Extending the OneSAF Testbed into a C4ISR Test Bed

Joseph A. Giampapa, Katia Sycara, Sean Owens, Robin Glinton, Young-Woo Seo and Bin Yu
The Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213
{garof, katia, owens, rglinton, wyseo, byu}+@cs.cmu.edu

Charles E. Grindle and Michael Lewis
School of Information Sciences, University of Pittsburgh, Pittsburgh, PA 15213
{cgrindle, ml}@sis.pitt.edu

February 16, 2004

Abstract

This paper describes how the modeling and simulation environment of the *OneSAF Testbed Baseline (OTB)* v1.0 has been extended to enable the testing of heterogeneous algorithms that are being designed for real world *Command, Control, Communications, Computers, Intelligence, Surveillance, Reconnaissance (C4ISR)* applications, such as the automated performance of high-level *Information Fusion*, the automation of the development and analysis of *Courses of Action (COAs)*, the automation of the *Intelligence Preparation of the Battlefield (IPB)* process, and the automated development of the *Modify, Combine, Obstacle Overlay (MCOO)*, to name a few. This has been accomplished by building an architecture that extends functional and logical components of the OTB system in six ways. One of those extensions is based on the reuse of the OTB *Compact Terrain Database (CTDB)*. The other five extensions are: (1) the addition of the *RETSINA-OTB Bridge* for the real-time query and control of OTB entities via TCP-based messages, (2) the addition of the *SAF Broker Agent* and the *SAF Manager Agent* to transform OTB *Distributed Interactive Simulation (DIS) entity state Protocol Data Units (PDUs)* into entity state TCP messages for interoperability with a TCP-based virtual reality (VR) modeling and simulation system, (3) the addition of new DIS-based sensor entities, *SARSim*, *GMTISim*, and *EOSim*, and their associated TCP-based *Sim Manager Agents*, for interoperation with *Command and Control (C2)* algorithms, (4) the modification of the line-oriented OTB command parser to permit supervised, on-line "batch-mode" human and software agent interactions with OTB, and (5) the implementation of an off-line *Batch Mode Control Agent* that permits the unsupervised configuration and control, state and execution monitoring of OTB simulations via the RETSINA-OTB Bridge. This paper describes these extensions and other data that is accessed from OTB for the purposes of simulating perceptual inaccuracies, along with examples of how they are used to create a test bed for C4ISR algorithms. These extensions enable the integration with OTB of C2 algorithms that would not normally integrate with its event-based program control modeling and simulation engine. The work described in this paper illustrates how it is possible to make a few small but general extensions to a modeling and simulation system to create a larger test bed system with minimum impact on the native system and with great potential for the range of applications that can exploit it.

1 Introduction

In order to research and design the automation of real world intelligence gathering, analysis and fusion systems, it is necessary to have a test system that models uncertainty of information, behavior, and environment. For example, if poor visibility leads to navigational errors of an unmanned air vehicle (UAV), and hence it will either be too far away from its target site to adequately gather intelligence — or worse, it may report "intelligence" for the wrong location — such errors must be detected and mitigated by the algorithms that are controlling the UAV and using its data. It is very difficult and expensive, however, in terms of time, cost and labor, to acquire such uncertainty models, let alone to develop a model and simulation system for them, and there is always the risk that the models that researchers create are biased towards their own algorithms and approaches. To address the need for such models, we have adopted the use of the *OneSAF Testbed Baseline (OTB)* v1.0 [12] as a modeling and simulation environment. It models common military vehicles, aircraft, sensors and munitions, and simulates uncertainty for entities' individual and doctrinal behaviors in the operating environment (e.g. firing from behind tree lines) and relative to each other (e.g. attack or retreat), and information uncertainty from the sensors that OTB models, such as being unable to target an entity that is concealed behind a tree line.

OneSAF was written to be extensible in two ways: by compiling new entities, entity behaviors, and functionalities into its code base of nearly one million lines of C code and over 500 software libraries [1], and by adding other simulators that can communicate with it via multicast-based *Distributed Interactive Simulation (DIS) Protocol Data Units (PDUs)* [2]. This has some drawbacks, however, particularly for the use of OTB as a test bed for new algorithms for *Command, Control, Communications, Computers, Intelligence, Surveillance, Reconnaissance (C4ISR)* applications, such as the automated performance of high-level *Information Fusion* (described in Section 2), the automated development and analysis of *Courses of Action (COAs)*, the automation of the *Intelligence Preparation of the Battlefield (IPB)* [9] process, and the automated development of the *Modify, Combine, Obstacle Overlay (MCOO)* [8] artifact. Namely, direct integration of external software entities requires that they be modified to be invoked through OTB's data- and event-driven software architecture, and many C4ISR algorithms do not lend themselves easily to such conversions. Communication by DIS PDUs does not effect interoperability with such algorithms, either, since DIS PDUs are bit-encoded words that represent a hierarchy of OTB system control, communication, and entity state information. A *Command and Control (C2)* algorithm for the automatic role assessment and assignment of two autonomous entities [4, 5], for example, requires the exchange of messages following a different protocol or knowledge representation scheme that cannot map to PDUs. There are additional problems derived from the fact that DIS packets are transmitted via multicast, which is a stateless protocol that is prone to high rates of packet loss and suppression by network routers. Not only would distributed C4ISR algorithms need to be modified to handle such transmission unreliability, but they would also need to communicate significantly more state information in order to be effective. Such requirements would actually be counter to the proposed environments in which such algorithms would be used.

The algorithms that we use for gathering, analyzing, and fusing the information derived from OTB are written and maintained in a non-OTB, *native* format. That is, the designs of algorithms, data structures, and communication protocols are made with consideration of the problems that they address, not the implementation of the specific testing environment in which they are evaluated. This has been accomplished by building an architecture that extends functional and logical components of the OTB system in six ways. One of those extensions is based on the reuse of the OTB *Compact Terrain Database (CTDB)*. The other five extensions are: (1) the addition of the *RETSINA-OTB Bridge* for the real-time query and control of OTB entities via TCP-based messages, (2) the addition of the *SAF Broker Agent* and the *SAF Manager Agent* to transform OTB *Distributed Interactive Simulation (DIS) entity state Protocol Data Units (PDUs)* into entity state TCP messages for interoperability with a TCP-based virtual reality (VR) modeling and simulation system, (3) the addition of new DIS-based sensor entities, *SARSim*, *GMTISim*, and *EOSim*, and their associated TCP-based *Sim Manager Agents*, for interoperation with *Command and Control (C2)* algorithms, (4) the modification of the line-oriented OTB command parser to permit supervised, on-line "batch-mode" human and software agent interactions with OTB, and (5) the implementation of an off-line

Batch Mode Control Agent that permits the unsupervised configuration and control, state and execution monitoring of OTB simulations via the RETSINA-OTB Bridge. This paper describes these extensions and other data that is accessed from OTB for the purposes of simulating perceptual inaccuracies, along with examples of how they are used to create a test bed for C4ISR algorithms.

A recurring motif of this paper is that many of the integrations and extensions to OTB are with agent-based systems. The reasons for this are discussed in Section 2 along with some of the background of this work. After an overview of the C4ISR architecture in Section 3, the extensions that involve the CTDB (Section 4), line-oriented OTB command batch interface (Section 5), the SAF Broker and SAF Manager Agents (Section 6, RETSINA-OTB Bridge (Section 7), and the simulated mounted sensors (Section 8) are described. The Batch Control Agent is briefly described in Section 3, and examples of C2 applications using these components are described in Section 8.2 and Section 8.3. We conclude in Section 9.

2 Motivations and Background

Our particular motivation for having a C4ISR test bed is to have an environment in which algorithms for higher levels of information fusion can be developed and tested. Information fusion is described in terms of levels by some U. S. Department of Defense (DoD) organizations. The lowest levels, 0 and 1, are concerned with the identification of individual entities (e.g. US M1A1 tank, USSR T80 tank, etc.) from the fusion of often low-confidence data from multiple types of sensors. Level 2 fusion attempts to associate the individual entities into larger organizational structures such as force echelons in order to perform reasoning at the third level of information fusion, on the expected behavior, intent, or threat that those organizational structures may pose. Level 4 fusion is concerned with the information acquisition process that was used throughout the lower levels, and on performing meta-level reasoning about how that process may be adjusted to be more accurate or use resources more efficiently in the gathering of that intelligence. For example, if the surveillance and reconnaissance of an area yields high-confidence information that suggests that the intelligence gathering activities should be redirected to another area, the algorithms that made that assessment would be classified as Level 4 information fusion algorithms.

Information fusion occurs at a variety of levels and in a variety of circumstances. An example of a military process that exercises all four levels of information fusion is the *Intelligence Preparation of the Battlefield (IPB)* [9], an intelligence gathering process that begins with terrain analysis as the foundation for identifying *Named Areas of Interest (NAIs)*, which are tactically significant entities such as, for example, terrain, bridges, or buildings, the control of which would offer superior or decisive advantage in a battle. One of the procedures for performing terrain analysis is a create-and-revise process known as the *Modify, Combine, Obstacle Overlay (MCOO)* [8], which produces annotations of terrain, known obstacle and force deployments, and the identification of NAIs. Once the MCOO artifacts have been generated, military intelligence officers then generate best-case, most probable, and worst-case scenarios, as time permits, called *Courses of Action (COAs)*, that they then *war game*, or simulate, to imagine how the COAs might evolve. Through this human, mental simulation exercise, intelligence officers can determine the consistency of the information that they gather. If information that can be critical to a scenario is missing, the intel officers may request that a commander task assets to attempt to acquire the missing information.

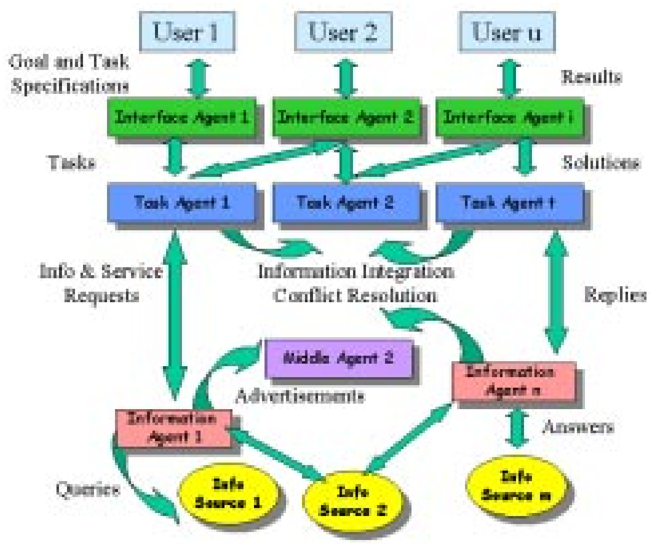

Figure 1: The RETSINA Functional Architecture

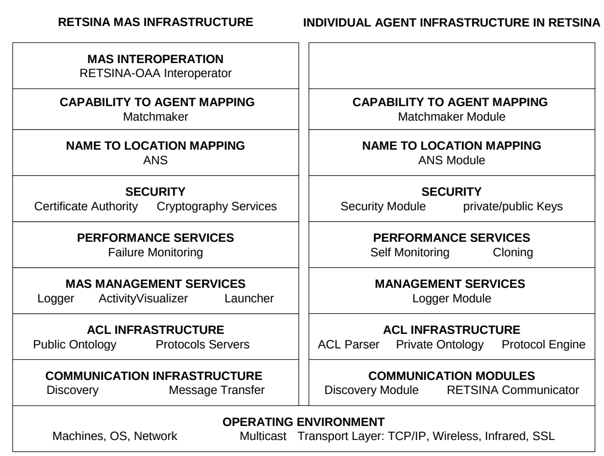

Figure 2: The RETSINA MAS and Individual Agent Infrastructures

The modeling and automation of these types of information-fusion processes, in particular those of a goal-directed and dynamic nature, lend themselves to solutions based on a *multi-agent system (MAS)* such as RETSINA [15]. In our vision and implementations of RETSINA agent-based systems, agents demonstrate autonomy at three levels. An agent is autonomous *toward the human user* if it is capable of understanding the human user's intentions, goals, and the context in which the user acts on his intentions or attempts to achieve his goals. An agent can exhibit *decision autonomy* if it demonstrates goal-directed behavior, and that it is capable of achieving that goal via diverse techniques that are chosen based on the agent's sensitivity to its operating environment and knowledge of past performance. And, an agent is autonomous *toward other agents* (we also call this *system-level autonomy*) when it demonstrates behaviors of seeking and attempting to *semantically interoperate* with other autonomous agents. By *semantic interoperation*, we refer to the ability of agents to collaboratively perform a task (e.g. solve a problem, or produce a service) based on the exchange of meaningful information, and not based on the choreographed timing of their collective program executions.

Another reason for using MAS technology for the research, development, and testing of information fusion algorithms is that multi-agent systems presume a common abstract architecture of functional services that can be implemented in heterogeneous ways. This facilitates the integration of a myriad of disparate software systems and components. These abstract architectures also guide decisions about how components within the architecture will interface with each other. Figures 1 and 2 illustrate two such perspectives. Figure 1 illustrates the functional division of roles that individual agents have in any one application, while Figure 2 illustrates one perspective of how functionalities within an individual agent map to functionalities in a multi-agent system, and how functionalities in either the MAS or individual agent interface with each other. The reader is referred to [15] for a complete explanation of these architectures (and other features) and justifications for developing applications as multi-agent systems.

3 The RETSINA C4ISR Architecture

Figure 3 illustrates the RETSINA extensions to OTB. The first extension (lower left corner) is the use of the OneSAF compact terrain database (CTDB) for purposes other than OTB's internal modeling and simulation. Information flow, indicated by the solid black line, "logical connection", is unidirectional, from the CTDB to the components that use CTDB terrain data. The entities of Figure 3 that receive CTDB data do not communicate directly with each other. The module labeled *Terrain Analysis* identifies recent work that is described in Section 4. The module labeled *Other TA Systems* (Other Terrain Analysis Systems) refers to other uses of CTDBs, such as for agent-based route planning [13], or for its inclusion in a virtual reality simulation system [11].

The second extension to OTB is indicated by the boxes of Figure 3 that are labeled, *Line-Oriented OTB Command Batch Interface*, and *Line-Oriented Commands*, respectively. They are explained in Section 5, below.

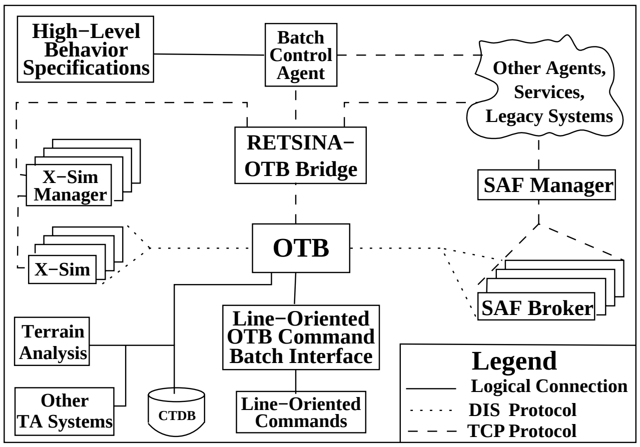

Figure 3: The RETSINA C4ISR Architecture

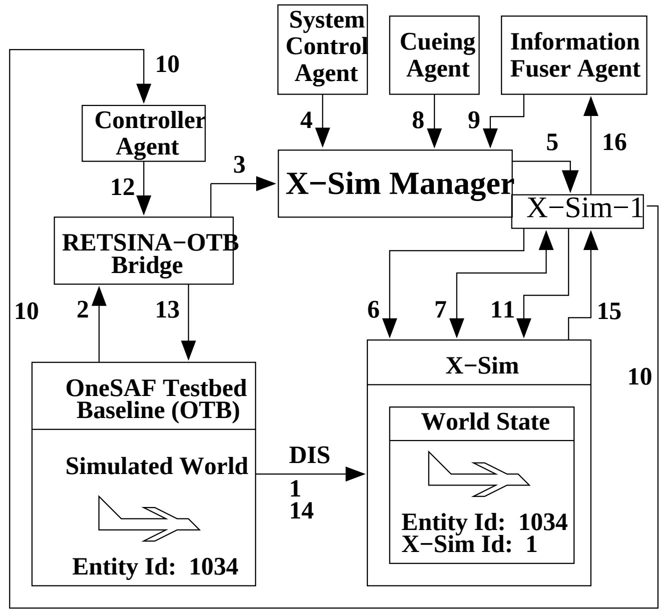

Figure 4: An application of the RETSINA-OTB Bridge and *X-Sim* modules. The activity sequence is described in Section 8.2.

The third extension to OTB is the addition of the *SAF Broker* and *SAF Manager* agents, which are described in Section 6. As can be seen from Figure 3, the SAF Broker agents listen to DIS PDUs, and then transmit them to a SAF Manager agent, which collects and organizes the information that they contain about any one entity for any other agent or system that subscribes to its information updates.

The *RETSINA-OTB Bridge*, described in Section 7, was a significant fourth extension to the OTB simulation environment. It enables the direct creation, addressability, and tasking of any SAF entity. It also allows for the custom specification of OTB tasks, and for entities' partially-executed tasks to be interrupted or modified.

The fifth component of the RETSINA C4ISR test bed is the addition of a TCP-based *Batch Control Agent* that can configure and execute experiments in OTB that are expressed in a *High-Level Behavior Specification*. Thus, it is possible to specify the creation of SAF entities, their being ordered to execute a certain task, and to specify the termination conditions of their task (e.g. "fight for X minutes," or "fight until either side sustains more than 80% losses," etc.). We have used the Batch Control Agent to run hundreds of unsupervised batch experiment iterations in which random tank configurations (e.g. *vee*, *echelon right*, *wedge*, etc.) were evaluated to determine their affects as force multipliers.

The final component shown in Figure 3, the *X-Sim Manager Agents* and the *X-Sim Agents*, illustrates how completely novel sensor types can be added to OTB, mounted on SAF entities, and integrated in a C2 application. The sensors are described in Sections 8, 8.1, and their integration is described in Section 8.2.

4 Compact Terrain Database Component

Although the compact terrain database component is not used as a dynamic interface to our extensions to OTB, being able to automatically read and process CTDB data in a relevant way is a significant accomplishment in demonstrating OTB's usability as a C4ISR test bed. OTB CTDB terrain data represents: elevation, slopes, vegetation, soil type, surface drainage, soil characteristics due to weather conditions, bodies of water, minefields and trenches. Our terrain analysis modules used this information to evaluate terrain in terms of the types of echelons that can use or traverse it (trafficability and configuration space analysis), and for the identification of avenues of approach, engagement areas and areas of interest [6, 7]. Future plans include the use of automatic terrain analysis to automatically generate courses of action (COAs).

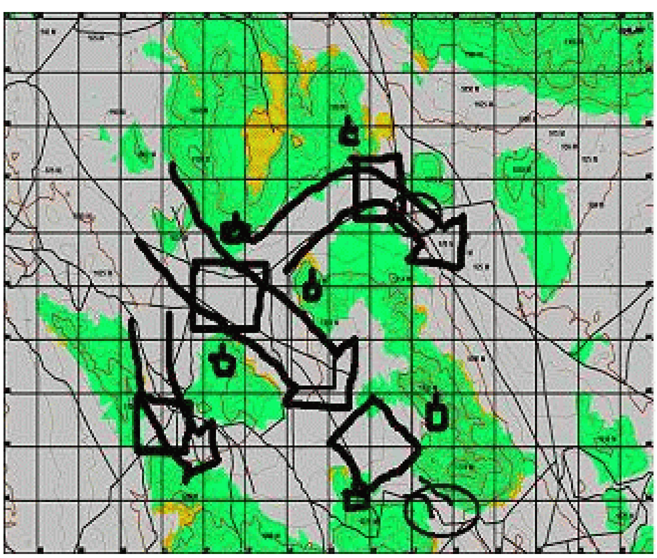

Figure 5: An expert's analysis of OTB terrain, obtained via the MCOO process.

Figure 6: The results of automatic terrain analysis, obtained via the automated MCOO process.

Figure 5 shows the results a subject matter expert performing the MCOO process on terrain from the OTB CTDB. Figure 6 shows the results produced by our algorithms which automate the MCOO process. The algorithms worked directly with data from the CTDB, creating this overlay in approximately one minute on a 2.4GHz Intel P4 processor, with 512MB RAM. The particular overlay shown in Figure 5 was produced in a little more than 30 minutes. Other subject matter experts, who produced results closer to those represented in Figure 6, needed closer to 60 minutes to produce their overlays. More details of the algorithms and experiments are provided in [6, 7].

5 Line-Oriented OTB Command Batch Interface

The "unextended" version of OneSAF has two interfaces that can be used to place, query, and control entities. One interface is the *Command Editor*, a GUI that allows a user to place SAF entities on its rendering of a CTDB map. The other interface is a text-based command-line parser. This interface allows a user to create, place, and query entities in OTB through a command-line parser. This latter interface, also referred to as the OTB debug interface, is highly interactive, processing one human-entered command line at a time. While faster than navigating the GUI (for an expert user), it quickly becomes evident how tedious this interface is for effecting any complex and non-trivial operations in OTB.

We modified the library that manages the processing of commands through this interface to also read and write files that contain such commands. Such files are logically indicated in the box labeled, *Line-Oriented Commands*, in Figure 3. The extension is designed to poll a directory for the existence of a file containing line-oriented commands. If the process detects such a file, it renames the file so that it will not be detected again, opens it for reading, and begins to execute the commands that it contains, one line at a time. The executions are blocking, meaning that no command line or batch file will execute before its predecessor has completed. If one of the commands is to query the status of an entity, then the LOOCBI (line-oriented OTB command batch interface) will write the status information to a file.

Since files can be created, renamed, or accessed by humans, agents, or web services, this interface can be used to perform rudimentary batch mode experiments. While this type of extension is fairly quick to implement and easy to learn to use, the drawbacks of this method are that it: (1) requires meticulous manual preparation and editing of the command files, (2) requires the meticulous tracking of SAF entity addition/deletion requests in order for a person

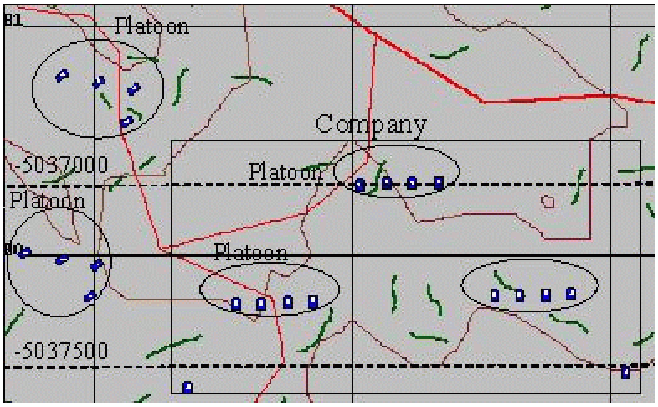

Figure 7: Level 2 information fusion: recognizing echelon types and behaviors.

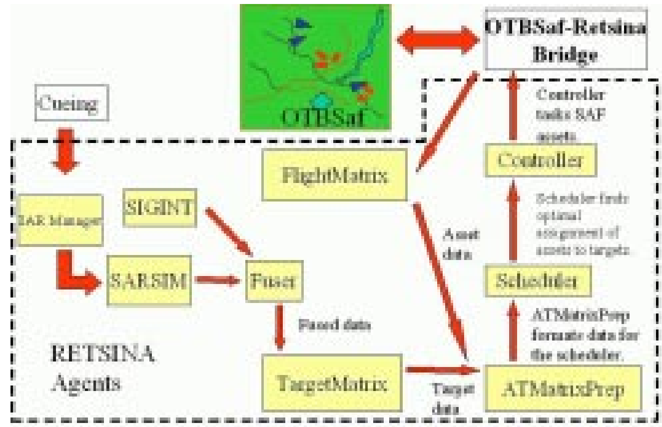

Figure 8: The command and control (C2) of simulated sensors and sensor platforms in OTB. The activity sequence is described in Section 8.3.

or program to know the OTB identification number of an entity, and (3) only offers coarse-grain query and control capabilities. That is, a command file must first finish executing before OTB will execute another command file.

Despite its limitations, this extension has been successfully used to test the coordination of three M1 tank platoons by autonomous agents in a dynamic environment [4, 5]. In more recent experiments, this interface proved useful in the rapid development and testing of algorithms for Level 2 fusion, as illustrated by Figure 7, such as the recognition of tank platoons, companies, and their behaviors (e.g. bounding overwatch movement).

6 SAF Broker and SAF Manager Agents

SAF Broker agents listen to DIS PDUs that are in the same multicast group as the OTB that transmits them. Each SAF Broker can potentially listen to as many multiple OTB simulations as are on that multicast channel, but SAF Brokers cannot listen to multiple multicast channels. Other agents, such as a SAF Manager agent, can subscribe to SAF Broker services, and request that the Brokers filter only PDUs that originate from a certain OTB simulation image, or that pertain to a specific entity. If multiple simulators produce PDUs about the same SAF entity, a SAF Manager agent will accumulate such updates, add them to its internal database, and forward only those updates that have been requested by a subscribing program or agent.

Applications that use this agent system (ex. [11]) should follow OTB expectations of performing their own *dead* *reckoning*. As unsequenced, stateless UDP packets that are sent to ethernet addresses within the same multicast group, DIS PDUs may be lost or dropped without consequence. At the typical rate of 30 DIS PDUs per second, even if tens of PDU packets never reach their destination, the next packet that does will contain all of the current state information of the simulation environment. In a network with a high loss rate due to high volumes of message traffic and congestion, it is expected that the entity reading the DIS packets will perform its own *dead reckoning*, which is an extrapolated estimation of an entity's state until the next PDU to update its state is received.

Although it was recognized that converting DIS PDU messages into TCP messages could potentially dramatically increase network traffic, our use of this system did not cause any perceptible degradation of the quality of the OTB updates. We believe that this has been because: (1) clients to the SAF Broker typically only need to read state information for visualization effects, and the overhead of parsing TCP messages for such state information is enough to handle multiple messages per millisecond, (2) the datacomm chain from OTB to SAF Broker client is unreliable. The TCP messages are generated from transport-unreliable UDP packets, so if the client needs to have high-fidelity knowledge of entity state, it must implement dead reckoning; and (3) because of this, the use of dead reckoning obviates the need to improve the data communications model of the SAF Brokers and SAF Manager.

7 The RETSINA-OTB Bridge

The purpose of the RETSINA-OTB Bridge is to allow for the finer-grained access and control to OTB entities and the simulation system, itself. It was implemented by adding a reduced (optimized for speed) C version of the RETSINA Communicator [14] program library to OTB, and building lightweight message processing routines to translate Communicator messages to and from OTB events and callback registrations. This internal library is called *libretsina*. The *libretsina* module receives specially-formatted TCP messages, and depending on the content, dispatches the content to the appropriate OTB event handling routine. If the message contains a query, then a RETSINA callback is registered with the OTB event processor. If the message contains a command or a task, then the corresponding OTB function is registered for execution.

The RETSINA-OTB Bridge, proper, resides outside of OTB so that it can optimize the streaming of messages to and from *libretsina* in OTB. Since OTB executes as a single-threaded process, any backups due to incoming message queue overflows will cause a degradation of system performance. As an external process, the Bridge can manage the message pacing into OTB without adversely affecting its performance. Messages leaving OTB have less of an impact on the system, but can still reduce the accuracy of simulation of OTB if queried too frequently.

Number of Platoons	**Number of Entities**	**Threading**	**OTB Optimized**	**RETSINA Updates per Second**
69	276	single	optimized	0
68	272	single	optimized	1
66	264	single	optimized	5
65	260	single	no	0
65	260	single	no	1
64	256	multi-	no	0
63	252	multi-	no	5
61	244	single	no	5
60	240	multi-	no	5

Table 1: Impact of the RETSINA libraries on OTB.

Table 1 illustrates the impact of polling once and five times per second on OTB. Zero *RETSINA updates per second* (cf. Table 1) indicates that *libretsina* has not been registered with the OTB event processor. These results were produced by using the native OTB benchmark program to determine how many entities OTB can simulate in parallel at real time speed without OTB reporting that it is not able to "keep up" with internal entity state updates. On a 2.4 GHz dual processor Intel XEON computer with 2.0 GB of RAM, running a multi-threaded RedHat Linux 7.1, kernel 2.4.20, connected to the 100mbs campus ethernet network, that limit has been around 272 M1A1 tanks, with the variations due roughly to the complexity of the terrain, and the degree of interaction among the simulated entities. The reader should note that OTB was designed as a single-threaded architecture, hence its abysmal performance when multi-threading was enabled. Many parameters can be tweaked in an attempt to tune the performance of an OTB system, and many of these parameters depend on the nature of the operating environment and what is being simulated. Since most of our simulation exercises have been at the platoon and company levels, we have typically run the simulator with $50 - 75$ internal, SAF-native entities plus another $30 - 50$ external SAF entities, such as the SARSim, EOSim, GMTISim, etc., associated with some of the SAF-native entities, all in the same image.

Communications based on the RETSINA-OTB Bridge need more careful considerations of the implications of transport reliability, since both the Bridge and the TCP communication end-point are considered to be transport reliable. While we have been able to achieve a message signalling throughput[1] of multiple messages per millisecond via the RETSINA-OTB Bridge, the effective throughput[2] depends greatly on the type of message processing that must be performed by the Bridge client application. Some applications, such as one that animates point-to-point communications between SAF entities, are not able to keep pace with the RETSINA-OTB Bridge updates, and so would block the reception of further Bridge updates until it could empty its network incoming message queues. This blocking of the transmissions would cause the Bridge, in turn, to block the iteration of SAF simulation cycles, and thus OTB would appear to freeze at times. This problem was resolved by enabling a RETSINA Communicator option whereby input messages can be discarded if the client application could afford to drop messages and its input message queues were full. Other areas where performance tuning can be effected are in adjusting the size of the message queues, adjusting the size of the messages — we noticed increases in performance as smaller messages were joined into larger messages, streamlining the structure of a message so that its content can be quickly extracted, and ultimately, implementing dead reckoning in the client application, if appropriate.

8 Simulated Mounted Sensor

One of our significant contributions to OTB is to add three simulated mounted sensors, *SARSim* (synthetic aperture radar simulator), *EOSim* (electro-optical simulator), and *GMTISim* (ground moving target indicator simulator), to the simulation environment, thereby increasing the types of surveillance and reconnaissance that can be performed, and augmenting the type of Level 1 fusion data that can be used for the development of our fusion algorithms. The simple models were generously provided by Northrop-Grumman and we assisted with their integration into the RETSINA multi-agent system. High fidelity versions of such systems exist, but they are either classified or prohibitively expensive, and ultimately, inaccessible to a research group that is interested in developing and testing

[1]The signaling rate indicates how fast we can generate and transmit TCP a message.

[2]The effective throughput is how many messages can the client program effectively process.

command and control algorithms that manage the scheduling and tasking of such simulated platforms as a C-130, F-16, UAV (unmanned air vehicle), or *WASM* (wide-area search munition), on which the sensors can be mounted. Considering that multiple sensors may be mounted on the same platform, or that some platforms may double as a munition as well as a sensor, the command and control of such platforms with such wide-ranging and diverse capabilities is a non-trivial task, and using OTB for the simulation of their real world dynamics and behaviors would allow us to begin to investigate such problems.

The SARSim simulates an automatic target recognition (ATR) system that receives its input from a synthetic aperture radar (SAR) that operates in *spotlight-mode*. In spotlight-mode, a SAR scans an area of terrain, and the ATR will attempt to recognize any *stationary* object that is either completely or partially within the bounds of that scanned area. Just like for a real SAR/ATR system, the output from the SARSim is a list of all friendly and threat candidate target types (e.g. US M1, USSR T80, USSR 2S6, US M977, etc.) with confidence levels indicated as percentages for every potential target type that the SAR/ATR system can recognize. While a real SAR/ATR system will report confidence levels for around three dozen entities, the SARSim will report confidence levels for a dozen entities and clutter. Just as for a real SAR/ATR system, it may report entities where they do not exist in the simulation (e.g. "false positives") and fail to recognize entities that, in fact, are present (e.g. "false negatives").

The GMTISim simulates a ground moving target indicator (GMTI) radar, which focuses a radar beam on one spot, and if it detects a moving target there with its automatic target recognition (ATR) system, a motion tracker mechanism follows the movement of the target. While very similar in output and behavior to the SARSim, it is complementary, because it only recognizes entities that are moving, while the SARSim only recognizes entities that are stationary. The EOSim simulates an electro-optical sensor that detects targets at distances and in conditions in which they would be detectable in the ultraviolet, visible and infrared light spectra.

Other features that the above three sensors have in common with real sensor systems are that multiple simulators may be used in the same exercise and on the same platform[3], the simulated sensors must be within sensor range in order to observe an area, they must be aimed with a certain angle at the target area, and, in the case of SARSim and GMTISim, the simulated sensor must be moving at a certain speed to be able to scan an area. These sensing constraints impose flight path constraints on the OTB assets on which the simulators are mounted that can leave

[3]We currently only allow one sensor type per asset, so it is possible to have a GMTISim, SARSim, and EOSim on the same F-16, but not to have the same F-16 mount two SARSims, for example.

some of the assets vulnerable to anti-aircraft missiles, thus injecting elements of risk, vulnerability, and uncertainty into the scenarios that our C4ISR algorithms are being designed to address. The only critical functional divergence between the above three simulators and the real systems that they model is that the simulators do not yet model occlusion effects caused by terrain and vegetation, such as mountains or tree canopies. This shortcoming is partially obviated by the low accuracy of the recognition, which is described in Section 8.1.

8.1 Sensor Characteristics and Algorithms

The three sensor simulators have many characteristics in common, so we will first describe the SARSim so as to establish an understanding of their characteristics. Real SAR sensors have the capability of maintaining the same spatial resolution (i.e. feet, meters, etc.) independent of range, although the farther a sensor is from a target in some modes of operation, the more prone they are to error due to atmospheric conditions. Standoff ranges for publicly documented SAR sensors range from 10km to 100km, and have resolutions ranging from 1 meter to less than 1 foot. For example, the SAR mounted on a GlobalHawk UAV flying at 650 km/hr at an altitude of 65000 feet is capable of imaging an area in spotlight mode that is 100km away at a resolution of 1 foot. *Integration time*, or the time that it takes the SAR to acquire an image, is roughly a function of parameters such as: arclength, velocity, subtended angle, angle from broadside, radar wavelength, and desired resolution. For a platform flying at 560 km/hr, looking straight on at its target, imaging in 1 foot resolution spotlight mode with a 3 cm (10 GHz) radar wavelength, the time to acquire an image is roughly 7.7 seconds. The SARSim model that Northrop-Grumman developed allows for the setting of all of these parameters. For the purposes of our simulations, and given that our terrain maps are typically around 75km X 75km, we have configured the SARSim to sense at a range of 25 km, mounted on an F-16D, flying at around 600 km/hr. Combined with a negligible time for ATR processing, the total integration time for scanning an area is usually completed in 10 seconds, or less. The GMTISim has roughly the same configurable parameters as the SARSim, so for the purposes of our simulations, we also mount the GMTISim on an F-16D, flying at around 600 km/hr, and activating at a standoff range of 25 km to the target. The standoff range for EO sensors is much shorter, so we configure our EOSim to sense at 15 km from the target.

The three simulators interoperate with OTB by receiving *ground truth* data about the OTB entities and "confusing" it according to a sensor-specific *confusion matrix* model. The sensors begin by subscribing to entity updates that arrive via DIS packets, and maintain an internal table of all such entities and their ground truth status. When the sensor is tasked to scan an area, and the sensor is within sensor range, it reads the entity ID, its orientation on the ground relative to the sensor, and then produces a list of possible target identities with associated levels of confidence according its confusion matrix model. The confusion matrix does not model the real fidelity of the sensor, as that is classified, but it does provide a series of low-confidence estimates in which the highest-confidence identification is not necessarily the correct classification — just like for the real sensor — that can be used in algorithms of higher-levels of information fusion. Thus, it is possible for a simulated sensor to confuse an M1 tank with a T-80 tank. So as to produce false positives and false negatives, the sensors have random functions that either generate entities where they do not exist or that suppress the reporting of entities where they do exist.

8.2 OTB, *X-Sim* and RETSINA Integration

X-Sim is the abstraction used to describe the architecture that is common to the *SARSim*, *EOSim*, and *GMTISim* simulators. The following enumerated sequence describes the activity represented in Figure 4.

An entity is created within OTB, and has an identification number assigned to it by OTB. In this example, that number is "1034". **(1)** As soon as an entity is created in OTB, entity state PDUs are multicast via DIS packets to all programs that are part of the same multicast group and exercise identifier as the OTB simulation. The X-Sim creates an entry in its "World State" table for every entity that is created in OTB. **(2)** Entity state information is transmitted to the *RETSINA-OTB Bridge* via RETSINA messages, which are based on the TCP protocol. **(3)** The RETSINA-OTB Bridge transmits all entity state information messages to whichever agents are subscribed to receive its update notifications, in Figure 4, the *X-Sim Manager*. **(4)** A *System Control Agent* contacts the X-Sim Manager and issues the command to associate an X-Sim instance (e.g. "X-Sim-1") with a particular OTB entity (e.g. "1034"). **(5)** The X-Sim Manager spawns a new thread which creates a new RETSINA *Communicator* [14] proxy with the identity of the newly requested X-Sim instance (e.g. "X-Sim-1"). The X-Sim Manager will route all communications for the X-Sim entity to that proxy. **(6)** The *X-Sim-1 proxy* sends an "install" command that notifies the X-Sim module that there should be a new instance of a simulation model, it should be associated with a specific OTB entity (e.g."1034"), and that it will base its behavior on the simulated behavior of that specific asset (e.g. entity "1034"). **(7)** Should the "install" command be received before the X-Sim has received the DIS packet announcing the existence of the entity, the X-Sim will reply to the X-Sim-1 proxy that the entity was not found. The X-Sim-1 proxy will continue to resend its "install" request after a small delay until the X-Sim acknowledges the entity, or until the "install" command is canceled. **(8)** A *Cueing Agent* sends a message request to the X-Sim Manager to scan an area. Requests are queued in the order that they are received and assigned to the proxy of the first available OTB asset (e.g. proxy "X-Sim-1" for entity "1034") without consideration of that asset's proximity to the target area. Requests to cancel a scan may be sent to the X-Sim Manager. **(9)** At any time, a service requesting agent such as the *Information Fuser Agent*, *Belief Display Agent*, or an interface agent that displays the X-Sim data, can submit a *monitor query* request to receive any and all notifications from the X-Sim Manager. Those notifications will be generated by the X-Sim sensor instances that the X-Sim Manager controls. **(10)** The assigned X-Sim proxy (e.g. "X-Sim-1") generates a new OTB destination and path plan for its asset to travel. This "order" is submitted to a *Controller Agent* which manages the scheduling of multiple requests from other agents in the simulation system. **(11)** In parallel with the asset order, the assigned X-Sim proxy "orders" the simulation model instance to survey the area when the asset on which it is mounted is within the sensor's range. **(12)** The Controller Agent sends the next scheduled task for the asset to the RETSINA-OTB Bridge. **(13)** The RETSINA-OTB Bridge checks the received task for syntactic and semantic correctness and if valid, forwards the request to OTB. The RETSINA library within OTB interprets and applies the task to the entity (e.g. "1034"). **(14)** When an entity state PDU indicates that the asset on which the sensor is mounted is within range of the area to scan, the sensor simulator, "X-Sim", "turns on" its simulation instance (e.g. "X-Sim Id: 1") for the duration required by that modeled sensor to perform its task. **(15)** Once the sensor simulator has completed its imaging, it sends the results back to its X-Sim Manager proxy. If the asset on which the sensor was mounted moved out of range before the sensor finished the task, the sensor returns partial results, if that is what it would do in reality, otherwise no results are returned. If the asset on which the sensor is mounted is destroyed, then the X-Sim Manager will delete its proxy and notify the X-Sim to delete the simulator instance that was mounted on that asset. The task that was assigned to the asset and sensor is also lost and will need to be rescheduled. **(16)** The X-Sim Manager proxy sends any and all results from the simulator to all agents that are subscribed to its notification service.

8.3 C2 of Agent Sensors on OTB Platforms

This section describes a scenario that illustrates how the RETSINA-OTB Bridge provides support for the real time interoperable tasking and execution of agents, sensors, and simulated vehicles in the OneSAF Testbed Baseline simulator. The scenario presumes that other intelligence, surveillance, and reconnaissance (ISR) events have already taken place. Such events may have been the human or automated analysis of terrain and the creation of a MCOO, similar to what is described in Section 4 and in Figures 5 and 6, or that long-range sensors, such as a 100km SAR, have already scanned the terrain in order to provide a first pass detection of units that might be situated at significant *areas of interest*[4]. This scenario also utilized a *SIGINT*[5] *Sensor Agent*, which models a sensor that can detect enemy signals from radio and radar emissions, has been activated and is listening for enemy radio emissions as further evidence of where the enemy might be located.

Part of the initialization of the scenario requires the System Control Agent to install instances of the X-Sim sensors on particular OTB entities, as described in Step 4 of Figure 4. At this point, the sequence of steps that follow make reference to Figure 8. **(1)** The Cueing Agent is triggered, and it announces *areas of interest* to the X-Sim (e.g. SARSim, EOSim, GMTISim) Managers for them to scan. **(2)** As described in Steps 9 – 15 of Figure 4, the OTB entities on which the X-Sims are mounted are tasked by their respective managers to fly certain routes to begin sensing the areas of interest. **(3)** As the X-Sims sense entities in OTB, each updates their respective information displays, and sends their low-confidence data to the Information Fuser Agent. **(4)** The Information Fuser Agent maintains all intelligence reports that it receives from the various sensor and simulated optical, *VSpotter* (discussed at the end of this section) sources, but, only displays one report with the highest confidence rating for any given entity. The Information Fuser Agent also displays the name of the originator of the report. If a human is interested in reviewing all of the reports for an entity, he may do so via the interface. **(5)** When an area of interest has been sufficiently scanned to indicate a concentration of entities that requires a closer inspection, the human commander, via the System Control Agent, may order a UAV to fly in for a closer look. **(6)** If a UAV is tasked, the Information Fuser Agent will subscribe to and receive the UAV's visual VSpotter reports. Such reports are considered to have

[4] We use the term, *areas of interest* to indicate areas on the map that our algorithms must further analyze, and distinguish the term from the typical military term, *named area of interest (NAI)*, which indicates a militarily-significant object, which could be key terrain, a bridge, or even a building.

[5] SIGINT is the acronym for *Signals Intelligence*.

a much higher confidence rating than the other sensor reports. **(7)** When enough high-confidence reports of enemy units have been accumulated for an area of interest, the commander may then decide to attack them. If he does, he will order the attack based on the fused data from from the Information Fuser Agent.

To begin planning the attack, **(8)** the Target Matrix Agent will receive the list of entities from the Information Fuser Agent. The Target Matrix Agent will convert the entities into a prioritized list of targets. For example, if an entity is a Scud missile, it will have a much higher target priority than, say, a T-72 tank. The Target Matrix Agent is designed to be modified to associate the names of weapons that can be used effectively against that target, at a later date. **(9)** The ATMatrixPrep Agent will receive target data from the Target Matrix Agent, and asset data from the Flight Matrix Agent that indicates which planes are in the sky, and what is their current location. It will format both types of information into a form that is suitable for the Scheduler Agent.

(10) There are really a few types of scheduling agents represented by the box labeled "Scheduler": the main *Scheduler Task Agent (STA)*, an *Optimized Scheduling Agent (OSA)* that implements algorithms described in [10] via a commercial off the shelf (COTS) scheduling system [3], and a *Naïve Scheduler Agent*. The STA first checks if the OSA is available. If the Optimizing Scheduling Agent is available, then other agents are invoked to prepare the input data for it before it is invoked. If the Optimizing Scheduling Agent is unavailable, then the Naïve Scheduler Agent is tasked to schedule the air strike. The Scheduler sends a completed schedule to the Controller Agent. The schedule consists of a prioritized list of targets for each asset to attack.

(11) The Controller Agent then tasks the OTB assets according to the schedule it receives, one target at a time, through the RETSINA-OTB Bridge. If, while the Controller Agent is executing a schedule with OTB assets, a *time critical target (TCT)* is identified by the Information Fuser Agent, the scheduler will send a new schedule with the TCT in the position of the highest priority. The Controller Agent will then discard its current schedule and substitute it with the new schedule. The schedules that the Controller Agent will receive will always contain all known targets that have not been killed, with the highest priority targets in the first positions of the schedules. **(12)** The RETSINA-OTB Bridge checks the received task for syntactic and semantic correctness and if valid, forwards the request to OTB. It also receives entity status updates from OTB, and routes those messages to the requesting agents. **(13)** The Controller Agent subscribes to the Information Fuser Agent to receive battle damage assessment (BDA) reports for the entities being targeted. As targets are killed, the Controller Agent removes those targets from the target lists for the OTB assets and reassigns the asset tasked with killing that target to the next live target in the asset's schedule. **(14)** Both the Flight Matrix Agent and the Controller Agent receive health assessments from the RETSINA-OTB Bridge for the assets that they are managing. If a plane is shot down, the Controller Agent will cease issuing task orders for that plane. All targets that were assigned to the plane will remain untargeted until a new schedule is issued, or unless they are opportunistically targeted by other assets that are in the vicinity. **(15)** Since our C2 loop is designed for rudimentary human-in-the-loop control, the whole process can be repeated only if the human commander reissues an order to attack the remaining targets that were identified by the Information Fuser Agent.

The RETSINA-OTB Bridge reports observational data from the perspective of OTB entities by exposing three OTB-internal data structures: the *VSpotter, VTargetAsses*, and *VKillAssess* lists. If any OTB entity can visually sense the presence of another entity, such as a pilot or gunner would see its target, detection events are triggered that register that entity in the VSpotter list. Since there may be terrain, smoke, dust, darkness or atmospheric conditions that affect the visual sighting of one OTB entity by another, the data that is represented in the VSpotter list does not represent ground truth, but a perceived reality. VKillAssess simulates the reliability with which battle damage assessment would be effected by a real world entity in a combat situation. It reports the amount of damage — such as a *firepower kill*, *mobility kill* or *catastrophic kill* — that another simulated entity within visible range of the observer, has suffered. VKillAssess is not provided for aircraft such as A-10s, F-16s, and UAVs, but by ground-based vehicles. VTargetAssess is reported by all OTB entities, and is reported whenever the observing entity is within sight of the observed entity. All three libraries are used by higher-level OTB libraries to determine what the simulated OTB entity can shoot at, and its precision and accuracy for striking its target.

The Information Fuser Agent subscribes to the RETSINA-OTB Bridge to receive all visual reports that are generated by a friendly entity of type *UAV*. Since this is a report of visual information taken from close range and low altitude, it has a much higher confidence rating than any of the individual sensors, so VSpotter reports from the UAVs will be displayed before the much lower-confidence SARSim, EOSim, or GMTISim reports, although the lower-confidence reports will still be accessible. However, since OTB UAVs do not report VKillAssess information, the RETSINA-OTB Bridge adds that information to the messages that the UAV reports to its subscribers.

9 Conclusions and Future Work

This paper has demonstrated the need and ability to extend the OneSAF Testbed Baseline modeling and simulation system into a C4ISR test bed for the research and development of algorithms for higher levels of information fusion, and for the automatic command and control of military assets. We do this via a variety of interfaces and extensions to the native OTB platform, both through the OTB interfaces that were intended for system expansion, and by making small but very effective modifications and additions to the system. Some of these small but effective modifications were the addition of libraries to enable OTB to communicate with agent-based systems. This permits the expansion of OTB into a system that is highly-interoperable with a heterogeneous array of other C4ISR components and test platforms. Future work will continue to refine and enhance these interoperability components as we continue to research and develop high-level information fusion and C2 algorithms and test scenarios.

Acknowledgments

The authors would like to thank past members of the Intelligent Software Agents Lab who contributed to the development, testing and use of our system: Jason Ernst, Brent Langley, Martin Van Velssen and Wei Yang. We would also like to thank our partners at Northrop-Grumman, led by Dr. Robert Mitchel, for their assistance in developing some of the external simulators. This research has been sponsored in part by: AFOSR Grant F49620-01-1-0542, Office of Naval Research Grant N00014-96-16-1-1222, and by DARPA Grant F30602-98-2-0138.

References

[1] Advanced Distributed Simulation Technology II. OneSAF testbed baseline assessment final report. Commissioned Report for NAWCTSD/STRICOM ADST-II-CDRL-ONESAF-9800101, Lockheed Martin Information Systems, Lockeed Martin Corporation, P. O. Box 780217, Orlando, FL 32878-0217, 8 May 1998. Available at http://www.onesaf.org/OTBFinalReport.doc, accessed January 2004.

[2] J. J. Brann. Enumeration and bit encoded values for use with protocols for distributed interactive simulation applications. Technical Report IST-CF-02-01, Institute for Simulation and Training (IST), 3280 Progress Drive, Orlando, Florida 32826, 16 April 2002.

[3] ILOG CPLEX scheduler. `www.ilog.com/products/cplex/`, February 2004. Accessed.

[4] J. A. Giampapa and K. Sycara. Conversational case-based planning for agent team coordination. In *Case-Based Reasoning Research and Development: Proceedings of the Fourth International Conference on Case-Based Reasoning (ICCBR 2001)*, volume 2080, pages 189–203, Berlin Heidelberg, July 2001. Springer-Verlag.

[5] J. A. Giampapa and K. Sycara. Team-oriented agent coordination in the RETSINA multi-agent system. Technical Report CMU-RI-TR-02-34, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213-3890, December 2002. Presented at AAMAS 2002 Workshop on Teamwork and Coalition Formation.

[6] R. Glinton, J. A. Giampapa, S. Owens, K. Sycara, C. E. Grindle, and M. Lewis. Integrating context for information fusion: Automating intelligence preparation of the battlefield. In *Proceedings of the* 5^{th} *Conference on Human Performance, Situation Awareness, and Automation Technology*, Daytona Beach, FL, 22 – 25 March 2004. to appear.

[7] R. Glinton, C. E. Grindle, J. A. Giampapa, M. Lewis, S. Owens, and K. Sycara. Terrain based information fusion and inference. In *Proceedings of the* 7^{th} *International Conference on Information Fusion*, Stockholm, Sweden, 28 June – 1 July 2004. to appear.

[8] Headquarters, Department of the Army, Washington, D.C. *Terrain Analysis*, July 1990. FM 5-33.

[9] Headquarters, Department of the Army, Washington, D.C. *Intelligence Preparation of the Battlefield*, 8 July 1994. FM 34-130.

[10] I. Karaesmen, P. Keskinocak, S. Tayur, and W. Yang. Scheduling multiple types of fractional ownership aircraft with crew duty restrictions. *Operations Research*, December 2003. Submitted.

[11] J. Manojlovich, P. Prasithsangaree, S. Hughes, J. Chen, and M. Lewis. UTSAF: A multi-agent based simulation bridge for distributed military simulation interoperability. *Simulation: Transactions of the Society for Modeling and Simulation International, Special Issue on Military Simulation Systems and Command and Control Systems Interoperability*, 2004. submitted.

[12] OneSAF Testbed Baseline (OTB). `http://www.onesaf.org`, accessed: January 2004.

[13] T. R. Payne, T. L. Lenox, S. Hahn, M. Lewis, and K. Sycara. Agent-based team aiding in a time critical task. In *Proceedings of Hawaii International Conference on System Sciences*, 2000.

[14] O. Shehory and K. Sycara. The RETSINA Communicator. In *Proceedings of the Fourth International Conference on Autonomous Agents (Agents 2000)*, 2000.

[15] K. Sycara, J. A. Giampapa, B. K. Langley, and M. Paolucci. The RETSINA MAS, a case study. In A. Garcia, C. Lucena, F. Zambonelli, A. Omici, and J. Castro, editors, *SELMAS*, volume LNCS 2603, pages 232–250. Springer-Verlag, 2003.