UTSAF: A Multi-Agent-Based Software Bridge for Interoperability between Distributed Military and Commercial Gaming Simulation

Phongsak Prasithsangaree
Joseph Manojlovich
Stephen Hughes
Mike Lewis
Department of Information Sciences and Telecommunications
University of Pittsburgh
135 N. Bellefield Avenue, Pittsburgh, PA
phongsak@sis.pitt.edu

Rapid advances in consumer electronics have led to the anomaly that consumer off-the-shelf gaming hardware and software provide better interactive graphics than military and other specialized systems costing orders of magnitude more. UTSAF (Unreal Tournament Semi-Automated Force) is bridging software written to take advantage of the power of gaming systems by allowing them to participate in distributed simulations with military simulators. UTSAF illustrates the use of multiagent technology to flexibly interconnect otherwise incompatible systems. This article describes an architectural approach for rapidly constructing middleware by taking advantage of built-in capabilities for processing, communication, and interoperation that a multiagent infrastructure provides. Several software agents based on Reusable Environment for Task-Structured Intelligent Networked Agents (RETSINAs) are used to support interoperability between military simulation nodes based on distributed interactive simulation and Unreal game simulators. Using a multiagent system, UTSAF can be expanded to support several network environments and interact with other agent-based software.

Keywords: Semi-automated force, Unreal Tournament, simulation interoperability, multiagent system, game simulation

1. Introduction

Distributed simulation has long been used in military applications for simulated battlefield training and strategic planning. Distributed simulations often involve different systems such as the core simulation engine, terrain database, synthetic theater, computer-generated forces, command-and-control center, decision support systems, and more. The distributed simulation may also involve different operators, agencies, or partners. Due to this heterogeneity, interoperation among systems is both critical and problematic. From the beginning, a major problem for distributed simulation has been defining a standard for interoperability. The SIMulator NETwork (SIMNET) project was begun in 1984 [1] to network different military simulation systems, using information units called protocol data units (PDUs) to describe typed data structures to be exchanged between the systems. Building on SIMNET protocols, a family of distributed interactive simulation (DIS) protocol standards for PDU exchange was later defined and published by the Institute of Electrical and Electronics Engineers (IEEE) in 1993 [2] and became the basis for interoperability among networked military simulations. By using the DIS protocol, military simulations could be extended to a very large network of simulation nodes [3].

However, DIS has several problems. First, the DIS PDUs transmitted between simulation nodes tend to overwhelm network bandwidth [4]. Since the DIS protocol works in a stateless manner, all information for each entity and event is transmitted to other simulation nodes, even though there may be no status changes in those entities or events. A second problem of DIS simulation is lack of a common and affordable three-dimensional (3-D) simulated environment. Using DIS for training requires very expensive customized systems such as vehicle or dismounted infantry simulators. Last, the DIS environment was originally built to link simulation nodes together but does not provide the additional information or interfaces needed for interoperating with other operational systems, such as a decision-making, a human behavioral, or a situation awareness module.

DOI: 10.1177/0037549704050907

To address some of these deficiencies of DIS and further increase interoperability among simulation systems and reusability of simulation components, the Defense Modeling and Simulation Office (DMSO) developed the High-Level Architecture (HLA) specification in 1996. IEEE later published HLA in 2000 as an open standard to increase interoperability for both military and nonmilitary simulations. However, legacy simulation systems that are not HLA compliant cannot be used in HLA-based federations. To overcome the interoperability problem among non-HLA-compliant systems, several mitigation approaches have been proposed. However, there are still several issues that have slowed HLA adoption by the simulation community. First, although several commercial off-the-shelf (COTS) simulation products now support HLA, they are not yet mature in terms of interoperation between products of different vendors [5]. Second, while there are several COTS products that use a gateway to bridge legacy systems to HLA, the gateway poses restricted access to some HLA capabilities [6]. There are also compatibility issues arising from limitations in programming language, operation systems, and platforms for available HLA-based products.

Recently, there has been increasing interest in forms of interoperation not supported by HLA [7] such as between modeling and simulation (M&S) systems and Command, Control, Communication, Computers, and Intelligence (C4I) systems. A new framework called XMSF (Extensible Modeling and Simulation Framework) has been proposed to solve this interoperability problem [8]. The XMSF is based on the idea of integrating Web services into M&S using open standards such as the Extensible Markup Language (XML) and the Simple Object Access Protocol (SOAP). The XMSF provides a common interface to the C4I systems, which is similar to that recently proposed for the C4I architecture, called NCES (Network Centric Enterprise Services) [9].

The work to standardize a framework for M&S and C4I interoperability is just one of many efforts to interconnect military information systems designed to operate in isolation. In this and a companion paper [10], we propose an architecture for interoperability among heterogeneous systems. The proposed architecture uses a multidomain, multiagent system (MAS) in which each agent performs a specific task and interacts with other agents through a standard infrastructure to achieve interoperability between otherwise incompatible systems. For example, an agent could be used as an interface to provide data from DIS protocol communications to an information-filtering agent that passes on only task-relevant information in the agent communication language to a decision-making agent. The agents are built using the Reusable Environment for Task-Structured Intelligent Networked Agent (RETSINA) environment, which has been widely used in many applications, including serving as the primary interoperator for the Coabs's grid [11].

In this article, we demonstrate this architectural approach by describing an implemented system, UTSAF, which integrates a military simulation, OneSAF Testbed Baseline (OTB), with a 3-D game engine, Unreal Tournament (UT), to provide a low-cost, high-quality solution to the problems of stealth viewing and simulation for human task allocation studies requiring 3-D visualization. UTSAF was developed to support human-in-the-loop studies by simulating remote video feeds and driving immersive panoramic displays as part of a study of high-level information fusion [12, 13]. Since the system provides a bridge between the OTB simulation and UT game engine, we have called it the Unreal Tournament Semi-Automated Force (UTSAF).

UTSAF illustrates an agent-based divide-and-conquer strategy for interoperability between heterogeneous systems. Use of an MAS allows processing to be distributed among a group of agents with preexisting infrastructure for communication and coordination. Agents act as wrappers around the incompatible systems and use this communication and coordination infrastructure to provide an abstraction layer for exchanging data and messages. In our implementation, the SAF manager agent plays the role of the HLA Federation Object Model (FOM) by specifying the semantic correspondence among wrapped systems. Because system-specific processing is encapsulated at the wrapper level, new systems can be added simply by writing additional wrappers and specifying correspondences to the SAF manager. This MAS approach allows rapid development of interoperability middleware for systems not adhering to interoperability standards, such as HLA, without requiring any modification to these systems themselves.

UTSAF solves practical problems as well. ModSAF, OTB's precursor, was originally developed as a large-scale, special-purpose military simulation. Over the past decade, rapid increases in processing power have allowed it to migrate from expensive workstations to inexpensive Linux PC systems. While ModSAF and its descendents have often been used [14] as servers for maintaining ground truth for realistic task simulators, they provide only a map-like graphical user interface (GUI) for user interaction. Software for generating the realistic 3-D views of the simulated battlefield needed for training and research has remained expensive and proprietary. UTSAF solves this problem by generating high-quality 3D graphics using a COTS game engine on a commodity-priced PC.

2. Background

In this section, we first explain what an MAS is and how we can deploy an MAS-based architecture to offer interoperability between heterogeneous military simulation systems. Then, we provide some background for the military simulator, ModSAF, and its related communication protocol, the DIS protocol. We introduce the essential components of a game engine and the use of the GameBots modification to control entities in the game simulation.

2.1 ModSAF, OneSAF, and DIS Protocol

One of the most popular ground forces modeling simulations is ModSAF, which was released as a joint effort between the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Simulation, Training, and Instrumentation Command (STRICOM) in 1993 [15]. ModSAF consists of SAFsim and SAFstation processes that communicate with each other and with other distributed applications by sending PDUs that describe the current state of entities being modeled across a network. Each SAFsim and SAFstation reads PDUs from the network and places processed message data into its own state database. The "semi-automated forces" of the simulation's name are vehicles and troops that have been programmed to respond to simulated events in accordance with rules of engagement. So a tank, for instance, might fire upon an enemy vehicle if it entered the tank's field of view. A new HLA-compliant, general-purpose SAF simulation, OneSAF Objective System, is scheduled to replace ModSAF with an accompanying change in terrain format from the Compact Terrain Database (CTDB) to the Synthetic Environment Data Representation and Interchange Specification (SEDRIS). The UTSAF middleware described in this article was developed to interoperate with an interim version of the simulation, OTB (http://www.onesaf.org), that retains ModSAF protocols and formats.

ModSAF was created as a DIS-based simulation. Each simulator communicates with one another by using the DIS protocol standardized by IEEE [2]. 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. The messages are in the form of PDUs. The primary PDUs used to convey entity interactions are the Entity State PDU (ESPDU), Fire PDU (FPDU), and Detonation PDU (DPDU). The ESPDU contains information about the entity's current status, including its type, location, orientation, velocity, and articulations. Most of the network traffic between simulation nodes is ESPDU messages. The FPDU is used to convey information about the firing of a weapon. The FPDU contains munitions type, a firing entity, a target entity, a launching location, and munitions velocity. The DPDU is used to convey information about the impact of munitions. The DPDU contains a detonation location, a detonation result, and information contained in the FPDU. More details about PDUs can be found in IEEE [2]. Distributed simulators can synchronize their models of the world by modifying their databases to reflect the PDUs they receive and sending PDUs in response to changes within their local simulation.

2.2 Game Engines and Simulation

Both military simulation and computer games have a long and related history but with different foci. While military simulations concentrate on high-fidelity validation and verification (e.g., trying to reproduce as closely as possible real-world effects such as the damage associated with a projectile), video games focus on pushing computer hardware to its limits to produce graphically rich simulations. In many areas, the cutting edge of computer development in both software and hardware is now being driven by the needs of large audience games [16]. The most powerful graphics-processing units, for example, are now found on game-oriented video cards rather than the refrigerator-sized graphics workstations of the 1990s. In 2002, the $10.3 billion in U.S. revenues for video games exceeded that from movies ($9.27 billion) for the first time. In this high-volume competitive market, the cost of developing ever more compelling simulations has grown so huge that even game developers can no longer rely on recouping their entire investment from a single game. This has led to the emergence of game engines, modular simulation code written for a specific game but general enough to be used for a family of similar games. This separation of function from content now allows game engines to be used as general-purpose, high-fidelity simulators [16].

The game's engine refers to the collection of modules of simulation code that do not directly specify the game's behavior (game logic) or game's environment (level data). The engine includes modules handling input, output (3-D rendering, 2-D drawing, and sound), networking, and generic physics and dynamics. The levels define 3-D environments using open data formats accessible to standard 3-D modeling tools. The game code handles most of the basic mechanics of simulation, including simple physics, display parameters, networking, and the base or atomic-level actions for animations, and can be modified using a game-specific scripting language. Multiplayer games use a client-server architecture in which the server maintains the reference state of the simulation while clients perform the complex graphics computations needed to display their individual views.

Unreal Tournament (UT) is a multiplayer, network-based video game for personal computers (http://unreal.epicgames.com). It runs on various operating systems, including Microsoft Windows, Linux, MacOS, and even the game consoles Sony Playstation and Microsoft XBox. The game engine used by UT provides a high-resolution simulation environment at a very low price, which can run on commodity personal computer hardware. Like most other modern video games, UT is designed to be easily programmable and highly modularized. End users are 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. The Unreal engine is arguably the best-suited game engine for general simulation because of its clean, object-oriented architecture and easy-to-use end-user tools and documentation. Recent work at the MOVES Institute at the Naval Postgraduate School has produced the America's Army video game, using the Unreal engine to realistically portray life as a new U.S. Army recruit [17]. The Unreal engine has also been used as an interface to human behavior modeling in the Soar architecture [18] for research in multiscreen displays [19, 20], architectural re-creations [21], and as a testbed for studying human–robot interaction [22].

There are two possible approaches to adapting the UT game engine for simulation: licensing the engine or modifying the UT game to provide an interface for simulation. Licensing the engine provides full access to source code and the ability to modify rendering and networking protocols but costs in the range of $250,000 to $500,000 because of the high commercial potential of successful games. This approach gives the most flexibility but is extremely expensive and does not provide migration to new releases of the engine. Modifications can be made using the game's scripting language for free under a GNU public license (GPL) by which anyone is allowed to modify the game, provided the modifications are freely distributed. These modifications can be wide ranging and include the ability to provide an interface for controlling entities within the game.

UTSAF uses game modifications to provide interoperability between OTB and the Unreal engine without modifying the underlying code. Subsequent efforts to provide interoperability between SAF simulations and Unreal have relied on licensing and modifying the game's engine. A military simulation vendor, MÄK, reported in a 2003 press release [23] that it had licensed the engine and successfully integrated the Unreal game engine Game Developer's Toolkit with its VR-Link product. A more recent effort at the Institute for Creative Technologies, University of Southern California (http://www.ict.usc.edu/disp.php?bd+proj_ia) is including the Unreal engine in a suite of applications being linked to the new OneSAF Objective System simulation. Although it would be possible to provide interoperation without modifying the Unreal engine's virtual machine using another architecture, the encapsulation and abstraction provided by agent wrappers makes the task much easier using an MAS.

UTSAF follows the modification route by using the GameBots [24] modification to provide the simulation interface. UT has two types of entities: human players who run individual copies of the game and connect to the server (typically running on the first player's machine) and "bots" (short for robots), simulated players running simple reactive programs. GameBots is a modification to the UT game that allows bots to be controlled through a normal Transport Control Protocol/Internet Protocol (TCP/IP) socket. GameBots talks to the game engine directly and opens its own networking sockets. A protocol for interacting with UT is defined in the GameBot Web site (http://gamebots.sourceforge.net). With a simple text-based TCP/IP protocol, we are able to create and manipulate players in an UT instance using GameBots. UTSAF uses GameBots to update Unreal entities in accordance with the PDUs it receives.

3. UTSAF Architecture

Our example architecture is based on using software agents to solve the interoperability problem. The architecture uses an MAS in which each specific task is assigned to each type of agent. Because of their common communication infrastructure and language, agents can freely exchange information with one another. Thus, the architecture is extensible and able to support novel systems mediated by new agents. In this section, we describe the MAS architecture and discuss our proposed MAS-based framework for interoperability among C4I systems and distributed simulation.

3.1 Multiagent System (MAS)

A single intelligent agent is created to solve only a particular, small problem due to its limited knowledge, its computing resources, and its perspective. However, in a more complex, heterogeneous system, a single agent may not perform its task well. Multiagent systems were introduced to offer modularity as a tool to handling complexity and heterogeneity [25]. In the MAS, each agent is implemented to use the most appropriate paradigm to solve a specific problem. Having several agents, each with their own paradigm working together, increases the capability for solving larger, more complex, and more dynamic problems than a single monolithic agent could handle. Because MAS agents are created with a common architecture, they share communication modules and a common language, allowing them to address problems of an open system where other agents come and go or a heterogeneous network where agents mediate among legacy, modern, or dynamic systems. Thus, MAS is an appropriate tool to deal with integrating heterogeneous systems such as military simulation and command-and-control systems in which (1) each system uses its own standard or is proprietary, (2) having a centralized database is not possible, or (3) computation is asynchronous.

3.2 RETSINA Multiagent Infrastructure

RETSINA provides a domain-independent, componentized, and reusable substratum to (a) allow heterogeneous agents to coordinate in a variety of ways and (b) enable a single agent to be part of a multiagent infrastructure. To this end, RETSINA provides facilities for reuse and a combination of different existing low-level infrastructure components, and it also defines and implements higher level agent services and components that are reconfigurable and reusable. Low-level categories include communication protocols, such as TCP/IP, Hypertext Transfer Protocol (HTTP), and Wireless Access Protocol (WAP); generic application programming interfaces for relational database access and query, such as Java Database Connection (JDBC), Simple Query Language for Java (SQLJ),

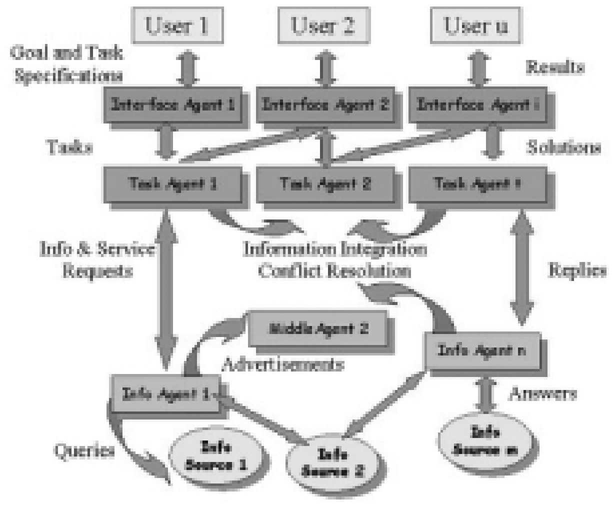

Figure 1. The multiagent system (RETSINA 1999)

Open Database Connection (ODBC), and Open Knowledgebase Connection (OKBC); distributed framework facilities, such as Java Remote Message Interface (RMI); and security services, such as Secure Socket Layer (SSL) and Public Key Infrastructure (PKI). On top of such services and components, RETSINA implements agent-level services such as distributed Agent Name Servers (ANS), the RETSINA ANS; agent communication languages, such as Foundation for Physical Agent (FIPA), Agent Communication Language (ACL), and Knowledge Query Model Language (KQML); and mechanisms for peer-to-peer communications through the abstractions in the RETSINA *communicator*. Figure 1 shows an example of MASs in which each agent communicates with each other to perform a specific task and to achieve a single goal.

3.3 Single-Agent Infrastructure

Each individual RETSINA agent consists of agent *reusable and reconfigurable components* (i.e., a set of generic software components for knowledge representation, agent control, and interaction with other agents). Each RETSINA agent consists of several modules, each of which is implemented as multithreaded code. The main modules of a RETSINA agent are as follows: *communicator*, *planner*, *scheduler*, and *execution monitor*. The modules can operate asynchronously and concurrently. For example, as the planner module is engaged in planning, the communicator module can receive and process messages. This architecture allows RETSINA agents to keep up with information assessment, planning, and replanning in the face of a rapidly changing and dynamic environment.

Communicator. This module provides an abstraction that supports peer-to-peer communication between agents based on the agent names. The default communication language is KQML, and the messages are transmitted through TCP/IP sockets. The communicator has application programming interfaces (APIs) for other agent languages (such as FIPA) as well.

Planner. The planner receives goals through communication messages and finds alternative ways to fulfill them. The planning component is reusable and capable of accepting different planning algorithms.

Scheduler. The agent's scheduling component takes as input the agent's current set of task structures, particularly the set of current executable actions, and schedules them. The default scheduling algorithm uses the earliest-deadline-first heuristic. A list of all actions is constructed (the schedule), and the action with the earliest deadline is chosen for execution. When a periodic action is chosen for execution, it is reinstated into the schedule with a deadline equal to the current time plus the action's period.

Execution monitor. The execution monitor takes the agent's next intended action and prepares, monitors, and completes its execution.

Creating a new instance of an agent to interact with a new information source such as OTB requires that the agent be provided with a schema definition, desired domain concepts, their mapping to data types, and which ones are selectable as inputs or outputs of a query. In addition, a source-specific function, the *external query function*, must be written to translate inputs and outputs into formats compatible with the information source. All other agent components are parts of the agent infrastructure reused between different agent instances.

3.4 An MAS-Based Architecture for C4I and Simulation Interoperation

In heterogeneous systems such as military simulation, command and control, and operation and planning systems, a multi-agent-based architecture can provide a general solution to the integration and interoperability problem. Rather than declaring a common standard such as HLA to which all participants must adhere, an MAS uses specialized agents to wrap each participant, letting the MAS infrastructure serve as the medium for interoperation. Because of this generality, an MAS could join legacy DIS with newer HLA- and XMSF-based systems without requiring internal modifications to the systems themselves by serving as a bridge between legacy systems to mitigate incompatibilities, as described in Tolk [9].

As shown in Figure 2, each component, such as a human behavior model, connects to an agent domain to transfer its tasks to agents inside the domain. In each agent domain, there are several types of agents. An *interface agent* communicates with a system through the DIS, HLA, or XMSF interface. On the other side, the interface agent is connected to one or many *task agents* to perform a task such as language/protocol interpretation. The task agent may translate a DIS protocol into a common language such as Battle Management Language (BML) [26] or vice versa for direct

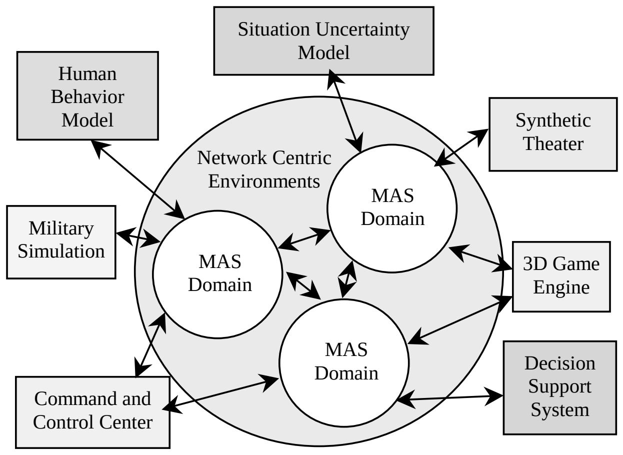

Figure 2. The multidomain, multiagent framework for integration and interoperability

communication to a C4I system. The task agent may communicate with an *information agent* to access databases before or after an information processing.

In each domain, agents may communicate using a language such as BML; thus, systems using the same language can connect to each other in the same domain to reduce overhead and to increase scalability. Using the multidomain architecture, communication due to unnecessary information that is only relevant to a particular domain can be reduced to increase the scalability of the whole system. Inside a domain, scalability can be achieved through the use of social interaction among agents, in which each agent performs one task and contacts other agents to achieve an overall goal.

Interoperability is also achieved using the multidomain, multiagent architecture. Similar systems (e.g., systems that employ the same communication language) are grouped together and connected to the same domain. Between domains, a common agent language such as KQML is used, and an interpreter agent is implemented to translate KQML into the local domain language. With M systems that speak N communication languages, only O(N) interpreter agents are needed instead of O(MN).

4. UTSAF ARCHITECTURE

As a demonstration of the integration and interoperability possible within MAS architectures, we have implemented a middleware system, UTSAF, which serves as a bridge between the OTB military simulation and the Unreal game engine. As shown in Figure 3, a multiagent environment is used to span the gap between the two simulations. In an effort to keep the system scalable, diverse, and extensible, this architecture allows subtasks to be assigned to specialized agents. A communication protocol between agents is

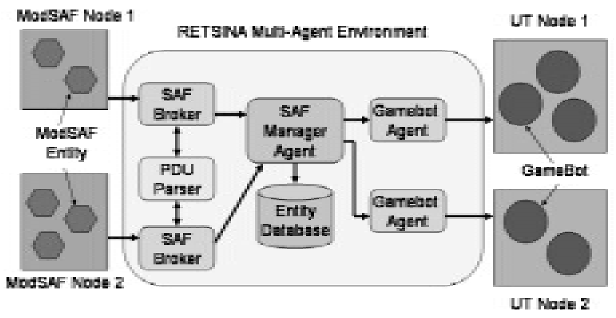

Figure 3. The UTSAF (Unreal Tournament Semi-Automated Force) system architecture

necessary to allow tasks and data to be distributed. The assignments of each of the agents as well as the communication protocol are discussed in detail below.

4.1 Agent Tasks

SAF brokers listen to DIS network traffic from an OTB-SAF simulation node over a multicasting group. To get information from the traffic, a PDU parser extracts relevant information (entity type, location, velocity, and orientation) from each DIS PDU and sends this information to a SAF manager agent. The SAF broker is able to listen to multiple OTBSAF simulations on a single multicast group, giving the user the ability to only view entities from a single simulation.

SAF manager agents control the flow of information between OTBSAF and UT at the entity level. These agents receive information from an SAF broker, update an internal database, and forward relevant updates to an agent representing the entity on the UT side of the simulation. The SAF manager interface (shown in Fig. 4), allows the operator to view entity-specific information such as its location, velocity, and other state variables.

A GameBot agent serves as the final connection between the agent space and the UT representation of the entities. GameBot agents await updates from SAF manager agents. When triggered, the GameBot agent manipulates the properties of the appropriate GameBot within a UT node. Figure 5 shows the interface used to track the communication between the GameBot agent and its associated GameBot.

Although each GameBot agent connects to all GameBots running on a UT game server, it may still be beneficial to employ multiple GameBot agents. For example, if several dozen OTBSAF entities are being tracked, processing the updates via GameBots on a single UT server can cause a bottleneck in the computer’s resources. Fortunately, the GameBots can be distributed over multiple UT servers, spreading the computational load over multiple machines. In this scenario, a GameBot agent would be required for each UT server. This does not affect the rest of the agent

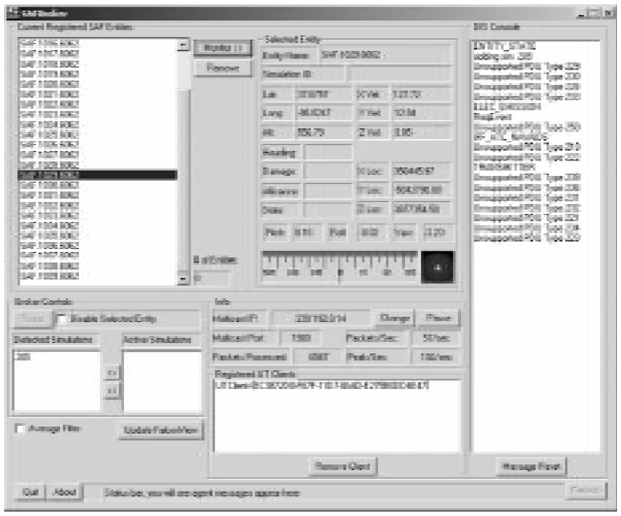

Figure 4. The SAF manager agent graphical user interface

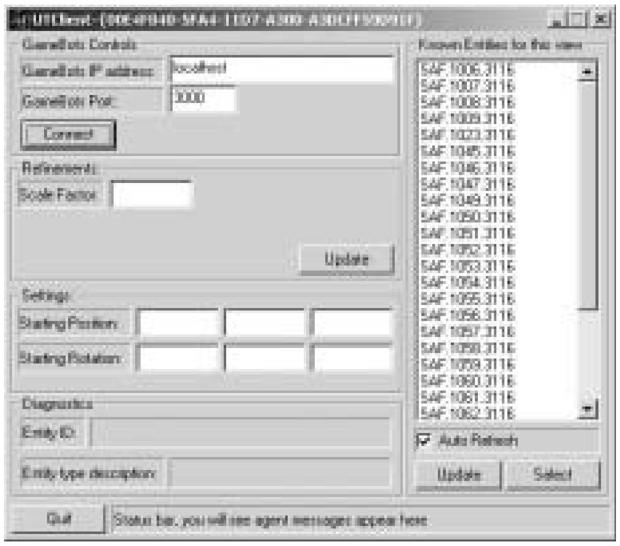

Figure 5. The GameBot agent graphical user interface

architecture; several GameBot agents may connect to a single SAF manager agent.

4.2 Communication Protocol Translations

OTBSAF simulations use the DIS protocol to communicate between each simulation node; entities in UT are manipulated via the GameBot protocol. To bridge these heterogeneous simulated environments, a protocol conversion is necessary. Fortunately, both protocols are well documented. The DIS protocol is purely based on exchanging messages in a standard format described in IEEE [2]. Likewise, the GameBot network API is defined on the GameBot Web site (http://gamebots.sourceforge.net). However, there are differences that must be addressed to ensure compatibility.

The first difference is the coordinate system used to track the location of entities. In DIS, location 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. To perform a conversion, the world location is first translated into latitude and longitude coordinates. Then, a specific latitude and longitude point is selected 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.

The systems also differ on the units used to measure entity velocity and acceleration. DIS uses meters and seconds, while UT worlds are modeled in terms of arbitrary "unreal units" (1 uu $\approx$ 1.9 cm), and time is an adjustable setting for UT simulations. 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 65,535 for each axis. For aerial vehicles, the entity's velocity is used to compute the orientation instead, as it provides a more accurate representation of the entity's true orientation in real time.

4.3 UTSAF Virtual Environment Design

To view a military simulation in a 3-D virtual world using our UTSAF architecture, several modifications to the existing game environment were necessary. First, we needed to create an entity in the game simulation to represent each entity in the military simulation because none of the required vehicles for representing OTBSAF entities was available by default in Unreal Tournament. A collection of 3-D models was acquired from Internet sources. Additional models were designed manually if no appropriate model could be found online. These models were entered into the user programming tools provided by Unreal Tournament to create appropriate entities. This may be time-consuming at first, but once a model (e.g., a ground vehicle model) is created, it can be reused to create other ground vehicles.

Second, we needed to translate the terrain stored in the CTDB format for OTBSAF into the terrain format used by Unreal. This was accomplished through a translation chain using Database Automatic Re-use Technology (DART) utility (http://www.terrex.com) to convert CTDB terrain to the OpenFlight format (http://www.multigen-paradigm.com/products/standards/ openflight/index.shtml), NuGraf (www.okino.org) to convert OpenFlight to the 3ds format, and finally 3ds2unr (http://unreal.epicgames.com/Models.htm) to convert into the Unreal format. Third, we

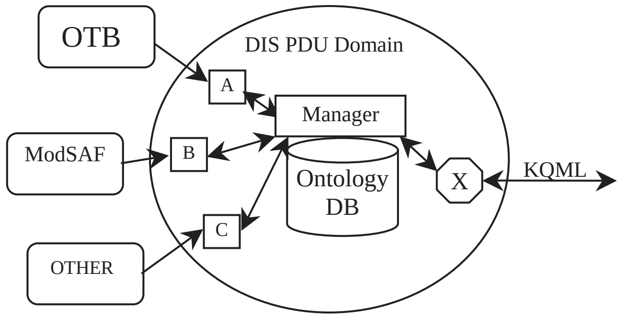

Figure 6. Sample architecture of a multidomain, multiagent system (MAS) domain

used a feature of the UT game that allows a simulation operator to "attach" itself as a spectator to any entity in the UT game to allow viewing the simulation from the perspective of that entity. In this mode, the user's view moves in conjunction with the entity to which the user is attached, allowing the user, for example, to attach to an uninhabited aerial vehicle and view the simulation from the vehicle as it flies around the map.

4.4 Extending UTSAF Architecture

UTSAF is an example of interfacing two different domains, where the SAF domain employs the DIS PDU language and the UT domain uses the GameBot language. To integrate a new system, a new interface agent is required to serve as a wrapper, and new correspondences need to be added to the SAF manager. There are two kinds of interface agents: the *intradomain* and *interdomain* interface agents. The intradomain interface agent needs a protocol parser to parse and filter necessary information from the connected system to its domain. This information is maintained and processed by an agent, such as the SAF manager agent shown in Figure 3. An interdomain interface agent, which was not needed in our prototype, would be implemented to translate its domain language to KQML. All domains need to have a common ontology to understand the content of the KQML language [11]. In Figure 6, each system such as OTB is connected to intradomain interface agent A to filter necessary information to the manager agent. The information may be interpreted into KQML by the interdomain interface agent X.

5. Experimental Testbed

We are using UTSAF to convert UT into a stealth viewer and in experiments involving the control of wide-area search munitions. For experiments, we have set up a testbed in a local-area network, as shown in Figure 7. We run a OTBSAF simulation engine on a Pentium III Linux-based machine. The DIS protocol traffic from this machine is broadcast to a multicast group that includes

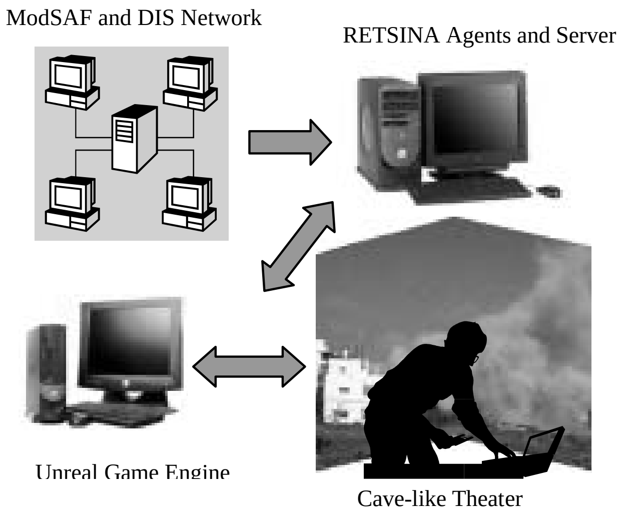

Figure 7. Our experimental testbed

the server where RETSINA agents operate. Within the MAS system, the information from OTB is processed and passed to the UT game server. PC clients connected to this game server then generate the graphical views needed to provide camera views to a targeting interface or drive a cave-like theater described on the CaveUT Web site (http://www.planetjeff.net/ut/CaveUT.html).

Figure 8 shows the user interface of an OTB simulation of 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 9 shows the chemical depot and T-80 tanks in the OTB simulation interface. The same simulation scenario in 3-D visualization using UT simulation engine is shown in Figure 10, in which 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 3-D visualization, we are able to change a view of a simulated battlefield using our agent. Figure 11 shows another view of the simulation from a T-80 tank. It shows that several A-10 Thunderbolts are flying over the biochemical depot and other T-80 tanks in the biochemical depot area. Figure 12 shows another view from a T-80 tank where other T-80 tanks are being attacked by A-10 Thunderbolts.

5.1 Experience and Performance

In our testbed, we have run scenarios with more than 35 ground vehicles and six aircraft. On average, each vehicle and aircraft generates PDU packets at the rate of 100 ms/packet and 20 ms/packet, respectively. There are also other management PDUs that are used for verification and validation of the simulation. Thus, the packet rate in

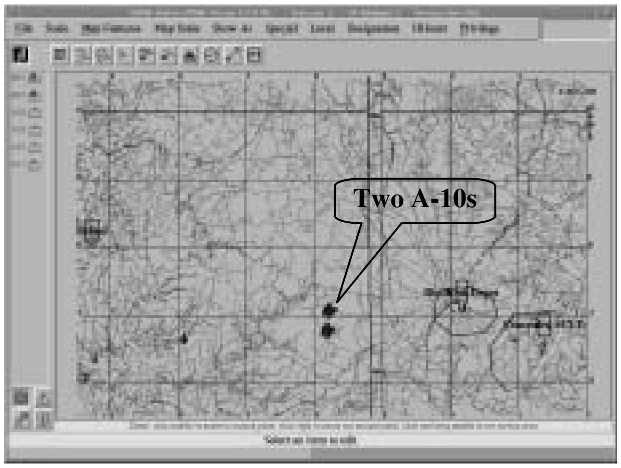

Figure 8. OTBSAF simulation command and control graphical user interface

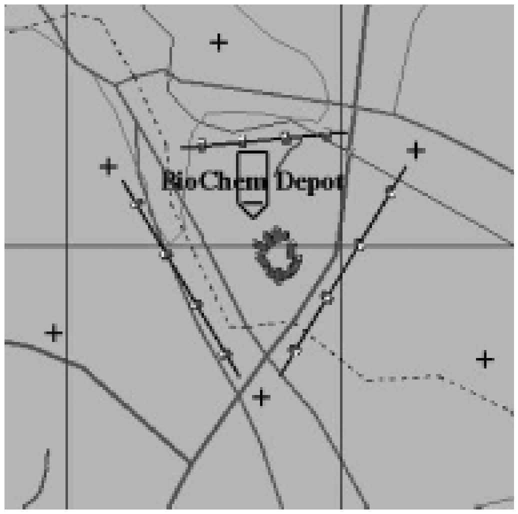

Figure 9. A biochemical depot in OTBSAF

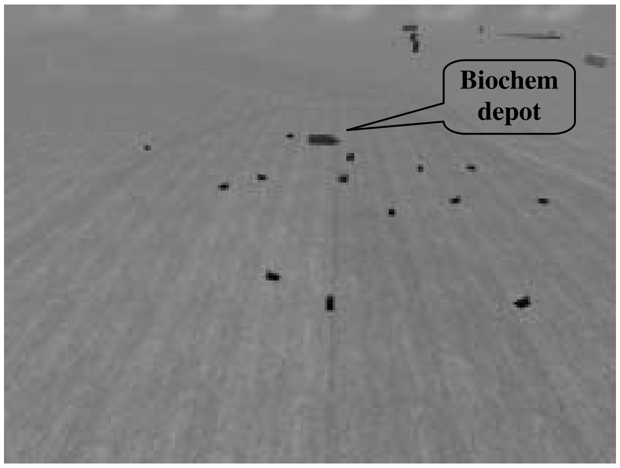

Figure 10. A biochemical depot in Unreal Tournament (UT)

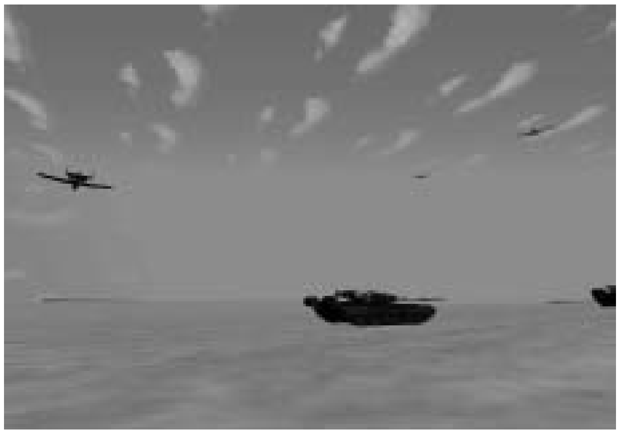

Figure 11. A view from a T-80 tank showing A-10 Thunderbolts flying over

the testbed is above 1000 packets/sec or more. Since we use MAS as middleware, we can reduce the traffic flowing from the OTB to UT game engine down to about 450 packets/sec since we use PDU filters such as a motion filter to reduce PDU packets of the ground vehicles that have no movement. By reducing the PDUs flowing through UTSAF, we also reduce about 35% of the computational resources used by UTSAF.

Some delay between OTB and the UT engine occurs in our testbed. Delay is unavoidable due to overhead in our system. However, we use dead-reckoning models such as a cubic spine function to predict future locations of each entity and asymmetric directional filtering to prevent jittering and to reduce the lag of the entity location between OTB and UT systems. This enables the reproduction of OTB entities in the UT system in close to real time and with verisimilitude sufficient for use in human experimentation.

6. Conclusion and Future Work

In this article, we describe the use of a multidomain, multiagent architecture to integrate a military simulation with a COTS game engine. Our approach can easily be extended to integrate other components such as human behavior models, intelligent information gathering and filtering, and decision-making modules or C4I systems. Using agents to wrap incompatible systems lacking other provisions for

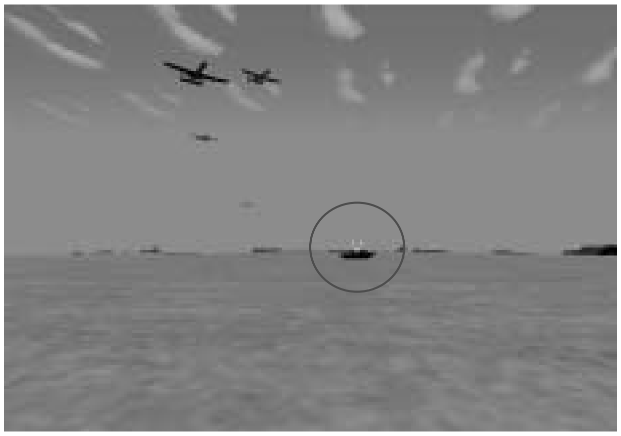

Figure 12. A spectator view showing T-80 tanks being attacked by A-10 Thunderbolts

interoperability allows the rapid construction of middleware. Although such middleware is not open in the sense of federations complying with interoperability standards such as HLA, it is readily extensible. As an example of this architecture, we created a software bridge for visualizing a DIS-based OTB simulation in a 3-D virtual world using the Unreal game engine. Unlike alternative approaches, this software bridge did not require any modification to the OTB application or game engine. Instead, we introduced a multiagent system that provided wrappers for both OTB-SAF and Unreal simulations to allow them to interoperate.

Obstacles remain to the seamless integration of OTB-SAF and Unreal. The Unreal engine currently has limits on both the possible map size and the number of entities allowed in a given simulation. To simulate very large OTB areas in Unreal, beyond the allowed map size in the game, we are currently investigating ways of dividing up the map into subunits. Each submap would be run on an individual game server and then linked together to create a seamless representation of the overall map.

7. Acknowledgments

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

8. References

[1] Garvey, R. E., and P. Monday. 1988. SIMNET SIMulator NETwork. BBN Technical Note, Ft. Knox, KY.

[2] Institute of Electrical and Electronics Engineers. 1993. IEEE P1278. International standard. ANSI/IEEE standard 1278-1993. In *Standard for information technology: Protocols for distributed interactive simulation*. Piscataway, NJ: IEEE.

[3] Brunett, S., and T. Gottschalk. 1997. An architecture for large ModSAF simulations using scalable parallel processors. Tech. Rep. CACR-155, California Institute of Technology.

[4] Stone, S., M. Zyda, D. Brutzman, and J. Falby. 1996. Mobile agents and smart networks for distributed simulations. In *Proceedings of the 14th DIS Workshop*, Orlando, FL.

[5] Ryde, M. D., and S. J. E. Taylor. 2003. Issues using COTS simulation software packages for the interoperation of models. In *Proceedings of the 2003 Winter Simulation Conference*, December 7-10, New Orleans, LA.

[6] Harkrider, S., R. Long, and R. Miller. 1999. AC-130U testbed migration to HLA. In *Proceeding of the Spring Simulation Interoperability Workshop*.

[7] Tolk, A., and M. R. Hieb. 2003. Building and integrating M&S components into C4ISR systems for supporting future military operations. In *SCS Proceedings of the 2003 Western Multi Conference* (WMC'03), International Conference on Virtual Worlds and Simulation (VWSIM'03), January, Orlando, FL.

[8] Brutzman, D., K. Morse, J. M. Pullen, and M. Zyda. 2002. Extensible Modeling and Simulation Framework (XMSF): Challenges for Web-based modeling and simulation. In the *Technical Challenges Workshop, Strategic Opportunities Symposium*, Naval Postgraduate School, Monterey, CA.

[9] Tolk, A. 2003. A common framework for military M&S and C4I systems. In *Proceedings of the 2003 Spring Simulation Interoperability Workshop*, April, Orlando, FL.

[10] Giampapa, J., K. Sycara, S. Owens, R. Glinton, Y. Seo, and B. Yu. 2004. Extending the OneSAF testbed into a C4ISR testbed. *SIMULATION* 80 (12).

[11] Sycara, K. P., M. Paolucci, M. van Velsen, and J. Giampapa. 2001. The RETSINA MAS infrastructure. Tech. Rep. CMU-RI-TR-01-05, Robotics Institute, Carnegie Mellon University.

[12] Hughes, S., J. Manojlovich, M. Lewis, and J. Gennari. 2003. Camera control and decoupled motion for teleoperation. In *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics*, October 5-8, Washington, DC.

[13] Lewis, M., J. Wang, J. Manojlovich, S. Hughes, and X. Liu. 2003. Experiments with attitude: Attitude displays for teleoperation. In *Proceedings of the 2003 IEEE International Conference on Systems, Man, and Cybernetics*, October 5-8, Washington, DC.

[14] Sardella, J. M., and D. L. High. 2000. Integration of fielded army aviation simulators with ModSAF: The eighth army training solution. In *Proceedings of Interservice/Industry Training, Simulation and Education Conference*, November 27-30, Orlando, FL.

[15] Calder, R. B., J. E. Smith, A. J. Courtemarche, J. M. F. Mar, and A. Z. Ceranowicz. 1993. ModSAF behavior simulation and control. In *Proceedings of the Second Conference on Computer Generated Forces and Behavioral Representation*, STRICOM-DMSO, July.

[16] Lewis, M., and J. Jacobson. 2002. Game engines in scientific research. *Communications of the ACM* 45 (1): 27-31.

[17] Zyda, M., J. Hiles, A. Mayberry, C. Wardynski, M. Capps, B. Osborn, R. Shilling, M. Robaszewski, and M. Davis. 2003. The MOVES Institute's Army Game Project: Entertainment R&D for defense. *IEEE Computer Graphics and Applications*, January/February, 23 (1): 28-36.

[18] McAlinden, R. 2003. Human behavior models and Unreal Tournament. In *The 23rd Soar Workshop Schedule*, June 23-27, Ann Arbor, MI.

[19] Jacobson, J., and Z. Hwang. 2002. Unreal Tournament for immersive interactive theater. *Communications of the ACM* 45 (1): 39-42.

[20] Baudisch, P., N. Good, and P. Stewart. 2001. Focus plus context screens: Combining display technology with visualization techniques. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, November 11-14, Orlando, FL.

[21] DeLeon, V., and R. Berry. 2000. Bringing VR to the desktop: Are you game? *IEEE Multimedia* 7 (2): 68-72.

[22] Lewis, M., K. Sycara, and I. Nourbakhsh. 2003. Developing a testbed for studying human–robot interaction in urban search and rescue.

In *Proceedings of the 10th International Conference on Human Computer Interaction* (HCII'03), June 22-27, Crete, Greece.
[23] MÄK Technologies. 2003. Integrates HLA networking interoperability in Epic Games' Unreal engine. News release, December 1.
[24] Adobbati, R., A. N. Marshall, A. Scholer, S. Tejada, G. Kaminka, S. Schaffer, and C. Sollitto. 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*, May 28–June 1, Montreal, Canada.
[25] Sycara, K. P. 1998. Multi-agent systems. *AI Magazine* 10 (2): 79-93.
[26] Carey, S. E., M. S. Kleiner, M. R. Hieb, and R. Brown. 2001. Standardizing battle management language: A vital move towards the army transformation. 01F-SIW-067. In *Proceeding of the Fall Simulation Interoperability Workshop*, September, Orlando, FL.

***Phongsak Prasithsangaree** is a research assistant at the Department of Information Sciences and Telecommunications, University of Pittsburgh, Pennsylvania.*

***Joseph Manojlovich** is a graduate student researcher at the Department of Information Sciences and Telecommunications, University of Pittsburgh, Pennsylvania.*

***Stephen Hughes** is a PhD student at the Department of Information Sciences and Telecommunications, University of Pittsburgh, Pennsylvania.*

***Mike Lewis** is an associate professor at the Department of Information Sciences and Telecommunications, University of Pittsburgh, Pennsylvania.*