MOBILE AGENTS AND SMART NETWORKS FOR DISTRIBUTED SIMULATIONS

Steve Stone
***Mike Zyda**
Don Brutzman
John Falby
Computer Science Department, Code CS/Zk
Naval Postgraduate School
Monterey, California 93943-5118
E-Mail: {stonesw, zyda, brutzman, falby} @cs.nps.navy.mil
*** contact author**

KEYWORDS
Communications Architecture, Communication Network, Multicast, Scalability

ABSTRACT

As the use of Distributed Interactive Simulations has grown, the need to support a large number of players in the environment has become apparent. DIS has not been able to support a large number of entities because of its high network bandwidth requirements and the large computational loads it places on host computers.

A combination of an agent based architecture and smart networks provides a promising solution to these problems when implementing large-scale distributed simulations. An agent system using the remote programming paradigm, transporting the necessary parameters and the necessary code for remote execution [WHITE95], could reduce the network bandwidth requirements and large computational loads associated with a large distributed simulation. This reduction would occur by eliminating unnecessary PDU traffic through the use of mobile agents that represent the originating entity. These agents would travel to, and reside on, the host computer of other entities and provide the necessary state information for stationary entities without using network resources.

Smart Networks could be used to create a flexible area of interest manager, allowing entities to specify their area of interest and the information that they require from within that area [HARR95a]. This approach allows an entity to get all of the information it requires to represent its view of the simulated world while eliminating unnecessary information processing.

DISTRIBUTED INTERACTIVE SIMULATIONS:

DIS Protocol:

Development of the DIS protocol began in 1989, jointly sponsored by the United States Army Simulation, Training and Instrumentation Command (STRICOM), ARPA and the Defense Modelling and Simulation Office (DMSO). DIS was based on SIMNET and designed as a man in the loop simulation in which participants interact in a shared environment from geographically dispersed sites. The initial objective was to develop a standard that provided guidelines for the interoperability of defense simulations. DIS provides a basis of interoperability for large scale virtual environments using a wide variety of different hardware and software platforms. DIS has been adopted by the Institute of Electrical and Electronics Engineers (IEEE) as an International Standard (IEEE Std 1278-1993).

While the DIS standard adopted many aspects of the earlier SIMNET protocol including its general principles, terminology and PDU formats, it is intended to overcome the limitations of SIMNET and includes packet definitions not found in SIMNET [DURLACH95]. DIS is also designed to use the Defense Simulation Internet (DSI), an ARPA project designed to allow thousands of players to link using real and simulated forces to create a Synthetic Theater of War (STOW) [DURLACH95]. Another difference between SIMNET and DIS is that DIS uses the TCP/IP suite of protocols and is thus an application level protocol, allowing it to be used on any network topology that uses TCP/IP.

Protocol Data Units: The heart of DIS is a set of protocols that are used to convey messages about entities and events, via a network, among the simulation nodes that are responsible for maintaining the status of the entities in the virtual world [DIS94]. Simulation and event information is conveyed by the twenty-seven PDUs defined by the IEEE 1278 DIS standard. Four of the PDUs are used for sending information about entity interaction. The other twenty-three are used for transmitting information on supporting actions, electronic emanations and simulation control [MAC95a]. DIS PDUs are independent of the network media and network protocols being used to transmit them and can be used on most current network topologies.

The PDUs used to transmit information about entity interaction are: Entity State PDU (ESPDU), Fire PDU (FPDU), and Detonation PDU (DPDU). The ESPDU is used to communicate information about an entity's current state, including its position, orientation, velocity and appearance. The ESPDU is the most commonly used PDU and, in most instances, it dominates the network traffic [MAC94]. The format of the ESPDU is shown in Figure 1. The FPDU contains data on any weapon that is fired or dropped. The DPDU is sent when a munition detonates or an entity crashes. The actual structure of each PDU is very regimented and is explained in full detail in [IEEE 1278].

Problems with the DIS Protocol:

Using the current method, a large-scale simulation would require enormous bandwidth and place a huge computational load on host computers. It has been estimated that using DIS, a simulation with 100,000 players would require approximately 375 Mbps of network bandwidth to each computer participating in the simulation [MAC95a]. Since this is not within the realm of possibility in the near future, the implementation of large-scale virtual environments will require a communications protocol with a much lower bandwidth requirement. Computational load is another problem that must be overcome. In 1994 the U.S. Army attempted to demonstrate the feasibility of large-scale simulations in the Synthetic Theater Of War - Europe (STOW-E) exercise. The goal of the exercise was to simulate 10,000 entities over a wide area network connecting a number of sites in the United States and Europe. However, only 1,800 entities could be represented because of computational bottlenecks in both the simulators and support equipment [MAC95a].

Because DIS is a stateless system and does not utilize servers, all data must be distributed to all entities in the system. Thus when the status of one entity changes, it must send an update, as an ESPDU, to every other entity in the simulation. As such, ESPDU's can account for up to 70% of the network traffic [MAC95a]. For entities that are moving it is necessary to send these updates as often as needed. Not doing so would corrupt the coherence of the virtual world. However, there is no such problem with entities that are stationary. An

Field Size (bytes)	Entity State PDU Fields	
12	PDU Header	Protocol Version Exercise ID PDU Type Padding Time Stamp Length in Bytes
6	Entity ID	Site Application Entity
1	Force ID	
1	Number of Articulation Parameters	
8	Entity Type	Entity Kind Domain Country Category Subcategory Specific Extra
8	Alternate Entity Type	Same information as above.
12	Linear Velocity	X, Y, & Z (32 bit components)
24	Location	X, Y, & Z (64 bit components)
12	Orientation	Psi, Theta, Phi (32 bit components)
4	Appearance	
40	Dead Reckoning Parameters	Algorithm Other Parameters Linear Acceleration Angular Velocity
12	Entity Markings	
4	Capabilities	32 Boolean Fields
N * 16	Articulation Parameters	Change ID Parameter Type Parameter Value

Figure 1: Entity State PDU. [IEEE93]

analysis of an actual combat scenario at the National Training Center showed that in ten hours of simulated combat approximately 33% of 2,191 entities did not move. As the exercise continued, over half of the vehicles stopped all movement. [MAC95b] If there were a method where each entity could have knowledge of every stationary entity without transmitting an ESPDU every five seconds there would be a significant reduction of PDU traffic. This would also remove some of the computational burden on the host computers by removing the need to read these PDU's off of the network. Mobile agents using the remote programming paradigm could be a solution to these problems.

MOBILE AGENTS AND SMART NETWORKS:

Overview:

Mobile Agents are programs, typically written in a script language, that are sent from a client computer and transported to a remote server for execution. A mobile agent contains both the necessary state information and the executable code for the agent to complete its mission. This approach to computer communication is known as the Remote Programming Paradigm. A Smart Network is a framework for the execution of mobile agents. This framework allows numerous heterogeneous servers to offer a host-independent execution environment for mobile agent programs and a standard communication language with which agents and servers can engage in dialogs [HARR95a].

Remote Programming:

Current Method: The basis for today's computer communications networks is the Remote Procedure Call (RPC) Paradigm. RPC views computer to computer communications as enabling one computer to call procedures on another. The messages that transit the network either request or acknowledge a remote procedure's execution. A request contains the data necessary for the execution of the procedure. A response contains the return value of the procedure. The format of the messages and the effects of each procedure call constitute a protocol. A client computer with work to accomplish on a remote server bundles the necessary parameters and ships them to the remote server for execution. The server begins execution and responds with an acknowledgment. This process continues until the interaction is complete. The DIS protocol can be viewed as a form of RPC without the acknowledgment. When an entity detects that its current state exceeds a threshold from its previous state, or that a certain time interval has passed, it packs up its current state information and sends it in a procedure call to every entity in the simulation. Upon receipt, each entity runs a procedure to update its view of the world with the new state information [WHITE95].

New Method: An alternative method is Remote Programming (RP). RP views computer communication not only as calling procedures on another computer, but also providing the procedure to be called. Each message that the network transports is a procedure that the server is to execute and the data necessary for execution. Of note is that the procedure's execution was begun on the sending computer but is continued on the receiving computer. The procedure and its state are called a mobile agent. The important part of RP is that the sending computer and the receiving computer can interact without using the network once the network has transported the agent between them [WHITE95].

The first advantage of RP is performance. When the client computer has work to do on a remote server, it can send the work and supervise locally using an agent, rather than continually sending instructions over the network. The network is called on to carry fewer messages. "The more work to be done the more messages remote programming avoids [WHITE95]." This performance advantage is dependent on the network. The lower the throughput or availability, the higher the performance gain. The second advantage of RP is customization. A mobile agent allows the sender to customize the functionality of the receiver. New procedures can be sent to the server with little effort. This turns the network into a platform for which new applications can be developed, allowing the network's behavior to be easily modified.

Mobile Agent Concepts:

Mobile Agent: The main components of an agent system, or framework, are mobile agents, an agent language, agent meeting places and security services. The primary part of an agent framework is the mobile agent itself. The agent can be written in various programming languages and can transport knowledge expressed in various forms. The agent must be able to engage in a dialog with the agent meeting place until they execute or are rejected. An agent can execute until completion or it can elect to suspend its activity and move to another meeting place and resume execution there. In this case, the state of the agent and its execution environment must be moved to the new meeting place. An agent's structure contains a passport, a table of contents and components. The passport contains the agent's identification and the identification of the owner which together establish the authority of the agent. For an agent to receive service, the server must be able to identify the owner of the agent. The passport also contains the permit of the agent. The permit establishes what the agent can do. An agent can be limited in the number of CPU cycles it can use or in the amount of time it can live. The permit also gives the agent the right to execute certain instructions. For example, an agent's permit can allow it to spawn other agents. An agent that tries to exceed one of these limits is prevented from doing so. An agent is aware of its authority and permit, but cannot increase them. This is critical to the security of the agent framework. Without these limitations an agent could run amok causing irreparable damage. The passport also contains error actions and addresses should problems arise. The table of contents of the agent provides a map of its structure. This allows an agent meeting place to determine which components of the agent are required for execution. The components of the agent are the executable code that the agent needs to accomplish its mission. Ideally, these components are instances of class libraries of the agent language [CHESS95].

Agent Language: A mobile agent can be written in one of a number of programming languages. There are certain languages that are more useful for implementing agents than others. An agent programming language should be a high level language for ease of use, object oriented, support agent mobility and provide constructs that support distributed computing. Several languages have been designed to support agent implementation. The newest of these are Telescript and Java [WHITE95] [SUN95]. Other agent frameworks have been written in Lisp, Scheme, Tcl/SafeTcl, KQML, and KIF [CHESS95].

Agent Meeting Place: The next important part of an agent framework is the Agent Meeting Place (AMP). An AMP offers service to the mobile agents that enter it. An engine (Figure 2) is a program residing on the server that implements the agent framework by maintaining and executing the AMPs it contains, as well as the agents that occupy those AMPs. In general, the engine is an interpreter for the language used to implement the framework. An interpreted approach is used to help provide security for the system. By interpreting the agent code, it is easier to prevent the agent from directly accessing memory and other system resources. The engine interfaces with the host system through three application program interfaces (APIs). The APIs are used to manage storage, transport agents and interface with external applications. The storage API lets the engine access the nonvolatile memory it requires to preserve the places and agents in case of a computer failure. The transport API lets the engine access the communications network so that it can transport agents to and from other engines. The external applications API lets the agent interact with other applications on the host computer.

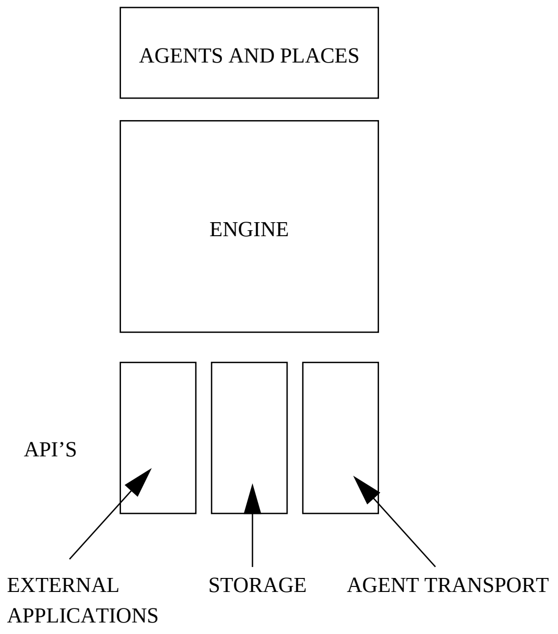

Figure 2: Agent Engine [WHITE95]

Agent Security: The final component of an agent framework is its security services. It is critical that an agent framework maintain the security of the host computer. Without proper security measures, an agent is really just a virus. The first and most important security measure is interpreted rather than executed code. As discussed above, interpreting the agent's code allows the engine to have some control over the agent's ability to access the systems resources. The agent's authority and permit are also essential in providing security. Without these controls, an agent could take over a system. These are the major security methods that an agent framework makes use of. There are many others required to prevent agent tampering and ensure data integrity and privacy [CHESS95].

Agent Execution: When a mobile agent arrives at an AMP, the Agent Transport API removes the headers associated with the transport protocol and assembles the agent. Once assembled, the agent is passed to the manager process for the AMP. The manager process authenticates the agent's passport, conducts a data integrity check and decrypts the components of the agent, if necessary. Once the authentication is complete, the manager process registers the agent with the engine and distributes the agents components to the elements of the engine necessary to support them. After all of this is completed, the agent begins executing. If it had begun executing on another host, its execution state is restored. The engine allocates host resources to the agent within its permit's limits and the agent continues to run until it has exceeded its limits or has completed its work. When the agent has completed its work, it is packaged and either sent back to its originator or off to another AMP that the agent has specified. A much more through explanation of this process can be found in [CHESS95].

Smart Networks:

Overview: The fundamental model of networks is that they provide communications between network nodes and that the routing and priority of the communication is independent of the content. A Smart Network's focus is on delivering information to users with the user in control of how, when, where and whether the information is delivered. The key point of this statement is that the user is in control of the network's behavior. This control is exercised through instruction provided by the user and through observations of the user's interaction with the network.

Components of a Smart Network: For a smart network to work, it requires several forms of intelligence. It requires an inferencing system for collecting information about the user's interaction with the network. It requires mobile agents with the associated framework for processing messages and routing them through a network of servers. And it requires gateways to allow access to the network. The inferencing system usually would consist of an inferencing engine, a set of rules and a method of delivering event information to the engine. This system would reside at the network gateway on a server at the user's point of access to the network and would analyze the user's interaction with the network and assist in customizing the network's behavior for the user. [HARR95a]

Customizing the Network's Behavior: The inferencing system described above works in conjunction with the user's expressed instructions in order to customize the user's interface with the network. This is accomplished through mobile agents that control the flow of information to the user's host computer. These agents can screen incoming messages, forward them to other sites, deliver messages according to a certain priority or perform other services. Most of the research in this area has been focused on using smart networks to aid in personal communications and mobile computing. While none of these smart networks is fully implemented, much work is being done in the commercial arena. Although not specifically intended for distributed simulations, the concept of a tailorable network could have a significant impact on the communications problems plaguing DIS.

APPLICATION TOWARDS DISTRIBUTED SIMULATIONS:

Overview:

The greatest problem currently facing the progress of distributed simulations is scalability. The current methodology cannot scale up beyond approximately 2000 entities because of its tremendous requirements for network bandwidth and the computational loads it places on the host systems. The root of this problem is twofold. First, the current DIS standard requires every entity to broadcast an ESPDU on the network whenever its state has changed enough to exceed a specified threshold or when a certain time limit has been reached. While the ESPDU that reflects a change in state is critical, the ESPDU that is sent even though the entity's state has not changed can waste a significant amount of network resources. The second problem is that currently an entity must process every PDU from every other entity even though it has no need to know anything about the other entities. This again results in large amount of unnecessary network traffic and computation by each host. Large-scale distributed simulations require that significantly reduced communications loads be placed on the system. Mobile agents and smart networks provide a method for accomplishing this.

Network Bandwidth Requirements:

DIS's Problem: The ESPDU accounts for approximately 70% of the network traffic in a distributed simulation. A large portion of this load comes from stationary entities that are broadcasting their 'heartbeat' message. DIS uses this heartbeat message so that an entity joining the simulation late will get an accurate picture of the state of the simulation within five seconds. While this method avoids the use of a server to maintain the current simulation state, it results in a large amount of network traffic. If somewhere between 33% and 60% of the entities are stationary then 23% to 42% of the network traffic is not necessary [MAC95b]. If this PDU traffic is removed from the network, there is a significant decrease in the bandwidth requirements. How then, do stationary entities communicate with other entities in the simulation? An agent based system using an engine process on each host is the answer.

An Agent Based Solution: The current DIS architecture could be modified to support an agent based system designed to reduce the communications load on the network. This system would consist of an engine process running on each host computer that would support the AMP and agents necessary to reduce the PDU traffic. The premise of this architecture is that an entity that does not change state for a specified time would, instead of transmitting heartbeat PDUs, send a mobile agent to each of the other entities in the simulation. The agent's passport would contain the entity identification and other information necessary to identify the agent. This agent would travel to the AMP on each host and would begin to execute there. The agent would communicate with the simulation program, providing all of the state information that was previously sent in an ESPDU. The simplest method of doing this would be for the agent to send the necessary data in the ESPDU format through the engine's external application API to the simulation's network interface. Then the simulation would read the ESPDU as if it had been transmitted over the network. While this technique would be the easiest to implement and would reduce the network traffic, it increases the computational load on the host. The simulation's network interface would still be required to read and process the ESPDU. Another method would be to write the agent and the simulation to allow direct communications between them. For example, the agent could, through the engine's APIs, write all of the necessary state information directly to the simulation's entity management table. This would eliminate the need to read and process an ESPDU.

Eliminating the ESPDUs from stationary entities creates a problem in updating entities that are late in joining the simulation. Without the heartbeat PDUs, newcomers to the simulation will not receive an accurate representation of the virtual world. An agent is also the solution to this new problem. An entity that joins the simulation will receive ESPDUs from every entity that is currently changing state, but not from stationary entities. To get the state information about these entities, the new entity will send out a query agent. This agent would ideally go to the nearest host computer and query the AMP about other agents that are located there. Through interaction with the AMP and other agents, the query agent would cause the agents representing stationary entities to spawn a copy and send it to the new entity's host. The copies of the agents then travel to the host, register with the AMP and begin providing state information. This would take some amount of time but the new entity would soon have an accurate view of the virtual world. This delay in time required to get up to date is an acceptable trade off for the decrease in network traffic.

Entity Interaction Using Agents: The agent would continue executing in the AMP until the originating entity changes state. At that point, the entity would begin to send ESPDUs as is currently done. Upon receipt of an ESPDU over the network, the agent representing the sender would be killed and the state information would be received from the ESPDUs until another agent is received. Collision detection and other entity interactions would continue to be implemented as they are today. A stationary entity that has sent an agent to other entities continues to behave normally, except that it no longer transmits ESPDUs as long as its state is constant. The entity continues to read PDUs from the network and respond appropriately to any interaction. A stationary vehicle entity that reads a Detonation PDU would check the PDU for a possible interaction. If it determined that it had been hit, it would calculate the damage received and send an appropriate ESPDU. As before, the receipt of this ESPDU would kill the vehicle's agent and its state information would be received over the network. After a specified time interval had elapsed, the vehicle entity would transmit another agent representing its damaged state.

Problems With This Approach: The most significant problem with this approach is the additional computational load placed on the host computer by the agent engine. On a multi-processor machine, one processor might be dedicated to the engine with little slowdown. However, on a single processor machine, the CPU cycles needed by the agent engine could outweigh any processing saved by the reduced network load. Current research work has not examined an agent based system incorporated into a distributed simulation and no quantifiable results exist. Such a system needs to be implemented and experiments conducted.

The other problem that requires more research is the performance of agent systems in a real-time environment such as a distributed simulation. In this architecture, the agents would be simple and it is possible that it would meet the latency requirements set forth in the DIS standard. Again, this is an area that requires thorough experimentation.

Computational Loads:

The Problem: The agent based approach above does little to reduce the computational loads associated with processing a large number of ESPDUs. In fact, the agent approach may increase the computational burden by adding another process to the host computer. The root of this problem is DIS's requirement that each entity know about the state of every other entity in the simulation. In a large scale simulation, an entity would have to process thousands of ESPDUs a second and maintain a record of each entity. In real combat, most entities have little interest in events outside of their immediate area. A tank on the ground has no interest in another tank that is fifty kilometers away. Yet in the current DIS architecture, it is required to maintain a record of the distant tank's state. A solution to this problem is to partition the virtual world into smaller 'Areas of Interest' (AOI) using multi-cast channels [MAC95a]. Each area of interest would have an associated multi-cast channel. An entity in that AOI would transmit its PDUs only on the multi-cast channel for that area. Entities would subscribe to the multi-cast channels associated with each AOI it is interested in. Thus, each entity would receive only the PDUs from entities within the areas it is interested in. AOIs would be from one of three classes: Spatial, Functional, and Temporal. Spatial AOIs would be used to group entities that are in close physical proximity to each other. Entities such as a Battalion of dismounted infantry would normally all be within the same Spatial AOI. Research has shown that a 4 kilometer hexagon is optimal for combat simulations [MAC95a]. Functional AOIs would be used to group entities that need to communicate, but are not in close physical proximity. The radio communications of a dispersed unit would be transmitted using a functional AOI. Temporal AOIs would be used for entities that have different real-time communications requirements. A JSTARS Aircraft might need entity position updates every five minutes instead of every five seconds. This could be done using a temporal "ALL" AOI, where every entity would transmit to this multicast channel once every five minutes. Simulations of this architecture have shown that there would be a dramatic drop in the network traffic and computational loads of the host computers [MAC95a].

Smart Networks for Distributed Simulation: Another approach would be to use a smart network to allow individual entities to create their own area of interest. This way an entity could design an area of interest so that it receives all of the information it needs to portray reality but receives no unnecessary information. An infantryman could set his AOI to be a 1000 meter circle around his position and the JSTARS could establish an AOI of thousands of square kilometers. Entities could also tailor the information they receive. The infantryman could specify that he does not want to get PDUs from high performance aircraft in the area and the JSTARS could tell the network that it does not want to get PDUs from dismounted troops within its AOI.

The key to implementing such an AOI manager is the network gateway containing its inferencing engine. An entity would be able to tailor its AOI by establishing a set of rules that describe the information (PDUs) that it wants to receive. These rules could outline the locations from which the entity wants to receive PDUs, effectively establishing a spatial class for the entity. For example, send me PDUs from each entity within ten kilometers. Or the entity could establish a set of rules so that it would receive PDUs from all aircraft type entities. These rules would be encoded into a mobile agent that is sent to the network gateway's AMP where its set of rules would be given to the inferencing agent. The inferencing agent would then use these rules to tailor the PDUs that are sent to that entity.

The main advantage of this approach is the flexibility that it gives to the AOI manager concept. By changing its set of rules, an entity can change its AOI. Using the examples from the proceeding paragraph, an entity could combine the two sets of rules to form a new AOI that is interested only in all aircraft within ten kilometers. An entity can make this change rapidly. Once the new set of rules is created, a mobile agent is dispatched and the gateway begins to use the new rule set.

Of course, just as using multi-cast channels to partition the world requires more work, the idea of a smart network for distributed simulations is a long term goal. Smart networks are just now beginning to be implemented and there are many questions to be answered. The foremost are about the real-time performance of such a network. Would a smart network be able to meet the latency requirements of a distributed simulation? The design of a smart network is also an area for future research. Would a smart network's server architecture move distributed simulations away from a stateless system. Would this create reliability problems that the DIS standard has attempted to avoid? As smart networks evolve these questions need to be answered.

CONCLUSIONS:

As the use of Distributed Interactive Simulations has grown, the need to support a large number of players (more than 1000) in the environment has become apparent. The current approach, the Distributed Interactive Simulations Protocol (DIS), has not been able to support large numbers of entities because of its requirement that each entity transmit a heartbeat message at a certain time interval (usually every five seconds). For an entity that is stationary, these messages serve the purpose of saying "I'm still here". This requirement has lead to high network bandwidth requirements and unacceptable computational loads on the host computers. Several methods of solving this problem have been suggested. Using multi-cast channels to partition the Virtual Environment and reducing redundant information contained in the Protocol Data Units (PDUs) are the two most common suggestions. While both of these concepts have merit, they do little to reduce the total number of PDUs that must be transmitted.

An agent based architecture and smart networks provide a promising solution to the problems of implementing large-scale distributed simulations. An agent system using the Remote Programming paradigm could reduce the network bandwidth requirements and computational loads associated with a large distributed simulation. This reduction would occur by eliminating unnecessary PDU traffic through the use of mobile agents that represent the originating entity. These agents would travel to and reside on the host computer of other entities and provide the necessary state information for stationary entities without using network resources.

Smart networks could be used to create a flexible area of interest manager that allows entities to specify their area of interest and the information they require from within that area. This approach allows an entity to get all of the information it requires to represent its view of the simulated world while eliminating all unnecessary information processing.

Further research must be done to examine the real-time performance of both agent systems and smart networks to ensure that they can meet the real-time requirements of distributed simulations and to measure the reduction in network traffic and computational loads.

ACKNOWLEDGEMENTS:

This work would not have been possible without the support of our research sponsors: the Advanced Research Projects Agency (ARPA), the US Army Research Laboratories (ARL), the Defense Modeling and Simulation Office (DMSO), US Army TRADOC Analysis Center (TRAC), US Army Topographic Engineering Center (TEC), the Office of Naval Research (ONR), USA STRICOM and the Research, Development, Test, and Engineering Division (NRaD) of the Naval Command, Control and Ocean Surveillance Center.

LIST OF REFERENCES:

BRU95 Brutzman, Donald, Macedonia, Michael and Zyda, Michael, *Internetwork Infrastructure Requirements for virtual Environments*, in the Proceedings of the First VRML Symposium, 1995.

CANTER95 Canterbury, Michael, *An Automated Approach to Distributed Interactive Simulation (DIS) Protocol Entity Development*, Masters Thesis, Naval Postgraduate School, Monterey, California, September 1995.

CHESS95 Chess, David, Grosof, Benjamin, Harrison, Colin, Levine, David, Parris, Colin, and Tsudik, Gene, *Itinerant Agents for Mobile Computing*, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1995.

DIS94 DIS Steering Committee, *The DIS Vision - A Map to the Future of Distributed Simulation*, Institute for Simulation and Training, Orlando, Florida, May 1994.

DURLACH95 Durlach, Nathaniel I. and Mavor, Anne S., *Virtual Reality: Scientific and Technological Challenges,* National Academy Press, Washington, D.C. 1995.

HARR95a Harrison, Colin G., *Smart Networks and Intelligent Agents*, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1995.

HARR95b Harrison, Colin G. Chess, David M., Kershenbaum, Aaron, *Mobile Agents: Are They a Good Idea?*, IBM T.J. Watson Research Center, Yorktown Heights, New York, 1995.

IEEE1278 Institute of Electrical and Electronics Engineers, International Standard, ANSI/IEEE Std 1278-1993, *Standard for Information Technology, Protocols for Distributed Interactive Simulation*, March 1993.

LING95 Lingnau, Anselm and Drobnik, Oswald, *An Infrastructure for Mobile Agents: Requirements and Architecture*, Proceedings of the 13th DIS Conference, Orlando, 1995.

MAC94 Macedonia, Michael R., Zyda, Michael J., Pratt, David R., Barham, Paul T., Zeswitz, Steven, *NPSNET: A Network Software Architecture for Large Scale Virtual Environments,* Presence, 3, 4, Winter 1994.

MAC95a Macedonia, Michael R., *A Network Architecture for Large Scale Virtual Environments*, Dissertation, Naval Postgraduate School, Monterey, California, June 1995.

MAC95b Macedonia, Michael R., Zyda, Michael J., Pratt, David R., Barham, Paul T., *Exploiting Reality with Multicast Groups*, IEEE Computer Graphics and Applications, September 1995.

MEYER95 Meyer, Tom and Conner, D. Brookshire, *Adding Behavior to VRML,* Brown Computer Graphics Group, August 95.

MOR95 Morrison, John, *The VR-Link, Networked Virtual Environment Software Infrastructure*, Presence, 4,2, Summer 95.

SAPAT95 Sapaty, P.S., Corbin, M.J., and Borst P.M., *Mobile WAVE Programming as a Basis for Distributed Simulation and Control of Dynamic Open Systems*, Proceedings of the 15th International Conference on Distributed Computing Systems, Vancouver, June 1995.

SUN95 Sun Microsystems, Inc., *The Java(tm) Language Environment: A White Paper*, Sun Microsystems, Mountain View, California, 1995.

WAY95 Wayner, Peter, *Agents Unleashed, A Public Domain Look At Agent Technology,* AP Professional, Boston, 1995.

WHITE95 White, James E., *Telescript Technology: Mobile Agents*, General Magic White Paper, Mountain View California, 1995.

ZES93 Zeswitz, Steven R., *NPSNET: Integration of Distributed Interactive Simulation (DIS) Protocol for Communication Architecture and Information Interchange,* Masters Thesis, Naval Postgraduate School, Monterey, California, September 1993.