Simulating Teamwork and Information-Flow in Tactical Operations Centers using Multi-Agent Systems

Yu Zhang
Linli He
Keith Biggers
Dr. John Yen
Dr. Thomas R.. Ioerger
Department of Computer Science
Texas A&M University
College Station, Texas 77843-3112
979-845-5457
{yuzhang, linli, kbiggers, yen, ioerger}@cs.tamu.edu

Keywords: Teamwork, Multi-Agent Systems, Distributed Artificial Intelligence, Knowledge Acquisition, Communication

ABSTRACT: *Battlefield simulations are playing an increasing role in training within the military. This is especially true for training staff officers in tactical operations centers (TOCs) at intermediate echelons like battalions and brigades. Currently, distributed battlefield simulations such as ModSAF and JANUS provide semi-autonomous control of entities (e.g. tanks), and smaller units (up to company size). However, larger units, such as battalions, are difficult to simulate because of the high level of human decision-making (i.e. in their own TOCs), and the coordination and exchange of information within and between battle staffs. Teamwork-oriented skills are needed for simulating these high-level types of information-oriented behaviors. In the University XXI project, we are developing a multi-agent architecture, called TaskableAgents, which focuses on dynamic plan execution and monitoring through the application of procedural knowledge, coupled with rule-based inference. Separate agents can have unique goals and task definitions, along with their own knowledge bases (with dynamic state information); they can communicate by sending messages to each other. In this paper, we describe how decision-making and behavior of a battalion TOC can be simulated in TaskableAgents through a decomposition of the activities into multiple agents that represent various functional areas within a battle staff. Then we show how communication actions between staff members (within the battalion TOC, and also interactions with the brigade TOC) can be generated from common operating procedures (tasks and methods) for each functional area. Extending our agent-based TOC model to incorporate multiple agents working collectively as a team allows for a more realistic simulation of the complex behaviors of (and interactions with) aggregates like battalions. This is important for enhancing the capabilities of large-scale simulations with features that better support the teamwork-focused and cooperation-oriented training of staff officers at higher echelons.*

1. Introduction

Teamwork is an essential element of training. While soldiers and other members of the Armed Forces are taught basic skills for individual tasks, such as how to operate various pieces of equipment like communications devices, weapons, vehicles, etc., and procedures, such as making requests and reports or setting up or mobilizing camps, it is critical for them to learn how to bring these individual abilities together to produce coordinated teamwork that enables the unit to accomplish its mission. Each member typically plays a role in the team, and they must learn how their individual actions fit into the context of the whole team to be maximally effective. This requires that they have a chance to refine their communication and coordination skills, practice group decision-making procedures, and achieve a general understanding (mental model) of each others' responsibilities and capabilities. In the military, the team structure is very hierarchical and extends upward through many echelons, forming many levels of teamwork.

A particularly important example of teamwork is within tactical operations centers (TOCs). In mid-level echelons like battalions and brigades, TOCs consist of a variety of staff officers for areas including intelligence (S2), maneuvers (S3), fire-support (FSO), air-defense (ADO), engineering (ENGR), and logistics. Each of the staff members plays a unique role; however, their primary goal as a group is to support the decision-making of the commander. Individual activities include collecting information (about the enemy) and assimilating it into a common relevant picture (situation assessment), tracking friendly assets and combat strength, battle-damage

assessment, maneuvering to meet the OPFOR on favorable terrain, and maintaining supplies and security. However, there are many interactions among TOC staff officers that must be honed too, such as cooperation among the S2, S3, and FSO to prevent fratricide from indirect fire, engineering support to remove obstacles for friendly maneuvers or placing obstacles to impede the enemy and protect the troops, etc. Much of this relies on information flow within the TOC. For example, the S2 is responsible for identifying and reporting to the commander when decision points (DPs) and other priority information requirements (PIRs) have been reached, and this often involves working closely with recon and surveillance (R&S) elements. Information flow also occurs between staff officers at different echelons, such as submitting situation reports, battalion scouts interacting with the brigade reconnaissance team, locating obstacles placed by the enemy in the routes planned by the S3, or acting as forward observers for artillery, the S2s at brigade- and battalion-level collaborating to form the larger picture of the enemy's intent, and adjustment of priorities for fire-support (brigade asset) or close-air support (division or higher). These interactions must be practiced to ensure efficient operation and success of the unit in real battles.

In the past, this type of training (i.e. for teamwork) has been accomplished primarily through live exercises. For example, a commander might present a hypothetical tactical situation to his troops and ask them to go through the motions necessary to handle the situation. This approach requires an enormous investment of time and resources, especially for other role players (e.g. at higher or lower echelons, or enemy units) who are needed to make the training more realistic. More recently, constructive battlefield simulations such as ModSAF and JANUS have been used to run virtual scenarios in which the movement and contact of friendly and enemy forces on the ground can be simulated, allowing the commander's battle staff to practice their teamwork skills in handling a variety of challenging tactical situations without excessive cost or inconvenience. However, aggregate units, such as battalions, brigades, and divisions are difficult to simulate because of the high level of human decision-making (i.e. command and control), and the coordination and exchange of information within and between battle staffs. Fortunately, intelligent agent technology can be used to simulate these high-level types of information-oriented behaviors and complex decision-making. Agents are software processes that can utilize domain knowledge (e.g. strategies, effects of terrain on mobility, enemy weapons capabilities) to achieve goals that they are assigned (e.g. mission objectives, reconnaissance, attacks, defense, deterence). Intelligent agents have a great potential to improve the effectiveness for training in such environments, e.g. by making certain entities/units appear to have intelligent reactions (in the case of enemies) and interactions (like with neighboring friendly units).

For training staff officers, it is useful to simulate corresponding staff officers at higher, adjacent, and lower echelons, which we view as virtual team members. This is the goal of our University XXI project [1] [16]; we are building a digital battle-staff trainer (DBST) using distributed systems and intelligent agent methodologies, which link with OneSAF Testbed Baseline (OTB) as our underlying simulator. The interacting agents naturally form a multi-agent system (MAS), and, as in other MASs, they must have methods for coordinating, communicating, cooperating, resolving conflicts, etc. In this paper, we describe a way of decomposing the complex activities and teamwork of a battalion TOC staff into a set of tasks that individual agents can execute, with sufficient interactions built-in to produce collaborative behavior and flexibly carry out missions in a battlefield simulation. While the external behavior of battalions can potentially be simulated by other methods, our work has focused on reproducing the internal information-flow and distributed decision-making within the TOC, which is essential to generating the types of outputs (e.g. reports, communications, requests) that facilitate training of human staff officers who must learn how to interact and coordinate within them.

The rest of the paper is organized as follows. Section 2 describes the TaskableAgents Architecture, which focuses on dynamic plan execution and monitoring through the application of procedural knowledge, coupled with rule-based inference. In section 3, we extend our agent-based TOC model to incorporate multiple agents working collectively as a team through the use of a defined communication framework. The knowledge acquisition of TOC operations and the functional decomposition are described in detail in sections 4 and 5. Finally, we summarize our work and discuss issues to be addressed within future work.

2. TaskableAgents Architecture

The agents in our DBST are implemented in the TaskableAgents Architecture, which is an evolution from our previous research [1]. The TaskableAgents Architecture can be described in terms of the standard model of a situated agent interacting with its environment, in which the agent performs an iterative sense-decide-act loop [2]. The core of the TaskableAgents Architecture is a knowledge representation language called TRL (Task Representation Language), which is used for describing procedural information (e.g. about tasks and methods) needed by the agent. The application of this knowledge

to select actions for an agent is carried out by an algorithm called APTE (Adaptive Protocol for Task Execution), which serves as a kind of interpreter for TRL. We start by describing these components of the TaskableAgents Architecture from the perspective of individual agents, and then discuss its multi-agent extension.

2.1 Task Representation Language

The most unique feature of TaskableAgents is the way of encoding knowledge through the use of TRL [1]. TRL provides descriptors for representing four fundamental types of information: goals, tasks, methods, and operators. Goals represent conjunctive combinations of conditions that the agent is supposed to achieve, such as: attack the northern region and secure the area to the east. Tasks represent generic types of activities that the agent can engage in, such as attacking, reporting, maneuvering, intelligence gathering, etc. Tasks can have several different methods associated with them, and unique preference conditions can be used to characterize when it is most appropriate to use each method to carry out the task. Methods define specific procedures, encoded in an expressive process language with sequential, iterative, conditional, and parallel constructs. Examples of methods would be like the specific sequence of steps involved in securing an area, performing a search-and-rescue, or issuing a fragmentary order. The processes may refer to and invoke other tasks (as sub-tasks), or may ground out in operators, such as sending a message, requesting information, calling for fire, or moving units forward. These operators, which can be implemented in arbitrary Java code, might produce effects on the trainees' interfaces (e.g. displaying a request for support), they might directly talk to the simulation (move units in OTB), or they might be issued to a "puckster" as a surrogate to carry out the action.

While goals and operators are the focus of typical AI-based planning systems [3], in the TaskableAgents Architecture, greater emphasis is placed on encoding tasks and methods. These might be extracted from field manuals, documents analyzing staff functions, SOP's (standard operating procedures), TTP's (techniques, tactics, and procedures), METL's (mission essential task lists), etc. Much of this material is oriented toward helping staff officers understand what to do under various circumstances by providing doctrinal procedures to follow.

TRL uses a LISP-like syntax (i.e. with nested parentheses), though it does not rely on a LISP interpreter in any way. Each descriptor starts with a keyword, such as :TASK or :METHOD, a symbolic name, and a list of formal parameters. The parameters allow arguments to be passed in when a task is invoked, such as (Monitor unit-57). The task for Monitor might look like:

```
(:Task Monitor (?unit)
 (:Term-cond (destroyed ?unit))
  (:Method (Track-with-UAV ?unit)
    (:Pref-cond (not (weather cloudy))))
  (:Method (Follow-with-scouts ?unit)
    (:Pref-cond (ground-cover dense))))

(:Method Track-with-UAV (?unit)
 (:Pre-cond (have-assets UAV))
 (:Process
 (:seq
  (:if(:cond(not(launched UAV)))(launch UAV))
   (:let((x y)(loc ?unit ?x ?y))(fly UAV ?x ?y))
   (circle UAV ?x ?y))))
```

Notice that several types of conditions are referred to in this example. These are logical conditions that can be evaluated by an inference engine (the '?' prefix signifies variables). TaskableAgents uses a backward-chaining theorem prover implemented in Java, called JARE (Java Automated Reasoning System). A knowledge base of battlefield domain knowledge (e.g. definitions of unit types and organization, weapons capabilities, terrain effects, threat assessment, etc.) can be given in the form of Horn clauses (i.e. if-then rules), which JARE can use to determine whether conditions are satisfied as queries. Tasks and methods may both have termination conditions, which are capable of interrupting their operation whenever they become true. Methods may also define a set of pre-conditions, which state under what circumstance the method can be initiated; for example, Track-with-UAV would require access to UAV (Unmanned Aerial Vehicle) assets.

We have implemented a parser and graphical tool for displaying, browsing, and editing TRL files. The tool is able to draw graph-based visual representations of process-nets and task-method relationships, and allows browsing through a point-and-click interface. In addition to facilitating developers, this tool could also be useful to brigade staff trainees by showing rationale and explanations for recommended courses of action.

2.2 APTE Agent Algorithm

The tasks and goals assigned to the agent are carried out by the APTE algorithm (Adaptive Protocol for Task Execution). APTE can be thought of as a set of algorithms for operating on task-decomposition hierarchies (trees). Conceptually, there are two phases to APTE. In the very first time step of the simulation, APTE takes the top-level tasks given to the agent and expands them downward by: 1) selecting appropriate methods for tasks, 2) instantiating the process networks for selected methods, 3) identifying sub-tasks that could be taken as "first steps" (nodes in the network without predecessors), and recursively expanding these sub-tasks further

downward. Once this tree is fully expanded, APTE collects the set of all viable concrete actions (operators) that could be taken in the initial situation, selects one (possibly based on priority), and executes it. In each subsequent time step, APTE must repair the task-decomposition tree. This partly involves marking the action just taken and moving tokens forward in the corresponding process net. More importantly, APTE also re-checks each of the termination conditions associated with the tasks and methods in the tree. If a termination condition has been reached (indicating failure), APTE back-tracks and tries to find another method that would satisfy the parent task. If a task at some level has successfully completed, then a step forward can be taken in the process net of its parent.

Much of the intelligence in the TaskableAgents Architecture comes from: 1) reasoning about how to select the most appropriate method for any given task, and 2) being able to react to significant changes in conditions and find alternative methods when necessary. TRL is expressive enough to allow the specification of complex procedures for the agent to follow under nominal circumstances (reducing the cost of online plan construction), while the APTE algorithm enables flexible behavior in the form of reactivity to changes in conditions. It is important to note that the conditions in tasks and methods are evaluated dynamically as needed. For example, the preference conditions to select a method for a task are only evaluated at the time in the simulation when the task is invoked, not statically beforehand. This focus on dynamically selecting and managing procedures places the TaskableAgents Architecture to the area of "plan execution and monitoring" [4][5], as opposed to typical AI planning systems which often rely on goal-regression to select sequences of actions prior to the execution of any single step that are expected to achieve a set of goals; in environments with high uncertainty, planning for contingencies is necessary but difficult.

3. Agent Teamwork and Communication

Extending the TaskableAgents Architecture to incorporate multiple agents working collectively as a team allows for a more realistic model of the individuals the agents are simulating. A multi-agent system can be used in several ways in TOC simulations. First, an agent could play the role of each of the battalion staff members (S1, S2, S3, etc.). Secondly, agents could play adjacent battalions working under a single brigade. These agents must work together through the use of teamwork, proactively share information based on reasoning about each other's roles, beliefs, and responsibilities, and collectively and effectively work towards the common team goals. Extending the TaskableAgents Architecture to allow agents to work together in a team requires one major extension, a method of communication between agents. This feature added to the TaskableAgents Architecture allows for teamwork to be used by the agent teams and allows multiple agents to work together to accomplish collective goals.

Communication can be implemented quite easily within TRL. Communication can be broken down into three levels of complexity. The lowest level consists of primitive communication methods. Communication methods are the low-level implementations of the simple communication acts. These work in a similar fashion to those protocols described within the networking domain [6]. These methods include such things as *Broadcast-and-Wait*, *Broadcast-and-Do-No-Wait*, and *Broadcast-With-a–Time-Limit*. These can also be implemented for Multicast and Unicast alternatives. Communication methods are the ways in which agents can physically speak with each other. The second level is the simple communication acts (or performatives). There can exist many different alternatives of these basic acts, but they are based on three basic categories: 1) Inform, 2) Request, and 3) Query. KQML defines an extensive list of these performatives [7][8]. These second-level acts can all be built from the low-level communication methods. The level-two acts are a higher abstraction of the level-one acts, and define the various ways in which individuals can interact with each other. The third level consists of the more complex forms of communication. These include such things as synchronization, coordination, and negotiation. These complex communicative protocols require multiple level-two acts and require many interactions among different team members spanning across a period of time.

Communication is a powerful tool that allows agents to more efficiently and effectively work together as a team. Fundamentally, an agent is an active entity with the ability to perceive, reason, and act. The communication ability is part perception (the receiving of messages) and part action (the sending of messages) [9]. First, we introduce the basic communication protocol in TaskableAgents. Agents communicate by interchanging messages through the use of a SEND operator, a "MsgQueue" to store incoming messages, and the relevant message-handling methods in TRL.

A good communication framework is the main foundation on which a multi-agent system can be built. A SEND operator allows for basic message passing and information exchange between agents. Messages are stored in queues local to each agent and can be retrieved and handled through tasks and methods within the agent's TRL knowledge base. The syntax of the SEND operator is as follows:

(:operator SEND (?recipient ?message))

The first argument refers to the id of the receiver and the second argument is the message itself. The message can be any s-expression (i.e. a list of symbols and numbers enclosed in parentheses). The message type (for different functions, such as TELL, ASK, BID, etc.) can simply be prefixed as a symbol at the front of the list (message). The SEND operator, can be used as a basis to implement more elaborate communication protocols such as coordination, synchronization, and negotiation.

Each agent has a simple queue-based system for receiving messages. Other agents (or client processes such as the brigade trainee interfaces) can send messages to the agent. Messages automatically get stamped with the sender's id and time. Each iteration through the loop, the agent checks for messages, removes them from the queue, and asserts them into the agent's JARE knowledge base. The format of the asserted messages is a fact with a predicate name 'message', a message counter (initially 0), the sender's ID, the time, and finally the message. Visually:

(message [count] [sender] [time] [message])

Having the above communication protocol, we can design TRL methods to evaluate and handle messages. A procedure that runs in parallel with the agent's other tasks continuously monitors for new messages and handles messages as needed. The communication architecture for agents is depicted in Figure 1.

The dashed box is the internal structure of an agent which is communicating with other agents. Each agent has the same structure for communication purposes. The interactive methods include message sending methods and message receiving methods. Both are written in TRL. If the facts in the agent's knowledge base satisfy the conditions in rules that motivate sending message tasks, they will trigger the sending-message methods. Meanwhile, the message sending state (the message has been sent) will be asserted into the agent's knowledge base that prevents it from sending the same message repeatedly. If other agents send messages back to the agent, they will be stored in the MsgQueue and later asserted into the agent's knowledge base when the queue is emptied. Within TRL, there is a receiving-message task running in parallel, which checks for new messages in the agent's knowledge base. These messages will trigger the relative receiving-message methods, which will extract the facts contained within these messages and assert them into the knowledge base. Finally, the message is retracted from the knowledge base since it has been handled.

There is a simple, but general approach to implementing a TELL act in TRL which is built on top of the SEND operator and encapsulates it. T-Tell is a message-sending task that allows one agent to send a given piece of information to another. M-Tell is the method used to carry out the T-Tell task.

```
(:Method M-Tell (?receiver ?message)
 (:Process (:seq
  (SEND ?receiver ?sender (?message))
  (RETRACT (tell-state ?x))
  (ASSERT (tell-state done)))))
```

After sending the message, the old communication state (need to tell *p*) will be deleted and updated with the new state (told *p*).
Message-receiving methods are handled in a similar way in TRL. Messages, which are automatically retrieved from an agent's Message Queue and recorded in its knowledge base, are recognized by procedures such as the simple one that follows:

```
(:Method M-Receive ( )
 (:Process
   (:while (:cond (TRUE))
     (:if (:cond (message ?count ?sender ?time
                                    (tell ?what)))
       (:seq
        (ASSERT ?what)
        (RETRACT (message ?count ?sender ?time
                   (tell ?what))))))))
```

In this case, the TELL message is deleted from the knowledge base and the information it was carrying is asserted as a fact. More complex actions can also be incorporated. For example, if a message is received then the agent might forward it on to other agents and then send orders to its subordinates to perform a defined action.

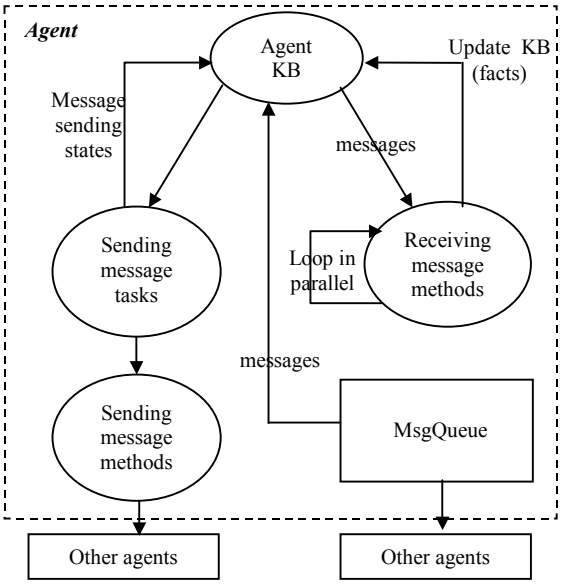

Figure 1 Multi-agent Communication Architecture

4. Knowledge Acquisition for TOC Operations and Tactical Decision Making

Previous research on knowledge acquisition has focused on improving and partially automating the acquisition of knowledge from human experts by knowledge engineers [10]. The goal is to construct an accurate and efficient formal representation of the domain expert's knowledge. It is one key step in building a knowledge base and is a pre-requisite to the knowledge engineering phase (i.e. transcribing knowledge into TRL and JARE). In University XXI, our knowledge acquisition effort sought to gain knowledge of both previous and ongoing staff training efforts, obtain domain knowledge of tactical Army operations, examine the various technical alternatives, and understand the flow of a typical tactical scenario over a duration of time that includes the major decision-making principles.

4.1 Knowledge Acquisition Approach

The knowledge acquisition methodology we adopted was the PARI methodology [11], which is a cognitive task analysis method incorporating features for several task analysis techniques, and a focus on a structured interview procedure for acquiring complex problem solving expertise from human experts. Because the PARI methodology was initially developed for acquiring expert knowledge for diagnostic problems whose solution is typically linking the source of a problem with its symptoms, we had to make several modifications to adapt it for our work. Our ultimate goal through the use of the PARI methodology was to obtain clear and accurate descriptions of the various tasks and then correctly model them in TRL.

During our knowledge acquisition, we were able to acquire a great deal of knowledge about Army organization, terminology, and the operational methods and procedures. We have spent a great deal of time examining field manuals, such as Staff Organization and Operations (FM 101-5 [12]), Tank and Mechanized Infantry Task Force (FM 71-2 [13]), The Armored and Mechanized Brigade (FKSM 71-2(2010) [14]), and The Armored and Mechanized Infantry Battalion Task Force (FKSM 71-2(2010) [15]). We then defined and categorized the major functional areas and the tasks performed by the various staff members within each area. We also had access to a retired military commander who we interviewed on several occasions to gain a better understanding of the information. To further understand the staff functions and interactions, we discussed and analyzed a variety of vignettes within a specific tactical environment (involving a hypothetical task force) that focused on high tempo events and significant information flow.

4.2 Tactical Operation Scenarios

The scenario we selected to study and build our knowledge from in University XXI was a heavy armor brigade movement-to-contact (MTC). The brigade is given the mission to attack to the north in a zone for a limited distance and restore the International Boundary (IB). For knowledge acquisition, we focused on the actions of a single task force (TF 1-22IN) in the MTC scenario. A tactical operation model was designed to interact upward with a brigade staff and downward with the sub-units of the battalion such as scouts and companies. Essentially, the tactical mission is to identify the enemy situation/intent and transition to a hasty attack or defense when the opportunity presents itself.

Figure 2 [16] portrays a partial virtual scenario that was used in knowledge acquisition. The scenario has been loaded into OTB, which provided management of many aspects of a digital force. The MTC has progressed to the point that the 1BCT, 4ID Recon troop is screening on south of the IB. Task Force scouts man OPs (observation posts) as well as TF 1-22 Main CP south of the IB too and companies have advanced to Phase Line BOB. Nine enemy units are approaching the TF 1-22 lead elements along roads south of IB. The main task for MTC is to establish contact with the enemy.

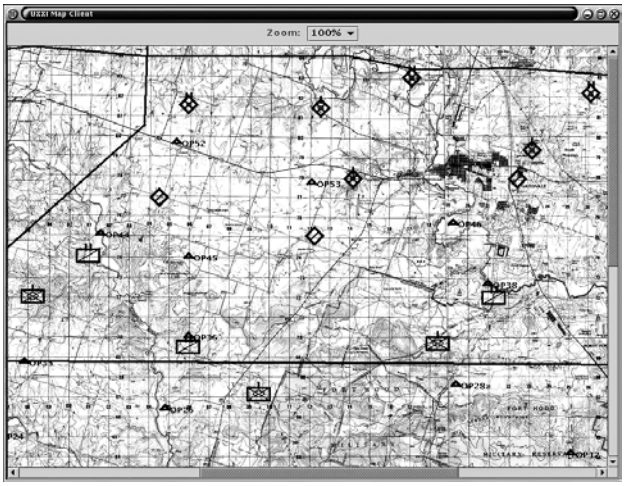

Figure 2 MTC Scenario Screen Shot

To adequately perform this task, the agents have to take into account the area of responsibility of the battalion, the avenues of approach (which is the direction and routes from which the enemy is expected to approach), and the engagement area (which is the optimum position on the avenues of approach for the units to coordinate their fire

upon). The agents have the capability to submit requests or reports to human brigade staff officer trainees, such as size, activity, location and time (SALT), intelligence summary (INTSUM), call for fire (CFF) and request for support (RFS) reports. A puckster serves as an intermediate actor between the agents and OTB to carry out needed actions within the simulation on behalf of the agents. The agents know the tactical setting such as terrain and battlefield geometry within their knowledge bases. Unit location updates are read in dynamically by monitoring entity state packets via distributed interactive simulation (DIS) PDUs. Figure 3 depicts a high level architecture of the interactions between OTB, agents, puckster and the brigade interfaces that are all distributed across multiple machines.

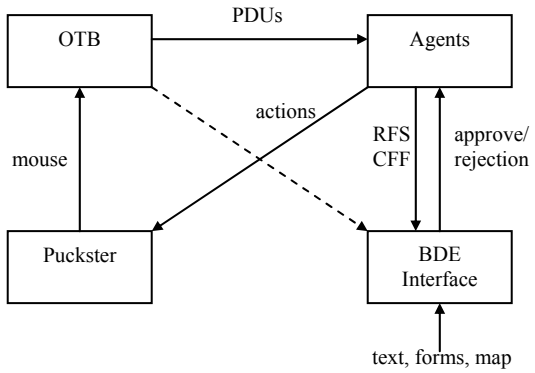

Figure 3 High-level Architecture of DBST

5. TOCs Officer Decomposition

We have applied the multi-agent extension of the TaskableAgents Architecture to decompose the complex activities within a battalion TOC into a set of tasks that individual agents can execute, along with the specific interactions among them based on needed information flow. There are many possible ways to decompose TOC activities, as opposed to having a single monolithic agent that makes all decisions and handles all external inputs and outputs. One possibility is to represent each of the battle-functional areas (BFAs) as an agent, e.g. one for security, one for maneuvering, one for reconnaissance etc. However, these do not map one-to-one onto staff officers, since some officers are involved in several functions, and the execution of some functions are distributed over multiple officers. Instead, we have chosen to represent each staff officer in the TOC as an agent. While this complicates the implementation (since decisions within one BFA often require interaction among multiple agents), it more faithfully represents the internal operation of real TOCs and therefore has greater utility for various training purposes (e.g. for training individuals by simulating the rest of the TOC with virtual staff officers).

Agents play the roles of battalion-level battle staff located in the tactical operations center. Our current work focuses on four specific staff officers: the commander, the S2 (intelligence officer), the S3 (maneuvers/operations), and the FSO (fire-support officer). Additionally, scouts and companies were represented by agents to facilitate dynamic information exchange and a natural battle environment. Each of these agents must be supplied with knowledge of the tasks they are supposed to carry out, and also how and when to interact with the other agents. Note that the basic actions (operators) that can be taken within this DBST system include:

1) issuing commands to a "puckster" to cause friendly units on the battlefield to move, fire, etc. in OTB,
2) sending messages to each other, such as commands and/or information (TELL actions)
3) sending messages to graphical interfaces seen by brigade staff trainees, such as requests for support or information, calls for fire, status/situation reports, etc.

Next, we examine the tasks and interactions for each of the agents that we have accumulated through our knowledge acquisition work and are encoded in TRL.

5.1 The Commander Agent

In a MTC scenario, the enemy positions are not known or are changing, and the battalion is moving into the area to make contact with them. In such scenarios, the commander has the responsibility of moving his units forward in the area of operations and deciding where and how to handle the contact. Within this command-and-control activity, the commander has ultimate decision-making authority. For example, one of his primary functions is to decide how to respond to threats. However, he does not personally keep track of all the details. Therefore, he must rely on the S2 to keep him informed of significant developments. For example, the commander agent allows the S2 agent to detect and assess the level of threats, which requires detailed analysis of enemy positions and movement, relative to terrain and friendly positions. Also, as in real TOCs, the commander may specify specific DPs or priority information requirements (PIRs). These are spelled out in the operations orders (OPORDs) specifically to let the S2 know what he (and his staff) should be looking for and reporting back to the commander. The point is that the commander agent does not have to have tasks/knowledge for determining these things, but must at least have tasks set up to wait for messages from the S2 regarding them and to then decide appropriate action.

There are a variety of possible responses to threats. When enemy location is reported within a Targeted Area of Interest (TAI), the FSO can call for indirect fire. If scouts or a company are nearby, they might use direct fire. If a scout/company spots an enemy, but is out of range for direct fire, they might request for mortar assets which requires a message to be sent to the FSO. When the threat level is high, the S3 may recommend committing its reserve company (if not already committed). If extra support is needed, the S2 could request for support (RFS) from brigade, possibly including FASCAM (family of scatterable mines) or CAS (close-air support). The appropriate response depends on the level of the threat, and also the current situation, so there is a lot of knowledge and decision-making involved. As we mentioned earlier, tasks are expanded by selecting the most appropriate method to accomplish it in a given situation. For instance, to get enemy information, the scout might move to an observation post (OP) close to the enemy. When there is high uncertainty about the enemy, the commander might request for information (RFI) from a BDE, which may in turn obtain it from a UAV. The following flow chart (Figure 4) describes the commander agent's decision-making process.

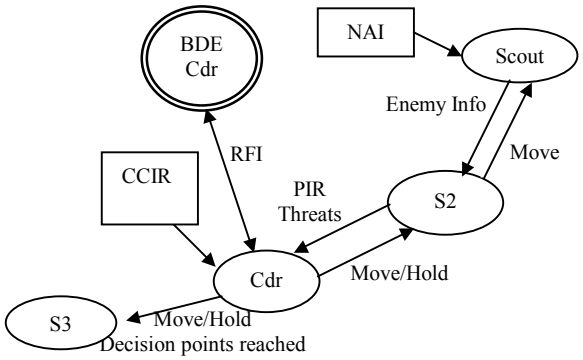

Figure 4. Commander Agent Information Flow

In addition to making decisions about how to deal with threats, the commander must also ensure that his battalion carries out its mission. Prior to contact, there is a great deal of time in which the battalion elements are not under threat. Under these conditions, it is the responsibility of the commander agent to keep his battalion moving forward (as part of the MTC). Again, he does not worry about specific route planning or destinations for specific units, which is delegated to the S3 agent, as described below. However, the commander agent is responsible for issuing the commands to move forward, hold, or halt. We model this in our commander agent as a simple three-state machine (finite automata). Under nominal conditions (no threat), he alerts the S3 to tell each battalion element to move forward along their respective routes, but if a threat occurs, he halts movement until the threat is adequately handled and then starts movement again. When the MTC is terminated, the state changes to halt.

5.2 The Scouts Agent

When the simulation starts, the enemy units are set in motion in OTB. They must be discovered and reported by the scouts (or their locations may be reported through brigade or division via higher sensor assests). The scouts essentially move according to some simple constraints: 1) stay behind the Brigade Recon Team (BRT), and 2) try to spread out horizontally across the zone.

Being the primary tool used for situation assessment, the scout team (we simulate multiple individual scout teams as a single agent) follows a brigade recon team forward through a collection of OPs commanded by the S2 and as described within the OPORD. Additionally, the scout team is typically instructed by the S3 to fire and destroy enemy reconnaissance teams as soon as they are identified. This is implemented as a simple task in TRL that runs in parallel with the others. As enemy units come into sight, scouts report the enemy as spotted to the S2. This message includes the enemy size, location, speed, direction, type, etc. The message will allow the S2 to recognize the current enemy situation and allows him to infer enemy intent. If the enemy force is stronger/larger than the scout team, and is in a Named Area of Interest (NAI which are key areas identified earlier), scouts can request for fire from the FSO. If it is also a Targeted Area of Interest (TAI), the FSO will put this request into a fire mission list including the information of the requester, target number, fire request type and request status. The scouts can also be asked to do battle damage assessment (BDA) to help with the situation assessment performed by both the BN TOC and BDE. When there is an obstacle on a road, the scouts will alert the S3 so he can re-route units or ask engineers to remove it.

5.3 The S2 Agent

The S2 agent (Figure 5) plays a key role in maintaining situation awareness as well as monitoring enemy locations, threats and actions. Potential enemy missions, intent, and avenues of approach have been identified previously (in the OPORD). Enemy forces can be detected by UAV, recon and scouts. To maintain situation awareness, the S2 is in charge of combining all of the information into a common and understandable picture that can be used to determine the proper course of action (COA). In some situations, the enemy's intent is not clear. The S2 can contact the BDE S2 for a better picture/understanding of what is happening. Also, in certain situations BDE may request information from the

BN S2, and thus he has to obtain and return the needed information.

To monitor the current enemy situation, the S2 tells scouts to move from OP to OP gradually moving closer to contact with the enemy. The scout agent infers (in JARE) its next OP based on its knowledge of the OPORD and the current situation. After combining enemy information detected by scouts and expected position, attributes and whether location known, the S2 tries to identify the enemy intent and calculate its level of threat. Currently we have three levels of threat: low, medium and high. The threat level is calculated based on enemy situation, size, movement patterns, etc. An enemy company or recon element is considered a low level threat. A lead battalion or higher is an example of a high level threat. If adjacent units are not protecting the battalion flank and rear, and the enemy mounts a flanking attack, then the threat level is marked as flanked which is an extreme threat. The threat level satisfying PIRs or DPs will trigger notifications to be sent to the commander agent. For example, in the TF 1-22 Operation Plan (OPLAN) within our scenario, PIR 3 is the location of the enemy BDE recon and lead battalion of the enemy 238th BDE. If known, DP 1, 2, 3 are reached. The S2 will alert the commander who could in turn shift the main effort, commit the reserved company, or ask for BDE support.

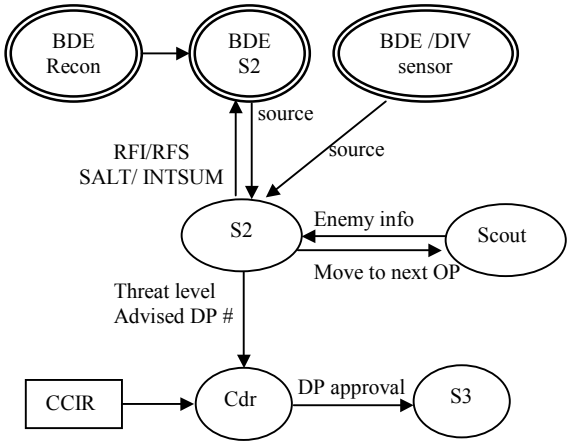

Figure 5 S2 Agent Information Flow

translates this into specific way-points along pre-planned routes for each subunit and tells them to go. The S3 will track the progress of all subunits and make acceptable situational adjustments accordingly. At the same time, the S3 maintains the terrian information in the battle field and gives it to the relevant subunits. For example, if a road or bridge that a unit will use has been destroyed by the enemy, the S3 will either ask engineer to deal with the obstacle or re-route the unit around it.

One of the most important responsibilities for the S3 is to respond to reached-DPs, which are determined by the S2 and approved by commander. If the S3 has been informed of the DP, it will perform relevant maneuvers according to the plan described in the OPORD. In our scenario, DP1 is to shift the main effort. There are three directions of main effort: left, center, and right. Once the main effort has been shifted, the S3 will tell subunits to move to their new locations, as well as alerting the FSO to shift the priority of fire to the new main effort area. DP2 is to commit the reserved company if not already committed. DP3 and DP4 will request for support (RFS) from the BDE S3, such as requesting commitment of the brigade reserve. If the S3 has found that the situation is critical, but it hasn't reached any DP, it will ask the commander to approve a new course of action (COA).

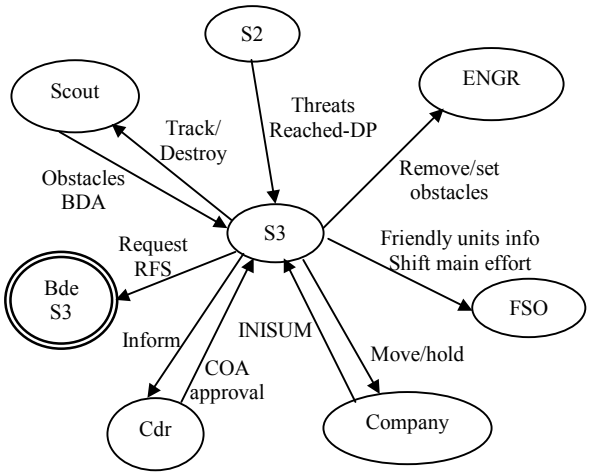

Figure 6 S3 Agent Information Flow

5.4 The S3 Agent

In order to carry out the commander's order to move, the S3 agent (Figure 6) is responsible for maintaining the current information about the friendly situation, and maneuvering the units as needed. First of all, the S3 is responsible for maneuvering all subunits in the battalion based on the planned progress of a mission. After receiving the commander's order to move, the S3

5.5 The FSO Agent

The FSO agent is shown in Figure 7. Usually the FSO doesn't perform any real fire actions because the real fire actions are excuted by the brigade artillery. In the beginning of the mission, prioriy of fire is asigned to areas of most probable approach by the enemy, based on analysis in the OPORD. After combining the information from other agents, the brigade shifts control of all

available fires to the observer who is in the best position to control fires against the enemy.

One of the most important responsibilities of the FSO is to respond to the CFF from the scout agent. If scouts identify that the enemy has entered an NAI, it will send a CFF to the FSO. After getting a CFF from scouts, the FSO will check whether this CFF matches a Targeted Area of Interest (TAI). The FSO plans TAI's based on key locations. A TAI is usually placed at key areas along avenues of approach, and can usually be observed by nearby OPs. The FSO checks the enemy information which comes from the S2 to fix the target, and they also check location of nearby friendly units (which comes from the S3) to avoid fratricide. Finally, it sends the relevant Fire Mission Combat Message to the BDE FSO.

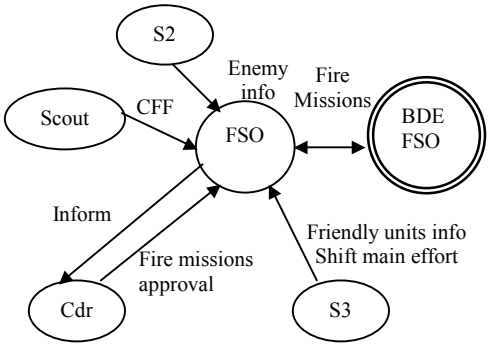

Figure 7 FSO Agent Information Flow

Another important responsibility of the FSO is to adjust the arrangement of the fire resources on the battlefield to the handle different situations throughout the development of the battle. The typical example of this is Shift Main Effort. When the S3 knows DP1 has been reached, he will tell the FSO to Shift the Main Effort, then the FSO will switch the priority of fire to the new main effort. Similarly, if there are some Courses of Action (COA) which involve the FSO, he will get the approval from BDE. We can view the FSO as a liaison for the fire support from the higher level TOC when the battlion needs that kind of support.

6. Conclusion

In this paper, we have presented a multi-agent architecture, called TaskableAgents. By extending the TaskableAgents Architecture to incorporate multiple agents working collectively as a team, we can simulate how staff officers in a battalion TOC make complex decisions that attempt to carry out their operation orders within the current situation. The battalion TOC officers are decomposed into individual agents that cooperatively handle battle-functional areas. Through the proper use of communication among individuals, more complex team behaviors such as coordination, cooperation, and synchronization can be exhibited.

The TaskableAgents Architecture could be used to simulate the intelligent behaviors of a wide variety of units in military combat simulations. We also are attempting to simulate the more complex decision-making aspects of such units, where they must follow orders under nominal conditions, yet react appropriately to unexpected situations as they arise. In the future, we will simulate higher-level decision-making, especially command and control (C2), which should facilitate more reactive behavior that is also more general. This requires performing actions to uncover the enemy's intent and choosing the appropriate COA accordingly. A step in this direction would be to try to implement a version of naturalistic decision-making [17]. This involves cognitively realistic methods for dealing with uncertainty, intelligence gathering, and situation assessment.

Acknowledgements

The authors would like to thank Dr. James Wall and Randy Elms in the Texas Center for Applied Technology on the Texas A&M University for their valuable input on this project. We also would like to acknowledge the cooperation of personnel at Ft. Hood.

References

[1] Ioerger, T. R., Volz, R. A., Yen, J., Modeling Cooperative, Reactive Behaviors on the Battlefield with Intelligent Agents, In *Proceedings of the Ninth Conference on Computer Generated Forces* (9th CGF): 13-23, 2000.

[2] Rao, A. S., Georgeff, M. P., BDI Agents: From Theory to Practice, *Proceedings of the First International Conference on Multi-Agent Systems*: 312-319, 1995.

[3] Georgeff, M. P., Planning, *Annual Review Computer Science*, 2:359-400, 1987.

[4] Ambros-Ingerson, J. A., Steel, S., Integrating Planning, Execution, and Monitoring, In *Proceedings of AAAI*: 83-88, 1998.

[5] Ingrand, F. F., Georgeff, M. P., Rao, A. S., An Architecture for Real-time Reasoning and System Control, *IEEE Expert* 7(6): 33-44, 1992.

[6] Tanenbaum, A., *Computer Networks*, Upper Saddle River, New Jersey. Prentice Hall. 7-9, 1996.

[7] Finin, T., et. al. 1994, KQML as an Agent Communication Language. In *Proceedings of the 3rd International Conference on Information and Knowledge Management* (CIKM94), ACM Press, 1994.

[8] Finin, T., Weber, J., et.al., Specification of the KQML Agent-Communication Language, DRAFT, 1994.

[9] Weiss, G. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, MIT Press, 1999
[10] Tecuci, G. *Building Intelligent Agents*, Academic Press, 1998.
[11] Hall, E. *et. al*., A Procedural Guide to Cognitive Task Analysis: The PARI Methodology, RPF F41624-93-R-1011, 1990.
[12] Field Manual No. FM 101-5, Staff Organization and Operations. Department of the Army, Washington, D.C., 1995.
[13] Field Manual No. FM 71-2, Tank and Mechanized Infantry Task Force. Department of the Army, Washington, D.C., 1995.
[14] FKSM 71-2(2010), The Armored and Mechanized Brigade, Department of the Army, Washington, D.C., 1993.
[15] FKSM 71-2(2010), The Armored and Mechanized Infantry Battalion Task Force. Department of the Army, Washington, D.C., 1993.
[16] Y. Zhang, T. R. Ioerger, *et. al*. A Distributed Intelligent Agent Architecture for Simulating Aggregate-Level Behavior and Interactions on the Battlefield, In *Proceedings of the 5th World Multi-conference on Systemics, Cybernetics and Informatics 2001*, To appear.
[17] Zsambok, C. & Klein, G., editors (1993), *Naturalistic Decision Making*, Ablex Publishing: Norwood, NJ

Author Biographies

YU ZHANG is a PhD student in the Department of Computer Science at Texas A&M University. She received her B.S and M.S degrees in Computer Science from Central South University in China in 1995 and 1998, respectively. Her research interests are in the area of artificial intelligence, intelligent agents and teamwork.

LINLI HE is a PhD student in the Department of Computer Science at Texas A&M University. She received her M.S degree in Computer Science from Southwest Petroleum Institute in China in 1999. Her research interests include intelligent agents, machine learning, data mining, fuzzy system, GIS (geographic information system), information retrieval, network security, and software testing.

KEITH BIGGERS is a Masters student in the Department of Computer Science at Texas A&M University. He received his B.S. degrees in Computer Science and Mathematics from Texas Wesleyan University (Fort Worth, Texas) in 1999. His research is in the area of teamwork and communication within intelligent agent teams, and modeling Command and Control (C2) within agents.

JOHN YEN received his B.S in Electrical Engineering from National Taiwan University, Taipei, Taiwan in 1980, his M.S. in Computer Science from University of Santa Clara, CA in 1982, and his PhD in Computer Science from the University of California, Berkeley in 1986. Dr. John Yen is currently a Professor of Computer Science and the Director of the Center of Fuzzy Logic, Robotics, and Intelligent Systems at Texas A&M University. Before joining Texas A&M University in 1989, he had been conducting AI research as a Research Scientist at Information Sciences Institute of University of Southern California. His research interests include intelligent agents, fuzzy logic, and evolutionary computing. He has published more than 100 technical papers in journals, conference proceedings, and edited volumes. Dr. Yen is the Vice President for Publication of IEEE Neural Networks Council. He is a member of Editorial Board of several international journals on fuzzy logic and intelligent systems. He co-authored the textbook (with R. Langari) Fuzzy Logic: Intelligence, Control, and Information published by Prentice Hall in 1999. Dr. Yen received an NSF Young Investigator Award in 1992 and he is a Fellow of IEEE.

THOMAS R. IOERGER is an Assistant Professor in the Department of Computer Science at Texas A&M University. He received his B.S. degree in Molecular and Cellular Biology in 1989 from the Pennsylvania State University, and his M.S. and PhD degrees in Computer Science in 1992 and 1996, respectively, from the University of Illinois.