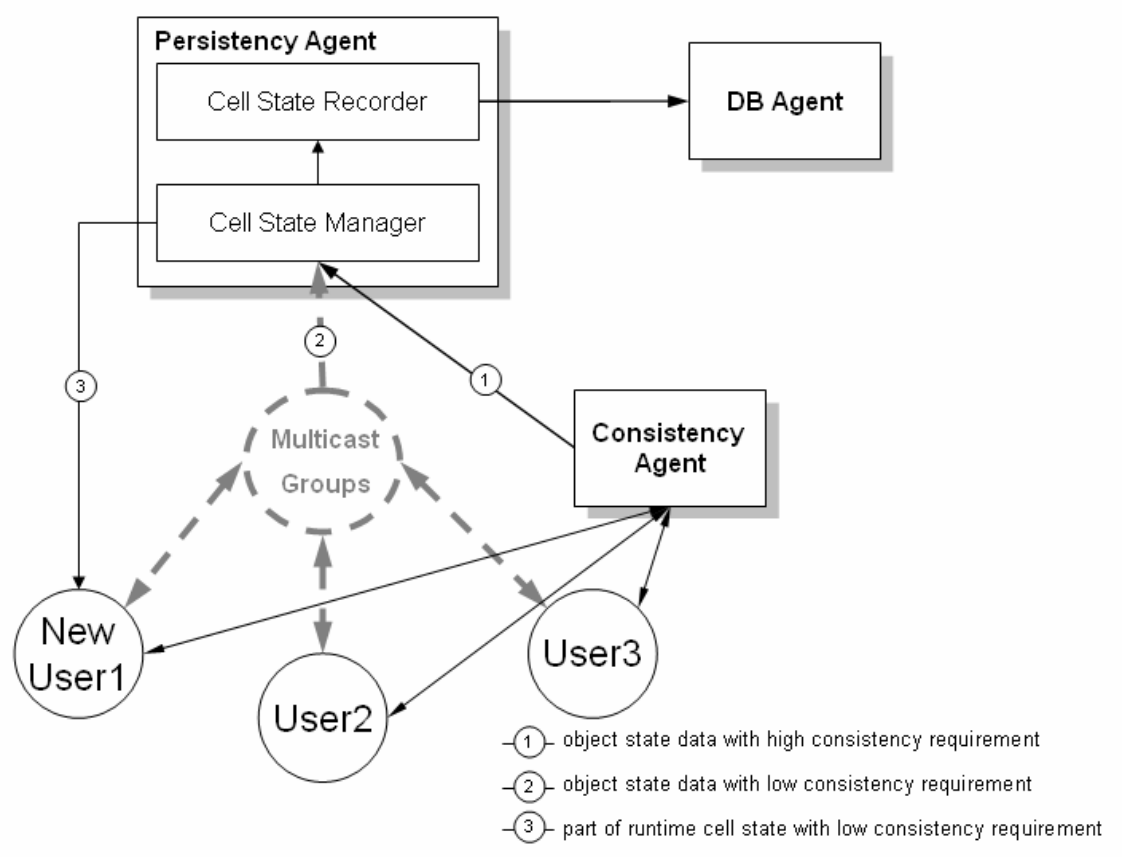

Figure 3-13 Persistency Agent architecture

- Cell State Manager

Cell State Manager preserves the real-time VE state of the cell. It receives the object state data with high consistency requirement from the Consistency Agent and object state data with low consistency requirement directly from users by multicast. The newly arriving users get the part of cell states with low consistency requirement from the Cell State Manager. In addition, the Persistency Agent and Consistency Agent usually run at different SC Nodes, so the Cell State Manager of Persistency Agent is also a real-time backup of the cell state for the Consistency Agent. If a Consistency Agent crashes, a new Consistency Agent can get the latest cell state from its Persistency Agent.

- Cell State Recorder

Cell State Recorder sends the changes that happen in the cell to the corresponding DB Agent periodically to maintain the persistency of the cell. According to different application scenarios, persistency management has different requirement to save the changes of the cell states. Normally, it does not care about virtual entity's intermediate state data, e.g. when a user move an object, only the finally position of the object need to be recorded. So Cell State Recorder throttles the real-time updates of object state data, which avoids the object state data flow overwhelming at the DB Agent. As persistency data flow transmission is independent of the real time VE interaction, Cell State Recorder can queue and schedule the persistency data according to the available system bandwidth, which saves more resources for the time-critical data flows such as object state data flow.

Persistency Agent supports migration and remote cloning. Serving as the real-time VE state backup for the corresponding Consistency Agent, Persistency Agent can only migrate/clone at Controlling Nodes for the stability reason. The mobility of Persistency Agent brings the following benefits:

(1) By migrating, Persistency Agent can work at the Controlling Node with sufficient resources, which provides better service and avoids the bottleneck emerging at the original Controlling Node.

(2) By remote cloning, multiple Persistency Agents work together to preserve the VE states and to update changes to the corresponding DB Agent, which avoids the potential bottleneck emerging at a single point.

The detailed mechanism for agent migration and agent remote cloning is discussed in Section 3.7.

3.5 VE Directory Management

VE Directory Management acts as an abstraction layer between CVE users and CVE content layer, which provide the bridging information for both sides. Since the CVE is managed in a distributed manner, VE Directory Management layer provides the service to link the users to the agents at content layer so that users can enter the cells which

includes their intended place, interested entities or user, etc. This layer also provides users' information to the agents at content layer to offer user customized service. VE Directory Management is achieved by a group of Gateway Agents.

3.5.1 Gateway Agent

Gateway Agents provide the service to store and manage user information and to lead users to their intended places. When a CVE system is initialized, Gateway Agents are deployed to multiple well-known Controlling Nodes on the Internet.

Gateway Agent maintains user information. It provides user registration, authentication, and user entry log services. A new user needs to register and provide his profile information to Gateway Agent. Each time when a user enters the CVE, he need authenticate his identity at Gateway Agent. At the same time, Gateway Agent records the user's entry log.

Gateway Agent also provides a CVE content directory service. It directs the user to the agents at content layer to enter the CVE. Gateway Agent collaborates with the agents in content layer to lookup the agent addresses for the cells so that user can connect to the Cell Agent and Consistency Agent to join the cell activity. Moreover, Gateway Agent can provide virtual place location, virtual entity, and online user searching service.

Gateway Agent supports migration and remote cloning. Serving as a portal and a search engine for CVE, Gateway Agent can migrate/clone at multiple Controlling Nodes. The mobility of Gateway Agent brings the following benefits:

(1) By migrating, Gateway Agent can work at the other Controlling Node with sufficient resources or near to the majority of users, which can provide better service and avoid the bottleneck emerging at the original Controlling Node

(2) By remote cloning at other Controlling Nodes, multiple Gateway Agent work together to process the users request, which avoids bottleneck emerging at a single point.

The detailed mechanism for agent migration and agent remote cloning is discussed in Section 3.7.

3.6 User Application

User application is a program by which user navigates and interacts within the CVE. It communicates with agents in content layer and gateway layer to take part in the CVE activities. As shown in Figure 3-14, the architecture of user application for MACVE can be divided into three layers: User Communication Manager, CVE Synthesizer, and User Interface.

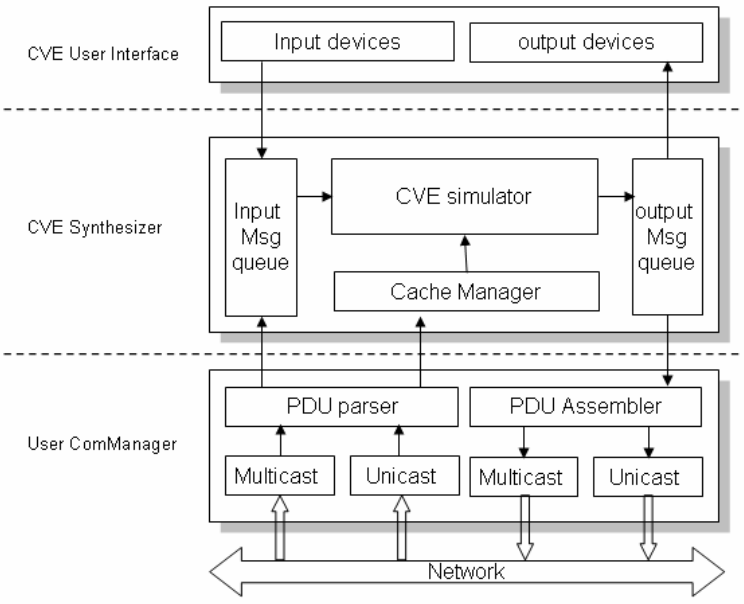

Figure 3-14 User application architecture

3.6.1 User Communication Manager

User Communication Manager (UserComManager) is a software module in the user application which functions as an application level network interface to send and receive CVE data. UserComManager needs to communicate with the agents at the gateway layer and content layer to log in the CVE, to download the VE scene data, VE state data, and to exchange the object state data. When transmitting within the network, CVE data is capsulated as protocol data unit (PDU) messages. PDU Parser and PDU Assembler are

two components in UserComManager to pack and unpack the PDU Messages according to the PDU format in MACVE Protocol (defined in Subsection 4.1.3). UserComManager provides both unicast and multicast capability. It separates communication network from the CVE scene construction so that the low level communication protocol can be dynamically changed without affecting the high level components.

3.6.2 CVE Synthesizer

CVE Synthesizer is a software module in user application to constructs the required CVE scene graph that the user navigates and interacts with. It performs the CVE simulation to make the scene graph consistent across the network. CVE synthesizer loads the CVE scene data from the UserComManager or Cache Manager. Cache Manager is a component to save the scene data on the local storage for possible future use. CVE Simulator receives the local user actions from User Interface module and the remote user actions from the PDU parser of UserComManager. Then it computes the effects of local and remote user's actions on the CVE and updates the scene graph. At the same time, the updates of local user's action propagate to network by the UserComManager.

3.6.3 User Interface

User interface is a software module which includes input device driver and output device driver. Input device driver capture user actions and commands from the input devices, such as keyboard, joystick, 3D space ball, data glove, etc. Output device driver presents the CVE to user by using different multimedia components, such 2D/3D graphics, audio, video, text, etc. User interface layer is independent of CVE Synthesizer, which makes it possible to used different technology to present the CVE.

3.7 Agent Mobility Algorithm

After decomposing the CVE system into agents, we propose an algorithm to make the agents to dynamically migrate/clone from the heavily-loaded node to lightly-loaded node to avoid the potential bottlenecks. We call this algorithm agent mobility algorithm.

As introduced in Section 3.1, agents in MACVE have two forms of mobile actions: agent migration and agent remote cloning. Agent migration requires that agents that run on the overloaded nodes migrate to the less loaded nodes. Agent remote cloning requires that a new agent created on less loaded nodes and share the workload with the original agents. Both agent migration and remote cloning help to distribute the system workload in order to avoid potential bottleneck. The agent mobility algorithm offers an autonomous method to dynamically make mobile action decision to distribute the system workload.

In MACVE, tasks are performed by agents. Because of the dynamic user interaction, the fluctuation of available resources on each node (background load of node may change), and the ever changing nature of network traffic, the workload of each agent/node vary over time in a nondeterministic way. On the other hand, each agent is an autonomous entity which is self-awareness. They have their own performance and resource requirements. For example, for Cell Agent, it needs to deliver large volume of VE scene data to users timely, so it requires large outgoing bandwidth and the network traffic are often in a bursting manner. For Consistency Agent, it processes the object state data for the required consistency control, maintains the real-time VE state; routes object state data to the relevant users, so it is computing intensive and requires quick routing process. For Persistency Agent, it maintains the backup of VE state and updates the VE state into database periodically, so it needs high stability. Therefore, the decisions, whether and when to initiate agent migration/cloning; which agent needs to migrate or clone; and where the location for agent migrating/cloning is, are nontrivial. The agents can not be treated as common tasks as the conventional load distributing algorithms do[120, 121]. In MACVE, there is no central scheduler to make migration/clone decisions. The decision is made by the collaboration of multiple agents.

As shown in Figure 3-15, the mobile action decision process includes three sub processes. They are mobile action initiation, mobile action reasoning, and node discovery.

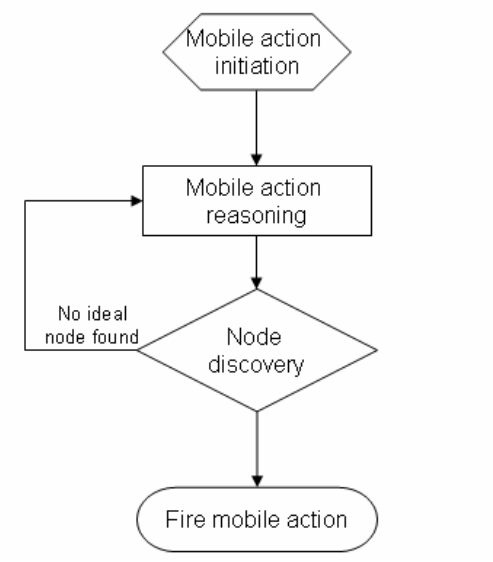

Figure 3-15 Mobile action decision

Before we further present the proposed algorithm, let us define the following notation.

a_s = the selected agent to perform mobile action

n_s = the source node for the mobile action

n_d = the destination node for the mobile action

t_i = the time to initiate the mobile action reasoning

k = iteration times of unsuccessful node discovery process

The general idea for the agent mobility algorithm is shown in Table 3-1:

Table 3-2 Algorithm for overall agent mobility algorithm

```
//main agent mobility algorithm
as = NULL;
ns = NULL;
nd = NULL;
ti = NULL;
k = 0;
While (1){
        If  MobileActionInitiation() == FALSE
                continue;
        else
                ns, ti will be set value inside MobileActionInitiation();
MAR:  as = MobileActionReasoning();
        if(as == NULL){
```

```
                wait for a certain time OR other event triggered mobile action initiation;
                continue;
        }
        nd = NodeDiscovery();
        if(nd == NULL)
                goto MAR;
        else
                FireMobileAction(as, ns, nd);
}
```

The mobile action initiation, mobile action reasoning, and node discovery take care of determining when, which and where for the mobile action. After determining n_s, a_s, n_d, the selected mobile action will be fired. If unable to find n_d node for a_s, MobileActionReason() will execute again to select another agent as a_s. If unable to determine a_s, the algorithm fail to make a decision for mobile action. Under this situation, the algorithm will wait for a certain time or wait for other mobile action initiation events to re-compute again to determine possible n_s, a_s, n_d. Warning messages will also prompt the system administrator to add more Controlling Nodes in the system to relieve the workload from n_s. In the following subsection, we describe each of the sub-process in detail.

3.7.1 Mobile Action Initiation

A mobile action initiation process determines when it should propose an agent mobile action to avoid the degradation of the CVE performance. Before we further present the proposed algorithm, let us define the following notation:

t_{span} = a time interval for the calculation of average workload

$W(n_s, t)$ = the average workload of n_s during t-t_{span} and t

$T_e(n_s)$ = export threshold for n_s

$Logoff(n_s, t)$ =at time t, n_s requests to log off

The algorithm for mobile action initiation is shown in Table 3-3.

Table 3-3 Algorithm for mobile action initiation

```
//sub-process of mobile action Initiation
```

```
MobileActionInitiation(){
        At time tx, If ∃n_x : Logoff(n_x, t_x) == TRUE {
                ns = nx;
                ti = tx;
                k = 0;
                Return TRUE;
        }
        At time tx, tx is the end of the time interval for the calculation of the average
workload {
                If ∃n_x : W(n_x, t_x) > T_e(n_x) {
                        ns = nx;
                        ti = tx;
                        k = 0;
                        Return TRUE;
                        }
                }
        }
        Return FLASE;
}
```

A mobile action decision process is initiated by any of the following two event rules:

(1) Any SC Node logoff

In MACVE, Trusted User Nodes log in/off at will. Controlling Nodes may also need to shut down for the maintenance reasons. $Logoff(n_x, t)$ indicates that at time t, n_x requests to log off. Therefore, if there exists a n_x and $Logoff(n_x, t) ==$ TRUE, the mobile action reasoning process and node discovery process will be executed to ensure all agents running on this node migrate to other SC Nodes one by one.

(2) Potential overloading of any SC Node

When a SC Node is overloaded, the agents running on it will not have enough resources to provide the proper services. Mobile action is needed to share the workload to the possible lightly loaded SC Nodes. As introduced in Section 3.3.2.1, each Node Agent has an export threshold. We define $T_e(n_s)$ as the export threshold for n_s. $W(n_s, t)$ is the average workload of n_s during t-t_{span} and t. when $W(n_s, t) > T_e(n_s)$, it indicates that the node is near overloaded and the mobile action reasoning process needs to be initiated.

Mobile action initiation is realized by Node Agent. When the node is preparing to log off or the node has the potential to be overloaded, Node Agent will start to perform the mobile action reasoning process.

3.7.2 Mobile Action Reasoning

Once initiated, the mobile action reasoning process reasons and proposes which agent should perform a mobile action and what mobile action the agent performs (migration or remote cloning).

Researches[122] shows that optimal task selection for load distribution in distributed systems is a NP-complete problem, and only approximation algorithms are applied to provide solutions in polynomial times. As the agents in MACVE have their own performance requirements and more essentially, agents, such as Consistency Agents, need real-time communication with users, round-robin algorithms, random algorithms, FIFO algorithm[123] for agent selection are not applicable for MACVE. The mobile agent reasoning process needs to consider the following factors when proposes a mobile action.

- The mobile action should bring as little impact to the CVE performance as possible;
- After the mobile action, the workload at original node should decrease considerably which is worthwhile to incur the transfer overhead;
- The overhead incurred by the mobile action should be minimal;

Based on the above considerations, the algorithm for mobile action reasoning is proposed and presented in Table 3-4.

Table 3-4 Algorithm for mobile action reasoning

```
//sub-process of mobile action reasoning
MobileActionReasoning(){
        A_t = Φ;                                  // a temporary set of agents
        If Logoff(n_s, t_i) == TRUE {             //algorithm is initiated by n_s logoff request
                A_t = {a_i|a_i is running at n_s AND PC(a_i) = minimal exiting agent priority
                class};
```

```
            as = ai, where W(ai) = max W(a), ai ∈ At ;
        }
        Else{                                   //algorithm is initiated by ns potential
        overloading
            At = {ai|ai is running at ns AND PC(ai) = (k+1)th maximal exiting agent
            priority class };
            ∀ai ∈ At : Wd(ai) =| W(ns,ti) − Wr(ns,ai) − Wi(ns) | ;
            as = ai, where Wd(ai) = min Wd(a),  ai ∈ At ;
        }
        k++;
        Return as;
}
```

Where

A_t = a temporary set of agents, which store the intermediate decision results.

$PC(a_i)$ = a integer to indicate the priority class of an agent (a_i) for performing a mobile action.

As the services provided by different types of agents vary greatly in MACVE, mobile actions of the agents also have different impacts on the performances of these services. To ensure their performance, agents with less performance impacts during their mobile actions are more willing to perform a mobile action. We define $PC(a_i)$ based on the services provided by the agents. For the existing agents, there are four priority classes, as shown in Table 3-5.

Table 3-5 Agent priority classes

$PC(a_i) =$	1	if a_i = Consistency Agent
	2	if a_i = Gateway Agent OR Cell Agent
	3	if a_i = Persistency Agent
	4	if a_i = other Agents

Consistency Agents belong to the lowest priority class for mobile action, because any mobile action of them will directly affect the real-time user interaction. The lowest priority minimizes the possibility for Consistency Agents to perform mobile action. Gateway Agents and Cell Agents belong to the second lowest priority class, because although they are not as critical as Consistency Agents to processing the frequent user

interaction messages, they still need to provide the time-sensitive services for the users' requests, such as virtual place searching service. Persistency Agents belong to the third priority class, because they do not provide the time-sensitive service, but they need high stability, which expects to reduce the chance of mobile actions if possible. Finally, other agents except the above agents belong to the last priority class, because they provide the resource management services and are not time-sensitive. They are more willing to take mobile actions.

If the mobile action reasoning is initiated by a node logoff event, all agents running on the node need to migrate to other nodes. Two steps are performed to select the first migrating agent:

(1) Agents that provide the critical service for real-time user interaction should migrate to other node as early as possible, which ensure critical agents has more chances to find an ideal destination node that has enough resources to provide a better service. Therefore, the agents with the minimal priority class on n_s are put in a temporary set, A_t, for further selection.

(2) For the agents in A_t, the one with the maximal agent workload, $W(a_i)$, should be selected firstly, which ensure agent with maximal workload has more chances to find an ideal destination node that has enough resources to provide a better service.

After above steps, a_s is selected and it will migrate to the ideal node found in the subsequent node discovery process. If n_s still has agents running on it, the next circular MobileActionInitiation() will return TRUE again and MobileActionReasoning() will select another agent for migration. Like this, until all the agents on n_s move out, n_s can log off from MACVE at last.

On the other hand, if the mobile action reasoning is initiated by a node potential overloading event, another two steps are performed to select the agent:

(1) Agents that belong to the existing (k+1)th highest priority class are put in a temporary set, A_t, for further selection. Agents with higher maximal priority class

are more willing to perform mobile action. But the agent in the high priority class may be unable to find an ideal node in the subsequent node discovery process. k indicate the iteration time for unsuccessful NodeDiscovery(). If the selected a_s can not find ideal node in NodeDiscovery(), k will increase by 1. The unsuccessful Node Discovery process triggers Mobile Action Reasoning process again. MobileActionReasoning() will select another agent in the next highest existing priority class indicated by (k+1).

(2) In the algorithm, we hope the relieved workload caused by the mobile action of the selected agent, a_s, should be large enough so that the source node, n_s, will not become overloaded again due to the future small increase in its workload. For each node, we define its ideal workload as $W_i(n)$. After a_s performs the mobile action, n_s will relieve part of its workload to the destination node (n_d). We define $W_r(n_s, a_i)$ as the relieved workload at n_s after performing the mobile action of a_s. (For agent migration, $W_r(n_s, a_i)$ is the workload of the migrating agent. For agent remote cloning, $W_r(n_s, a_i)$ is the shared workload of the cloned agent.) To reduce the frequency of future mobile actions and increase the efficiency of resource utilization on n_s, after the mobile action, the workload of n_s should approach $W_i(n_s)$. We define $W_d(a_i)$ as the deviation of the workload on n_s after the proposed mobile action and the ideal workload, $W_i(n_s)$.

$$W_d(a_i) = | W(n_s, t_i) - W_r(n_s, a_i) - W_i(n_s) |$$

For the agents in A_t, the agent with the minimal $W_d(a_i)$ is selected as a_s.

As for the decision of what mobile action (agent migration or agent remote cloning) to perform, it is agent type and CVE scenario dependent. For example, Cell Agent prefers clone to migration when there exist Trusted User Nodes in the system which have already cached the scene data. This is because remote cloning at Trusted User Node to share the workload is more efficient than to find a node with enough resources to migrate. Consistency Agent is more like migration, since the clone of Consistency Agent causes the problem of state synchronization among the original agent and the cloned ones, which

cost extra computing and networking resources. Therefore in MACVE, agents itself decide the mobile action whether to migrate or clone.

Mobile action reasoning is not a process performed by a single agent. As shown in Figure 3-16, Node Agent negotiates with other agents running on n_s to make the decision.

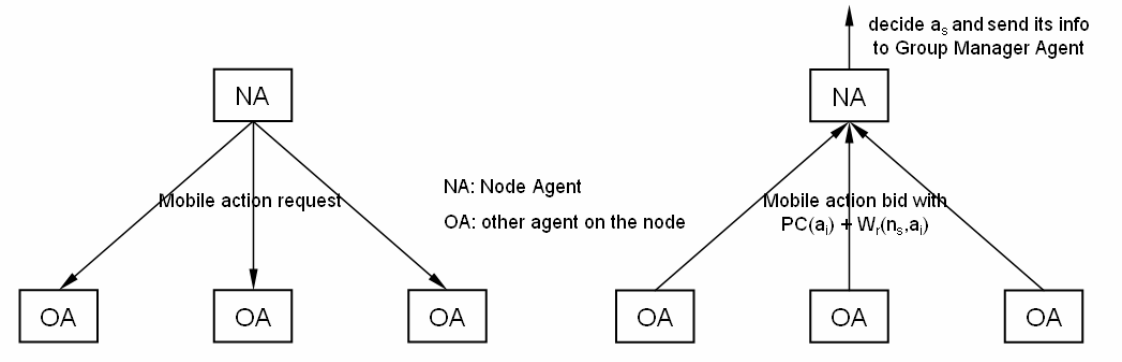

Figure 3-16 Agent collaboration for agent action reasoning

Node Agent polls the mobile action request to all other agents on n_s. Other agents bid a mobile action with the information of its $PC(a_i)$ and $W_r(n_s, a_i)$. After receiving all bids, Node Agent performs the reasoning algorithm and selects the a_s. The information of a_s is sent to the Group Manager Agent for the further node discovery process, which will be discussed in detail in the next section.

3.7.3 Node Discovery

Node discovery is responsible for finding an ideal node to host the migrating agent or to create the cloned agent. The algorithm for node discovery is presented in Table 3-6.

Table 3-6 Algorithm for node discovery

```
//sub-process of node discovery
NodeDiscovery(){
        N_t = Φ ;                                   // a temporary set of nodes
        N_ms(a_i) = {n_i | n_i belong to the mobile space scoping of a_i} ;
        N_t = {n_i | W(n_i,t_i) < T_i(n_i) ∧ W(n_i,t_i) + W_r(n_s,a_s) < T_e(n_i) ∧ n_i ∈ N_ms} ;
        If( N_t = Φ )
                If Logoff(n_s,t_i) == TRUE
                        N_t = N_ms(a_i) ;
```

```
                Else
                        Return NULL;
        n_d = n_i where Distance(n_s, n_i) = min Distance(n_s, n), n_i ∈ N_t ;
        Return n_d;
}
```

Where

N_t = a temporary set of nodes, which store the intermediate decision results.

N_{ms} (a_i) = a set of nodes that belong to the mobile space of a_i.

The algorithm for node discovery includes three steps:

(1) Limit mobile space scoping for a_s.

Mobile actions for different types of agents have their own mobile space scoping. Mobile space is the possible nodes that the mobile action can perform. For example, Persistency Agent can only work at Controlling Nodes for the stability reasons. When searching an ideal node, The nodes belonging to the mobile space for a_i are put in $N_{ms}(a_i)$.

(2) Apply available resource constraint

To ensure the migrating/cloned agent work properly, the ideal node should have enough resources to host the migrating agent or to create the cloned agent. As introduced in Subsection 3.3.2.1, each node has an import threshold $T_i(n)$.

$$W(n_i,t) < T_i(n_i) \text{ AND } W(n_i,t) + W_r(n_s,a_s) < T_e(n_i) \quad \textbf{(3-3)}$$

For n_i, if it is subject to Equation 3-3, n_i has spare resources and the available resources is enough to perform the mobile action of a_s. The nodes in $N_{ms}(a_i)$ that are subject to Equation 3-3 are stored in a temporary set, N_t, for further selection.

After applying available resource constraint, if $N_t = \Phi$, it means no node has enough resources to host the migrating agent or to create the cloned agent. When the mobile action reasoning is initiated by a node logoff event, node discovery must select a node to receive the a_s, even though there is no node with enough resources for the time being. Therefore, all the node in $N_{ms}(a_i)$ are used for further selection. On the other

hand, when the mobile action reasoning is initiated by a node potential overloading event, NULL result is returned. Since the agent mobile decision is an iterative process, if n_s = NULL, Mobile action reasoning will be restart to propose a new mobile action. Mobile action reasoning and node discovery work together to finally decide a mobile action.

(3) Identify n_d for Optimize mobile action

To optimize the mobile action with minimal overhead, the time used for transferring the agent code and agent state should be minimized. Network distance is metrics to represent the network speed, which is measured as the packet round-trip transmission delay. Network distance is used for the final decision of the destination node. N_d should have the shortest network distance to n_s. Therefore

$$n_d = n_i \text{ where Distance}(n_s, n_i) = \min \text{Distance}(n_s, n),\ n_i \in N_t$$

where Distance(n_s, n_i) is the network distance for n_s to n_i.

As shown in Figure 3-17, the node discovery algorithm is realized by the collaboration of Node Agents, Group Manager Agents and CRM Agent.

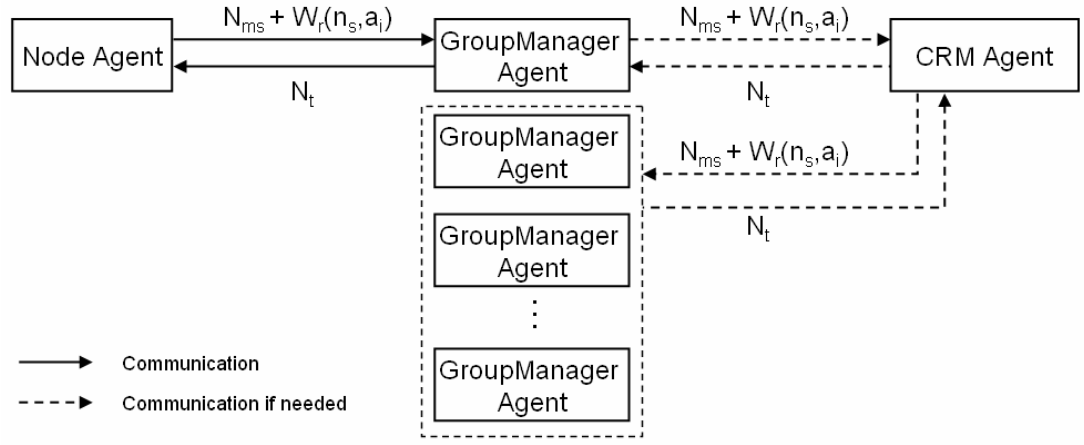

Figure 3-17 Agent collaboration for node discovery

Node Agent sends the node discovery request with its mobile space scoping information, $N_{ms}(a_i)$, to the Group Manager Agent. The Group Manager Agent selects the nodes based on the mobile space scoping and available resource constraint and stores the candidates

of ideal nodes in N_t. If the Group Manager Agent cannot find a candidate node within its own group, the Group Manager Agent will negotiate with CRM Agent to search for candidates of ideal nodes in other node groups. After generate N_t, Group Manager Agent sends N_t to the requested Node Agent, who will find the ideal destination node by comparing the network distance. After determining n_s, a_s and n_d, Node Agent will send migration commands to as to perform the agent action.

In summary, agent mobility algorithm enables the agents in MACVE to dynamically migrate/clone to any qualified SC Node to distribute the system workload and ensure the real-time CVE performance. By the algorithm, Trusted User Nodes can share the system workload by hosting agents when they log in and transfer all agents running on it to other nodes when they log off. And the potential overloaded node can dynamically select a mobile action and find an ideal node with the enough resources to distribute its workload. Both of them can improve the LCVE scalability.

3.8 Adaptive Consistency Control

To effectively support dynamic and various consistency requirements in a LCVE, we propose adaptive consistency control in MACVE to dynamically decide the message communication architecture for different object state data depending on their run-time application semantics.

In a real application, various activities happen in a LCVE and generate different and dynamic consistency requirements. The diversity of these consistency requirements has the following manifests:

(1) Different cells in a LCVE may have totally different consistency requirements according to the theme of the cells. For example, a cell that represents a virtual engineering design studio needs exactly identical state for each design action to ensure accuracy. Whereas, a cell that represents a virtual theater values the smooth performers' actions more than the absolute identical view of the actions among all participants.

(2) In addition, even in the same cell, object state data belong to different activities may also have different consistency requirement. For example, in a virtual auction room, the object state data for biding information needs to be consistent for all participants to avoid confliction during the auction. Whereas, the location of each avatar in the room may tolerate slightly short-term difference.

(3) Moreover, in highly interactive applications, with the changes of current task or application scenario, activities happened in the cell may change[124]. This may also lead the change of consistency requirements for the same object state data. For example, in a virtual playground, when a football match is in process, the location information of the players needs to be same to ensure fairness. Whereas, during the break or after the game, the players' location information does not have strict requirement on consistency.

Therefore, a LCVE may have dynamic and multiple levels of consistency requirements. A static consistency model can hardly provide effective techniques to control different level of consistency in a LCVE. An adaptive consistency control mechanism is needed to ensure the required consistency in a LCVE.

As introduced in Subsection 2.1.1, the essence of consistency control is the tradeoff between consistency degrees with the responsiveness requirements of different activities in CVE. The techniques for consistency control are strongly coupled with the data communication architecture[125, 126]. If a CVE attempts to guarantee the CVE state to be identical among all its participants, the most effective way is to implement the centralized repositories on a client-server communication architecture [2]. Since all of the object state data is sent to a server, the server preserves the snapshot of shared VE state. The shared VE state can be protected by locks to guarantee no conflict existing in the concurrent VE state updates. The server can also ensure the universal order of the object updates at each participant so that all participants display the identical views of the VE. However, client-server architecture requires considerable communication overhead and server computational overhead. And it may also affect the responsiveness of the object state updates at each participant. On the other hand, if a CVE attempts to maximize the

responsiveness of the VE state changes for all its participants, peer-to-peer architecture is a better choice. Each participant generates the VE state based on the object state data received from others. However, in peer-to-peer architecture, it can only maintain approximately consistent VE states for each participant.

Based on the dependence of consistency model and the system communication architecture, MACVE supports strict consistency model, weak consistency model and best-effort consistency model, as shown in Figure 3-18.

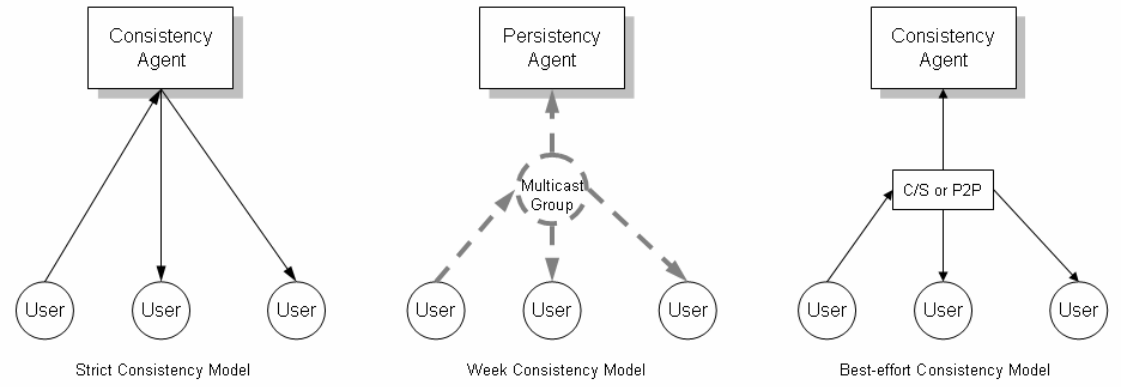

Figure 3-18 Consistency control models in MACVE

(1) Strict consistency model refers to the mechanisms to ensure the controlled VE states to be identical among all participants, although they may present at different time shifts. It is also defined as delay global consistency in [19]. This model is for the object state data that need reliable communications, synchronized state update or concurrency control. Object state data for strict consistency control is communicated in client-server manner. As the data that needs the strict consistency control may have different responsiveness requirements, strict consistency model assign different priorities to these data according their responsiveness requirements. The data with high responsiveness requirement will be processed and routed with high priorities.

(2) Weak consistency model refers to the mechanisms to ensure high responsiveness of the controlled VE state update rather than the absolute accuracy of the state [2]. This model is for the object state data that is generated and transmitted frequently.

Although the loss or disorder of these data may cause short-term inconsistencies among participants, the subsequent message can quickly correct the inaccurate state. Object state data for weak consistency is communicated in peer-to-peer manner. Although the VE state may be slightly inconsistent, the responsiveness of VE state update is ensured.

(3) Best-effort consistency model refers to the mechanisms that when Consistency Agent has enough resources in terms of computational cycles and network bandwidth, it tries its best to maintain the controlled VE state to be identical at each participant. But this model does not ensure the controlled states to be consistent all the time. This model is for the object state data that has neither strict consistency nor responsiveness requirements. The data for best-effort consistency model can be communicated in either client-server architecture or peer-to-peer architecture depending on the performance of the Consistency Agent. When the consistency agent still has bandwidth and computing resources left after it ensures its strict consistency service, the data for best-effort consistency requirement is communicated in a client-server manner to ensure an identical state. On the other hand, when the Consistency Agent do not have enough resources to ensure its strict consistency service, part of the object state data with best-effort consistency requirement will be communicated in peer-to-peer manner.

Adaptive consistency control in MACVE is the mechanism to dynamically map the consistency and responsiveness requirements for the run-time VE activities to the selection of corresponding consistency model for the object state data. Before formulate the mechanism, we make the following definitions:

$S = (s_1, s_2, s_3, \cdots, s_y)$, which represents all of the shared states in a cell. Each element of S can describe a single attribute of an entity/avatar, such as the location of a ball, or a group of attributes that have the same consistency requirement, such as the locations of all avatars in a cell.

$R = (r_1, r_2, r_3 \cdots r_y)$, which represents the consistency requirement for the shared state in S. Each element in R can have the value of 1, 2 or 3, which indicate the strict, weak and best-effort consistency requirement respectively.

$M = (m_1, m_2, m_3, \cdots, m_x)$, which represents all types of messages may be issued when activities happened in a cell, such as avatar walk message.

$C = (c_1, c_2, c_3 \cdots c_x)$, which represents the consistency model for the message types in M. Each element in C can have the value of 1, 2 or 3, which indicate the strict, weak and best-effort consistency model respectively.

$$MS = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \cdots & \alpha_{1y} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \cdots & \alpha_{2y} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \cdots & \alpha_{3y} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \alpha_{x1} & \alpha_{x2} & \alpha_{x3} & \cdots & \alpha_{xy} \end{bmatrix}$$, which is an $x \times y$ matrix.

MS represents the relationship between the shared states and the types of issued messages in the cell. When $1 \le i \le x$, $1 \le j \le y$ and message type m_i cause the changes of VE state s_j, $\alpha_{ij} = 1$; otherwise, $\alpha_{ij} = 0$. A single message may include the information to change more than one VE states.

For different CVE applications, various activities produce different consistency and responsiveness requirements on each state in S. Given t, we have

$$S_{Con\_R} = f_{semantic}(S, t) \qquad \textbf{(3-4)}$$

where $f_{semantic}$ is the function to match each consistency requirement for the activities to each shared state item. It is the subjective matching process given at the application level during scenario design phase.

As the adaptive consistency control applies to each message, we assume that a single message will not change the VE states that have different consistency requirements. Thus, the consistency model for each type of the message is given by the following model mapping equation.

$$C = ModelMapping(MS, R) \Leftrightarrow \ c_i = any\ r_j, where\ \alpha_{ij} = 1 \quad \textbf{(3-5)}$$

where, $1 \le i \le x$, $1 \le j \le y$.

C is calculated based on the S, R, M and MS provided by the system designer before the LCVE starts. All of the messages are processed and communicated using the calculated consistency model until the semantic of the activity happened in the cell changes. Because of the dynamic nature of LCVE, at a given time t, the changes of run-time activities in a cell may affect consistency requirements for the shared states, which leads to re-calculation of the consistency model for the affected types of messages. After the re-calculation, the processing and communication of affected types of messages will automatically change to adopt the new consistency model.

This adaptive mechanism is realized by the Consistency Agent of that cell. After each re-calculation of C, Consistency Agent informs the user application of the changes of consistency model for the affected message types by sending consistency control message to them.

An example is used to illustrate the adaptive consistency control. In a virtual training course, the teacher tries to teach the students how to assemble a machine. The course includes two sections. During the first section from t_0 to t_1, the teacher introduces the basic construction of the machine by showing and explaining each accessory to the students. During the second section from t_1 to t_2, the teacher demos how to assemble the machine by slowly and accurately put the accessories together. In the course, real-time voice and writeboard is provided the teacher and students to for their communication.

Based on the application semantic, there are two activities happened in the course: introduction activity and domo activity. The shared states in this application are denoted $S = (s_1, s_2, s_3, s_4, s_5, s_6,)$, where:

s_1 = the teacher and students voice

s_2 = the content in the whiteboard

s_3 = the location of the teacher and students

s_4 = the rotation of the teacher and students

s_5 = the hand gesture of the teacher

s_6 = the states of the accessories of the machine

The issued message types in this application are denoted M = (m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8), where:

m_1 = the teacher and students speaking message

m_2 = the teacher and students editing whiteboard message

m_3 = the teacher and students moving location message

m_4 = the teacher and students turning around message

m_5 = the teacher and students combined movement message

m_6 = grabbing accessory message

m_7 = moving a accessory location message

m_8 = rotating a accessory message

The relationship of M and S in the application is

$$MS = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

After analyzing the scenario of the introduction activity, we assume the consistency requirements for the shared states are R = (2, 1, 3, 3, 2, 2) from t_0 to t_1. Applying MS and R to Equation 3-5, the consistency model for types of messages are C = (2, 1, 3, 3, 3, 2, 2, 2).

After analyzing the scenario of the demo activity, we assume the consistency requirements for the shared states are R = (2, 1, 3, 3, 1, 1) from t_1 to t_2. The consistency requirement for s_5 and s_6 has been changed. This is because during the introduction activity, the teacher grabs and shows the machine accessories just in order to give the students a general and draft impression of each accessory, which does not need the teacher's hand gesture and state of the accessory to be identical among the students;

whereas during the demo activity, the detailed assemble process is need to be viewed identically by all students for accurate study, which depends on the strict consistency control on s_5 and s_6. Therefore, at t_1, Equation 3-5 is recalculated and the result is C = (2, 1, 3, 3, 3, 1, 1, 1), which indicates the consistency model for m_6, m_7 and m_8 will dynamically change from weak consistency model to strict consistency model. The communication architecture of m_6, m_7 and m_8 will also change from peer-to-peer to client-server architecture accordingly.

The above example illustrates that by dynamically adopting different consistency model and the communication architecture for object state data, the adaptive consistency control can ensure the required consistency, efficiently utilizes the system resources and hence improves the system scalability.

3.9 Agent Failure Recovery

As introduced in Section 3.1, one aspect of the basic idea behind MACVE is that the CVE system workloads can be shared by the Trusted User Nodes. However, User Nodes tend to have less stability compared with Service Provider Nodes. So MACVE provides a mechanism for agent failure recovery to ensure the system stability.

MACVE adopts a hierarchical monitoring mechanism to detect an agent failure. In MACVE, the management of all agents is organized as a tree-structure, such as Group Manager Agent manages the Node Agents which belong to its group; or Region Agent manages its Cell Agents. The agent at the root position is called the Parent Agent and the agent at the leaf position is called Child Agent. The Parent Agent is responsible for monitoring the proper execution of its Child Agents. Every Child Agent sends a "heartbeat" message (a living message) periodically to its Parent Agent. If the Parent Agent receives no heartbeat from the Child Agent within a timeout, it detects the crash of the Child Agent. Then, the Parent Agent asks ARM agent to send a new Child Agent to a suitable SC Node to resume the system tasks.

MACVE uses reliable redundancy to recovery agent failure. For example, because of system stability reason, Persistency Agent of a cell is not allowed to be transferred to

Trusted User Node, because it maintains the backup of the current cell state. When a Trusted User Node crashes and result in a Consistency Agent lost, the newly launched Consistency Agent will get the latest cell state from the corresponding Persistency Agent at the Controlling Node. Thus, it performs an effective recovery of the lost Consistency Agent and minimizes the impact of the crashes of Trusted User Node. If both Consistency Agent and Persistency Agent crash at the same time, the newly launched Persistency Agent will retrieve the cell state from the DB Agent. The reloaded cell state is the latest persisted state. The newly launched Consistency Agent gets the cell state from the Persistency Agent and in order to guarantee consistency, all users in this cell need to roll back their cell state the same as the Consistency Agent.

3.10 System Scalability

After the introduction of entire proposed framework, we will theoretically analyze the system scalability of MACVE in this section. Because of the mobility of agents and the computing resource sharing at Trusted Used Node, MACVE can autonomously improve the system scalability in multiple ways. We analyze the scalability of MACVE in the following two sections.

3.10.1 Distribution of the Data Flows

In MACVE, the five main data flows (defined in Section 1.2) in a cell have been controlled by Cell Agent, Consistency Agent and Persistency Agent respectively. Each agent performs an independent task to manage one or more data flows. We use a set Task to represent these tasks.

$$\text{Task} = \{\text{CelAgt}_i, \text{ConAgt}_i, \text{PerAgt}_i, | \ i=1 \ldots N_c\} \quad (3\text{-}6)$$

where CelAgt_i refers to the task performed by Cell Agent to manage VE scene data flow in Cell_i; ConAgt_i refers to the task performed by Consistency Agent to manage the VE state data flow, the object state data flow and the consistency control data flow in Cell_i; PerAgt_i refers to the task performed by Persistency Agent to manage VE persistency data flow in Cell_i; N_c is the number of cells in a LCVE.

These tasks need to be distributed to nodes to ensure that each of them has sufficient resources to execute. We use a Set S to represent the Controlling Nodes which are server nodes in the traditional CVE systems.

$$S = \{S_i | \ i = 1 \ldots N_s\} \tag{3-7}$$

where S_i refers to a Controlling Node; N_s is the number of Controlling Nodes in a LCVE. We use another Set U to represent the Trusted User Nodes in MACVE systems.

$$U = \{U_i | \ i = 1 \ldots N_t\} \tag{3-8}$$

where U_i refers to a Trusted User Node; N_t is the number of Trusted User Nodes in a LCVE. The distribution of the tasks is a dynamic mapping from the set Task to the set Node, which guarantee that the task of each node is less than its workload threshold.

In the traditional multi-server CVE system, the task distribution is the mapping:

$$f_{Traditional}: \text{Task} \rightarrow S \tag{3-9}$$

where $f_{Traditional}$ is a mapping function which implements the distribution/balancing algorithm in traditional multi-server CVE system. The tasks are only distributed among the server nodes, whereas in MACVE, these tasks can also be executed by Trusted User Nodes in addition to the Controlling Nodes (For system stability reason, Persistency Agents, namely task $PerAgt_i$, are not allowed to be transferred to Trusted User Node as discussed in Section 3.9). The task distribution is the mapping:

$$f_{MACVE}: \text{Task} \rightarrow S \cup U \tag{3-10}$$

where f_{MACVE} is a mapping function which implements the agent mobility algorithm of CRM in a MACVE system.

Qualitatively we can see that, in MACVE, the tasks for exchanging and processing the data flow in a CVE are distributed at $S \cup U$ nodes, which is more than the traditional multi-server CVE system. Before we further quantitatively discuss the workload of exchanging and processing the data flow needed by the system and provided by the nodes, let us define the following notation.

Δw_i = The workload resulted from user i at SC Nodes;

C_i^s = The available capability of Controlling Node (server) i;

C_i^t = The available capability of Trusted User Node i;

N_u = The number of concurrent users in a LCVE;

In our model, Δw_i may represent CPU load as well as bandwidth requirements, both of which can apply the following equations. Firstly, we make the following assumptions.

- The overall workload generated at SC Nodes is linear with the number of the users.
- Δw_i follows a Normal Distribution, then in a LCVE, when N_u is large enough, Δw_i may have a stable mean value $\overline{\Delta w}$;
- Usually more than one task is performed at a Controlling Node so that one or more tasks can be transferred to Trusted User Nodes.

To ensure no bottleneck among Controlling Nodes (servers) in the traditional CVE systems, we need to ensure:

$$\sum_{i=1}^{Nu} \Delta w_i \le \sum_{i=1}^{Ns} C_i^s \Rightarrow N_u * \overline{\Delta w} \le N_s * \overline{C^s} \Rightarrow \overline{C^s} \ge \frac{\overline{\Delta w}}{N_s} * N_u \qquad (3\text{-}11)$$

where $\overline{C^s}$ is the mean value of available capability of all Controlling Nodes. To ensure no bottleneck among SC Nodes in MACVE systems, we should ensure:

$$\sum_{i=1}^{Nu} \Delta w_i \le \sum_{i=1}^{Ns} C_i^s + \sum_{i=1}^{Nt} C_i^t \Rightarrow N_u * \overline{\Delta w} \le N_s * \overline{C^s} + N_t * \overline{C^t} \qquad (3\text{-}12)$$

where $\overline{C^t}$ is the mean value of available capability of all Trusted User Nodes. As the average available capability of Controlling Nodes is much more powerful than those of

the Trusted User Nodes, we assume $\overline{C^s} = k * \overline{C^t}$. In order to simplify the problem, we also assume there is a static ratio of the Trusted User Nodes to all participating user nodes, and we let $\lambda = N_t / N_u$. Then, Equation 3-12 can be changed into

$$\overline{C^s} \geq \frac{\overline{\Delta w} * N_u}{N_s + \frac{\lambda}{k} * N_u} \tag{3-13}$$

For traditional multi-server system and MACVE system, if the number of Controlling Nodes and the average workload produced by one user are same, i.e. N_s and $\overline{\Delta w}$ are the same value in Equation 3-11 and 3-13, we can compare the scalability of both systems by presenting the relationship of $\overline{C^s}$ (in the terms of bandwidth and CPU utilization) and N_u in Figure 3-19 and Figure 3-20.

In Figure 3-19, we discuss the average bandwidth requirement at Controlling Nodes. We assume $N_s = 10$, $\overline{\Delta w} = 0.5$ Mb/s, $k = 100$, $\lambda = 25\%$.

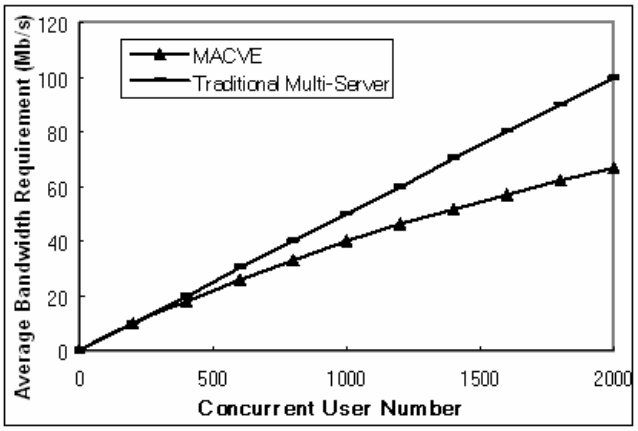

Figure 3-19 The average bandwidth requirement at the Controlling Node in MACVE system vs. in traditional multi-server system

In Figure 3-20, we discuss the average normalized CPU utilization requirement at Controlling Nodes. We assume $N_s = 10$, $\overline{\Delta w} = 0.5$, $k = 100$, $\lambda = 50\%$.

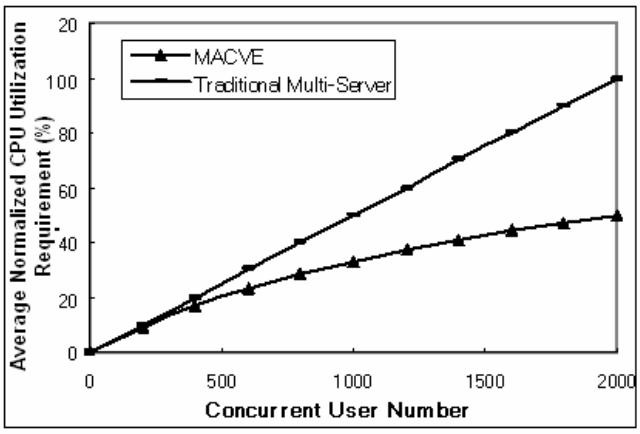

Figure 3-20 The average CPU utilization requirement at the Controlling Nodes in MACVE system vs. in traditional multi-server system

Figure 3-19 and Figure 3-20 illustrates that the same number of the Controlling Nodes and the same number of concurrent users in a LCVE, MACVE system has less resource capability requirements than the traditional multi-server system does. In other words, MACVE system can support more concurrent users than the traditional multi-server system with the same number of Controlling Nodes.

3.10.2 Pervasive VE Scene Data Caching

When a user navigates through a cell, it caches VE scene data of this cell. If other users who are near that user can download the needed VE scene data from that user, it will be faster than downloading from the traditional servers. This also reduces the burden on the servers. MACVE achieves the Pervasive VE Scene Data Caching by cloning Cell Agent and migrate it to the corresponding Trusted User Nodes. A Cell Agent may have multiple cloned ones running at different Trusted User Nodes, when a new user node needs to fetch the VE scene data, it will select "nearest" Cell Agent.

To discuss the effectiveness of the Pervasive VE Scene Data Caching mechanism to the scalability of the LCVE system, we define:

N_u^i = The number of users in $Cell_i$

D^i = The volume of the VE scene data in $Cell_i$

N_t^i = The number of Trusted User Nodes among N_u in $Cell_i$

N_k^i = The number users who can get the VE scene data from the Trusted User Node k in $Cell_i$

Without the Pervasive VE Scene Data Caching mechanism, all users in $Cell_i$ have to download data with the volume of D^i from a single traditional server. The total traffic at the server is

$$V = N_u^i * D^i \tag{3-14}$$

With the Pervasive VE Scene Data Caching mechanism in MACVE, if in each Trust User Node of N_u^i, there is cloning Cell Agent to deliver the VE scene data, the total traffic at the original Cell Agent is

$$V' = N_u^i * D^i - \sum_{k=1}^{N_t^i} N_k^i * D^i \tag{3-15}$$

The ratio of the decrease of traffic at the original Cell Agent is

$$\eta = \frac{V - V'}{V} = (\sum_{k=1}^{Nt} N_k^i * D^i) / (N_u^i * D^i) = \sum_{k=1}^{N_t} N_k^i / N_u^i \tag{3-16}$$

From Equation 3-16, the more Trusted User Node (N_t^i) and the more powerful (N_k^i), the larger the ratio of the decrease for the VE Scene Data traffic at a single SC Node is. The VE Scene Data has been replicated and the data flow has been distributed to multiple SC Nodes, which will enlarge the capability for Scene Data delivering.

Therefore, the mutual independence of LCVE tasks and the participating hosts and the ability for Trusted User Nodes to take over system mobile agents in MACVE make LCVE system more scalable than the systems with traditional multi-server architecture.

3.11 Summary

In this chapter, we present our proposed framework-MACVE, which includes the agents belonging to three layers and a user application for navigating in the CVE. Agent mobility algorithm is proposed to autonomously take mobile actions to distribute system tasks to both Controlling Nodes and Trusted User Nodes. Adaptive consistency control in MACVE ensures the required dynamic and various consistency requirements for different parts of LCVE and efficiently utilizes the system resources; hence it improves the system scalability. Agent failure recovery is also incorporated in MACVE to improve the system stability and robustness. Finally, we theoretically analyze the scalability of MACVE.

Chapter 4

Implementation of MACVE

In this chapter, we introduce the implementation details of a MACVE prototype system. A mobile agent system is specially design for MACVE to allow effective distribution of mobile agents. The mobile agent system includes a Mobile Agent Environment to help the agent to be mobile, a set of MACVE APIs to facilitate Agent implementation, and a MACVE Protocol to standardize MACVE communication and future extension. Based on the mobile agent system, we realized the agent mobility in MACVE that satisfied with the CVE application requirement. Eleven basic types of agents and a web-based user application are implemented for the evaluation of MACVE.

4.1 Mobile Agent System

To achieve our proposed framework, a mobile agent system is needed to support agent implementation, agent mobility, and agent communication. A mobile-agent system provides an infrastructure that implements the agent paradigm. It also provides a protective agent execution environment. Each machine that intends to host mobile agents installs the execution environment so that agents can run on the execution environment and communicate with one another to provide their services collaboratively.

Although there exists many mobile agent systems, such as Telescript, Aglet, Voyager, etc., they all only support agent synchronous migration[127]. During agent synchronous migration, the service provided by the agent suspends until the agent migration finishes. Since the services provided by agents in MACVE deal with user real-time interactions

within the CVE, long time of service suspension will destroy the real-time performance of CVE and frustrated the users. Therefore, synchronous agent migration can not be applied in MACVE. We designed the mobile agent system with asynchronous agent migration. The detailed implementation of our asynchronous agent migration is introduced in Subsection 4.2.1

The mobile agent system implemented for MACVE is called VEMAS (Virtual Environment's Mobile Agent System). VEMAS provide services for all the MACVE agents to make them to be mobile and pervasive to all of the SC nodes in the whole MACVE system. VEMAS includes a MAE (Mobile Agent Environment), MACVE APIs and MACVE Protocol.

4.1.1 Mobile Agent Environment

MAE (Mobile Agent Environment) is an agent container running on each SC Node. It provides a platform to host agents. It helps agents to communicate with each other, to access the machine level information and to perform primitive operations so that agent can be mobile around the network. Figure 4-1 shows the major components in MAE

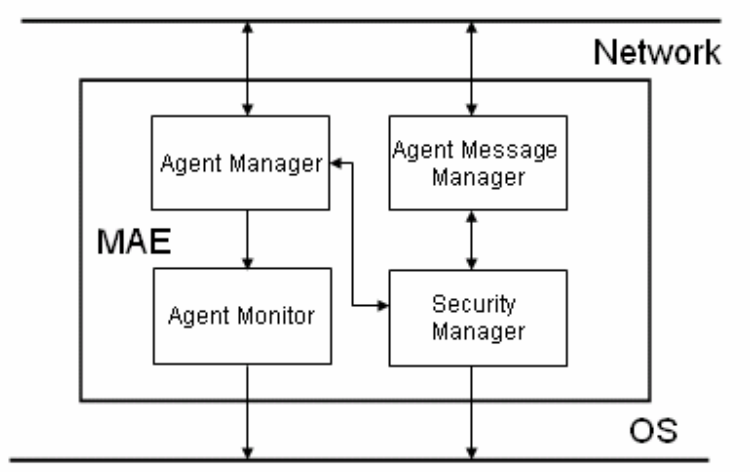

Figure 4-1 Main software components of MAE

- **Agent Manager**

Agent Manager provides the primitive agent operations to support agent mobility. The

primitive agent operations include: (1) creating an agent with its initial configuration or agent running state, (2) transferring agent code to a remote MAE, (3) receiving agent code from a remote MAE, (4) transferring agent running state to a remote MAE, (5) receiving agent running state from a remote MAE, (6) and disposing a running agent. Agent mobility is achieved by a series of these primitive agent operations and agent level synchronization. Agent Manager receives the agent operation commands from local agents or from remote MAE. For example, when ARM Agent plans to launch an agent remotely, it sends the operation command to its MAE. The local MAE forwards the command to the remote MAE to create an agent with its initial configuration file. When more than one operation commands are received, Agent Manager queues the commands for later processing. Agent Manager also provides the service to control the agents that run on the MAE. It supports manually launching, suspending, and terminating agent.

- **Agent Message Manager**

Agent Message Manager provides secure agent message passing service. It is responsible for sending or receiving messages for agents. As illustrated in Figure 4-2, when an agent needs to send a message to another agent, it pushes the message into the outgoing message queue in Agent Message Manager. The receiver agent may run on the same MAE or on a remote MAE. If the receiver agent runs at the same MAE, the message router in Agent Message Manager directly move the message to the incoming messages queue. If the receiver agent runs at a remote MAE, the message router sends the message to the remote MAE. When the Agent Message Manager in the remote MAE receives the message, it pushes the message into its incoming message queue. When any message comes into the incoming message queue, Agent Message Manager dispatches the messages to the corresponding agents that run on it. As Agent Message Manager helps agents to send and receive messages, agents themselves do not need to set up a mass of connections to their relevant agents, which reduces the agent communication complexity. Moreover, before forwarding the messages in the outgoing message queue, message router merges the consecutive messages which are to the same remote MAE, which improves the message transmission efficiency.

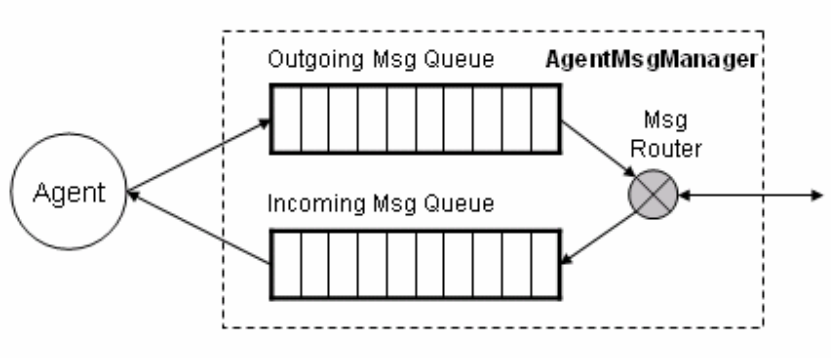

Figure 4-2 Agent message manager

- **Security Manager**

The security of mobile agent system includes protecting the host from attacks by agent, protecting the agent from attacks by host, by other agents or by other software entities. The security issues of mobile agent system are still under ongoing research. MACVE is an open framework which allows third party developers to add their new type of mobile agents into the system. We implement Security Manager in MAE to guarantee the integrity of the incoming agent code, and agent messages. Security Manager is responsible for encrypting the agent code and messages at the sender MAE and decrypting them at the receiver MAE. Security Manager provides two pair of functions EncryptAgentCode() v.s. DecryptAgentCode() and EncryptAgentMsg() v.s. DecryptAgentMsg() for Agent Manager and Agent Message Manager to protect the MAE from malicious code and agent messages.

- **Agent Monitor**

In order to cross different computer platforms, mobile agents in MACVE are platform independent software entities. The Agent Monitor component in MAE provides the service to invoke the system calls of the platform to help agents to query the resources and performance information of the node and the performance information of other agents that run on the node. Agents can request from Agent Monitor for their interested parameters such as the network traffic of the node, the CPU usage for each agent, and the traffic pattern of each agent. For example, Node Agent periodically queries the node workload index from Agent Monitor to decide when to initiate a mobile action.

MAE is implemented in C++, which have high efficiency to control the agents operations and communication. Moreover, it has strong ability to access the system level parameters. Figure 4-3 illustrates a screen shot of MAE.

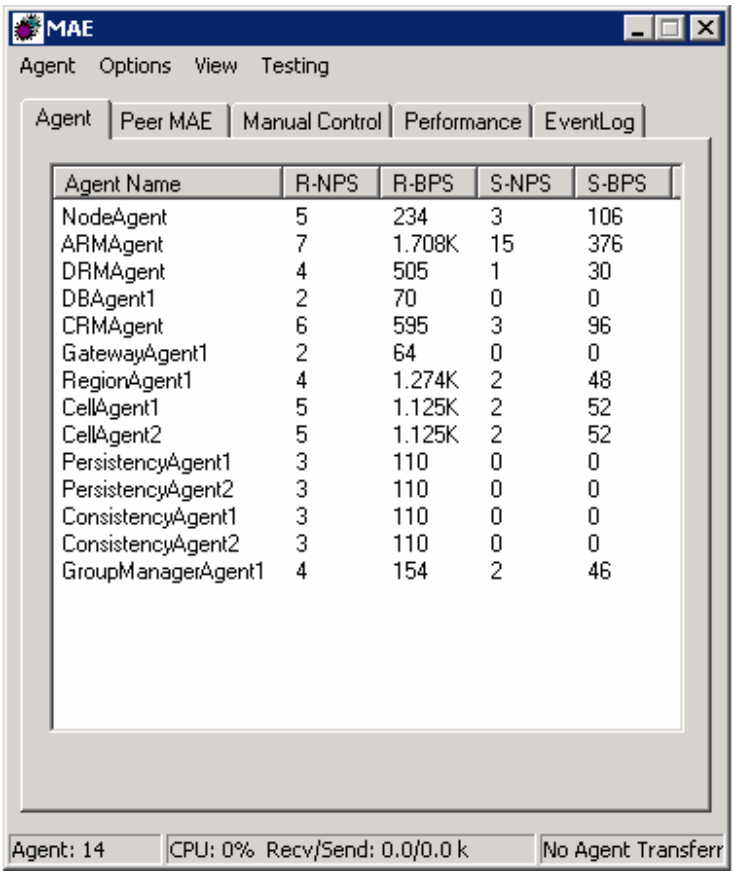

Figure 4-3: Screenshot of MAE

In the screen shot, the Agent page lists all the agents that run on the MAE. R-NPS is the number of the received messages per second for each agent; R-BPS is the number of bytes of the received messages per second for each agent; S-NPS is the number of the sent messages per second for each agent; S-BPS is the number of bytes of the sent messages per second for each agent. Peer MAE page list all the remote MAE which communicate with local MAE. Manual Control page provides the function to manually send agent operation commands to MAE. Performance page shows the performance parameters of the node. EventLog page presents the log of executed agent operations on this MAE.

4.1.2 MACVE APIs

As we cannot assume that all the SC Nodes have identical architectures or even run the same operating system, agents must be programmed in a widely available, platform independent language. The agents in MACVE are implemented in Java, since Java is platform-independent language[128]. The security and multithread programming model in Java also facilitate the development of our prototype system.

To facilitate the agent development, MACVE provides a set of Java APIs for programmer to create mobile agents. We call them MACVE APIs. It contains abstract classes and interfaces for initializing an agent, migrating and cloning agent, forwarding and processing agent messages, and disposing of agents.

MACVE APIs provide the interface and basic functions for agent implementation. Every agent class is derived for abstract class Agent. Besides class *Agent*, there are other classes providing the basic function for agent implementation. Class *AgentID* provides naming service for all agents; class *AgentSub* provides the functions to communicate with and control remote agents; class *AgentMsg* provides the method to deal with the agent messages; and class *RecoveryManager* builds in the basic mechanism for agent failure recovery, etc.

- Agent class

The Agent class is the key class in the MACVE APIs. It is an abstract class that all other agents derive from it. The Agent class defines basic methods for controlling its own execution, which are listed in Table 4-1.

Table 4-1 Main methods in class Agent

```
public abstract class Agent implements Serializable
{
        public void init();
        public void start();
        protected void run();
```

```
        protected void dispose();
        protected void dispatchMsg (AgentMsg msg);
        protected void migrate();
        protected void clone();
        protected void extendDispatchMsg (AgentMsg msg);
        protected byte[] serialize();
};
```

The *init* method reads the agent configuration file and initializes the agent before it begins to run. The *start* method triggers the agent to begin work. The *run* method is the entry point of the agent's own controlling thread. The *dispatchMsg* method dispatches the receiving agent messages to each individual message processing method. The *extendDispatchMsg* method is the method for the child agent class to override to dispatch messages to its own message processing method. The *migrate* method invokes an agent to move a remote SC Node. The *clone* method spawns a new instance of the agent itself locally or remotely. The *dispose* method stops the execution of the agent and disposes itself from MAE. The *serialize* method capture the agent running state and store them in secondary storage.

- AgentID class

As a large number of agent instances running in the mobile agent system, agents are needed to be assigned names which can uniquely identify them. In MACVE, we assign each agent an Agent ID. Agent ID is a 16 bytes number which contains 5 sub-ID as shown in Table 4-2:

Table 4-2 Agent ID structure

Agent Type ID	Agent Version ID	Agent Instance ID	Agent Clone ID	Checkout ID
1 2	4	8	12	16

(1) Agent Type ID: a 2 bytes integer number to identify the type of the agent. For example the type ID of ARM Agent is 1. A type of agent corresponds to a kind of service.

(2) Agent Version ID: a 2 bytes integer number to identify the version of the agent code. The same type of agent may have different versions corresponding to different inner algorithm and realizations. For example, according to CVE consistency requirement, some cells need aura based AOI algorithm while others need broadcast based AOI algorithm. We use the version ID to distinguish the two kinds of code for Consistency Agents.

(3) Agent Instance ID: a 4 bytes integer number to identify the instance of the same type of agent who belongs to a sub-area of CVE. For example Cell ID is used as the instance ID for Cell Agent to identify the Cell Agent instance working for which cell.

(4) Agent Clone ID: a 4 bytes integer number to identify the cloned agent instances which provide mirroring services to work for a same area of CVE. For example Cell Agent may have several cloned ones to provide the scene data delivery service for the same cell. We use Clone ID to identify the cloned Cell Agents.

(5) Checkout ID: a 4 bytes integer number to verify the agent code integrity. According to the checkout ID, MAE can verify if the local cached agent code or the receiving agent code is the same as the code in the agent code base.

To make the agent ID understandable for system developer and administrator, each Agent ID has name string to represent the agent ID. For example, a Cell Agent ID string is

0008-0003-00000001-00000000-06D66A0C

The numbers separated by the dash are each sub-IDs of the agent ID. The numbers are in hex form. 0008 is the agent type ID; 0003 is the agent version ID; 00000001 is the agent instance ID; 00000000 is the agent clone ID, 06D66A0C is the checkout ID.

The AgentID class defines basic methods for generating, processing agent ID which are listed in Table 4-3.

Table 4-3 Main methods in class AgentID

```
public class AgentID
{
        public String getAgentName();
        public short getAgentTypeID();
        public short getAgentVersionID();
        public int getAgentInstanceID();
        public int getAgentCloneID();
        public int getAgentCheckoutID();
        public void setAgentTypeID();
        public void setAgentVersionID();
        public void setAgentInstanceID();
        public void setAgentCloneID();
        public void setAgentCheckoutID();
        public String getAgentCodeFileName();
};
```

The *getAgentName* method generates a unique name string from the agent ID, which make the agent ID readable to both machine and people. The *getAgentCodeFileName* generate agent code file name for the agent. MAE uses this function to identify the agent code to load. The *getAgentTypeID*, *getAgentVersionID*, *getAgentInstanceID*, *getAgentCloneID*, and *getAgentCheckoutID* method exact the different ID information from the agent ID. The *setAgentTypeID*, *setAgentVersionID*, *setAgentInstanceID*, *setAgentCloneID*, and *setAgentCheckoutID* method write the different ID information to the agent ID.

- AgentStub class

An agent stub is a representative of a remote agent. The AgentStub class defines basic methods to communicate and control the remote agent, which are listed in Table 4-4. an agent stub acts as a handle of a remote agent and provides common way to access the agent. Any agent that needs to communicate with other agents need to create agent stubs, and then communicate with them through the stubs. AgentStub only provide the function for basic remote control. Agents with special communication and control requirements can create its own type of stub by deriving from AgentStub class.

Table 4-4 Main method in class AgentStub

```
public class AgentStub
{
        public AgentID getAgentID ();
        public byte getAgentState ();
        public void clone ();
        public void migrate ();
        public void create ();
        public void dispose ();
};
```

The *getAgentID* method gets the ID of the remote agent. The *getAgentState* gets the state of the remote agent. The *create* method launches an agent at the designated SC Node. The *dispose* method stops the remote agent execution and removes it from system. The *clone* method requests the remote agent clone a new instance. The *migrate* method requests the remote agent move to a new location.

- AgentMsg class

Agents in MACVE communicate with each other by sending and receiving agent messages. Agent message is composed of two parts: message header and message data. The detailed format of agent messages is introduced in Subsection 4.1.3.

The AgentMsg class defines basic methods for processing the message header and packs and extracts raw data from message data part. The basic methods in AgentMsg class are listed in Table 4-5.

Table 4-5 Main methods in class AgentMsg

```
public class AgentMsg
{
        public short getProtocolVer ();
        public short getMsgType();
        public int getMsgLength();
        public char readChar();
        public short readShort();
        public int readInt();
        public float readFloat();
        public double readDouble();
        public String readString();
        public void read (byte[] dst);
        public void writeChar(char value);
        public void writeShort(short value);
        public void writeInt(int value);
        public void writeFloat(float value);
        public void writeDouble(double value);
        public void writeString(String str);
        public write(byte[] src);
};
```

The *getProtocolVer* method gets the protocol ID of the message. The *getMsgType* method gets the message type ID from the message header. The *getMsgLength* gets the message length from the message header. The *readChar, readShort, readShort, readInt, readFloat, readDouble, readString, read* methods provide functions to pack basic type of

data into message buffer. The *writeChar*, *writeShort*, *writeInt*, *writeFloat*, *writeDouble*, *writeString, write* methods provide functions to extract get basic type of data from the message buffer.

4.1.3 MACVE Protocol

MACVE Protocol is an application level communication protocol. It defines how agents and user application communicate with each other and what is format of the messages. As MACVE need to be extensible and allow new type of mobile agents to be added in the future, MACVE Protocol also provide the standard for new agents to communicate with the system.

MACVE support both synchronous message and asynchronous message. According to the sender and receiver of the messages, the messages in MACVE can be grouped into four categories: user-to-agent message, agent-to-agent control message, agent-to-MAE control message, and agent persistency message.

(1) User-Agent message

User-Agent message convey the data between user application and agents. The type of agents that user applications communicate with includes Gateway Agent, Cell Agent, and Consistency Agent. The communication data include user registration/ login data, VE scene data, VE state data, and VE object state data.

(2) Agent-Agent message

Agent-Agent messages are sent and received among agents. It refers to all the messages which deal with agent operation, such as agent location registration agent migration/clone negotiation, agent running state monitoring and recovery, etc. Agent-Agent messages are distributed by MAEs, as introduced in Subsection 4.1.1.

(3) Agent-MAE message

Agent-MAE message refer to the agent operation commands sent form agent to its MAE. These messages include agent commands to launch agent, transfer agent code,

transfer agent state, kill running agent, request performance information of node and agents, etc.

(4) Agent-DB message

Agent-DB messages are sent between relevant agents and DB Agents. They are used to save and retrieve the persistency information from the database system. Agent persistency messages include directory of user information, directory of running agent information, directory of SC Node information, VE scene data, VE object states, etc.

All the above messages are in binary format which is efficient in communication. As shown in Table 4-6, agent message includes two parts: message header and message data. Message head includes three fields: protocol version, PDU type and PDU length.

Table 4-6 Agent message format

Protocol Version (1)	PDU Type (1)	PDU Length (2)
PDU Data		

Since the above 4 categories of messages have different sender and receiver, the field of PDU type have their own defining space which means they can have the same PDU type in each category of the messages to represent different messages.

As discussed in Section 3.8, MACVE supports both client server communication architecture and peer to peer architecture. Thus, MACVE protocol needs to support both unicast and multicast. Unfortunately, multicast-capable network interface and routers are not yet universally deployed in today's Internet. To multicast packets across such a deficient network, soft multicast protocol is designed on top of the underlying transport protocol UDP and TCP. Each multicast group corresponds to a multicast repeater to maintain the list of multicast subscribers and disseminate packets among the subscribers. All the multicast members subscribe to multicast repeater and then forward their packets

to multicast repeater. Multicast repeater is a simple and efficient software program which is only required to ensure forwarding of packets to all the other subscribers as the multicast routers do. We implement our soft multicast repeater as shown in Figure 4-4.

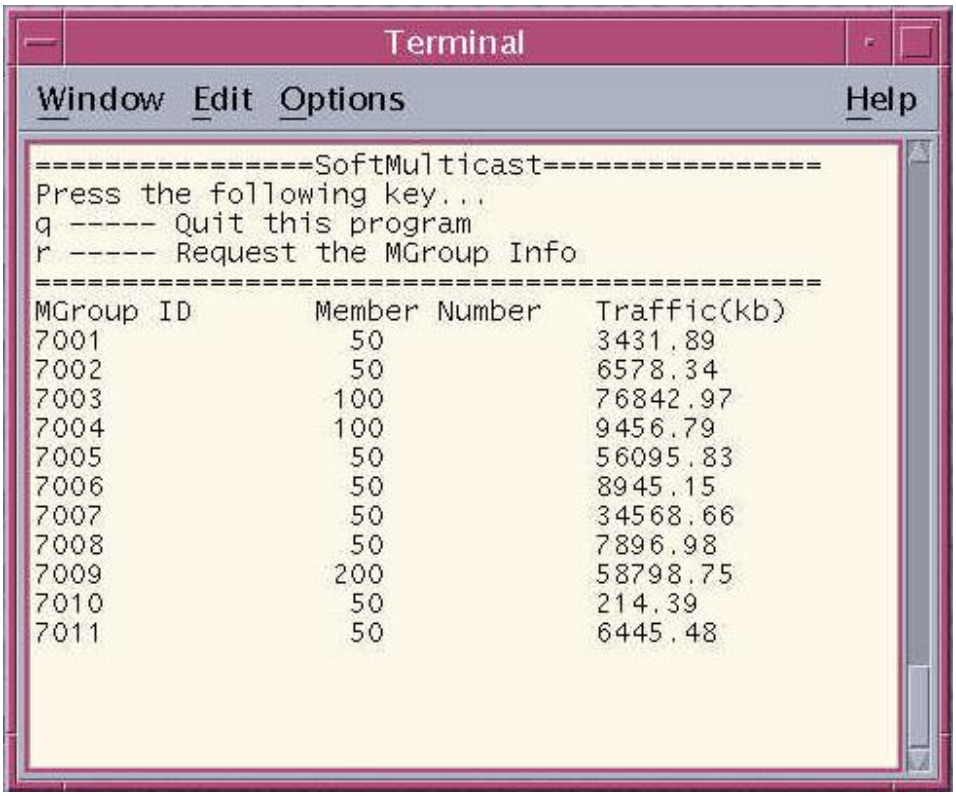

Figure 4-4 Soft multicast repeater

MGroup ID represent different multicast group. Member Number shows how many members subscribe in this multicast group; Traffic calculates the bytes received by the multicast group.

4.2 Agent Mobility Implementation

VEMAS supports agent weak mobility. During agent migration/remote cloning, the mobile agent system does not capture the thread-level execution state of the agent. It captures agent execution state at a higher level, in terms of application-defined agent state. Since agent in MACVE is autonomous, the migration of agents is under its own explicit control so that state capture at thread-level is not necessary.

The first issue in implementing agent mobility is the transfer of agent code. In one approach, agent code is transferred with the agent at every time. This approach causes bandwidth wastage when multiple same types of agents need to migrate to a remote MAE. It is not necessary to transfer the same piece of agent code multiple times. Another approach is that systems do not transfer any code at all and require that the agent's code be preinstalled on the destination server. Although this approach can save the bandwidth for the agent code transferring, it is not flexible for agent code updating or system extension. For examples, when agent code needs to be updated, all the MAE need reinstall the code. When a new type of agent is added into MACVE, the new type of agent code also needs to be installed at all the MAEs. Therefore, this approach is hard to guarantee the agent code consistency in large mobile agent system. In MACVE, we combine the two approaches. When a MAE receives a piece of agent code, it caches the agent code for the possible future use so that the agent code need not be transferred for each time. The mechanism of invalidation checking of the cached agent code guarantees the consistency of agent code in the system.

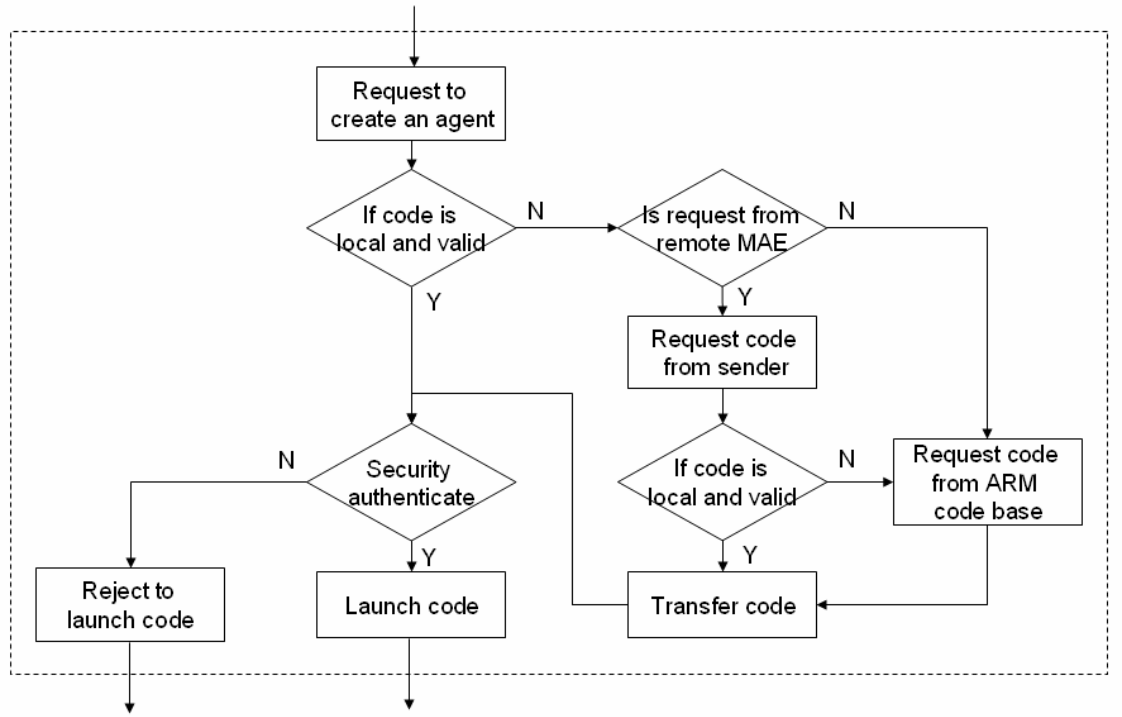

Figure 4-5 Agent code transferring

In our approach, when a MAE receives the primitive commands to create an agent, it initiates the code transferring process. The control flow of agent code transferring can be illustrated in Figure 4-5.

When a MAE receives the command to create an agent, it firstly checks whether the agent code is local and valid. The Checkout ID in the Agent ID is used to check the cached agent code is integrity or not. If the agent code is local and valid, it creates the agent instance immediately. If the agent code is not local or not valid, MAE checks whether the agent operation command is from a remote MAE. If it is from a remote MAE, MAE requests agent code from that remote MAE. If the command is from the local agents, the MAE will request the agent code from agent code base. Agent code base is the location of the ARM Agent. ARM is the agent code repository which manages the all agent codes. To avoid the potential bottleneck, ARM Agent can have multiple cloned instances running at different Controlling Nodes to provide the mirroring service. So MAE request the agent code from one of the ARM Agents. Similarly, if remote MAE doesn't have the agent code, the MAE will also request the code from ARM Agent.

Agent code transferring is a process common to agent migration and agent remote cloning. In the following discussion of agent migration and agent remote cloning implementation, we treat the agent code transferring as a single process and do not distinguish whether the agent code is really transferred.

4.2.1 Agent Migration Process

Agent migration process is to transfer an agent with its run-time state from one host to another. The agent migration process in most of the traditional mobile agent systems is a synchronous process. We discuss the agent migration process in Aglet[114] as an example to explain the synchronous agent migration. The agent migration process in Aglet can be illustrated by Figure 4-6.

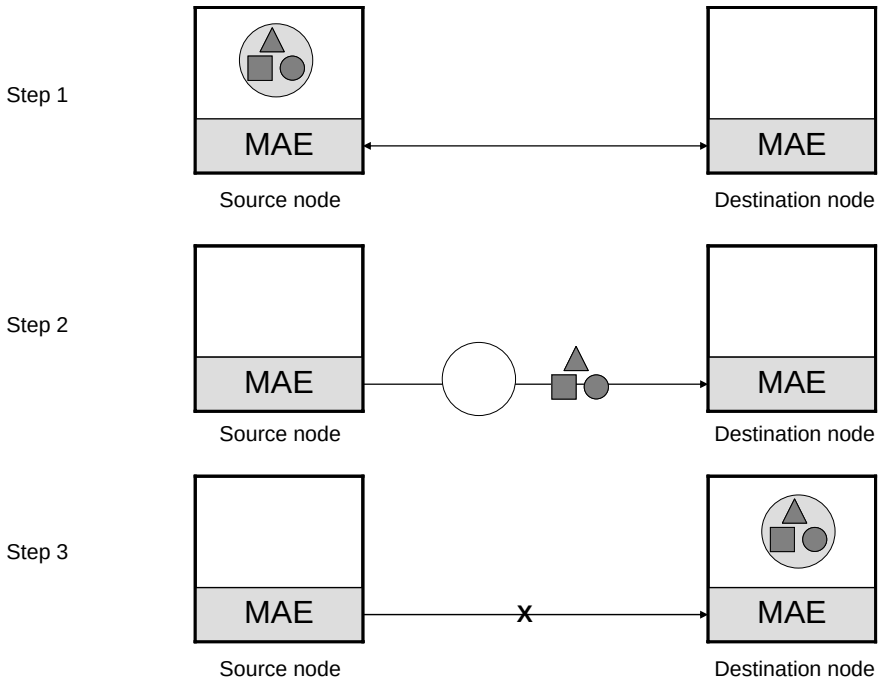

Figure 4-6 Agent migration process in Aglet

Step 1:

An agent migration request is issued from the MAE on the source node to a remote MAE on the destination node. After negotiation, migration request has been accepted by the remote MAE.

Step 2:

The migrating agent stops to provide its service. The agent state has been captured and serialized. Then, Agent stops running. MAE sends agent code and agent state to the remote MAE.

Step 3:

MAE at the destination node authenticates the agent code. After the success of authentication, the MAE launches the agent. This agent is called the new agent. Agent state is imported into the new agent. The new agent begins to work.

During the course of the agent migration, the agent service suspends for a period of time. The suspend time can be represent by Equation 4-1,

We define:

T_s = agent service suspended time;

T_{ss} = agent state serialization time;

T_{ct} = agent code transferring time;

T_{st} = agent state transferring time;

T_{ta} = the remote MAE security authentication time;

T_{la} = Agent launching time;

T_{ia} = import agent state time.

Then,

$$T_s = T_{ss} + T_{ct} + T_{st} + T_{sa} + T_{la} + T_{ia} \quad (4\text{-}1)$$

As the agent code transferring time T_{ct} and agent state transferring time T_{st} are dependent on the size of transferred data and the network traffic situation, they are non-determinate. From the experiment results in [115], other time is relative small or stable and neglectable. We neglect other times effects on the changes of T_s. Therefore, T_s will increase when the size of agent code or state increases. In MACVE, we need the agent migration process does not affect user real-time interaction in the CVE. However, in [115], when agent's size reaches 500 kbytes, the migration time is near 6 seconds. Such long time suspension of agent service in Aglet can not satisfy our framework's requirements.

In order to decrease the agent service suspended time during agent migration, we adopt an asynchronous agent migration mechanism, which is different from that of the traditional mobile agent systems. In asynchronous agent migration, the agent does not suspend its service during the whole process of agent migration. The detailed agent migration process in MACVE is illustrated by Figure 4-7.

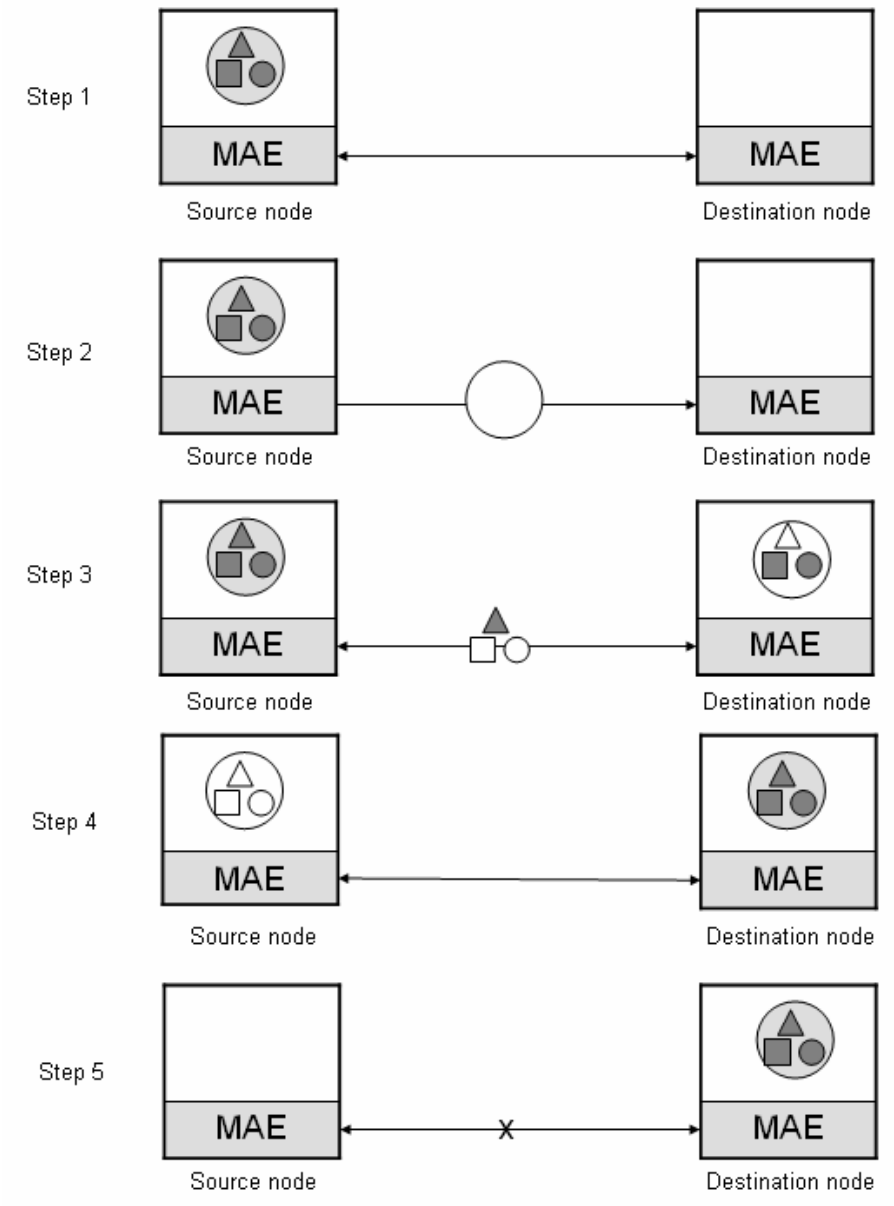

Figure 4-7 Agent migration process in MACVE

Step 1:

An agent migration request is issued from the MAE on the source node to a remote MAE on the destination node. After negotiation, migration request has been accepted by the remote MAE.

Step 2:

The agent on the source node still runs and holds the control to provide its service. This agent is called the old agent. At the same time, the agent code has been sent to the destination MAE concurrently.

Step 3:

MAE at the destination node authenticates agent code. After the success of authentication, the MAE launches the agent. This agent is called the new agent. The old agent still holds the control of its service and at the same time, it synchronizes the dynamic changing agent state with the new agent.

Step 4:

After the agent state synchronization, the old agent stops providing its service and it informs the new agent to takeover the service control. The new agent begins to hold the control to provide the agent service.

Step 5:

After the success of the handover of the agent control, the new agent informs the old agent to stop running. The old agent terminates to work and disposes.

During the process of the asynchronous agent migration, the agent service suspend time can be represent by the Equation 4-2,

We define:

T_{ch} = agent control handover time;

Then,

$$T_s = T_{ch} \tag{4-2}$$

As the old agent and the new agent run concurrently to make the time consuming steps execute in background, the agent service suspended time decreases. Moreover, the agent service suspended time tends to be a stable value theoretically, because the message size for handing over the agent control from the old agent to new agent is independent of the size of agent code and agent state. We conduct an experiment in Section 5.2 to study the overhead of agent migration.

4.2.2 Agent Remote Cloning Process

Agent remote cloning process is very similar with agent migrating process. The difference is when a new agent is created at a remote MAE, the old agent do not dispose, the old agent and the new agent work together to provide the mirroring service.

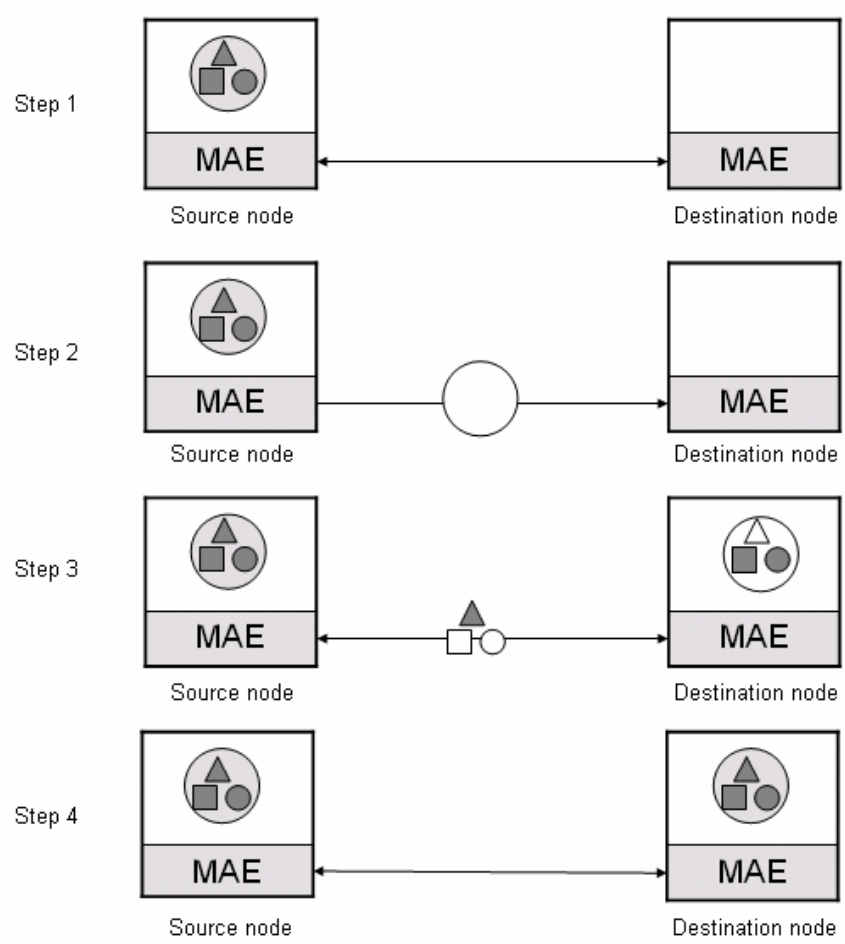

Figure 4-8 Agent cloning process in MACVE

Step 1:

An agent remote cloning request is issued from the MAE on the source node to a remote MAE on the destination node. After negotiation, agent remote cloning request has been accepted by the remote MAE.

Step 2:

The agent on the source node still runs and holds the control to provide its service. This agent is called the master agent. At the same time, the agent code has been sent to the destination MAE concurrently.

Step 3:

MAE at the destination node authenticates the agent code. After the success of authentication, the MAE launches the agent. This agent is called the cloned agent. The master agent synchronizes the dynamic changing agent state with the cloned agent. The master agent still holds the control of its service at this step.

Step 4:

After the agent state synchronization, the master agent informs the cloned agent to begin to provide the mirroring service. The master agent has a scheduler to allocate workloads to the cloned agent.

After the agent remote cloning, master agent and the cloning agent needs to exchange their own updates information with each other to guarantee the consistency of their agent state.

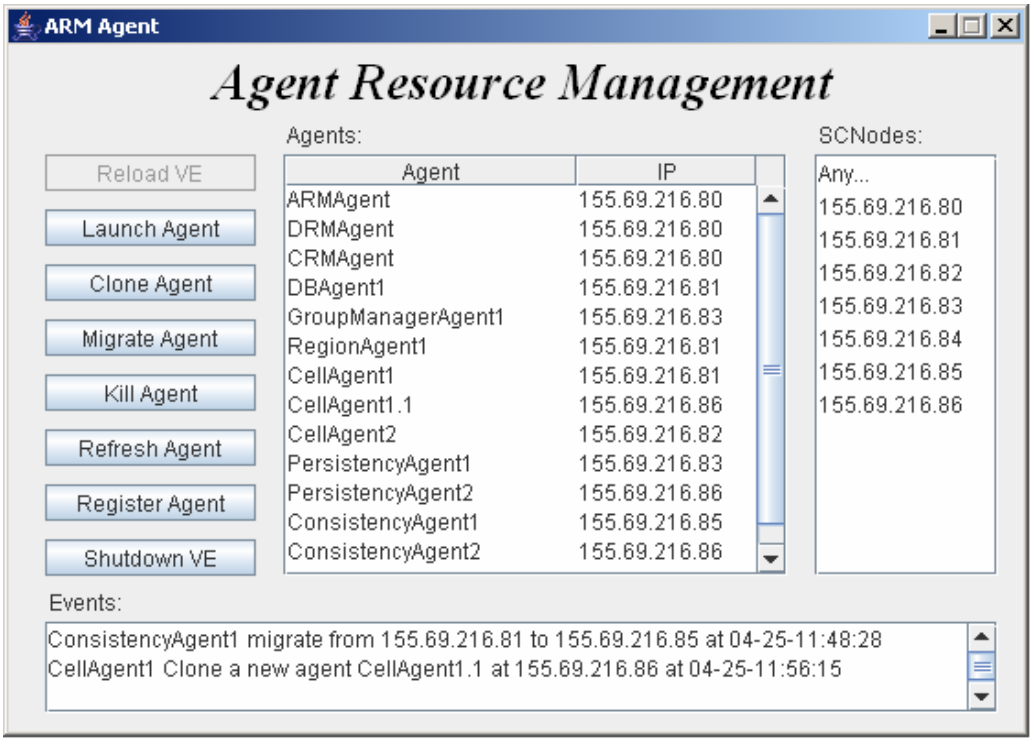

Figure 4-9 Screenshot of ARM Agent

As shown in Figure 4-9, we use a screenshot of ARM Agent to illustrate the event of agent migration and agent remote cloning in MACVE. During the execution of MACVE system, two mobile actions are automatically performed:

(1) ConsistencyAgent1 migrates from a node with IP of 155.69.216.81 to another node with IP of 155.69.216.85;

(2) CellAgent1 remote clones a new instance CellAgent1.1 on the node with IP of 155.69.216.86.

4.3 User Application Implementation

To allow users to enter and interact within CVE without any complex software pre-installation and configuration, we implement a web-based user application. The web-based user application takes the form of java applet and follows VRML standard to construct the 3D scene of the CVE. The applet and the VRML scene are embedded in the same web page. User only needs to install a VRML browser plug-in and then use a common web browser to enter the CVE from a URL (Uniform Resource Locator).

As introduced in Section 3.6, user application composes of three key modules: User Communication Manager, CVE Synthesizer, and CVE User Interface. The applet implements the User Communication Manager and CVE Synthesizer. The VRML browser works as the CVE User Interface. The VRML browser captures local users' interaction. It needs to send the user interaction events to the applet for processing. On the other hand, the applet receives the CVE scene data, state data and remote users' interaction message from the network. After processing the data, it needs to inform the VRML browser to render the updated 3D scene. Therefore, an interface is needed to communicate between the VRML browser and the Java applet.

VRML's External Authoring Interface (EAI)[129] can take charge of this task of communicating between the VRML world and its external environment such as java applet and java application . In essence, EAI provides a set of methods for developing custom applications that interact with, and dynamically update a 3D scene, so that the

outside applications can "talk" to the VRML scene in real time. As a result, EAI can be considered as a Java application programming interface. It defines a set of Java packages and classes for bridging Java applications to the VRML world.

The use of EAI in the applet is shown in Figure 4-10. EAI allows a two-way communication between the two components — the Java applet and the VRML web browser plug-in. When a user logs in, the applet adds the VE scene data, virtual entities and local/remote avatars into the existing 3D VRML world, and begins to communicate with the agents to maintain the consistency of the VE.

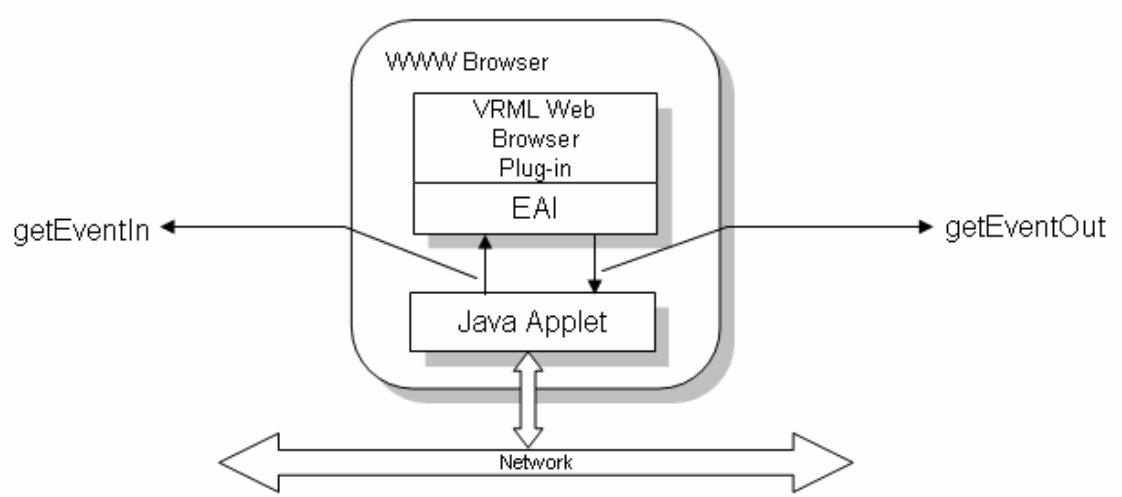

Figure 4-10 EAI in user application

EAI provides the Java applet with an interface to access the nodes in the VRML scene using the existing VRML event model. In this model, an eventOut of a given node can be routed to an eventIn of another node. To receive notification when an eventOut is generated from the scene, the applet implements the *EventOutObserver* class defined in EAI, and programs the *callback* method in this class. The *getEventOut* method is used to access the eventOut from the VRML node. The value of the eventOut is passed to the *callback* method and sent out to the Consistency Agent to provide information about the interactions occurring in the VE. The *getEventIn* method is used to trigger the animations in the user's local VE, through the Java applet.

Figure 4-11 is the screen shot of our web-based user application. Users in the CVE can navigate in the VE, can add, modify, delete entities and interact with entities, and also can communicate with other users through textual chat messages.

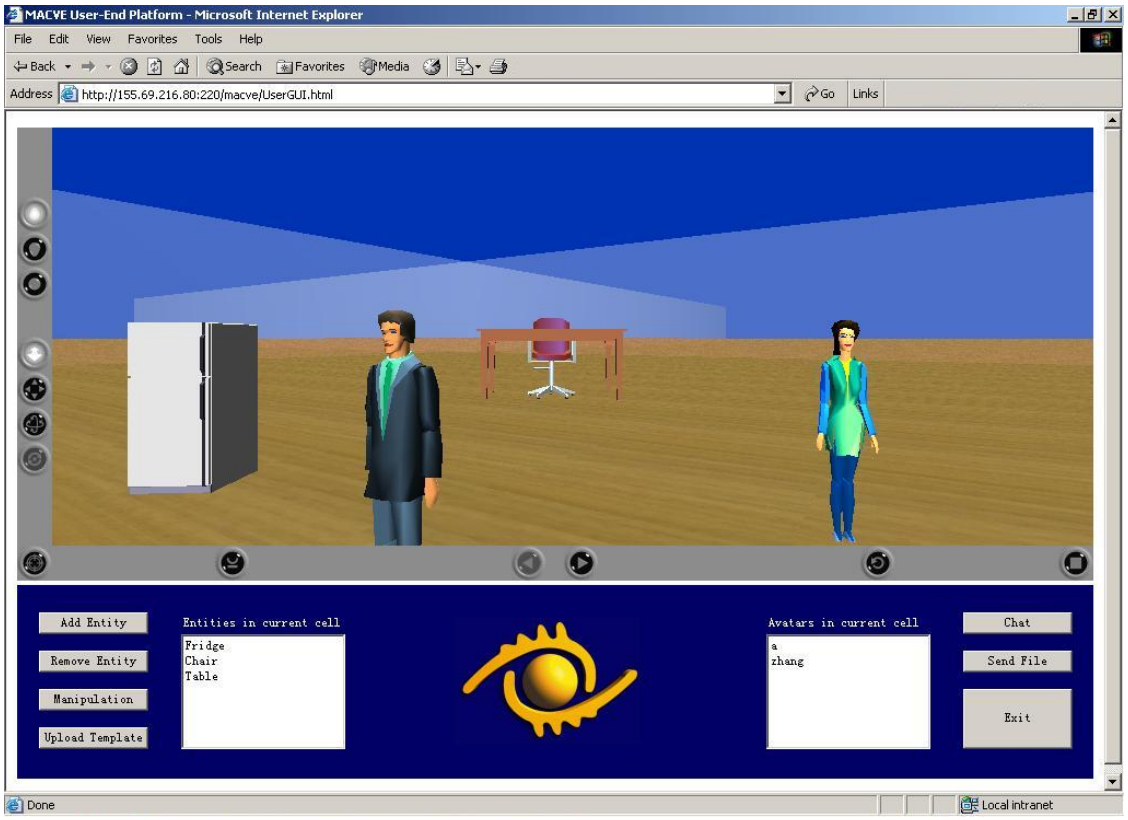

Figure 4-11 Screenshot of web-based user application

4.4 Summary

This chapter presents the implementation details of MACVE prototype system. Our implementation includes a mobile agent system -- VEMAS, eleven basic types of agent with mobility capability on VEMAS and a web-based user application to navigate and interact in the CVE. In the next chapter, we conduct experiments in our prototype system to evaluate the proposed framework on the scalability of CVE system.

Chapter 5

Experimentation and Evaluation

In this chapter, we design and conduct experiments on a test bed CVE scenario, a virtual playground, to evaluate MACVE. We measure the overhead of agent migration to examine the feasibility of agent migration as an approach to dynamically distribute system workload. We compare the performance of LCVE system with and without agent migration and agent remote cloning to investigate the effect of agent mobile action on the system scalability. We measure the workload of a consistency agent to investigate the effects of adaptive consistency control on the system scalability. Finally, we make a discussion of MACVE system and existing LCVE systems.

5.1 Introduction of the Experiments

To evaluate our proposed MACVE, we design a virtual world to conduct different experiments on it. The theme of the virtual world is a virtual playground. In the virtual playground, there are multiple types of virtual entities. Users can walk in the virtual playground, add/delete virtual entities, collaboratively interact with the virtual entities, and communicate with each other by text message. The playground is divided into eleven cells, which belongs to two regions. The boundaries of cells are transparent to users so that users will not feel the partition of the whole playground. By system configuration, we can add arbitrary number of interactive virtual entities which randomly located in the playground.

In order to simulate a large number of concurrent users in our evaluation experiments, we design an experiment to study interaction data generated by real users. In the experiment, we collected real interaction data with 10 simultaneous users during 30 minutes. The

client program recorded the number of the interactions occurred when the user navigated around and interacted with the interactive entities or other users in the CVE. After analyzing the data collected from the interaction among users in the virtual playground scenario, it was found that the users' total interaction messages increased at an almost constant rate. Thus, it is reasonable to use the statistic maximum message rate and minimum message rate, as the seeds to randomly generate the simulated user interaction messages, for the artificial users in the simulation of a CVE. Figure 5-1 shows the real data collected from 10 concurrent users' interaction in the sample CVE.

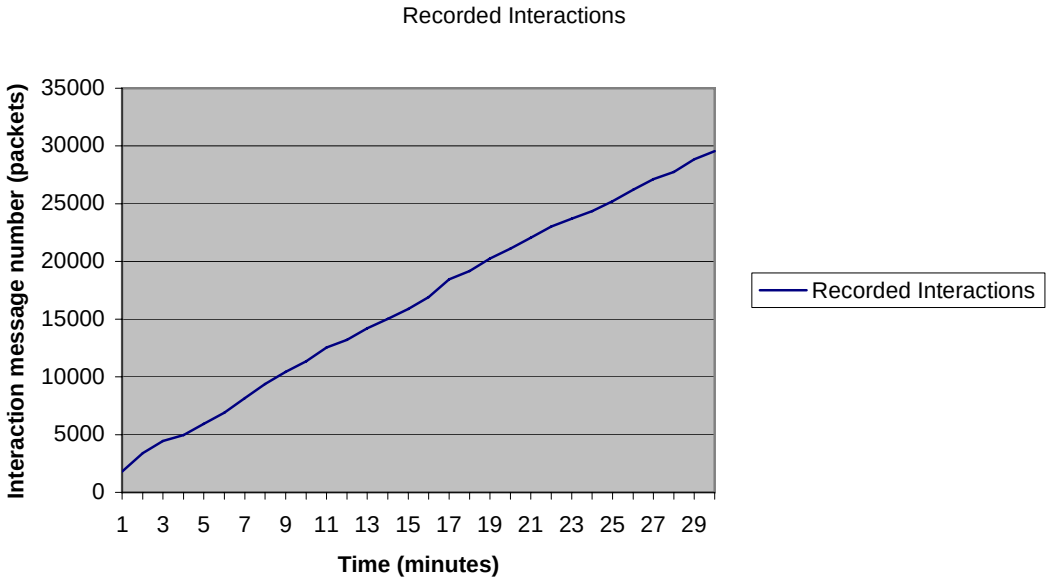

Figure 5-1 The total interaction message recorded in the experiment

From the data collected, we got the statistic maximum message rate and the minimum message rate:

$$f_{max} = 2.3 \text{ packets per second}; \quad f_{min} = 0.65 \text{ packets per second};$$

In the simulation, the artificial users' interaction message rates are generated according to the f_{max} and f_{min}.

We implement a RobotGroup Agent to simulate a group of concurrent users, whose interaction message rates randomly distribute between f_{max} and f_{min}.

As shown in Figure 5-2, we use 50 RobotGroup Agents running at different machines to simulate 990 artificial concurrent users and we use another 10 machines to execute the real user application to log on the virtual playground so that we can simulate the scenario of large number of concurrent users

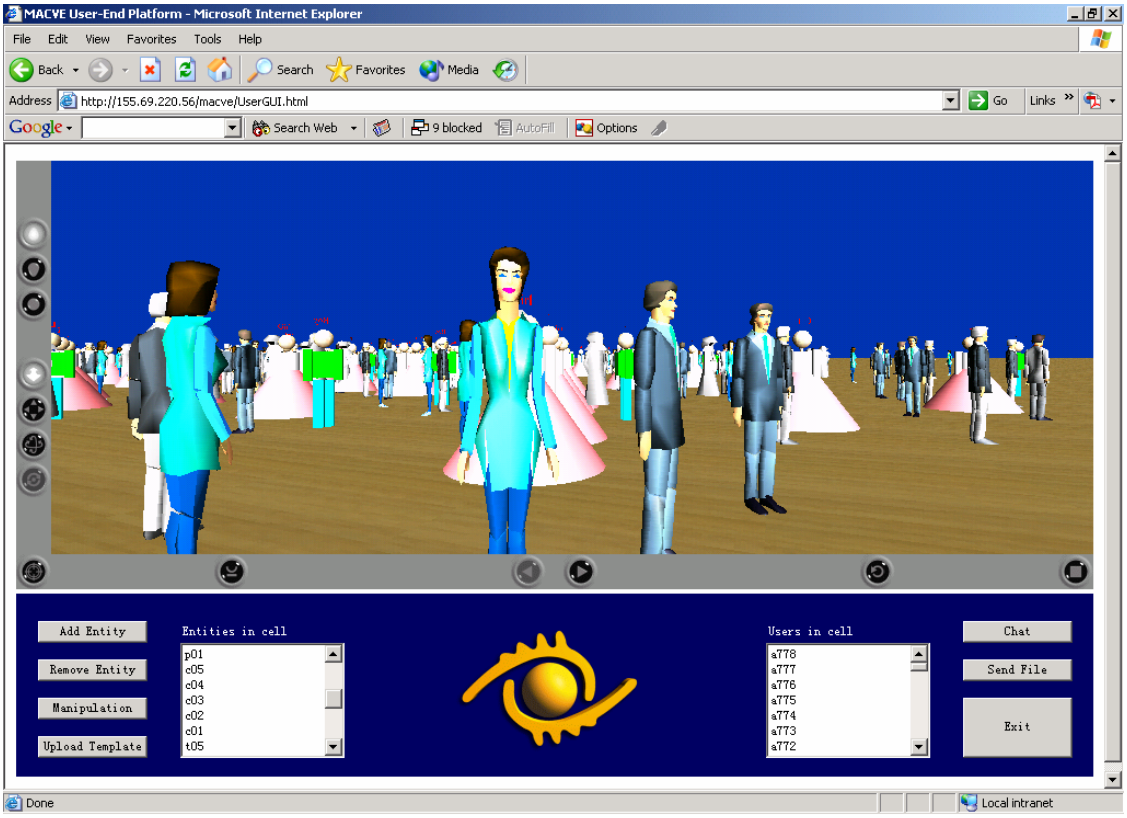

Figure 5-2 Screenshot of large number of concurrent users

As shown in Figure 5-3, we add 3000 virtual entities in the playground. Each entity has multiple interactive behaviors and attributes. Users can interact with the entities collaboratively. The added entities are randomly located within the virtual playground. We can define the types of virtual entities, the density of the virtual entities, and the scope of their deployment by system configuration, so that we can simulate the scenario of large number of virtual entities.

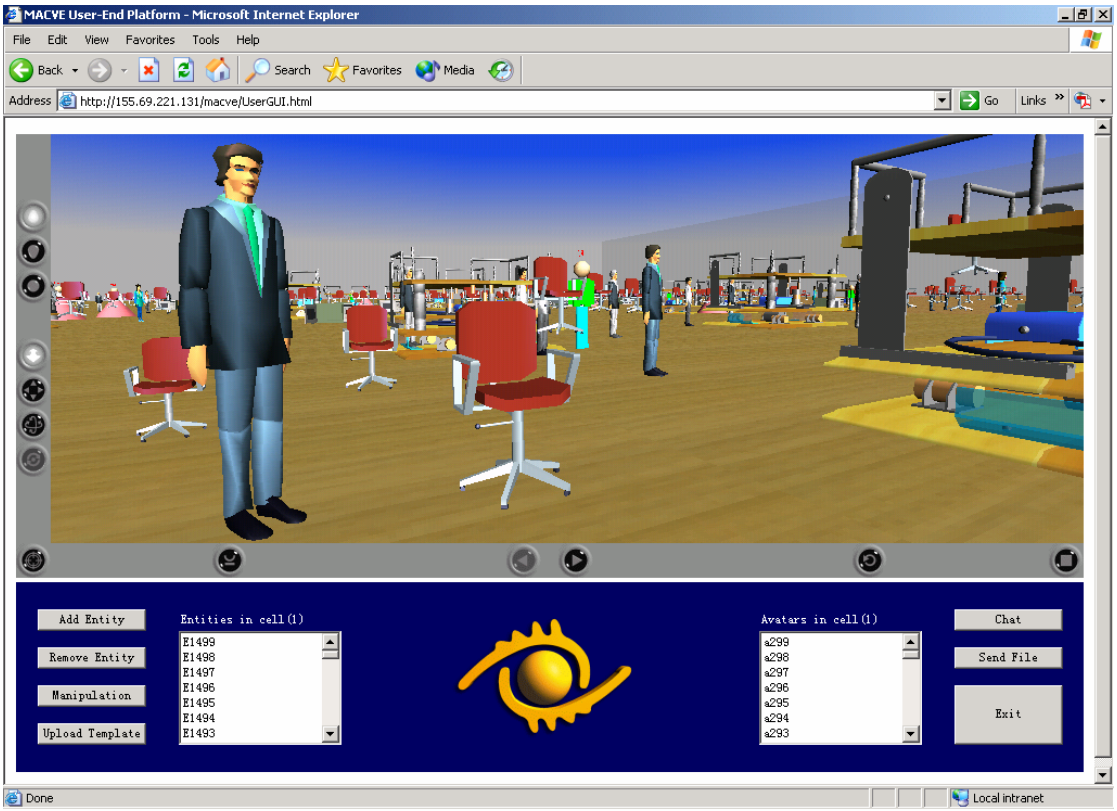

Figure 5-3 Screenshot of large number of virtual entities

In the following sections, we conduct experiments based on the virtual playground to measure the overhead of agent migration, to evaluate the feasibility of agent migration on the system scalability, to evaluate the feasibility of agent remote cloning on the system scalability, and to evaluate the effects of adaptive consistency control on LCVE scalability.

5.2 Agent Migration Overhead

Before evaluating the effectiveness of agent migration on the scalability of MACVE, we design experiment to measure the overhead of agent migration. From the result of the experiment, we can examine how much impact agent migration will have on the CVE real-time performance, and whether agent migration can be adopted in LCVE systems.

Instead of measuring the migration overhead of all agents in MACVE, we only conduct experiments to study the overhead of Consistency Agent migration. The reason is that

Consistency Agent is the most critical agents to the CVE real-time performance. The user messages which need consistency control pass through the corresponding Consistency Agent. The migration of Consistency Agent has the most remarkable impacts to the CVE real-time interactions. Moreover, during the Consistency Agent migration, real-time VE state of the cell need to be synchronized, Consistency Agent migration also has the most complexity, which will bring further impacts on the users' interaction within CVE. Therefore, from the study of the migration overhead of Consistency Agent, we can examine the maximum impact of agent migration to the CVE system.

As introduced in Subsection 4.2.1, the process of Consistency Agent migration can be described as three major steps: (1) transferring the agent code to the destination; (2) synchronizing the agent state; (3) shifting the agent control. Transfer & Synchronization Time measures the time required to transfer the agent code and synchronize the agent state. The Transfer & Synchronization process will not directly affect users' collaborative interaction in the CVE as it is done in a separate thread in the background. Handover Time measures the time required to shift agent control from the old Consistency Agent in the source node to the new one in the destination node. Handover process will cause a short delay on the messages processing at the agent, which may affect the users' collaborative interaction in the CVE. Thus it is particularly important to evaluate the delay caused by the Handover Time.

We design experiments to measure the Transfer & Synchronization Time and Handover Time during the Consistency Agent migration under different concurrent user number and interactive entity number in a cell.

We firstly conduct the experiment to measure the Transfer & Synchronization Time and Handover Time when the concurrent user number in a cell increases. As shown in Figure 5-4, we observe from the result of this experiment that the Transfer & Synchronization Time increases as the number of concurrent users grows. It is because when user number increases, the transmitted data volume of agent state also increases. To transfer large volume of agent state cost more time. However, the Handover Time is relatively stable which is 425.0 ms when the concurrent user number reaches 1000 in a cell.

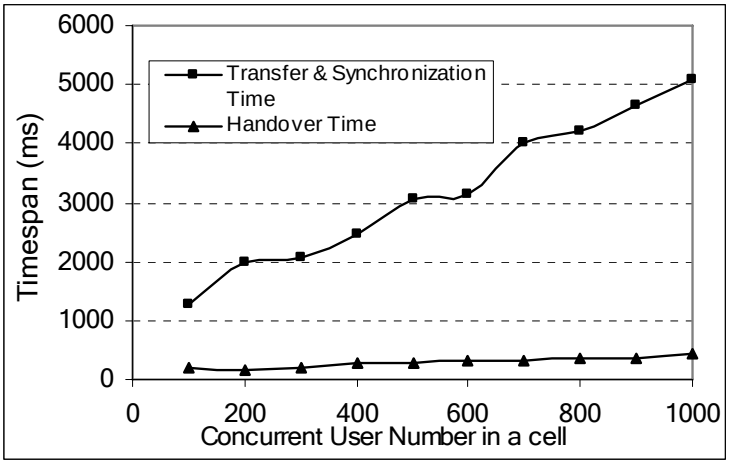

Figure 5-4: Consistency Agent migration time vs. concurrent user number

We conduct another experiment to measure the Transfer & Synchronization Time and Handover Time when the interactive entity number in a cell increases. As shown in Figure 5-5, we observe that the Transfer & Synchronization Time increases with the increase of the number of entities. It is because when entity number increases, the transmitted data volume of agent state also increases. To transfer large volume of agent state cost more time. However, the Handover Time is relatively stable which is 344.4 ms when the number of entities in the cell reaches 3000.

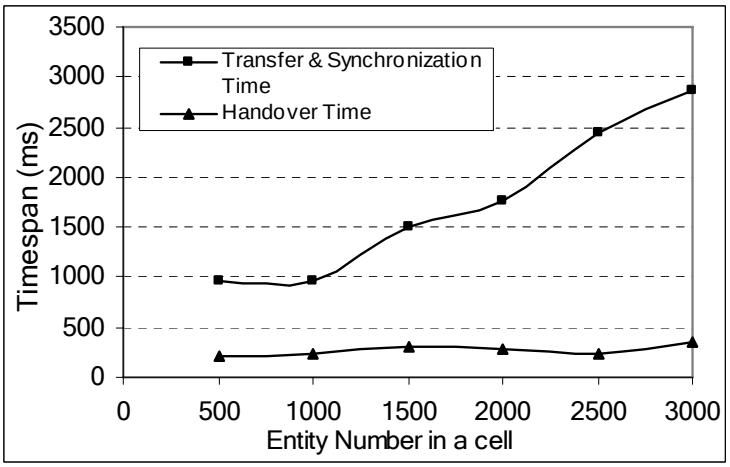

Figure 5-5: Consistency Agent migration time vs. entity number

From the above two experiments, we found the Handover Time is small and stable comparing with the Transfer & Synchronization Time. A temporary short delay of less than half a second in updating scene state caused by the Handover Time will have little impact on the performance of a CVE with a large number of concurrent users and virtual entities. We also found from the experiment that the users would not feel the migration of the Consistency Agent. This is because the user interactions in LCVE are not affected during the agent transfer and synchronization period while the agent control Handover Time is short enough to avoid apparent interruption.

Our experiments show that Consistency Agent migration does not affect the real-time interaction of the CVE users. Since the migration process of other agent is not as complex as Consistency Agent, and the services provided by other agents have lower real-time restriction, the migration of other agents will not affect the real-time interaction in MACVE either. Therefore, agent migration in MACVE can be used to improve the scalability of LCVE.

As introduced in Section 4.2.2, the process of agent remote cloning does not include an agent control handover step, so there is no handover time involved, i.e. it will not cause user interaction interruption. It is not necessary to measure the overhead of agent remote cloning.

5.3 Effects of Agent Migration on CVE System Scalability

To evaluate agent migration on the scalability of LCVE system, we conduct an experiment to make Consistency Agent migrate from Controlling Node to Trusted User Node for distributing the system computational workload. In the experiment, one of the Controlling Nodes hosts a Consistency Agent, a Persistency Agent, a Cell Agent and a Gateway Agent. We call the Controlling Node N_a. We execute the user application to log on the virtual playground from a host which works as a Trusted User Node. We call the Trusted User Node N_b. When the CPU workload of N_a exceeds its threshold, the Consistency Agent migrates from N_a to N_b to distribute the system computational workload to the Trusted User Node. The result of this experiment is shown in Figure 5-6.

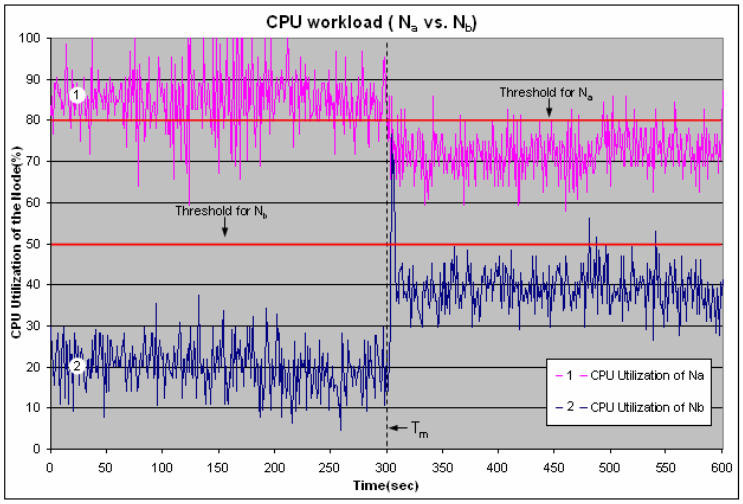

Figure 5-6 CPU workload on the Controlling Node (N_a) and Trusted User Node (N_b)

T_m --- the time when Consistency Agent starts to migrate from N_a to N_b.

CPU utilization --- the percentage of elapsed time that the processor spends to execute a non-Idle thread. It is the primary indicator of processor activity, and displays the average percentage of busy time observed during the sample interval.

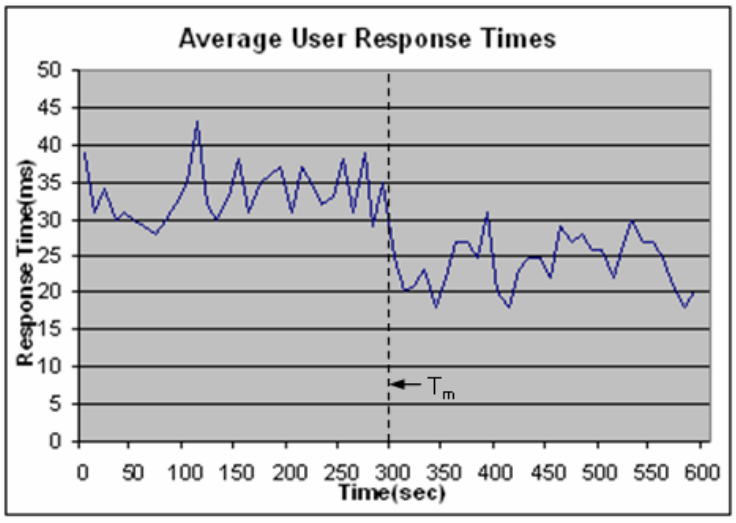

Figure 5-7 User Response Time during Consistency Agent Migration

As shown in Figure 5-6, the predefined threshold for CPU workload of N_a is 80%. Before T_m, the average of the CPU utilization of N_a is 85.17%. Based on the agent mobility

decision process, the Consistency Agent migrates to the Trusted User Node N_b at T_m to distribute its workload. After T_m, the average of CPU utilization of N_a drops to 72.27%. The migration of the Consistency Agent decreases the CPU workload considerably. At the same time, it does not cause N_b to be overloaded. Before T_m, the average CPU utilization of N_b is 20.38%. After the migration of the Consistency Agent, the average CPU utilization of N_b increase to 39.34%, which is still below the predefined threshold for CPU workload of N_b (50%).

As shown in Figure 5-7, the average user response times (ART) were sampled at every 10 second interval. There is a significant decrease of the ART at Tm. Before Tm, the ART has the mean value of 33:47ms and it ranges from 29ms to 43ms. After Tm, the ART's mean value is 24:13ms and it ranges from 18ms to 31ms. When the Consistency Agent migrates from a heavily-loaded node to a less-loaded node, the time for processing each user message is reduced. As the user response time includes network delay and Consistency Agent processing delay, and at the same time, the network delay of any node is same in our experiment, the migration of Consistency Agent results in the decrease of ART.

This experiment shows that agent migration is a feasible method to decentralize the system workloads and reduce the user response time. It can decrease the workload on Controlling Node and make the whole system to be more scalable.

5.4 Effects of Agent Remote Cloning on CVE System Scalability

To evaluate agent remote cloning on the scalability of MACVE system, we conduct an experiment to examine the performance of Cell Agent remote cloning. Since Cell Agent maintains the VE scene data which is static 3D description file or multimedia VE description, after the initial agent state synchronization, the master Cell Agent and its clones do not require any further synchronization. Therefore, the remote cloning of Cell Agent is an effective approach to improve the system scalability.

To study effect of Cell Agent remote cloning on the scalability of LCVE system, we conduct an experiment to compare the scene delivery performance of a cell having only

one Cell Agent with that of a cell having one master Cell Agent and three cloned Cell Agents. In the experiment, Master Cell Agent runs at a Controlling Node and the three Cloned Cell Agents run at three Trusted User Nodes. Under the same intensity of a CVE scenario, more concurrent users result in more user requests for the VE scene data from the Cell Agent. Therefore, we measure the CPU utilization at the Controlling Node, outgoing traffic rate at the Controlling Node, and the average scene data downloading time of users to examine the Cell Agent performance.

Our experiment is performed in the virtual playground. We use 20 RobotGroup Agents to simulate 50-300 concurrent users to download the scene data of a cell from its Cell Agent at a random time within every minute. We use another three hosts to work as Trusted User Nodes which meet the minimum capability requirements to take over system mobile agents. Once these Trusted User Nodes download the whole scene data of the cell, Cell Agent clone a new instance at the Trusted User Nodes. The master Cell Agent directs part of user requests to its cloned ones. These nodes are connected on a 100-Mb/s Ethernet. The results of this experiment are depicted in Figure 5-8, Figure 5-9, and Figure 5-10.

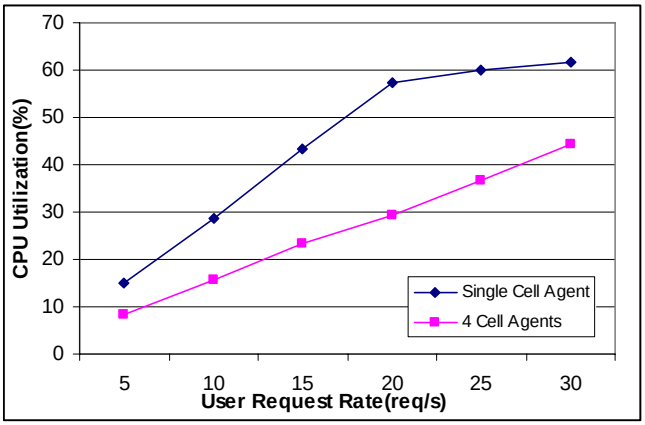

Figure 5-8: CPU utilization at the Controlling Node

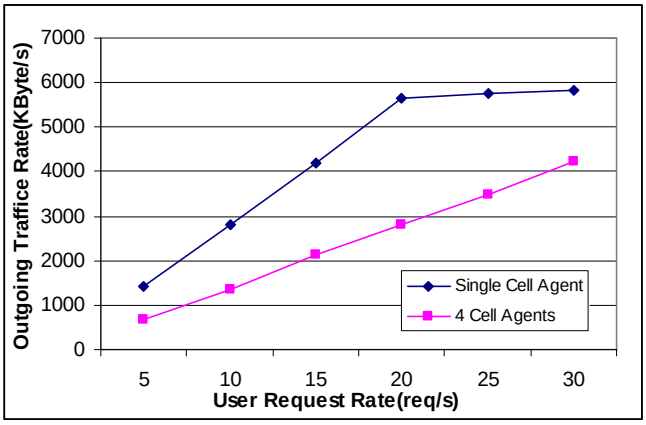

Figure 5-9: Outgoing traffic rate at the Controlling Node

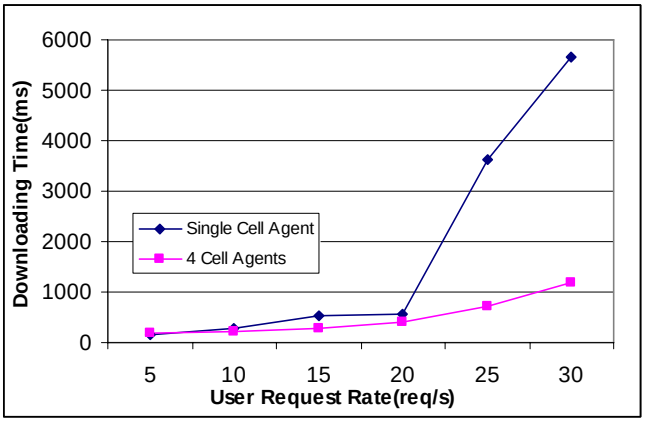

Figure 5-10: The average scene data downloading time

As shown in Figure 5-8 and Figure 5-9, when there is only one Cell Agent, the CPU utilization and outgoing traffic rate at the Controlling Node increase steadily as the user request rate increases. User request rate is the number of user requests for downloading VE scene data per second. However, after the Master Cell Agent spawns three Cloned Cell Agents at three Trusted User Nodes, the CPU utilization and outgoing traffic rate at the Controlling Node increase at a significantly slower pace. This indicates that remote agent cloning of Cell Agent at Trusted User Nodes can alleviate the workloads of delivering scene data at the Controlling Node so that the whole system can support higher user request rate.

As shown in Figure 5-10, the average downloading time is the average time for downloading a designated 3D description file of a virtual entity, which is 276kb in size. When there is only one Cell Agent, the average downloading time increase rapidly as the user request rate increases. When the user request rate is 30 requests per second, the average downloading time reaches 5,643ms. For the whole VE scene data of any cell which is much larger than 276kb, user will suffer much longer delay. So this delay will interrupt user's continuous navigation experience because the scene data of new cells can not be downloaded timely. Whereas, when the Master Cell Agent spawns three Cloned Cell Agents at Trusted User Nodes, the downloading time increases at slower pace. And the delay is only 1,176ms when the rate reaches 30 req/s. This indicates that remote agent cloning of Cell Agent at Trusted User Nodes can significantly reduce the time for users to download the scene data.

From Figure 5-8, Figure 5-9 and Figure 5-10, we also observe that when the user request rate is below 20 req/s, the difference between the two methods is marginal; but when the rate is above 20 req/s, remote agent cloning presents a much better performance. This indicates that remote agent cloning of Cell Agent at Trusted User Nodes is more effective for LCVE with large number of concurrent users.

For other type of agents, remote cloning can also make them provide better services. For example, ARM Agent can have multiple clones to distributed agent code on demand, which make MACVE system scalability at administration level. We will evaluate the performances of the remote cloning of other agents in our future works.

5.5 Effects of Adaptive Consistency Control on CVE System Scalability

In our prototype system, we only implemented strict consistency model and weak consistency model. Best-effort consistency model and other more fine-grained consistency models will be designed and implement in our future work. We will simply use these two models to evaluate the proposed adaptive consistency control in this section.

To evaluate adaptive consistency control on the CVE scalability, we conduct an experiment to measure the workload of a Consistency Agent and the round trip time

(RTT) of the affected user messages when the consistency model for part of the object state data is dynamically changed from strict consistency with client-server architecture to weak consistency with peer-to-peer architecture due to the change of activities happen in the CVE.

In the experiment, we launch a Consistency Agent on one of the Controlling Nodes. We use 20 RobotGroup Agents running on different machines to simulate 500 concurrent users to log on the CVE and connect to the Consistency Agent. In the initial stage, 40% of user messages adopt strict consistency model and the other 60% of them adopt weak consistency model. After a change of the activities happened in the CVE, the Consistency Agent dynamically changes the consistency model for the affected user messages. As a result, only 30% of user messages adopt strict consistency model and the other 70% of them adopt weak consistency control. The results of this experiment are depicted in Figure 5-11 and Figure 5-12.

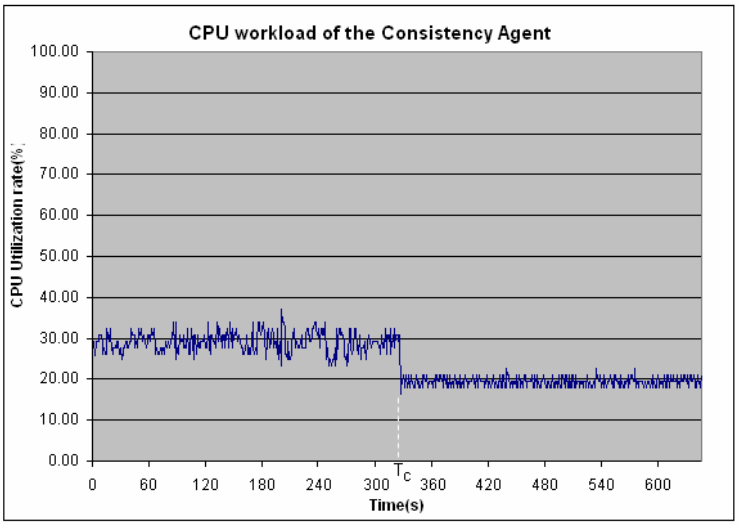

Figure 5-11 CPU workload of the Consistency Agent

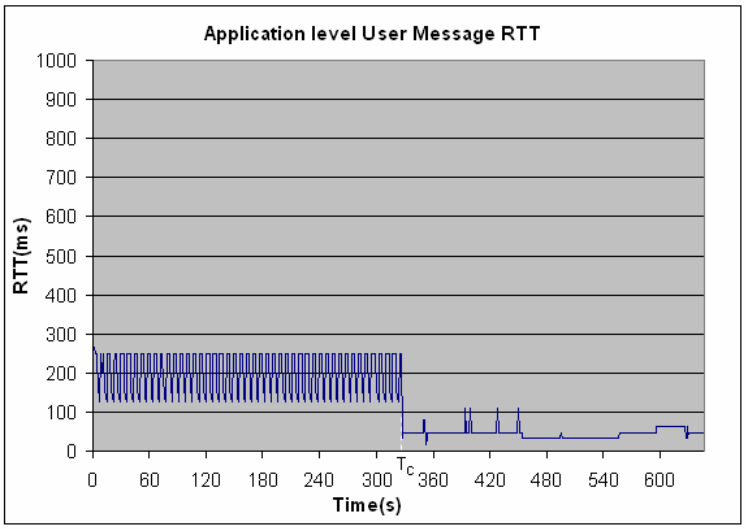

Figure 5-12 The affected user message RTT

T_c -- the time when Consistency Agent changes the consistency model for avatars' interaction information from strict consistency to weak consistency.

We observe that before T_c, the average workload of the Consistency Agent is 29.22%; after T_c, the workload drops to 19.29%. We also observe that before T_c, the average affected user message RTT is 203.14 ms; after T_c, the RTT drops to 44.53 ms. This indicates that when the changes of runtime activities in LCVE results in consistency requirement for certain object state data changes from strict consistency to weak consistency, adaptive consistency control in MACVE reduce the workload of the Consistency Agent, and at the same time, ameliorated the responsiveness of the affected activity in LCVE.

To further study the effects of adaptive consistency control on CVE system scalability, we conduct the above experiment to measure the average CPU workload and received data rate at a Consistency Agent under different concurrent user number and interactive entity number in a cell. The results of the experiments are depicted in the following figures.

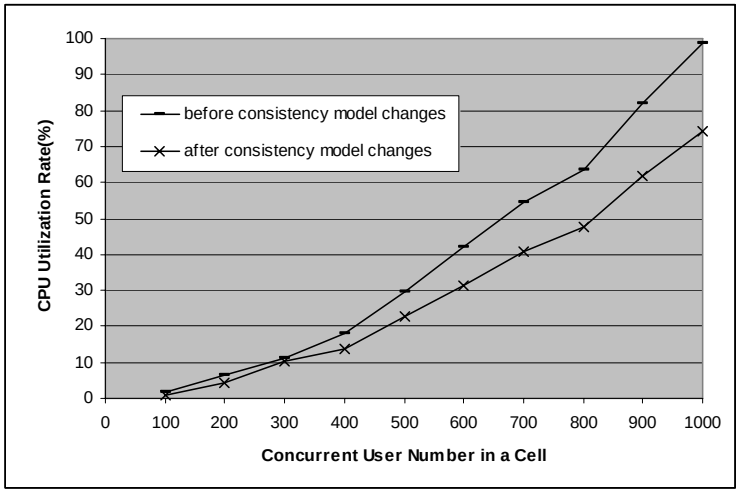

Figure 5-13 CPU workload of the Consistency Agent vs. concurrent user number

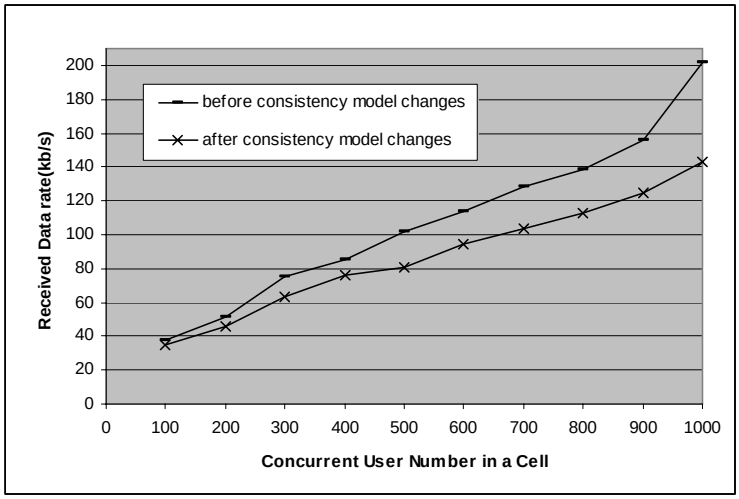

Figure 5-14 Received data rate at the Consistency Agent vs. concurrent user number

As shown in Figure 5-13 and Figure 5-14, when the concurrent user number in a cell is small, the CPU workload and received data rate do not have much difference before and the consistency model changes. With the growth of the concurrent user number, the dynamical changes from strict consistency to weak consistency for the affected user messages can save remarkable percentage of CPU cycles and bandwidth at Consistency Agent. Therefore, adaptive consistency control has more effect on the CVE with large number concurrent users.

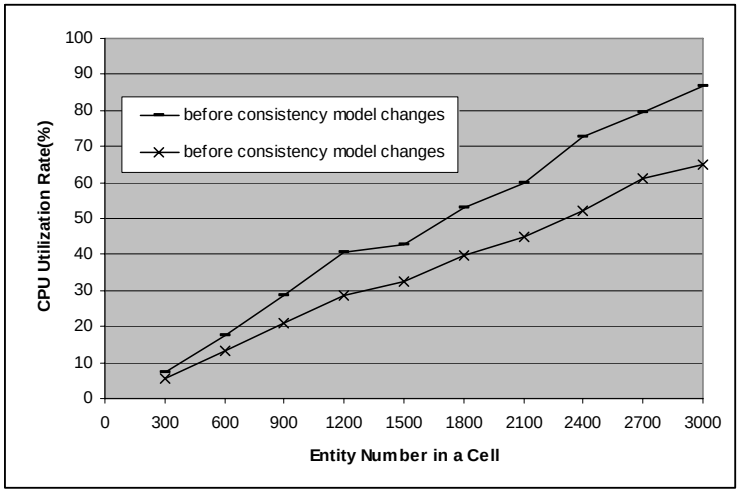

Figure 5-15 CPU workload of the Consistency Agent vs. entity number in a cell

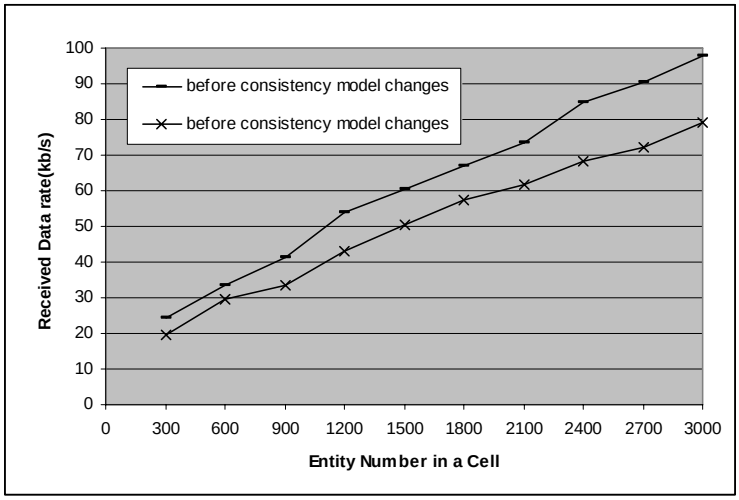

Figure 5-16 Received data rate at the Consistency Agent vs. entity number in a cell

As shown in Figure 5-15 and Figure 5-16, with the growth of the entity number in a cell, the dynamical changes from strict consistency to weak consistency for the affected user messages can save remarkable percentage of CPU cycles and bandwidth at Consistency Agent. Therefore, adaptive consistency control has more effect on the CVE with large number of entities.

From our experiments, we observe that by dynamically changing the consistency model, adaptive consistency control ensure the dynamic change of consistency requirements. It also efficiently utilizes the system resources and hence improves the system scalability.

5.6 Comparison of MACVE with Existing LCVE Systems

From the above experiments, it is shown that agent migration and remote agent cloning in MACVE can improve the LCVE system scalability. In this section, we compare MACVE with existing CVE systems to discuss the benefits and limitation of MACVE.

In comparison to NPSNET, MACVE has the following benefits:

(1) In NPSNET, the VE scene data is fully replicated at each participating host before the host joins the CVE. NPSNET did not provide an automatic and efficient way for scene data delivery. In MACVE, Cell Agents provide on demand VE scene data delivery services, which is more appropriate for LCVE with the evolving scene data.

(2) In NPSNET, the VE object state data is directly disseminated to participants by best-effort multicast. No consistency control is performed for the object state data. In MACVE, VE object state date is disseminated by different ways according to its consistency requirements. Object state data with low consistency requirement is directly disseminated to participants by multicast. Object state data with specific consistency requirement is sent to Consistency Agent first for consistency processing. It is then disseminated to related participants by unicast or multicast according to the semantic of the state data.

(3) NPSNET runs in a dedicated network and the variation of each participating host's processing power is related low. Hosts with low computing power or low connection speed are not well supported in NPSNET. In MACVE, Cell Agent and Consistency Agent can provide different services to various participating hosts to support more heterogeneous participants.

(4) NPSNET is complied with DIS, which is a protocol specially designed for war simulation. MACVE Protocol is more general. Developers can define the semantic of the fields for certain types of PDUs. So MACVE can support various genre of CVE application.

In comparison to RING, MACVE has the following benefits:

(1) In RING, the VE object state data of all users is firstly sent to servers. Servers perform visibility algorithms and only forward the object state data to the clients that have potential visual interactions. Although visibility algorithms at servers reduce the number of messages required to maintain consistency, the computational workload of visibility algorithms will increase dramatically with the increase of number of concurrent users. In MACVE, not all VE object state data need to be sent to Consistency Agents and not all of them is forwarded by unicast connections. Only the object state data that need certain level of consistency control is sent to Consistency Agents. Other state data is directly exchanged among users by multicast. Therefore, the Consistency Agents in MACVE consume few bandwidths and computing resources to control consistency than the servers in RING.

(2) In RING, no load balancing is proposed to redistribute the system workloads, which could results in partial capability wastage at certain servers whereas other servers may be overloaded. In MACVE, the agent migration and agent remote cloning provide the ways to dynamically distribute the system workloads to avoid the potential bottleneck.

(3) In RING, the scalability of the system solely depends on the number of servers. User hosts with extra computing power can not contribute theirs resources to the system to improve the system scalability. In MACVE, agents can autonomously migrate or clone at suitable Trusted User Nodes to relieve the workloads on the Controlling Nodes, so that the system can be more scalable.

In comparison to Spline, MACVE has the following benefits:

(1) In Spline, as the basic data of the world model, the scene data of the whole CVE is fully replicated at each participating host. Spline did not provide an automatic and efficient way for scene data delivery. In MAECVE, Cell Agents provide on

demand VE scene data delivery services, which is more appropriate for LCVE with the evolving scene data.

(2) In Spline, the VE object state data is directly disseminated to participants by best-effort multicast. No consistency control is performed for the VE object state data. In MACVE, different level of consistency control is provided. The object state date that required specific consistency control is sent to Consistency Agent firstly. After the consistency processing, it can be disseminated to related participants.

(3) In Spline, a virtual entity exists only when the application that owns it runs. By itself, Spline does not support virtual entities to persist over time. In MACVE, Persistency Agents can provide persistence services. They periodically record the cell state to database.

In comparison to Community Place, MACVE has the following benefits:

(1) Community Place system adopt client-server architecture. The server acts as a VE state maintainer and a VE object state data filter. Because the object state data filter is aura based and no multicast is used for message forwarding, the system is not scalable. The object state data filter in MACVE is zone based and subscription based, which use multicast for message propagation. Therefore, MACVE has less resource consumption in object state data filter than Community Place.

(2) In Community Place, static scene data is downloaded initially as part of the VRML file and replicated at all user browsers. For LCVE, the scene size is large. To initially replicate the entire scene data to user browser need a long waiting time, which is not practical. In MAECVE, each user downloads the most needed VE scene data on demand from Cell Agents, which is more efficient.

In comparison to VRTP, MACVE has the following benefits:

(1) VRTP provides each individual CVE user host with client, server, peer and network monitoring capability. User hosts can use unicast or multicast for data communication. MACVE also enable the user hosts to communicate by unicast

and multicast according to different consistency requirement of the data. Moreover, the adaptive consistency control in MACVE can dynamically change the data communication architecture for different object state data depending on their run-time application semantics. Therefore, the MACVE can effectively ensure the required consistency and efficiently utilizes the system resources and hence improves the system scalability.

(2) In MACVE, LCVE tasks for system data flows processing and exchanging do not bond with any fixed hosts. Agent migration and remote cloning can pervasively distribute system workloads at run-time to avoid potential bottleneck. It is not possible in VRTP.

From the above comparisons of MACVE with the existing LCVE systems, we argue MACVE systems combine the strength of both client-server and distributed peer-to-peer architecture while it avoids their weaknesses. Compared with the existing CVE systems using peer-to-peer architecture, MACVE have the advantages to support heterogeneous participant with low computer power; to support on demand VE scene data delivery; and to adaptively ensure different level CVE consistency. Compared with the existing CVE systems using multi-server architecture, MACVE has advantages to make LCVE tasks not bond with any fixed nodes; to decentralize system workloads to Trusted User Node by agent migration and agent remote cloning, which can make the system scalability less confined by the number Controlling Nodes.

At the same time, compared with existing CVE systems, MACVE has the following limitations.

(1) As MACVE is constructed with multiple collaborative mobile agents, the interactions among these agents make the software architecture more complex than the existing CVE system. Moreover, MACVE supports dynamic agent migration and agent remote cloning, which does not include in the existing CVE system. So MACVE has more implementation complexity. However, with the refinement of our MACVE APIs, most of the complexity will be transparent for developers.

5.7 Summary

This chapter presents the design and results of experiments for the evaluation of MACVE. From the presentation and analysis of experiments result, we demonstrate that agent migration is a feasible approach to dynamically distribute the system workload for LCVE; that agent migration and remote cloning can improve the system scalability by sharing the system workloads with Trusted User Nodes. Adaptive consistency control can ensure the dynamic change of consistency requirements and efficiently utilizes the system resources and hence improves the system scalability. Comparing MACVE with other LCVE systems, we discuss the benefits and limitations of MACVE.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis has been motivated by the challenge to improve the scalability of LCVE system in a heterogamous computing and networking environment from the data communication perspective. A mobile agent based framework – MACVE has been proposed to effectively control the data flow to avoid potential bottleneck and to adaptively control consistency of a persistent LCVE. A mobile agent system including Mobile Agent Environment, MACVE APIs, and MACVE Protocol, has been specially designed for MACVE to allow effective distribution of LCVE tasks. Based on our mobile agent system, we implement a prototype CVE system with eleven basic types of agent and a web-based user application. Experimentation results of the prototype system have demonstrated the feasibility of MACVE. It also has shown the advantages of the proposed MACVE for improving the scalability of the CVE applications.

There are three main unique advantages of the proposed framework:

- In MACVE, the tasks for controlling the data flows of a LCVE are modeled as mobile agents. The mobile agents are deployed to both Controlling Nodes and Trusted User Nodes. They can make decision to migrate or clone to reduce the workload of its hosting node before it becomes a potential bottleneck. The mutual independence of LCVE tasks and the participating hosts provide large freedoms to utilize the system resource efficiently.

- In MACVE, user nodes with spare capacity in terms of computing power, memory and network bandwidth can be registered as Trusted User Nodes. Based on agent mobility algorithm, suitable mobiles agents can migrate or clone at Trusted User Nodes to share the overall LCVE workload. The ability for qualified user hosts to take over system mobile agents enables the system to utilize the extra computing resources of user hosts and makes LCVE system scalability less confined with the number of Controlling Nodes.

- Adaptive consistency control enables MACVE to dynamically decide the consistency model and the communication architecture for object state data for different parts of LCVE, which effectively ensures the required consistency and efficiently utilizes the network bandwidth and hence improves the system scalability.

The main shortcoming of the proposed framework is the implementation complexity. The interactions among the mobile agents in MACVE make the software architecture more complex than the existing CVE system. Moreover, since MACVE support dynamic agent migration and remote cloning, it also has more implementation details. However, with the refinement of our MACVE APIs, most of the complexity will be transparent for developers.

6.2 Recommendations for Future Research

MACVE provide an adaptive framework for LCVE systems. Future research will refine the proposed framework from the following perspectives.

- As MACVE provides an adaptive layered architecture for LCVE system, it is easy for the future system functional extension. In this thesis, we only design and implement the most basic types of agents. New types of agents with other functionality can be incorporated in MACVE by using MACVE APIs and following the MACVE Protocol. For examples, various types of Robot Agent can be added in content layer to simulate autonomous computer generated avatar or entity; User Assistant Agent can be added in gateway layer to guide the user

within the CVE; User Proxy Agent can tailor the data communication to match the networking and computing capability of users without enough resources; etc. The new extensible agents inherit the ability to migrate or clone to powerful Trusted User Nodes to dynamically distribute the system workloads.

- Accurately predicting the resource requirements of an agent and the available resource of a SC Node remains a hard problem. Artificial Intelligence (AI), such as neural networks, seems to be a promising technology in computer science research. AI technology can be incorporated in the CRM to predict the future resource requirements of agents and available network and computational resources within the system. By the AI algorithm, MACVE can optimize the mobile action decision process and hence utilize system resources more effectively.

- In the prototype system of MACVE, only strict consistency model and weak consistency model have been implemented. More fine-grained consistency model can be incorporated in MACVE to improve the effectiveness of the proposed adaptive consistency control.

Publications

Journal Papers

- **Liang Zhang**, Qingping Lin, Robert Gay, Guang-Bin Huang, Hoon Kang Neo, "Enabling Large-Scale Collaborative Virtual Environments by Mobile Agents in Grid", Accepted by IEEE Transactions on Multimedia with minor revisions, 2007
- **Liang Zhang** and Qingping Lin, "MACVE: A Mobile Agent Based Framework for Large-scale Collaborative Virtual Environments," Presence: Teleoperators and Virtual Environments, Vol. 16, No. 3, pp. 279-292, MIT Press, 2007
- **Liang Zhang,** Qingping Lin, Robert Gay, Guang-Bin Huang, Hoon Kang Neo, "An Autonomous Decentralized Multi-Server Framework for Large Scale Collaborative Virtual Environments", International Journal of Image and Graphics, Vol. 7, No. 2, pp. 353-375, 2007
- **Liang Zhang** and Qingping Lin, "Support Dynamic Network Architecture for Large Scale Collaborative Virtual Environment", International Journal of Information Technology, Vol. 12, No. 1, pp. 26-36, 2006
- Qingping Lin and **Liang Zhang**, "Collaborative Virtual Environments: System Architectures ", accepted for publication in Encyclopedia of Computer Science and Engineering, Wiley Publishing, 2006.
- Qingping Lin and **Liang Zhang**, "Collaborative Virtual Environments: Web-based Issues ", accepted for publication in Encyclopedia of Computer Science and Engineering, Wiley Publishing, 2006.
- Qingping Lin and **Liang Zhang**, "Collaborative Virtual Environments: Applications ", accepted for publication in Encyclopedia of Computer Science and Engineering, Wiley Publishing, 2006.
- Qingping Lin, Weihua Wang, **Liang Zhang**, Jim Mee Ng, Chor Ping Low, "Behavior-Based Multiplayer Collaborative Interaction Management", Journal of Computer Animation and Virtual Worlds, Vol 16, pp1-19, Wiley Publishing, UK, 2005.

Conference Papers

- **Liang Zhang**, Qingping Lin, Robert Gay, Guang-Bin Huang, Hoon Kang Neo, "Support Large-Scale Collaborative Virtual Environment in Grid", The 3rd International Workshop on Grid Computing & Applications, Singapore, pp. 196-201, 2007
- Qingping Lin, Hong Koon Neo, **Liang Zhang**, Guang-Bin Huang, and Robert Gay, "Grid-based large-scale Web3D collaborative virtual environment " in Proceedings of the twelfth international conference on 3D web technology Perugia, pp. 123-132, Italy ACM Press, 2007
- Qingping Lin, **Liang Zhang**, I. Kusuma, N. Neo, "Mobile Agent Based Large Scale Collaborative Virtual Environment System", in Proceedings of the EUROMEDIA2006, Greece, 2006.
- Qingping Lin, **Liang Zhang**, I. Kusuma, N. Neo, "Addressing Scalability Issues in Large-Scale Collaborative Virtual Environment", accepted for publication in Computer Graphics International 2006, China, 2006.
- Qingping Lin, **Liang Zhang**, I. Kusuma, "Design of A Scalable Collaborative Virtual Environment System", in proceedings of EUROMEDIA2005, pp10-17, France, 2005.
- Qingping Lin, **Liang Zhang**, Wang W., I. Kusuma, "Design of Mobile Agent Based Large-scale Collaborative Virtual Environment System", in Proceedings of International Conference, Distributed and Parallel Systems and Architectures (DPSA2005), Portugal, 2005.
- Qingping Lin, Weihua Wang, **Liang Zhang**, Tian Fook Choo, "A Novel Method for Supporting Collaborative Interaction Management in Web-based CVE", in Proceedings of ACM VRST2003, pp124-131, Osaka, Japan, 2003.
- **Liang Zhang**, Qingping Lin, Tian Fook Choo, "Mobile Agent-Based Architecture for Large-Scale CVE", in Proceedings of Cyberworlds2003, IEEE Computer Society, pp115-122, Singapore, 2003.
- Tian Fook Choo, Qingping Lin, **Liang Zhang**, "A Novel Approach for Addressing Extensibility Issue in Collaborative Virtual Environment" in Proceedings of Cyberworlds 2003, IEEE Computer Society, pp162-169, Singapore, 2003.

Bibliography

[1] S. Benford, C. Greenhalgh, T. Rodden, and J. Pycock, "Collaborative Virtual Environments," *Communications of the ACM*, vol. 44, pp. 79 - 85, 2001.

[2] S. Singhal and M. Zyda, *Networked virtual environments: design and implementation*. New York, NY, USA ACM Press/Addison-Wesley Publishing Co., 1999.

[3] M. Meehan, "Survey of Multi-User Distributed Virtual Environments," in *Course Notes: Developing Shared Virtual Environments, SIGGRAPH 99*, 1999.

[4] I. Greif, *Computer-supported cooperative work : a book of readings*. San Mateo, California: Morgan Kaufmann, 1988.

[5] C. A. Ellis, S. J. Gibbs, and G. Rein, "Groupware: some issues and experiences," *Communications of the ACM*, vol. 34, pp. 39-58, 1991.

[6] C. R. Karr, D. Reece, and R. Franceschini, "Synthetic soldiers [military training simulators]," *IEEE Spectrum*, vol. 34, pp. 39-45, 1997.

[7] J. Han and B. Smith, "CU-SeeMe VR Immersive Desktop Teleconferencing," in *Proceedings of the fourth ACM international conference on Multimedia*. Boston, Massachusetts, United States, 1996, pp. 199-207.

[8] J.C.Oliveira, M.Hosseini, S. Shirmohammadi, E. P. M.Cordea, D.Petriu, and N.D.Georganas, "VIRTUAL THEATER for Industrial Training: A Collaborative Virtual Environment," in *4th World Multi-Conference on Circuits, Systems, Communications & Computers (CSCC 2000)*. Vouliagmeni, Greece, 2000, pp. 294-299.

[9] H. Y. Kan, V. G. Duffy, and C. J. Su, "An Internet virtual reality collaborative environment for effective product design," *Computers in Industry*, vol. 45, pp. 197-213, 2001.

[10] S. Su, R. B. Loftin, D. T. Chen, Y. C. Fang, and C. Y. Lin, "Distributed Collaborative Virtual Environment: Pauling World," in *Proceedings of The Tenth International Conference on Artificial Reality and Tele-existence*. Taipei, Taiwan, 2000.

[11] G. D. Kessler, D. A. Bowman, and L. F. Hodges, "The Simple Virtual Environment Library: an Extensible Framework for Building VE Applications," *Presence: Teleoperators and Virtual Environments*, vol. 9, pp. 187-208, 2000.

[12] M. Roussos, A. Johnson, T. Moher, J. Leigh, C. Vasilakis, and C. Barnes, "Learning and Building Together in an Immersive Virtual World," *Presence: Teleoperators and Virtual Environments*, vol. 8, pp. 247-263, 1999.

[13] J. Smed, T. Kaukoranta, and H. Hakonen, "A Review on Networking and Multiplayer Computer Games," Turku Centre for ComputerScience, TUCS Technical Report No 454, 2002.

[14] M. Matijasevic, D. Gracanin, K. P. Valavanis, and I. Lovrek, "A Framework for Multiuser Distributed Virtual Environments," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 32, pp. 416-429, 2002.

[15] R. Gil, J. P. Tavares, and L. Roque, "Architecting scalability for massively multiplayer online gaming experiences," in *Proceedings of Digital Game Research Association 2005 Conference: Changing Views - Worlds in Play*, 2005.

[16] L. Bernardo and P. Pinto, "Scalable Service Deployment Using Mobile Agents " in *Proceedings of the Second International Workshop on Mobile Agents* Springer-Verlag, 1999 pp. 261-272

[17] I. Vaghi, C. Greenhalgh, and S. Benford, "Coping with inconsistency due to network delays in collaborative virtual environments " in *Proceedings of the ACM symposium on Virtual reality software and technology* London, United Kingdom ACM Press, 1999 pp. 42-49

[18] D. Delaney, T. Ward, and S. McLoone, "On consistency and network latency in distributed interactive applications: a survey - Part I," *Presence: Teleoperators and Virtual Environments*, vol. 15, pp. 218-234, 2006.

[19] Q. Xiao, "Delayed consistency model for distributed interactive systems with real-time continuous media," *Journal of Software*, vol. 13, pp. 1029-1039, 2002.

[20] S. Zhou, W. Cai, B. S. Lee, and S. J. Turner, "Time-space consistency in large-scale distributed virtual environments," *ACM Transactions on Modeling and Computer Simulation*, vol. 14, pp. 31-47, 2004.

[21] L. Gautier, C. Diot, and J. Kurose, "End-to-end transmission control mechanisms for multiparty interactive applications on the Internet," in *Proceedings of the 1999 18th Annual Joint Conference of the IEEE Computer and Communications Societie, INFOCOM-99*, vol. 3. INRIA, Sophia Antipolis, Fr, 1999, pp. 1470-1479.

[22] T. Henderson, "Latency and user behaviour on a multiplayer game server," in *Third International COST264 Workshop, NGC 2001, Lecture Notes in Computer Science*, vol. 2233. London, UK: Springer-Verlag, 2001, pp. 1-13.

[23] L. Pantel and L. C. Wolf, "On the impact of delay on real-time multiplayer games," in *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*. Miami, FL, United States: Association for Computing Machinery, 2002, pp. 23-29.

[24] C. Diot and L. Gautier, "A distributed architecture for multiplayer interactive applications on the Internet," *IEEE Network*, vol. 13, pp. 6-15, 1999.

[25] D. J. Roberts and P. M. Sharkey, "Maximising concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimising the effect of network delays," in *Proceedings Sixth IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*. Cambridge, MA, USA, 1997, pp. 161-166.

[26] H. Tristan and B. Saleem, "Networked games: a QoS-sensitive application for QoS-insensitive users?," in *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS: What have we learned, why do we care?* Karlsruhe, Germany: ACM Press, 2003.

[27] M. Mauve, "Consistency in replicated continuous interactive media," *Proceedings of the ACM Conference on Computer Supported Cooperative Work*. Philadelphia, PA: Association for Computing Machinery, 2000, pp. 181-190.

[28] S. J. Kim, F. Kuester, and K. Kim, "A global timestamp-based approach to enhanced data consistency and fairness in collaborative virtual environments," *Multimedia Systems*, vol. 10, pp. 220-229, 2005.

[29] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, pp. 558-565, 1978.

[30] D. Roberts and R. Wolff, "Controlling Consistency within Collaborative Virtual Environments," in *Proceedings of the IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 2004, pp. 46- 52.

[31] W. Broll, "Interacting in distributed collaborative virtual environments " in *Proceedings of the Virtual Reality Annual International Symposium (VRAIS'95)* IEEE Computer Society, 1995 pp. 148

[32] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of Networking in Multiplayer Computer Games," in *International Conference on Application and Development of Computer Games in the 21st Century*. Hong Kong SAR, China, 2001, pp. 74-81.

[33] E. Frecon and M. Stenius, "DIVE: a scaleable network architecture for distributed virtual elements," *Distributed Systems Engineering*, vol. 5, pp. 91-100, 1998.

[34] T. Nedelec, "Data management in Distributed Shared Virtual Worlds," in *the 9th Doctoral Consortium of the Conference on Advanced Information Systems Engineering, CAISE 2002*. Toronto, Ontario, Canada, 2002.

[35] EPFL, Geneva, IIS, Nottingham, Thomson, and TNO, "Review of DIS and HLA techniques for COVEN," Project Report, ACTS Project N. AC040 1998.

[36] D. Roberts, "Communication Infrastructures for Inhabited Information Spaces," in *Inhabited Information Systems: Living with Your Data*: Springer-Verlag, 2003, pp. 233-267.

[37] J. Leigh, A. E. Johnson, and T. A. DeFanti, "CAVERN: A Distributed Architecture for Supporting Scalable Persistence and Interoperability in Collaborative Virtual Environments," *Journal of Virtual Reality Research, Development and Applications*, vol. 2.2, pp. 217-237, 1997.

[38] M. Capps, D. McGregor, D. Brutzman, and M. Zyda, "NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments," *IEEE Computer Graphics and Applications*, vol. 20, pp. 12-15, 2000.

[39] "The Virtual Reality Modeling Language." Web3d consortium, http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/, 1997.

[40] R. Lea, K. Matsuda, and K. Miyashita, *Java for 3D and VRML worlds*. Indianapolis: New Riders Publishing, 1996.

[41] R. C. Hofer and M. L. Loper, "DIS today [Distributed interactive simulation]," *Proceedings of the IEEE*, vol. 83, pp. 1124-1137, 1995.

[42] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen, "The SIMNET virtual world architecture," in *Virtual Reality Annual International Symposium*, 1993, pp. 450-455.

[43] D. C. Miller and J. A. Thorpe, "SIMNET: the advent of simulator networking," *Proceedings of the IEEE*, vol. 83, pp. 1114-1123, 1995.

[44] "IEEE standard for distributed interactive simulation - application protocols(IEEE Std 1278.1)," Simulation Interoperability Standards Committee of the IEEE Computer Society, 1995.

[45] M. Macedonia, M. Zyda, D. Pratt, P. Barham, and S. Zestwitz, "NPSNET: A Network Software Architecture for Large-Scale Virtual Environments," *Presence: Teleoperators and Virtual Environments*, vol. 3, pp. 265--287, 1994.

[46] J. S. Dahmann, "High Level Architecture for Simulation," in *the 1st International Workshop on Distributed Interactive Simulation and Real-Time Applications*: IEEE Computer Society, 1997, pp. 9-14.

[47] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules." Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society, http://standards.ieee.org/catalog/olis/compsim.html, 2000.

[48] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification," Simulation Interoperability Standards Committee of the IEEE Computer Society, http://standards.ieee.org/catalog/olis/compsim.html, 2000.

[49] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification." Simulation Interoperability Standards Committee of the IEEE Computer Society, http://standards.ieee.org/catalog/olis/compsim.html, 2000.

[50] D. P. Brutzman, M. Zyda, K. Watsen, and M. R. Macedonia, "Virtual Reality Transfer Protocol (VRTP) Design Rationale " in *Proceedings of the 6th Workshop*

on Enabling Technologies on Infrastructure for Collaborative Enterprises IEEE Computer Society, 1997 pp. 179-186

[51] K. Watsen and M. Zyda, "Bamboo - a portable system for dynamically extensible, real-time, networked, virtual environments," in *Proceedings of the 1998 IEEE Virtual Reality Annual International Symposium, VRAIS,*. Atlanta, GA, USA, 1998, pp. 252-259.

[52] T. A. Funkhouser, "Network topologies for scalable multi-user virtual environments," in *Proceedings of the IEEE 1996 Virtual Reality Annual International Symposium*. Santa Clara, CA, USA, 1996, pp. 222-228.

[53] D. Bauer, I. Iliadis, S. Rooney, and P. Scotton, "Communication architectures for massive multi-player games," *Multimedia Tools and Applications*, vol. 23, pp. 47-66, 2004.

[54] K. Saar, "VIRTUS: a collaborative multi-user platform," in *Proceedings of the fourth symposium on Virtual reality modeling language*. Paderborn, Germany: ACM Press, 1999, pp. 141-152.

[55] J. Calvin, J. Seeger, G. Troxel, and D. V. Hook, "STOW Realtime Information Transfer and Networking Architecture," in *12th DIS Workshop on Standards for the Distributed Interactive Simulations*. Orlando, FL, 1995.

[56] M. R. Macedomia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, "Exploiting reality with multicast groups: a network architecture for large-scale virtual environments " in *Proceedings of the Virtual Reality Annual International Symposium (VRAIS'95)* IEEE Computer Society, 1995 pp. 2

[57] H. Fisher, "Multicast Issues for Collaborative Virtual Environments," *IEEE Computer Graphics and Applications*, vol. 22, pp. 68-75, 2002.

[58] M. R. Macedonia and M. J. Zyda, "A Taxonomy for Networked Virtual Environments " *IEEE MultiMedia* vol. 4 pp. 48-56 1997

[59] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel, "Reality built for two: a virtual reality tool " in *Proceedings of the 1990 symposium on Interactive 3D graphics* Snowbird, Utah, United States ACM Press, 1990 pp. 35-36

[60] W. Bricken and G. Coco, "The VEOS Project (Virtual Environment Operating Shell)," *Presence: Teleoperators and Virtual Environments*, vol. 3, pp. 111-129, 1994.

[61] C. Shaw and M. Green, "MR Toolkit peers package and experiment," in *IEEE Symposium on Research Frontiers in Virtual Reality*. San Jose, CA, 1993, pp. 463-469.

[62] D. McGregor, A. kapolka, M. Zyda, and D. Brutzman, "Requirements for large-scale networked virtual environments," in *the 7th International Conference on Telecommunications*, vol. 1, 2003, pp. 353- 358.

[63] B. Blau, C. E. Hughes, M. J. Moshell, and C. Lisle, "Networked virtual environments," in *Proceedings of the 1992 symposium on Interactive 3D graphics*. Cambridge, Massachusetts, United States: ACM Press, 1992, pp. 157-160.

[64] C. Greenhalph, "Spatial Scope and Multicast in Large Virtual Environments," Technical Report NOTTCS-TR-96-7, 1996.

[65] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W.Yerazunis, "Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability," *Presence: Teleoperators and Virtual Environments*, vol. 6, pp. 461-81, 1997.

[66] W. Broll, "SmallTool - A Toolkit for Realizing Shared Virtual Environments on the Internet," in *Distributed Systems Engineering, Special Issue on Distributed Virtual Environments*, 1998, pp. 118-128.

[67] E. Lety, T. Turletti, and F. Baccelli, "SCORE: A Scalable Communication Protocol for Large-Scale Virtual Environment," *IEEE/ACM Transactions on Networking*, vol. 12, pp. 247-260, 2004.

[68] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu, "The state of the art in locally distributed Web-server systems," *ACM Computing Surveys*, vol. 34, pp. 263-311, 2002.

[69] B. Devlin, J. Gray, B. Laing, and G. Spix, "Scalability Terminology:Farms, Clones, Partitions, and Packs: RACS and RAPS," Microsoft Research, Technical Report MS-TR-99-85, 1999.

[70] G. Singh, L. Serra, W. Png, A. Wong, and H. Ng, "BrickNet: sharing object behaviors on the Net " in *Proceedings of the Virtual Reality Annual International Symposium (VRAIS'95)* IEEE Computer Society, 1995 pp. 19-25

[71] T. K. Das, G. Singh, A. Mitchell, P. S. Kumar, and K. McGee, "NetEffect: a network architecture for large-scale multi-user virtual worlds," in *Proceedings of the ACM symposium on Virtual reality software and technology*. Lausanne, Switzerland: ACM Press, 1997, pp. 157-163.

[72] H. Sugano, K. Otami, H. Ueda, S. Hiraiwa, S. Endo, and Y. Kohda, "SpaceFusion: a multi-server architecture for shared virtual environments," in *Proceedings of 2nd Annual Symposium on the Virtual Reality Modelling Language*. Monterey, CA, USA: ACM, 1997, pp. 51-58.

[73] B. Ng, A. Si, R. W. H. Lau, and F. W. B. Li, "A multi-server architecture for distributed virtual walkthrough," in *Proceedings of the ACM symposium on Virtual reality software and technology*. Hong Kong, China: ACM Press, 2002, pp. 163-170.

[74] T. A. Funkhouser, "RING: a client-server system for multi-user virtual environments," in *Proceedings of the 1995 symposium on Interactive 3D graphics*. Monterey, California, United States: ACM Press, 1995, pp. 85-92.

[75] R. Lea, Y. Honda, K. Matsuda, and S. Matsuda, "Community Place: architecture and performance," in *Proceedings of the second symposium on Virtual reality modeling language*. Monterey, California, United States: ACM Press, 1997, pp. 41-50.

[76] Q. Lin, W. Wang, J. Feng, B. Gao, T. O. Zaw, Y. H. Low, and K. Balasubramanian, "NATIVE: a network architecture supporting large-scale VRML based collaborative virtual environment," in *Proceedings of EUROMEDIA'99*. Munich, Germany, 1999, pp. 23- 27.

[77] A. Shaikh, S. Sahu, M. Rosu, M. Shea, and D. Saha, "Implementation of a service platform for online games," *Proceedings of the ACM SIGCOMM 2004 Workshops*, 2004, pp. 106-110.

[78] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *Proceedings of the IEEE INFOCOM*, vol. 1, 2004.

[79] K. L. Morse, L. Bic, and M. Dillencourt, "Interest management in large-scale virtual environments," *Presence: Teleoperators and Virtual Environments*, vol. 9, pp. 52-68, 2000.

[80] J. C. d. Oliveira and N. D. Georganas, "VELVET: An Adaptive Hybrid Architecture for VEry Large Virtual EnvironmenTs," in *IEEE International Conference on Communications*, vol. 4, 2002, pp. 2491 -2495.

[81] J. W. Barrus, R. C. Waters, and D. B. Anderson, "Locales: Supporting Lange Multiuser Virtual Environments," *IEEE Computer Graphics and Applications*, vol. 16 (6), pp. 50-57., 1996.

[82] T. T. Emmanuel lety, and Francois Baccelli, "SCORE: A Scalable Communication Protocol for Large-Scale Virtual Environment," *IEEE/ACM Transactions on Networking*, vol. 12, pp. 247-260, 2004.

[83] M. R. Macedomia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, "Exploiting reality with multicast groups: a network architecture for large-scale virtual environments," in *Proceedings of Virtual Reality Annual International Symposium, 1995*, 1995, pp. 2-10.

[84] Q. Lin, W. Wang, L. Zhang, J. M. Ng, and C. P. Low, "Behavior-Based Multiplayer Collaborative Interaction Management," *Journal of Computer Animation and Virtual Worlds*, vol. 16, pp. 1-19, 2005.

[85] W. Wang, Q. Lin, J. M. Ng, and C. P. Low, "SmartCU3D: a collaborative virtual environment system with behavior based interaction management," in *Proceedings of the ACM symposium on Virtual reality software and technology*. Baniff, Alberta, Canada: ACM Press, 2001, pp. 25-32.

[86] H. Abrams, K. Watsen, and M. Zyda, "Three-tiered interest management for large-scale virtual environments," in *Proceedings of the ACM symposium on Virtual reality software and technology*. Taipei, Taiwan: ACM Press, 1998, pp. 125-129.

[87] M. Hori, T. Iseri, K. Fujikawa, S. Shimojo, and H. Miyahara, "Scalability issues of dynamic space management for multiple-server networked virtual environments," in *Proceedings of IEEE Pacific RimConference on Communications, Computers and Signal Processing*, vol. 1, 2001, pp. 200-203.

[88] H. Eriksson, "MBONE: the multicast backbone," *Communications of the ACM*, vol. 37, pp. 54-60, 1994.

[89] C. Greenhalgh and S. Benford, "MASSIVE: a distributed virtual reality system incorporating spatial trading," in *Proceedings of 15th International Conference on Distributed Computing Systems*, 1995, pp. 27-34.

[90] I. S. Pandzic, T. K. Capin, N. M. Thalmann, and D. Thalmann, "VLNET: a networked multimedia 3D environment with virtual humans," *Multimedia Modeling. Towards Information Superhighway*. Singapore: World Scientific, 1995, pp. 21-32.

[91] S. Benford and C. Greenhalgh, "Introducing Third Party Objects into the Spatial Model of Interaction," in *Proceedings of the Fifth European Conference on Computer Supported Cooperative Work (ECSCW'97)*. Lancaster, 1997, pp. 189-204.

[92] C. Greenhalgh, J. Purbrick, and D. Snowdon, "Inside MASSIVE-3: flexible support for data consistency and world structuring " in *Proceedings of the third international conference on Collaborative virtual environments* San Francisco, California, United States ACM Press, 2000 pp. 119-127

[93] IBM, "Butterfly.net: Powering Next-Generation Gaming with On-Demand Computing," available at http://www.ibm.com/grid/pdf/bufferfly.pdf 2003.

[94] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "Grid services for distributed system integration," *Computer*, vol. 35, pp. 37-46, 2002.

[95] I. Foster, "Globus Toolkit Version 4: software for service-oriented systems," in *Proceedings of Network and Parallel Computing. IFIP International Conference, NPC 2005.* Beijing, China: Springer-Verlag, 2005, pp. 2-13.

[96] V. A. Pham and A. Karmouch, "Mobile Software Agents: An Overview," *IEEE Communications Magazine*, vol. 36, No.7, pp. 26-37, 1998.

[97] D. B. Lange and M. Oshima, *Programming and deploying Java mobile agents with Aglets*: Addison-Wesley Longman, Reading, Mass., 1998.

[98] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Communications of the ACM*, vol. 42, pp. 88-89, 1999.

[99] J. M. Smith, "A survey of process migration mechanisms," *ACM SIGOPS Operating Systems Review*, vol. 22, pp. 28-40, 1988.

[100] D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and Z. Songnian, "Process migration," *ACM Computing Surveys*, vol. 32, pp. 241-99, 2000.

[101] M. Persson, "Mobile Agent Architectures," Swedish Defence Research Establishment, Division of Command and Control Warfare Technology, Technical Report FOA-R-00-01700-503-SE, 2000.

[102] N. M. Karnik and A. R. Tripathi, "Design Issues in Mobile-Agent Programming Systems," *IEEE Concurrency*, vol. 6, pp. 52-61, 1998.

[103] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, vol. 24, 1998.

[104] G. Cabri, L. Leonardi, and F. Zambonelli, "Weak and Strong Mobility in Mobile Agent Applications," in *Proceedings of the Second International Conference on the Practical Application of Java*, 2000, pp. 143-156.

[105] D. B. Lange, M. Oshima, G. n. Karjoth, and K. Kosaka, "Aglets: Programming Mobile Agents in Java " in *Proceedings of the International Conference on Worldwide Computing and Its Applications* Springer-Verlag, 1997 pp. 253-266

[106] R. S. Gray, "Agent Tcl: a flexible and secure mobile-agent system," in *Proceedings of 4th Annual Tcl/Tk Workshop '96*. Monterey, CA, USA, 1996, pp. 9-23.

[107] H. Peine and T. Stolpmann, "The architecture of the Ara platform for mobile agents," in *Proceedings of the First International Workshop on Mobile Agents(MA '97)*. Berlin, Germany: Springer-Verlag, 1997, pp. 50-61.

[108] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet, "Concordia: an infrastructure for collaborating mobile agents," in *Proceedings of the First International Workshop on Mobile Agents(MA '97)*. Berlin, Germany: Springer-Verlag, 1997, pp. 86-97.

[109] J. Baumann, F. Hohl, K. Rothermel, and M. Stra?er, "Mole - Concepts of a mobile agent system," *World Wide Web*, vol. 1,3, pp. 123-137, 1998.

[110] D. Johansen, R. van Renesse, and F. B. Schneider, "Operating system support for mobile agents," in *Proceedings of the 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*. Orcas Island, WA, USA, 1995, pp. 42-45.

[111] D. B. Lange, "Mobile objects and mobile agents: The future of distributed computing?," *Lecture Notes in Computer Science*, 1998, pp. 1-12.

[112] W. Obeloer, C. Grewe, and H. Pals, "Load management with mobile agents," in *Proceedings of 24th Euromicro Conference*, vol. 2, 1998, pp. 1005-1012.

[113] J. Cao, Y. Sun, X. Wang, and S. K. Das, "Scalable load balancing on distributed Web servers using mobile agents," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 996-1005, 2003.

[114] D. B. Lange and M. Oshima, "Mobile agents with Java: The Aglet API " *World Wide Web* vol. 1 pp. 111-121 1998

[115] H. Daniel and L. Ismail, "A performance evaluation of the mobile agent paradigm," in *Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. Denver, Colorado, United States: ACM Press, 1999.

[116] M. D. Dikaiakos and G. Samaras, "Performance Evaluation of Mobile Agents: Issues and Approaches " *Performance Engineering: State of the Art and Current Trends, Lecture Notes in Computer Science*, vol. 2047, pp. 148-166, 2001.

[117] D. Schmalstieg and M. Gervautz, "Demand-driven geometry transmission for distributed virtual environments," in *European Association for Computer Graphics 17th Annual Conference and Exhibition, Computer Graphics Forum*, vol. 15. Poitiers, France, 1996, pp. 421-32.

[118] J. Chim, R. W. H. Lau, V. Leong, and A. Si, "CyberWalk: A Web-Based Distributed Virtual Walkthrough Environment," *IEEE Transactions on Multimedia*, vol. 5, pp. 503-515, 2003.

[119] A. S. Tanenbaum and M. v. Steen, *Distributed Systems: Principles and Paradigms*. PTR Upper Saddle River, NJ, USA: Prentice Hall, 2002.

[120] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems " *Computer* vol. 25 pp. 33-44 1992

[121] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, pp. 141-154, 1988.

[122] J. Watts and S. Taylor, "A practical approach to dynamic load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, pp. 235 - 248, 1998.

[123] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, pp. 141-54, 1988.

[124] D. Roberts, R. Wolff, O. Otto, and A. Steed, "Constructing a Gazebo: supporting teamwork in a tightly coupled, distributed task in virtual reality," *Presence: Teleoperators and Virtual Environments*, vol. 12, pp. 644-657, 2003.

[125] M. Dahlin, A. Brooke, M. Narasimhan, and B. Porter, "Data Synchronization for Distributed Simulations," in *2001 European Simulation Interoperability Workshop, SISO*, 2001.

[126] S. Shirmohammadi and N. D. Georganas, "An end-to-end communication architecture for collaborative virtual environments," *Computer Networks*, vol. 35, pp. 351-367, 2001.

[127] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding code mobility," *IEEE Transactions on Software Engineering*, vol. 24, pp. 342-361, 1998.

[128] D. Wong, N. Paciorek, and D. Moore, "Java-based mobile agents," *Communication of the ACM*, vol. 42, pp. 92-102, 1999.

[129] "The Virtual Reality Modeling Language," web3d consortium, http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/.