Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

UTSAF: A MULTI-AGENT-BASED FRAMEWORK FOR SUPPORTING MILITARY-BASED DISTRIBUTED INTERACTIVE SIMULATIONS IN 3D VIRTUAL ENVIRONMENTS

Joseph Manojlovich
Phongsak Prasithsangaree
Stephen Hughes
Jinlin Chen
Michael Lewis

School of Information Science
135 N. Bellefield Ave.
University of Pittsburgh
Pittsburgh, PA 15260, U.S.A.

ABSTRACT

A Military-based distributed interactive simulation (DIS) such as ModSAF has been used for many years. Several problems of the DIS-based simulation to support a large and heterogeneous virtual simulation environments have been discovered (Stone, Zyda, Brutzman, and Falby 1996). To solve these problems, we propose an architectural multi-agent-based framework to support a large military-based simulation with 3D visualization using inexpensive game simulators. Several software agents are used to support interoperability between DIS-based military simulation nodes and Unreal Tournament game simulators. An agent is used to reduce DIS traffic to efficiently utilize network bandwidth. It also performs protocol conversion between DIS protocol and a game engine protocol. Additionally, using a multi-agent system, our work is easily expandable to support several network environments and also to support agent-based intelligent operations. Our main contribution is twofold. We use a multi-agent system which is scalable to support our framework. In addition, our framework builds a simulation bridge that enables affordable high-quality 3D viewer node using affordable game simulations for military simulations.

1 INTRODUCTION

Military-based distributed interactive simulations (IS) have been in development for many years. Such simulations support multiple simulation environments and interactions between simulation operators and computer generated forces. A major example of such simulations is Modular Semi-Automated Force (ModSAF). The ModSAF simulator is composed of several simulation nodes, each of which can be a simulation engine (server), a simulation client, or both. Each simulation nodes communicates by using a DIS protocol through a TCP/IP network. The DIS protocol is used to exchange messages called Protocol Data Units (PDUs). The PDUs contain information about entities and events in simulation environments. By using the DIS protocol, a ModSAF simulation can be extended into a very large network of simulation nodes (Brunett and Gottschalk 1997).

However, DIS has several problems. First, the DIS PDU packets transmitted to other simulation nodes tend to overwhelm a network bandwidth (Stone, Zyda, Brutzman, and Falby 1996). Since the DIS protocol works in stateless manner, all information of each entities and events is transmitted to other simulation nodes even though there is no status changes in those entities or events. Therefore, there is a need of a "middle" agent between each distributed simulation nodes so that its network traffic is reduced and the members of military organizations are able to simulate a very large DIS network.

The second problem of the DIS environments is the lack of an efficient and affordable three-dimensional (3D) simulated environment. In the DIS environment, an operator may want to view a surrounding area in 3D view to determine an appropriate position of new entities before placing them in a simulation. The operator may also want to operate from the view point of an entity that provides a better 3D visualization environment. Few simulators provide this capability.

Lastly, the DIS environment was originally built to link several simulation nodes together, but it does not provide an interface for intelligent operations, which can be added to aid operators in making a critical decision. In a large DIS

environment, there is an enormous amount of information available for an operator and it is necessary to provide an intelligent information fusion so that the operator can make critical decisions in timely manner.

In our framework, we propose a DIS environment using software agents and a 3D game simulation. The software agents were built using the Reusable Environment for Task-Structured Intelligent Networked Agents (RETSINA) environment, which is a proven multi-agent system (MAS) (Sycara, Paolucci, van Velsen, and Giampapa 2001). In our framework, each simulation node is represented by a RETSINA agent, and it is used to filter redundant information so that the network traffic can be reduced. In addition, our framework uses a 3D game simulation which is efficient and inexpensive, and it also provides an amazing graphical view. Our work can be easily extended to support an interactive simulated 3D environment by using the game simulation. Finally, with our MAS-based framework, we can introduce an intelligent support to operators such as an intelligent information fusion agent.

For the 3D game simulation, we used the Unreal Tournament video game because of its affordability and its flexibility to support several modifications for our work (Epic Game Developers). We have named our work Unreal Tournament Semi-Automated Force (UTSAF).

In summary, the main contribution of our proposed work is threefold. First, our framework is based on software agents which can be distributed in any network environments to support a large distributed interactive simulation network. Second, our framework uses a game simulation environment for 3D visualization which can support interactive activities and is affordable and flexible for research in human visualization. Third, our framework is easily extensible to provide intelligent operations such as information fusion agent-based interface for operators.

In this paper, a background of our work is described in Section 2. We discuss our framework of a UTSAF architecture and visualization environment design in Section 3. Our experimental testbed setup is explained in Section 4, and other related works are described in Section 5. Finally, we conclude our work and discuss our future work in Section 6.

2 BACKGROUND

In this section, we explain military-based simulation such as ModSAF and its related communication protocol, the DIS protocol. We further explain in general what software agent is. Then, we focus on describing the essential components of a game simulation and how to use software such as GameBot to help in controlling a game simulation.

2.1 ModSAF and DIS Protocol

One of the well-known military simulation environments is ModSAF, which was released by joint effort between the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Simulation, Training and Instrumentation Command (STRICOM) in 1993 (Calder, Smith, Courtemarche, Mar, Ceranowicz 1993). Later, ModSAF has been integrated to a next generation semi-automated force (SAF) simulation called One Semi-Automated Force (OneSAF) which was released in early 2001 (OneSAF Website). ModSAF and OneSAF were created to support DIS environments. Each simulator communicates with one another by using DIS protocol which was standardized by IEEE (IEEE1278 1993). The DIS protocol is used to convey messages about entities and events, through a network, to simulation nodes that are involved in a distributed simulation of SAF entities in a virtual world. The messages are in form of Protocol Data Units (PDUs). The PDUs used to explain entity interactions are Entity State PDU (ESPDU), Fire PDU (FPDU) and Detonation PDU (DPDU). The ESPDU contains information about entity's current status including its type, location, orientation, velocity, and articulations. Most of the network traffic between simulation nodes are ESPDU messages. The FPDU is used to convey information about a firing of a weapon. The FPDU contains a munition type, a firing entity, a target entity, a launching location, and munition's velocity. The DPDU is used to convey information about the impact of a munition. The DPDU contains a detonation location, a detonation result, and information similar to FPDU. More details of PDUs can be found in (IEEE1278 1993).

2.2 Software Agents

A software agent in generally defined as an intelligent, collaborative software object that achieves a higher level of artificial intelligence through a global communication structure involving other software agents. Multi-agent systems (MAS) assume that no single agent can or does know all of the information in its given domain, and that only through collaboration with other agents can they achieve overall knowledge (Sycara 1998).

RETSINA is an example of a software agent test bed and development system (Rectenwald 2002). RETSINA is a mature MAS which has been at the core of many successful agent-based projects (SoftAgents Website).

2.3 Game Simulation

Both military simulations and personal computer video games have a long and related history, but with different focuses. While military simulations concentrate on reproducing as exactly as possible the real world, video games

tend to focus on pushing computer hardware to its limit to produce the most graphically intense simulation as possible. In many ways, the cutting edge of computer development in both software and hardware is being driven by the games themselves (Lewis and Jacobson 2002).

Games have found their way into more than just battle simulation. Recent work at the MOVES Institute at the Naval Postgraduate School have produced their own video game, "America's Army," which uses a pre-release copy of the Unreal Tournament 2003 video game to realistically portray life as a new U.S. Army recruit (Zyda, Hiles, Mayberry, Wardynski, Capps, Osborn, Shilling, Robaszewski, and Davis 2003). Video games today are a far cry from the once cutting edge of Pong and Space Invaders. Today's games are reminiscent of the operating systems on which they run. They have a core "kernel" engine, which abstracts the real operating system from the rest of the game. They employ networking to link multiple game engines across the actual network. Most now even have their own scripting languages and full development suites to facilitate user modifications to the games.

2.3.1 Game Engines

Figure 1 shows the structure of a modern computer game. In most games, the game engine, networking code, and graphics drivers are written in a native language of the operating system, such as C or C++. Game engines carry out all sorts of tricks to ensure that the maximum amount of processing is used for graphics rendering. All games use a process known as "culling," whereby parts of the simulation that are not visible to the user are not given any rendering time. While this may seem obvious, military simulations in general do not follow this guideline. Another trick is to render objects with less detail the further from the player that they are. This is known as level of detail optimization. The structure of objects are simplified based on distance.

2.3.2 Game Networking

Most interesting of all is the ability to link game engines across a network. Every modern game engine provides this capability, to allow multiple human users to play a single game together. Many of the problems of state replication and security that were first encountered with military simulators of years back, have now reappeared in these new games.

Computer games of today can link together in a peer-to-peer fashion, or users can run so-called dedicated servers. By using a dedicated server, most of the non-graphical game logic can be dumped onto an older PC, freeing up the user's gaming PC to use more of its resources on rendering the graphics.

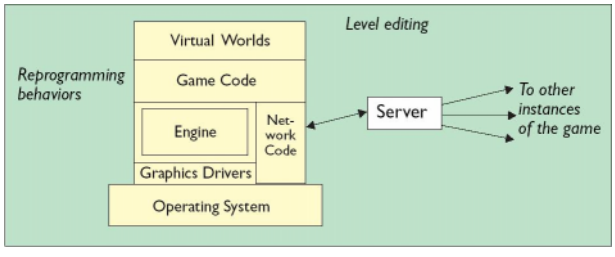

Figure 1: Modular Game Engine Structure from Lewis and Jacobson (2002)

2.3.3 Modern Personal Computer Graphics

Until quite recently, high resolution real-time computer graphics were confined to expensive and proprietary computer systems. However, the rapid development of personal computer hardware over the last 5 years has produced cheap computers that not only rival these setups, but in some ways even surpass them (Lewis 2002). PCs run both industry-standard OpenGL, giving them the ability to run older software projects. They also support other high-performance toolkits such as Microsoft DirectX and Direct3D.

2.3.4 Programmable Modules

Above the game engine lies the user-configurable parts of the game. The game code typically is some proprietary scripting language developed solely for the individual games. It abstracts out the low-level game engine details and lets the user concentrate on modifying the game with a minimum of knowledge of either the game engine itself or the computer languages used to write the engine and associated modules. Above the game code lies the virtual worlds of the game. A given game will come with 10 to 20 levels, and the game developers usually provide software that will allow the end user to write his or her own levels.

2.3.5 Unreal Tournament

Unreal Tournament (UT) is a multi-player network-based video game for personal computers. It runs on various operating systems, including Microsoft Windows, Linux, MacOS, and even the game consoles Sony Playstation and Microsoft XBox. Unreal Tournament provides a high resolution simulation environment at a very low price, that can run on commodity personal computer hardware.

Like most other modern video games, Unreal Tournament is designed to be easily programmable. The end user is able to modify easily most parts of the game above the actual rendering subsystem, to both manipulate default game behavior and to supplement the game with their own changes. These include both scenery and players.

2.4 GameBots

GameBots is a modification to the UT video game that allows control of game players through a normal Transmission Control Protocol and Internet Protocol (TCP/IP) socket (Adobbati, Marshall, Scholer, Tejada, Kaminka, Schaffer, and Sollitto 2001). Even though UT is designed to be played over a network, the actual protocol used by the game is undocumented and encrypted to prevent cheating. This makes the protocol unusable by third-party developers. GameBots overcomes this problem by simply side-stepping it. GameBots talks to the game engine directly, and opens its own networking sockets. A protocol for interacting with UT is defined in (GameBots Website).

With a simple text-based TCP/IP protocol, we are able to create and to manipulate players in an UT instance using GameBots. The real bonus of this system is that we can use the UT game itself to view, in full real-time detail, the appearance and movement of these GameBot players in the game.

3 AN AGENT-BASED UTSAF ARCHITECTURE AND DESIGN

Our proposed framework of an agent-based UTSAF architecture is composed of several types of agents residing in a multi-agent environment that is used to communicate between an ModSAF/OneSAF simulation world and the UT simulation world. Figure 2 shows the architecture of our framework based on different kinds of agents.

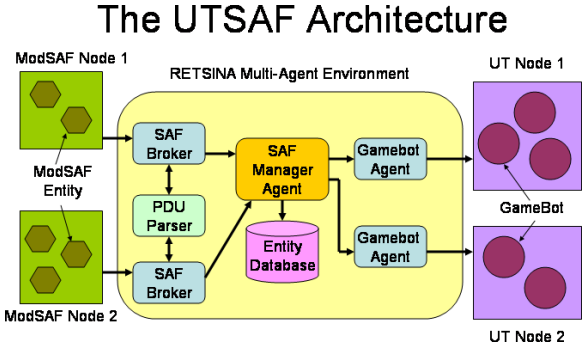

Figure 2: UTSAF System Architecture

In a multi-agent environment, agents have different tasks and are encouraged to communicate with other kinds of agents to get additional information that it does not maintain. This enables the multi-agent system to be scalable. In our framework, we use three different kinds of agents to bridge military simulation to game simulation. These agents are SAF Broker agents, SAF Manager agents, and GameBot agents. Later in this section, we also discuss the communication protocol that are used in our framework. Additionally, we discuss the modification to bridge a military simulation to a 3D virtual environment.

3.1 SAF Broker Agent and DIS Parser

Figure 2 shows the SAF Broker listening to DIS network traffic on a OneSAF simulation node over a multicast group. By using the DIS Parser, it takes each DIS PDU, parses out the relevant information (entity type, location, velocity, and orientation), and sends this information into a SAF Manager agent. The SAF Broker is able to listen to multiple ModSAF/OneSAF simulations on a single multicast group, giving the user the ability to only view entities from a single simulation.

3.2 SAF Manager Agent

Figure 3 shows the SAF Manager in action. The SAF Manager agent manages information about each entity between a ModSAF/OneSAF simulation world and a UT simulation world. After receiving information from the SAF broker, it updates each entity information in its database and passes this information to the GameBot agent that is linked to the entity. From the SAF Manager interface in Figure 3, we are able to view information of each entity such as location and its status.

3.3 GameBot Agent

Figure 4 shows a GameBot agent in action. GameBot agents are used to link the agent space to the GameBots in UT

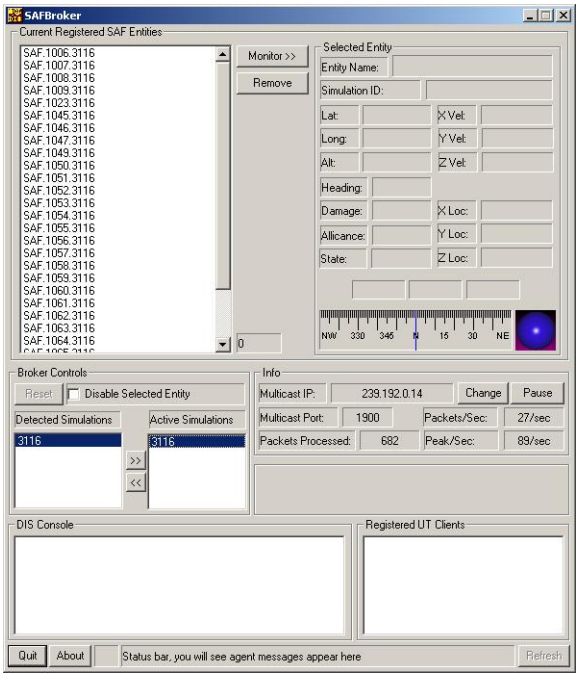

Figure 3: SAF Manager User Interface

nodes. GameBot agents listen for DIS updates distributed via the agent language from the SAF Manager agent. When an update occurred, the GameBot agent, which is responsible to this update, contacts the associated GameBot residing in the UT node to update its information such as location, orientation, and velocity. A message-based protocol is used for the GameBot agent to control or update the GameBot in a UT server.

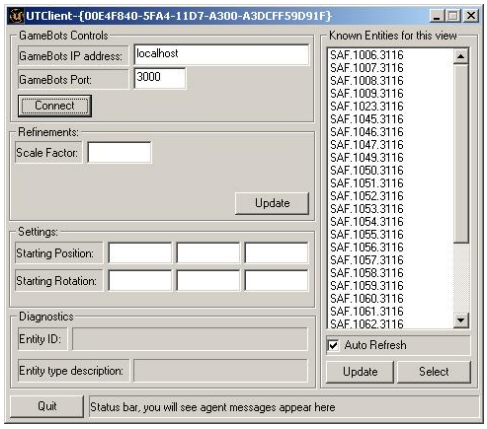

Figure 4: GameBots Agent

All GameBot agents are connected to a single SAF Manager agent, and each GameBot agent connects to all GameBots in one UT server. Many GameBots can be in one UT server. In some cases, it is desirable to have multiple UT servers. This could be the case when there are many dozens of ModSAF/OneSAF entities, and processing the updates via GameBots in one UT server can consume much of a computer's resources. To reduce the computational load, more than one UT servers are used and the GameBots are spread among them. Therefore, more than one GameBot agents are needed, and the number of the GameBot agents should be equal to the number of the UT servers.

3.4 Communication Protocol

To support a heterogeneous simulated environment, a protocol conversion is necessary. ModSAF and OneSAF simulations use a DIS protocol to communicate between each simulator node. The SAF broker listens to the DIS traffic, parses each entity information, and passes it to SAF Manager. A SAF manager forwards only this information to a GameBot agent. The GameBot agent converts this information into a meaningful form, and then uses a GameBots Protocol to transfer this information to a GameBots residing in a UT simulator node. Later in this section, we explain the DIS protocol conversion and the GameBots protocol.

3.4.1 DIS Protocol Conversion

The DIS protocol is purely based on exchanging messages in a standard format described in (IEEE1278 1993). It is a stateless protocol, and messages are always exchanged in a multicast or broadcast group. To communicate with a UT simulation engine, a sequence of protocol conversion must be taken so that we have a form useful for Unreal Tournament.

Location in DIS is defined in terms of the world coordinate system. As such, the zero point is the centroid of the earth, and the axes represent distances from this central point. In Unreal Tournament, the zero point is defined as the center of the level. We first convert the location into latitude and longitude coordinates. Then, we pick a particular latitude and longitude point to be the zero point in Unreal Tournament. The difference between the entity location and the new center point is then scaled and used as the Unreal Tournament entity location.

Entity velocity and acceleration in DIS is defined in meters and seconds. Both are simply scaled before being sent into Unreal Tournament.

The orientation of entities in DIS is specified in terms of Euler angles based on the entity's coordinate system. For ground vehicles these values are converted into roll, pitch, and yaw values and then input into Unreal Tournament, which uses a range from 0 to 65535 for each axis. For aerial vehicles, the entity's velocity was used to compute the orientation instead, as it provided a more accurate representation of the entity's true orientation in real time.

3.4.2 GameBots Protocol

The GameBots network API is defined in (GameBots Website). Incoming sensory messages from GameBots were generally ignored, as ground truth was chosen to be entirely in OneSAF. This also reduced the network and processing load required to generate, transmit, and parse the sensory messages.

Due to the extensive use of dead reckoning in ModSAF, OneSAF, and the DIS protocol, dead reckoning had to be added to the GameBots system. Other changes were made, including the ability to pick entity starting locations and orientations in the Unreal Tournament virtual world and the ability to pick which entity type to appear as.

3.5 A UTSAF Virtual Environment Design

In order to view a military simulation into a 3D virtual world using our UTSAF architecture, several modification to existing game environments is necessary. Later in this section, we explain how we implement UT player models, modify a UT game server, and create a special player for visualization.

3.5.1 Game Modification

None of the required vehicles for representing ModSAF/OneSAF entities were available by default in Unreal Tournament. A collection of 3D models were acquired from Internet sources. Additional models were designed manually, as no appropriate model could be found online. These models were input into the user programming tools provided by Unreal Tournament to create appropriate entities.

3.5.2 GameBot Server

A GameBot server is an Unreal Tournament server that is specifically running a GameBot session. GameBots is an extensive modification to the game, and therefore runs as a special case of the server, and not as a normal UT game server.

3.5.3 Unreal Tournament Spectators

The Unreal Tournament GUI has the ability to connect to a server as a spectator. In this mode, the user can survey the running simulation, but can't actually interact with any of the entities. The game provides a very useful function in spectator mode: a player can "attach" itself to an entity. In this special case, the user's view will move in conjunction with the entity to which the user is attached. This allows a user to attach to an uninhabited aerial vehicle, for example, and to view from the vehicle as it flies around the map.

4 EXPERIMENTAL TESTBED SETUP

For experiments, we set up our test bed in a local area network. We run a OneSAF simulation engine on a Pentium III Linux-based machine. The DIS protocol traffic from this machine is broadcast to a multicast group which includes a machine where RETSINA agents operate.

Figure 5 shows the user interface of OneSAF simulation on a Fort Knox terrain. In this scenario, two flights of A-10 Thunderbolts are flying over a biochemical depot which is surrounded by several T-80 tanks. Figure 6 shows the chemical depot and T-80 tanks. The same simulation scenario in 3D visualization using UT simulation engine is shown Figure 7 where the biochemical depot and T-80 tanks are depicted. This figure shows an example of the view from a spectator attached to an A-10 Thunderbolt flying over the biochemical depot. In the 3D visualization, we are able to change a view of a simulated battlefield using our agent. Figure 8 shows another view of the simulation from a T-80 tank. It shows several A-10 Thunderbolts are flying over the biochemical depot and other T-80 tanks in the biochemical depot area.

By using the UTSAF broker, we are able to reduce network traffic from the OneSAF simulation network to

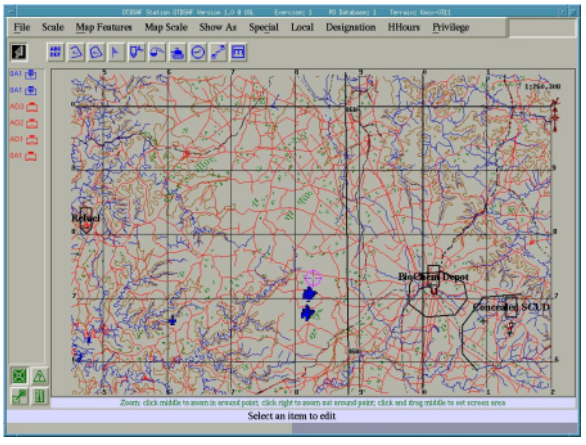

Figure 5: ModSAF

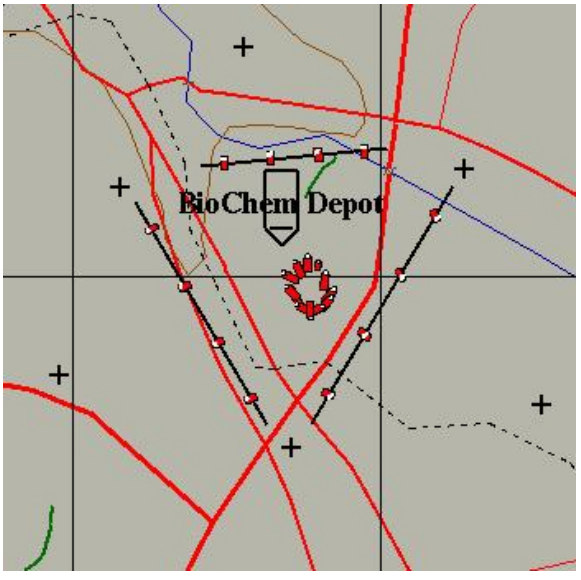

Figure 6: A Biochemical Depot in OneSAF

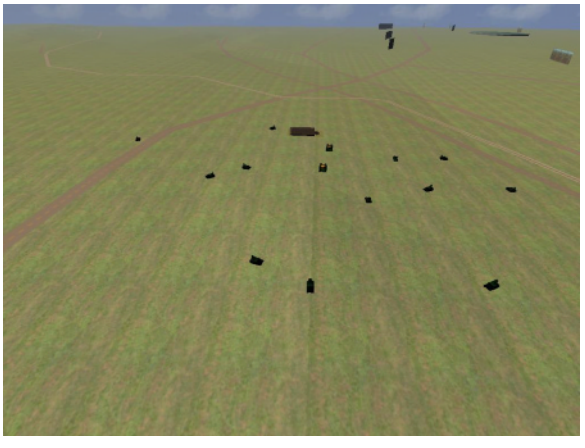

Figure 7: A Biochemical Depot in Unreal Tournament

Figure 8: A-10 Thunderbolts Flying Over a T-80 Tank Platoon

another end. The agent examines the entity information received by DIS PDUs, and simply ignores the PDU if the entity information is not changed.

When running, the UTSAF system will display in 3D a running ModSAF/OneSAF simulation. The movement and appearance of the ModSAF/OneSAF entities are reproduced faithfully in real time. The user is able to both spectate the simulation from any position in the game space, but to also connect their view to any of the entities in the simulation.

5 RELATED WORK

There have been several works related to creating a 3D visualization environment to support military simulation. The DIS-Java-VRML project was created to build a Java-based software suite to support communication using the DIS Protocol for the simulation over the Internet (DIS-Java-VRML Website). A simulation node listens to DIS traffic, and an entity object is viewed in 3D using virtual Reality Modeling Language (VRML). The entity is visualized by a web browser as a viewing tool.

Another project involving modeling a large network of military simulations is the NPSNET project (Macedonia, Zyda, Pratt, Barham, and Zeswitz 1994). The NPSNET project was initiated with a goal of expanding DIS simulations and virtual environments to support more than 1000 entities or players. This work proposed a testbed which utilizes DIS protocol (version 2.03), IP multicast protocol, and several parallelism techniques to reduce traffic in the network.

Schwamb et. al. proposed a program interface for intelligent agents to ModSAF simulation (Schwamb, Koss, and Keirsey 1994). In their work, the interface provides an intelligence to ModSAF to simulate a human behavior using a Soar agent (Rosenbloom, Laird, and Newell 1993). In this work, the Soar agent models a human behavior in a cockpit in an air-to-air combat simulated by ModSAF. The interface and Soar are integrated to ModSAF into a single process. This requires a modification to other ModSAF nodes and also this does not provide multi-agent environments to support heterogeneous intelligence.

Another work related to simulating ModSAF in heterogeneous environment is an integration of UH-60 and Ch-47 flight simulators to ModSAF simulation (Sardella and High 2000). This work proposed an architecture to integrate the flight simulators in a ModSAF simulation world using the DIS protocol.

6 CONCLUSION AND FUTURE WORK

In this work, we proposed an architectural framework for visualizing a DIS-based military simulation such as ModSAF in 3D virtual world using an Unreal Tournament game simulation. Our framework does not require any modification to ModSAF or OneSAF application. Instead, we introduce a multi-agent system (MAS) to support multiple ModSAF simulations to solve several problems. We use our agent to reduce DIS traffic among simulation nodes so this framework is able to support a large DIS-based simulation network requiring a fair amount of network bandwidths. Secondly, our MAS-based framework can be extended to support intelligent operations by introducing an intelligent agent to aid in user's operations such as time-critical tasks. Lastly, our framework uses a UT game simulator which is fast, efficient, and affordable. It also provides flexibility for modification to visualize different views in 3D virtual environments.

There remains much work to be done before UTSAF is a fully viable system. Most of the problems that the system currently faces stem from the Unreal Tournament video game and not ModSAF or the agent system. Unreal Tournament currently has limits on both the possible map size and the number of entities allowed in a given simulation.

To simulate very large ModSAF/OneSAF areas in Unreal Tournament, beyond the allowed map size in the game, we are currently investigating ways of dividing up the map into subunits. Each sub-map can be run on an individual game server, and then linked together to create a seamless representation of the overall map. Unreal Tournament itself does not support this feature, however, and a solution will have to be found outside of the video game itself.

GameBots currently has a preset limit of 16 entities that it will allow. This was easily extended to 64 by simply resetting this limit. Higher limits may be possible, and experimentation to determine the upper limit actually allowed by the game is being conducted.

A major problem that exists in UTSAF now is latency. The time from when a PDU is emitted by ModSAF or OneSAF and when the contained information enters Unreal Tournament is both significant (on the order of 1 or 2

seconds) and variable. It is felt that this is probably related to both the agent system, and overall network latency.

One final issue with UTSAF is user disorientation. A ModSAF or OneSAF map has both grid lines and a compass to orient users. UTSAF currently lacks both. Active research is being conducted into using the heads-up display provided by the game to provide these features.

ACKNOWLEDGMENTS

This project is supported by AFOSR contract F49640-01-1-0542. The authors would like to thank Martin van Velsen, and the entire Carnegie Mellon University RETSINA team.

REFERENCES

Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S. Kaminka, G., Schaffer, S., and Sollitto, C. 2001. GameBots: A 3D Virtual World Test-Bed For Multi-Agent Research. In *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*.

Brunett, S., and Gottschalk, T. 1997. An Architecture for Large ModSAF Simulations using Scalable Parallel Processors. *Technical Report CACR-155*. California Institute of Technology. October 1997.

Calder, R.B., Smith, J.E., Courtemarche, A.J., Mar, J.M.F., Ceranowicz, A.Z. 1993. ModSAF Behavior Simulation and Control. In *Proceedings of the Second Conference on Computer Generated Forces and Behavioral Representation*.

DIS-Java-VRML Website. Available via <http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/> [accessed June 20, 2003].

Epic Game Developers. Unreal Tournament Website. Available via <http://unreal.epicgames.com/> [accessed June 20, 2003].

GameBots Website. GameBots Network API. Available via <http://www.cs.cmu.edu/~galk/GameBots/WEB/docapi.html> [accessed June 20, 2003].

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

Lewis, M. 2002. The New Cards. *Communications of the ACM, January 2002*.

Lewis, M. and Jacobson, J. 2002. Game Engines in Scientific Research. *Communications of the ACM, January 2002*.

Macedonia, M.R., Zyda, M.J., Pratt, D.R., Barham, P.T., and Zeswitz, S. 1994. NPSNET: A Network Software Architecture For Large Scale Virtual Environments. In *Presence* 3(4): 265-287.

OneSAF Website. Available via <http://www.onesaf.org> [accessed June 20, 2003].

Rectenwald, M. 2002. *RETSINA AFC Developers' Guide*. Available via <http://www-2.cs.cmu.edu/~softagents/afc/> [access June 20, 2003].

Rosenbloom, P.S., Laird, J.E., and Newell, A. 1993. The Soar Papers: Readings on Integrated Intelligence. Cambridge, MA. MIT Press.

Sardella, J.M., and High, D.L. 2000. Integration of Fielded Army Aviation Simulators with ModSAF: The Eighth Army Training Solution. In *Proceedings of Interservice/Industry Training Systems and Education Conference*.

Schwamb, K.B., Koss, F.V., and Keirsey, D. 1994. Working with ModSAF: Interfaces for Programs and Users. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavior Representation*. Orlando, Florida, USA. Institute for Simulation and Training. University of Central Florida.

SoftAgents Website. Available via <http://www-2.cs.cmu.edu/~softagents/> [accessed June 20, 2003].

Stone, S., Zyda, M., Brutzman, D., Falby, J. 1996. Mobile Agents And Smart Networks For Distributed Simulations. In *Proceedings of the 14th DIS Workshop*. Orlando, Florida.

Sycara, K.P. 1998. Multi-Agent Systems. *AI Magazine* 10(2): 79-93.

Sycara, K.P., Paolucci, M., van Velsen, M., Giampapa, J. 2001. The RETSINA MAS Infrastructure. Technical Report CMU-RI-TR-01-05. Robotics Institute, Carnegie Mellon University.

Zyda, M., Hiles, J., Mayberry, A., Wardynski, C., Capps, M., Osborn, B., Shilling, R., Robaszewski, M., Davis, M. 2003. The MOVES Institute's Army Game Project: Entertainment R&D for Defense. *IEEE Computer Graphics and Applications*. January/February 2003.

AUTHOR BIOGRAPHIES

JOSEPH MANOJLOVICH is a graduate student at the School of Information Sciences at the University of Pittsburgh. His interests include using game engines for scientific research. His e-mail address is <josephm@sis.pitt.edu>.

PHONGSAK PRASITHSANGAREE is a doctoral student at the School of Information Sciences at the University of Pittsburgh. His research interests are position location system, networking protocols, and wireless network security. His e-mail address is <phongsak@sis.pitt.edu>.

STEPHEN HUGHES is a doctoral student at the School of Information Sciences at the University of Pittsburgh. His research interests include human navigation in 3D environments. His e-mail address is <shughes@sis.pitt.edu>.

JINLIN CHEN is a visiting professor in School of Information Sciences, University of Pittsburgh. Before joining University of Pittsburgh, he was a researcher in Microsoft Research Asia. He received his Ph.D. from Department of Automation, Tsinghua University in 1999. His e-mail address is <jlchen@sis.pitt.edu>.

MICHAEL LEWIS is an associate professor at the School of Information Sciences at the University of Pittsburgh. He is a principle investigator of Usability Laboratory. His email address is <mlewis@sis.pitt.edu> and his website is <http://www.pitt.edu/~cmlewis>.